UNIVERSITAT POLITÈCNICA DE VALÈNCIA

etsinf

Escola Tècnica Superior d'Enginyeria Informàtica

Escola Tècnica Superior d'Enginyeria Informàtica
Universitat Politècnica de València

Introduction to Deep Learning: a practical point of view

PROJECT WORK

Author: Piccione, Andrea

Tutors: Silla Jiménez, Federico
Reaño González, Carlos

Course 2017/2018

Abstract

Deep Learning is currently used for numerous Artificial Intelligence applications, especially in the computer vision field for image classification and recognition tasks. Thanks to the increasing popularity, several tools have been created to take advantage of the potential benefits of this new technology. Although there is already a wide range of available benchmarks which offer evaluations of hardware architectures and Deep Learning software tools, these projects do not usually deal with specific performance aspects and they do not consider a complete set of popular models and datasets at the same time. Moreover, valuable metrics, such as GPU memory and power usage, are not typically measured and efficiently compared for a deeper analysis.

This report aims to provide a complete and overall discussion about the recent progress of Deep Learning techniques, by evaluating various hardware platforms and by highlighting the key trends of the main Deep Learning frameworks. It will also review the most popular development tools that allow users to get started in this field and it will underline important benchmarking metrics and designs that should be used for the evaluation of the increasing number of Deep Learning projects. Furthermore, the data obtained by the comparison and the testing results will be shown in this work in order to assess the performance of the Deep Learning environments examined.

The reader will also deepen the following points in the next pages: a general Deep Learning study to acquire the main state-of-the-art concepts of the subject; an attentive examination of benchmarking methods and standards for the evaluation of intelligent environments; the personal approach and the related project realised to carry out experiments and tests; interesting considerations extracted and discussed by the obtained results.

Keywords: Deep Learning, Artificial Intelligence, Neural networks, Image Classification, GPU, Performance, Benchmark, Caffe, TensorFlow, PyTorch, CNTK, MXNet

Index of contents

Index of figures

CHAPTER 1
Introduction

1.1 Motivation

Nowadays our world is surrounded by technology and all the devices we use in our daily life have started to act intelligently. Everywhere there is a big talk about artificial intelligence since many applications have been developed and surprise us every single day with new features and capabilities. Several domains, from medicine to automotive and going through gaming (just to remark few examples), are affected by this phenomenon and it is logical to think that soon everything around us will be different but easier thanks to these improvements.

In the last few years, brand-new artificial intelligence methods and techniques have been developed and some of them have become very popular and well known for the huge advancements made in image and speech recognition, for their simplicity and general usability among beginners in programming and even for their unlimited and boundless possibilities. The mentioned reasons are a stimulus to study accurately and to understand in depth this field of interest, by analysing the means, both hardware and software, that allow this sort of revolution and foreseeing the possible future scenarios.

This motivation led me to discover the entire Artificial Intelligence methods and mostly one of its subareas, Deep Learning, which is considered one of the most fascinating topic in the entire Computer Science. It is no coincidence that Deep Learning related jobs are believed to be the most “attractive” among the companies and even universities carry out many researches in the field. So, gaining skills in this area can improve my professional possibilities in a very strong way, both in academia and industry. In addition, this opportunity of working with Deep Learning methods is meant to acquire a knowledge which was neither been taught in the previous studies nor personally known before this work. Certainly, the themes discussed require to be adequately proficient in Operating Systems (especially Linux) and have a basic knowledge of programming languages techniques (especially Python) and the courses studied in the last two years have been helpful without any doubt. Moreover, critical analysis was fundamental to examine the problem of benchmarking and review the pre-existent benchmarks.

1.2 Objectives

This work aims to provide valuable and useful data for comparing the performance of Deep Learning frameworks by using different hardware platforms. At the same time, a study of the technology and the interesting implications of the comparison will be carried on in order to master the main aspects of the domain and, thus, elaborate a personal project for this evaluation. The main contribution is to make available innovative testing results as a reference for other users to select the framework to adopt and the hardware configuration to build according to their preferences and circumstances. Furthermore, the projected performance model offers a starting point for further optimizations in system design and configuration for future researches.

However, there are already numerous available projects on the web which offer an assessment of hardware architectures and software frameworks related to Deep Learning. These existent works do not fully satisfy some requirements in terms of performance evaluation and they could seem to lack completeness and extensiveness on certain points of view since they do not take into consideration some metrics and parameters as much as they should.

1.3 Structure of the report

The proposed objectives will be pursued with an intended set of steps which follow the exact methodology used to carry out the whole work.

In Chapter 2, an introduction to Artificial Intelligence, Machine Learning and above all Deep Learning (the main topic of this work) will be carried out, with special attention to the problem of image recognition. The chapter will review the state of the art of the technology as well as the most important concepts of this area by providing the relevant context, background and glossary to understand the entire discussion which will be developed in the later chapters.

At that point, in Chapter 3, an analysis of the problem of benchmarking Deep Learning frameworks and hardware architectures will be provided: uppermost, a general overview will try to clarify all the possible variables, metrics and limitations of this activity in order to clearly understand the situation; afterwards, the pre-existing projects and solutions will be identified and studied in depth by pointing out advantages and disadvantages; eventually, a brief proposal of work will be provided with reference to the previous explained points.

After profiling the topic, in Chapter 4 the entire design and development of the personal benchmarking project will be fully and carefully described by illustrating the architecture and the technology, the software and the libraries, the models and the parameters and even the most relevant choices made in order to set the right values for a correct

evaluation. At the end of this section, the project structure and code implementation for the benchmark work will be also defined.

Later, in Chapter 5, the data and the results will be shown through tables and graphs in a sequential and logical way; also, a brief discussion for each figure will be put along to help the reader to “extract” the most important ideas and facts from the numbers. The chapter will end with an overview of the experiments with final considerations and a short final section dedicated to validation and testing of the correctness of data.

Finally, in Chapter 6, the conclusions of this work will be presented along with hints about how to extend it with future work.

CHAPTER 2
State of the art of Deep Learning

In this chapter, the principles and the basics of Deep Learning will be presented along with a brief introduction to the reference knowledge field, which includes the concepts of Machine Learning and Artificial Intelligence.

2.1 Artificial Intelligence

It has been always a goal to allow computers to model our world in order to show a certain form of intelligence and to perform various human tasks. Artificial Intelligence, a branch of Computer Science, aims to create intelligent machines as humans are. According to **John McCarthy**, the computer scientist who coined the term in the 1950s, Artificial Intelligence is «the science and engineering of making intelligent machines, especially intelligent computer programs» [15] that have the ability to achieve goals like humans do. Artificial intelligence studies how the human brain works and how people learn and take decisions when they face a problem: then these paradigms are applied to the development of intelligent systems.

The traditional problems of Artificial Intelligence include reasoning, knowledge representation, planning, learning, natural language processing, perception and it is a matter of fact that increasingly complex algorithms currently influence our lives and our civilization more than ever before. Artificial intelligence applications are everywhere and its possibilities are growing more and more thanks to recent improvements in computer technology: now some algorithms already surpass human abilities.

For instance, in the twenty-first century, Artificial Intelligence techniques had a renaissance following concurrent advances in computational power and large amounts of data; now, Artificial Intelligence methods are crucial for technology industry and for computer science as well. One of the most pervasive, attractive and potentially disruptive field of Artificial Intelligence which is growing more and more in the last few years is Machine Learning [26]

2.2 Machine Learning

Machine Learning, a subfield of Artificial Intelligence, was defined in 1959 by **Arthur Samuel** as a «field of study that gives computers the ability to learn without being explicitly programmed» [23]: a program, once created, will be able to learn how to do

some intelligent task “on its own”, differently from the classic way of programming. This contrasts with all those programs whose behaviour is defined point by point, explicitly and statically defined. An approach based on efficient Machine Learning algorithms opens new possibilities and advantages as well. In fact, instead of creating a distinct and specific program to solve each individual problem in a domain, the single Machine Learning algorithm simply needs to learn, through a process called “training”, to handle each new task.

A lot of researchers also think that this is the best way to make progress towards human level Artificial Intelligence and the increasing interest in this area has become more and more relevant.

There are several techniques to design and specify parameters for Machine Learning algorithms, known technically as **learning** or **training**.

The main learning techniques are:

- **unsupervised learning**, that is based on the ability to find patterns in a stream of input; no labels or targets are given to the learning algorithm, leaving it on its own to find structure in its input dataset;
- **supervised learning**, that includes both classification and numerical regression algorithms:
    - **classification** is used to determine which category something belongs in, after seeing numerous examples of things from several categories;
    - **regression** is the attempt to produce a function that describes the relationship between inputs and outputs and predicts how the outputs should change as the inputs change;
- **semi-supervised learning**, that is a mixture of the two above: the computer is given only an incomplete training set with several target outputs missing;
- **reinforcement learning**, that is based on rewards and punishments: the agent is rewarded for good responses and punished for bad ones; the agent uses this sequence of rewards and punishments to form a strategy for operating in its problem space.

Training is a compute intensive and supervised task: it deals with millions of parameters because it is about mathematical optimization of a cost function. It requires thousands of steps to achieve a correct value for the parameters, where the forward pass (making the computations starting from the data to the solution) and the backward pass (calculating the derivative of the cost function regarding each parameter) must be executed. Moreover, during the initial steps, the number of free parameters (or weights) are not known and must be found, which means that several training jobs must be carried out finding the best model [29].

2.3 Deep Learning

2.3.1. Basic concepts

Among the different Machine Learning methods, there is an area that is often referred to as “brain-inspired computation”: creating a program which reproduces some aspects from the way the brain works.

From a well-famous article,

> *Deep Learning enables computational models, composed of multiple processing layers connected by each other, to learn representations of data with multiple levels of abstraction. These methods have dramatically improved the state of the art in speech recognition, visual object recognition, object detection and also other domains and fields such as drug discovery and genomics.* [2]

Since the brain is currently the best “machine” we know for learning and solving problems, it seems logical to look at it for a Machine Learning approach. This contrasts with the attempt to create a brain, but rather the program aims to emulate how we understand the brain to operate. Such systems, called **Artificial Neural Networks**, "learn" how to do tasks by examples, generally without task-specific programming.

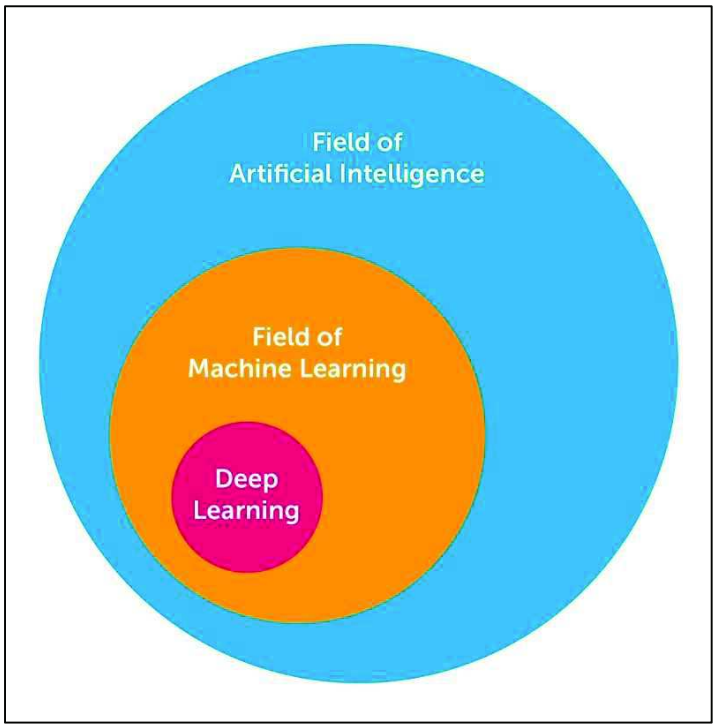


Figure 2.3.1.a - The relationship between Artificial Intelligence, Machine Learning, and Deep Learning [14]

Figure 2.3.1.a shows a simple scheme to illustrate the relations among Artificial Intelligence, Machine Learning and Deep Learning.

In a sense, Artificial Neural Networks "learn by example" as do their biological counterparts. Artificial Neural Networks (ANNs) are computer programs designed to simulate the way in which the human brain works. An artificial neuron is composed of a set of weighted inputs, a transformation and an activation function. The connections of the biological neuron are modelled as weights and all inputs are modified by a weight and summed. The activation function would be the axon of a biological neuron, while the weighted inputs would be electrical impulses which move through the brain to transmit the signal to subsequent layers of neurons. The connections of the biological neuron are modelled as weights. In Figure 2.3.1.b, the structure of an Artificial neuron is shown.

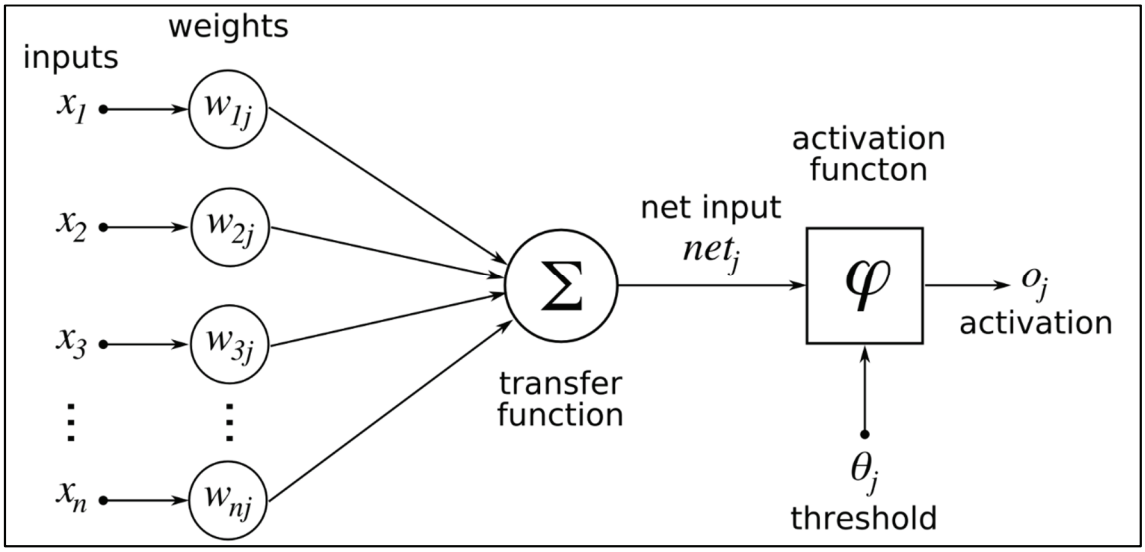


Figure 2.3.1.b – Artificial neuron structure and composition [25]

An ANN is formed from hundreds of single units, artificial neurons or processing elements (PE), connected with coefficients (weights), which constitute the neural structure and are organised in layers. The power of neural computations comes from connecting neurons in a network. Each PE has weighted inputs, transfer function and one output. The behaviour of a neural network is determined by the transfer functions of its neurons, by the learning rule, and by the architecture itself. The weights are the adjustable parameters and, in that sense, a neural network is a parameterized system. The weighted sum of the inputs constitutes the activation of the neuron. The activation signal is passed through transfer function to produce a single output of the neuron. The transfer function introduces non-linearity to the network [1].

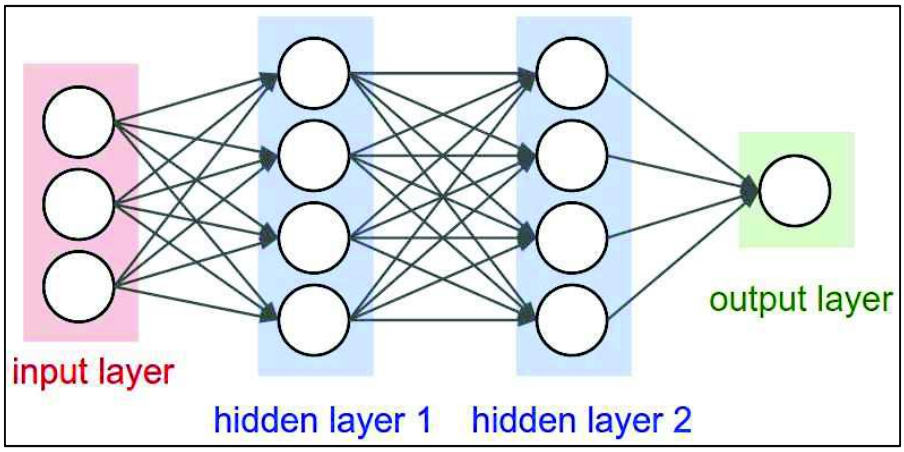


Figure 2.3.1.c – Input, hidden and output layer in Artificial Neural Networks [9]

In this domain of interest, there are networks, known as Deep Neural Networks (DNNs), that have more than three layers (including input and output), that means more than one hidden layer (a layer between the input and output layer), as shown in Figure 2.3.1.c. This is the so called **Deep Learning**. Today, the typical numbers of network layers used in Deep Learning range from five to more than a thousand layers.

When training a network, data is given to the first layer of the network and neurons assign a weighting to the input (that is, how correct or incorrect it is) according to the task to perform. The learning activity, which is crucial to perform tasks, involves determining the value of the weights in the network. Once trained, the program can perform its task by computing the output of the network using the weights determined during the training process.

> *The gap between the ideal correct scores and the scores computed by the DNN based on its current weights is referred to as the loss (L). Thus, the goal of training DNNs is to find a set of weights to minimize the average loss over a large training set* [3].

During the process of training, usually the weights need to be updated using the **gradient descent**: a multiple of the gradient of the loss relative to each weight (the partial derivative of the loss with respect to the weight) is used to update the weight. **Backpropagation** is the process to efficiently compute the partial derivatives of the gradient. It operates by passing values backwards through the network in order to compute how the loss is affected by each weight.

Deep learning discovers intricate structures in large data sets by using the backpropagation algorithm to indicate how a machine should change its internal parameters that are used to compute the representation in each layer from the representation in the previous layer [2].

The training process is affected by several parameters, the most significant ones are:

- the **training phase** is composed of several passes throughout the entire dataset and this is done more and more times in order to find the right weights;
- an **epoch** is one forward pass and one backward pass of all the training examples, i.e. one entire processing of all the images of the dataset. Usually several epochs are needed to obtain a high accuracy;
- a **batch size** is the number of training examples in one forward/backward pass. To note that the higher the batch size, the more the required memory space;
- the **number of iterations** is equal to the number of passes and each pass uses a specific number of examples according to the batch size.

A popular variant of DNNs are **Convolutional Neural Networks** (CNN), a class of deep, feed-forward artificial neural networks, so called because includes **convolutional layers**, that calculate weights by using only some input activations. They have a huge impact in several applications for processing images, video, speech and other audio.

Most modern CNNs use variations of the same 3 types of layers:

- **convolutional layers** take an image and perform several spatial transforms (convolutions), resulting in several transformed images of same size as the original, or slightly smaller depending on which convolution method is being used;
- **pooling layers** reduce data size, resulting in fewer computations in subsequent layers and introduce some location invariance;
- **fully connected layers** are traditional neural network layers where all nodes are connected with weights to all nodes in the next layer.

The applications domains where Deep Learning is today successful are:

- State of-the-art in speech and language recognition
- Gaming
- Robotics
- Visual object recognition
- Object detection
- Many other domains such as drug discovery and genomics

2.3.2. Brief History

The first traces of Deep Learning can be found in the **Rosenblatt's perceptron** (1957), a probabilistic model which aimed to simulate the learning mechanisms of the brain (in fact, it was called a "brain model"). The main idea was to recognize many objects

without storing information about these objects. The first version of this model was a learning system of 3 layers: the original deep learning idea was to have multiple association layers to describe more complex objects. This idea was rejected because it was considered too complex. In recent years, with much faster computers and more training data, Deep Learning has become again a reality and the first big advancement was the LeNet model.

> *Perhaps the most well-known example is **LeNet 5 by LeCun** in the late 90's, often referred to as **MNIST**[1] because of the dataset they were using. LeNet 5 is a deep Convolutional Neural Network (CNN) with 7 layers (2 convolutional, 2 pooling, and 3 fully connected) and 60,000 trainable parameters. It resulted in a commercial implementation for reading handwritten bank checks* [13].

The next big advancement in deep CNNs came in the 2010 and this was due to three factors:

- the first factor is the amount of available information to train the networks: learning requires a large amount of training data and the big companies have a huge amount of data to train their algorithms;
- the second factor is the increasing computing performance available: computer architecture advances have continued to provide increased computing capability, which is required for both inference and training, and learning can be performed in a reasonable amount of time: this is especially true for GPUs[2] that, thanks to their much greater power, are able to perform far better than CPUs in these kinds of tasks. The increasing computational techniques have also inspired the development of several frameworks that make it even easier for researchers and practitioners to explore and use DNNs;
- the third factor is the evolution of the algorithmic techniques that have improved application accuracy significantly, giving the possibility to apply DNN to different domains with good results.

An excellent example of the successes in Deep Learning can be illustrated with the **ImageNet Challenge**. This challenge is a contest that involves several different areas. One of the areas is an image classification task where algorithms must accurately identify

[1] MNIST stands for Modified National Institute of Standards and Technology and is the de facto "hello world" dataset of computer vision.

[2] A Graphics Processing Unit (GPU) is a specialized electronic circuit designed to rapidly manipulate and alter memory to accelerate the creation of images in a frame buffer intended for output to a display device [28]

objects in a test set of images, that it has not previously seen. The training set consists of 1.2 million images, each of which is labelled with one of 1000 object categories.

It needs to be mentioned **AlexNet by Krizhevsky,** a CNN for image recognition that won the yearly **ILSVRC** (Imagenet Large Scale Visual Recognition Challenge) in 2012. AlexNet consists of 11 layers (5 convolutional, 3 pooling and 3 fully connected) and 60 million trainable parameters. In order to train that many parameters, two GPUs were used for 5-6 days. The network was trained on ImageNet. This was a sort of revolution for computer vision, making a shift from traditional image classification methods to various and modern deep CNN architectures, which started to be used exclusively for training neural networks. In fact, it is not a case that these kinds of CNN have won ILSVRC every year since [12]

In addition to convolutional, pooling and fully connected layers, many competitors in ILSVRC add **new types of layers** to achieve various advantages. For example, AlexNet implements a dropout layer which randomly excludes parameters from being used during training. In conjunction with the trend to Deep Learning approaches for the ImageNet Challenge, there has been a corresponding increase in the number of entrants which started to use **GPUs** for training. From 2012 when only 4 entrants used GPUs to 2014 when almost all the entrants (110) were using them. This reflects the almost complete switch from traditional computer vision approaches to Deep Learning-based approaches for the competition.

In 2015, the ImageNet winning entry, **ResNet**, exceeded human-level accuracy and, since then, the error rate has dropped below 3% and more attention is now being placed on more challenging components of the competition, such as object detection and localization. These successes are clearly a contributing factor to the wide range of applications to which DNNs are being applied.

2.3.3. Main types of Deep Neural Networks

There is a wide variety of shapes and sizes for DNNs: this depends on the application they are applied to. The popular shapes and sizes are also evolving rapidly to improve accuracy and efficiency. In all cases, the input to a DNN is a set of values representing the information to be analysed by the network. For instance, these values can be the pixels of an image. The networks that process the input come in two major forms: feed-forward and recurrent.

In **Feed-Forward Neural Networks** (FFNN) all the computation is performed as a sequence of operations on the outputs of a previous layer and the direction is always the same as shown in Figure 2.3.3.a. The final set of operations generates the output of the network, for example a probability that an image contains a particular object, the probability that an audio sequence contains a particular word, a bounding box in an image around an object or the proposed action that should be taken. In such DNNs, the network

has no memory and the output for an input is always the same irrespective of the sequence of inputs previously given to the network.

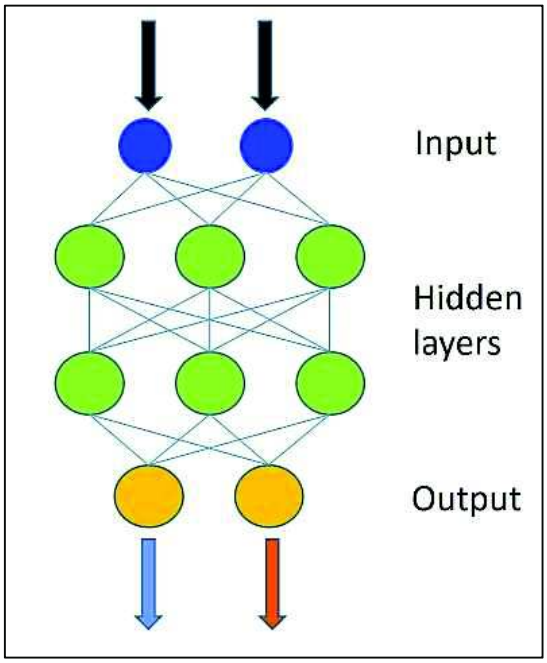


Figure 2.3.3.a – In a Feed Forward Network information always moves one direction [24]

In contrast, **Recurrent Neural Networks** (RNNs), of which Long Short-Term Memory networks (LSTMs) are a popular variant, have internal memory to allow long-term dependencies to affect the output. In these networks, some intermediate operations generate values that are stored internally in the network and used as inputs to other operations in conjunction with the processing of a later input as shown in Figure 2.3.3.b.

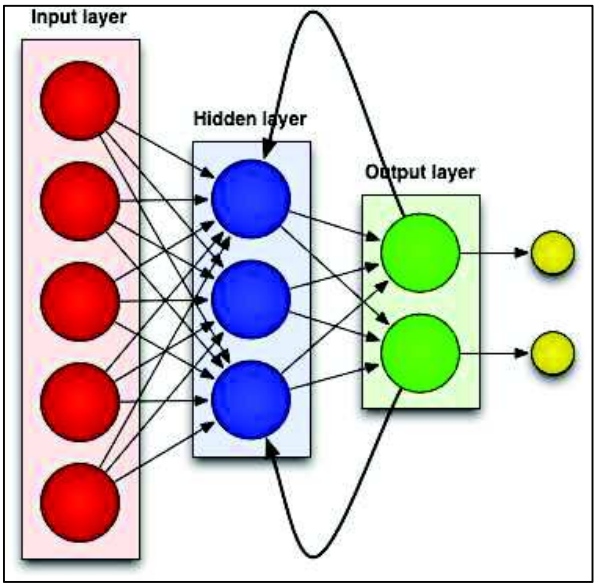


Figure 2.3.3.b – In a recurrent network, information comes back to previous layers [20]

DNNs can be composed only of **fully-connected layers (FC layers)**. In a FC layer, all outputs are connected to all inputs. This requires a significant amount of storage and computation but they offer learns features from all the combinations of the features of the previous layer.

In many applications, some connections between the activations can be removed by setting the weights to zero without affecting accuracy. This results in a **sparsely-connected layer**.

2.3.4. Popular Neural Networks

Lots of DNN models have been developed over the past few years. Each of these models has different specifications, such as a different architecture (number, type, shape and connections of layers), and several variations. In this section we provide an overview of some popular DNNs, which have been used for benchmarking and analyse performance in this case of study.

LeNet was introduced in 1989 and it was one of the first CNNs. It was designed for the task of handwritten digit classification.

> *The most known version, **LeNet-5**, contains two CONV layers and two FC layers. Each CONV layer uses filters of size 5×5 (1 channel per filter) with 6 filters in the first layer and 16 filters in the second layer. […] In total, LeNet requires 60k weights and 341k multiply-and-accumulates (MACs) per image. LeNet led to CNNs' first commercial success, as it was deployed in ATMs to recognize digits for check deposits* [3].

AlexNet, which has been already introduced before, is described in detail in the well-known paper *ImageNet Classification with Deep Convolutional Networks* [12] which is considered one of the most important Deep Learning publications. The network won the 2012 ILSVRC (ImageNet Large-Scale Visual Recognition Challenge).

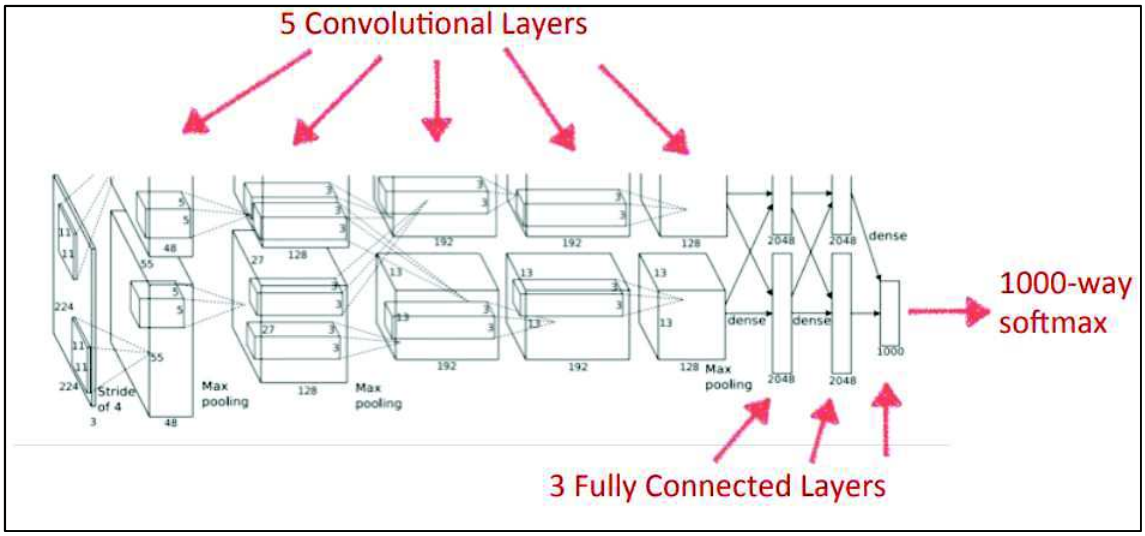


Figure 2.3.4 – AlexNet architecture [10]

As shown in Figure 2.3.4,

> *It consists of five CONV layers and three FC layers. Within each CONV layer, there are 96 to 384 filters and the filter size ranges from 3×3 to 11×11, with 3 to 256 channels each. In the first layer, the 3 channels of the filter correspond to the red, green and blue components of the input image. A ReLU (rectified linear unit)[3] is used in each layer. Max pooling of 3×3 is applied to the outputs of layers 1, 2 and 5. To reduce computation, a stride of 4 is used at the first layer of the network. […] One important factor that differentiates AlexNet from LeNet is that the number of weights is much larger and the shapes vary from layer to layer. To reduce the amount of weights and computation in the second CONV layer, the 96 output channels of the first layer are split into two groups of 48 input channels for the second layer, such that the filters in the second layer only have 48 channels. Similarly, the weights in fourth and fifth layer are also split into two groups. In total, AlexNet requires 61M weights and 724M MACs to process one 227×227 input image* [13].

VGG is a Convolutional Neural Network from the University of Oxford. It was first introduced in the paper *Very Deep Convolutional Networks for Large-Scale Image Recognition* [22].

[3] In the context of Artificial Neural Networks, the rectifier is an activation function defined as the positive part of its argument.

> *VGG-16 has 16 layers consisting of 13 CONV layers and 3 FC layers. In order to balance out the cost of going deeper, larger filters (e.g., 5×5) are built from multiple smaller filters (e.g., 3×3), which have fewer weights, to achieve the same receptive fields. As a result, all CONV layers have the same filter size of 3×3. In total, VGG-16 requires 138M weights and 15.5G MACs to process one 224×224 input image. VGG has two different models: VGG-16 (described here) and VGG-19* [3].

GoogLeNet was the winner of ILSVRC 2014 and was one of the first CNN that differentiated from the general approach of simply use convolutional and pooling layers on top of each other.

> *It introduced an inception module which is composed of parallel connections, whereas previously there was only a single serial connection. Different sized filters (i.e., 1×1, 3×3, 5×5), along with 3×3 max-pooling, are used for each parallel connection and their outputs are concatenated for the module output. Using multiple filter sizes has the effect of processing the input at multiple scales. For improved training speed, GoogLeNet is designed such that the weights and the activations, which are stored for backpropagation during training, could all fit into the GPU memory. In order to reduce the number of weights, 1×1 filters are applied as a 'bottleneck' to reduce the number of channels for each filter. The 22 layers consist of three CONV layers, followed by 9 inceptions layers (each of which are two CONV layers deep), and one FC layer* [3].

This model has improvements in terms of memory and power usage respect to older networks. There are several versions of GoogLeNet (also known as Inception): v1, v3 and v4.

> *Inception-v3 decomposes the convolutions by using smaller 1-D filters to reduce the number of MACs and weights in order to go deeper to 42 layers. […] Inception-v4 uses residual connections for a 0.4% reduction in error* [3].

ResNet was projected by Microsoft engineers and has 152 layers, so an incredibly depth architecture. ResNet won ILSVRC 2015 and set new records in several image classifications tasks.

> *Residual net introduces a 'shortcut' module which contains an identity connection such that the weight layers (i.e., CONV layers) can be skipped. […] It was the first entry DNN in ImageNet Challenge that*

> *exceeded human-level accuracy […]. ResNet-50 consists of one CONV layer, followed by 16 shortcut layers (each of which are three CONV layers deep), and one FC layer; it requires 25.5M weights and 3.9G MACs per image. There are various versions of ResNet with multiple depths (e.g., without bottleneck: 18, 34; with bottleneck: 50, 101, 152). The ResNet with 152 layers was the winner of the ImageNet Challenge requiring 11.3G MACs and 60M weights* [3].

As it can be easily guessed, the trend of all these popular DNNs was to increase the depth of the network in order to provide higher accuracy. Furthermore, most of the computation has been placed on CONV layers rather than FC layers. In addition, the number of weights in the FC layers is reduced. Thus, the focus of hardware implementations should be on addressing the efficiency of the CONV layers, which in many domains are increasingly important.

2.3.5. Deep Learning frameworks

In the last few years, with the increasing popularity and attention for Deep Learning, several Deep Learning frameworks have been developed from various sources. These open source libraries contain software libraries for DNNs.

Caffe is an open source framework and was developed in 2014 by the Berkeley Vision and Learning Center (BVLC); it supports C, C++, Python and MATLAB. There are several variants of the project, the most famous are NVIDIA Caffe (optimized for GPU training) and Intel Caffe (optimized for CPU training).

Caffe2 is the designated successor of Caffe and is developed by the Artificial Intelligence Department of Facebook. It is based on the old Caffe project but more focused on some features for new intelligent applications, above all mobile ones.

TensorFlow was released by Google in 2015. It supports many up-to-date networks such as CNNs with different settings and was mainly designed for flexibility and portability. It supports C++ and Python.

Torch is another popular framework and was created by Facebook and the New York University and supports Lua as the main programming language. Recently, a Python-version of Torch called PyTorch was developed and quickly acquired lots of users for the familiarity with a very common language like Python is.

CNTK is a unified computational network toolkit developed by Microsoft Research.

Apache MXNet is an open-source framework and has become very well-known in the last few years. It supports several languages.

There are also **higher-level libraries** that can run on top of the mentioned frameworks to provide a more universal experience and faster development. **Keras,** written in Python and supporting TensorFlow, CNTK and Theano, is an example of this category.

> *The existence of such frameworks is not only a convenient aid for DNN researchers and application designers, but they are also invaluable for engineering high performance or more efficient DNN computation engines. In particular, because the frameworks make heavy use of a set primitive operations, such processing of a CONV layer, they can incorporate use of optimized software or hardware accelerators. This acceleration is transparent to the user of the framework. Thus, for example, most frameworks can use Nvidia's cuDNN library for rapid execution on Nvidia GPUs* [3].

2.3.6. The importance of GPU training and CUDA

To process the data from scratch, neural networks need to do a lot of work and deal with a great amount of information. There are basically two ways to do so: with a CPU (Central Processing Unit) or a GPU (Graphical Processing Unit).

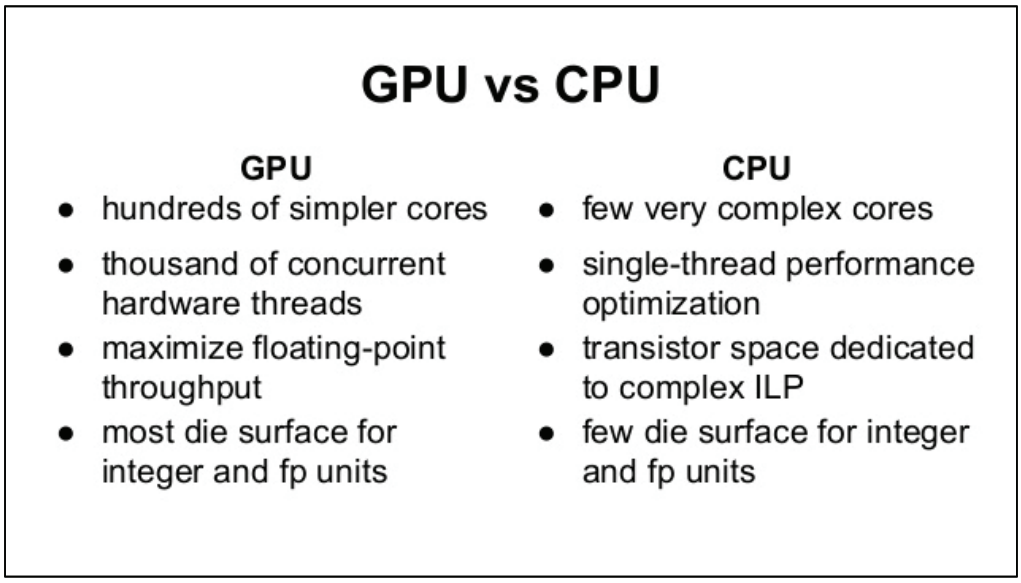


Figure 2.3.6 – High-level Comparison between CPU and GPU [21]

CPU is designed to do fast computation on a quite small amount of data due to their configuration as shown in Figure 2.3.6; for instance, multiplying some numbers on a CPU is fast but when a considerable amount of data is given (like multiplying very large matrices), CPU is not efficient in these types of tasks. And Deep Learning mostly deals with

operations like matrix multiplication which are very computationally expensive. Indeed, support of multiple GPUs has become a standard in recent Deep Learning tools.

> *Basically, a GPGPU is a parallel programming setup involving GPUs and CPUs which can process and analyse data in a similar way to image or other graphic form. GPGPUs were created for better and more general graphic processing, but were later found to fit scientific computing well. This is because most of the graphic processing involves applying operations on large matrices. The use of GPGPUs for scientific computing started some time back in 2001 with implementation of Matrix multiplication. One of the first common algorithm to be implemented on GPU in faster manner was LU factorization in 2005. But, at this time researchers had to code every algorithm on a GPU and had to understand low level graphic processing* [21].

In 2006, Nvidia created a high-level language to write programs from graphic processors, CUDA. This was probably one of the most noteworthy changes in the way users interacted with GPUs.

NVIDIA CUDA is a parallel computing platform and application programming interface (API) model created by NVIDIA. It allows to use a CUDA-enabled GPU for general purpose processing. With CUDA, developers are able to dramatically speed up computing applications by using the power of GPUs [17].

Thanks to this, researchers and scientists can significantly speed up deep learning training, otherwise the activity could take several hours or even days. Users can use GPU-accelerated platforms to have high-performance for the most computationally-intensive deep neural networks.

There was also a huge development of software libraries (e.g. cuDNN), both in academies (e.g. Berkeley, NYU) and industries (e.g. Nvidia): the most relevant and worth to mention are NCCL and cuDNN.

The **NVIDIA Collective Communications Library (NCCL)** «implements multi-GPU and multi-node collective communication primitives that are performance optimized for NVIDIA GPUs» [18].

The **NVIDIA CUDA Deep Neural Network library (cuDNN)** «is a GPU-accelerated library of primitives for deep neural networks. cuDNN provides highly tuned implementations for standard routines such as forward and backward convolution, pooling, normalization, and activation layers» [19].

In summary, all the frameworks mentioned in the previous section are very fast in training complex DNNs, thanks also to their strong CUDA backends since nowadays every model is trained by using Graphical Processing Units (GPUs).

2.3.7. Popular Datasets for image classification

The comparison of different DNN models must take into account the difficulty of the task. For instance, the task of classifying handwritten digits from the MNIST dataset is much simpler than classifying an object in ImageNet. For instance, LeNet is designed for digit classification, while AlexNet, VGG16, GoogLeNet and ResNet are designed for the 1000-class image classification. Public datasets are important for comparing the accuracy of different approaches. The most common task is image classification. Others, for example, are localization or detection.

MNIST is a widely used dataset for handwritten digit classification. It was introduced in 1998 and it consists of 10 classes, 60,000 training images and 10,000 test images[4] (28×28-pixel grayscale images of handwritten digits). «LeNet-5 was able to achieve an accuracy of 99.05% when MNIST was first introduced, nowadays the accuracy has increased to 99.79% using other techniques» [3].

CIFAR is a dataset which was released in 2009. It consists of 10 mutually exclusive classes, 50,000 training images and 10,000 test images (32×32-pixel coloured images of various objects). «A two-layer convolutional deep network was able to achieve 64.84% accuracy on CIFAR-10 when it was first introduced and since then the accuracy has increased to 96.53%» [3]

ImageNet is a large-scale image dataset that was introduced in 2010; the dataset became stable in 2012. There are 1.3M training images, 100.000 testing images and 50.000 validation images (256×256 pixel in colour with 1000 classes). There is a hierarchy for the ImageNet categories and the classes were selected so that there is no overlap.

In summary, MNIST (more) and CIFAR10 (less) are fairly easy datasets, while ImageNet is challenging for the number of classes. Therefore, it is important to consider the dataset every time it must be evaluated the performance of a network.

2.4 Goal of this work

After providing the reader with this necessary amount of information about Deep Learning state of the art, we briefly illustrate the aim and the structure of this work before going on: the analysis will focus on **3 metrics**:

- average **speed** for training;
- average **GPU power** usage;

[4] Training images are a set of images used for training the network. Test images are used instead for testing the performance and the accuracy of the network.

- average **GPU memory** usage;

and will be centered in the evaluation of the following hardware/software configurations:

- **5 networks:** LeNet, AlexNet, ResNet50, GoogLeNet, VGG16;
- **3 datasets:** MNIST, CIFAR10, ImageNet;
- **5 frameworks:** Caffe, TensorFlow, PyTorch, MXNet, CNTK;
- **6 hardware configurations:** CPU with 12 cores, CPU with 20 cores, 1 K20m GPU, 2 K20m GPUs, 1 P100 GPU, 2 P100 GPUs.

CHAPTER 3
Analysis of Deep Learning benchmarking methods

After reviewing the main Deep Learning concepts, this section is a theoretical introduction to Deep Learning benchmarks and it deals with the different approaches for evaluating an intelligent environment.

3.1 The problem of benchmarking

Several Deep Learning frameworks are now used by a wide range of people, from researchers to regular users, because their features and capabilities easily adapt to different scenarios, allowing efficient and fast training of deep networks especially through GPUs. Developers have constantly improved these frameworks by adding more and more features and performance improvements to attract more people. Recently, the efficacy of several Deep Learning frameworks has been evaluated in quite a lot of aspects. In particular:

- **Extensibility**: their capability to incorporate different types of Deep Learning architectures (convolutional, fully-connected and recurrent networks), different training procedures (unsupervised, layer-wise pre-training and supervised learning), and different convolutional algorithms (e.g. FFT-based algorithm);
- **Hardware utilization**: their efficacy to incorporate hardware resources in either (multi-threaded) CPU or (multi or single) GPU settings;
- **Performance**: their speed from both training and deployment perspectives.

Lots of researches have been carried on efficient processing of DNNs. Certainly, several key metrics should be considered to compare the various strengths and weaknesses of different designs, techniques and frameworks. These metrics should cover important attributes such as accuracy/robustness, power/energy consumption, throughput/latency and cost. Reporting all these metrics is important in order to provide a complete picture of the trade-offs made by an option.

For example, the difficulty of the dataset and/or task should be considered when measuring the **accuracy**. For instance, as it was discussed in the previous chapter, the MNIST dataset for digit recognition is significantly easier than the ImageNet dataset. As a result, a DNN that performs well on MNIST may not necessarily perform well on

ImageNet. Therefore, it is important that the same dataset and task is used when comparing the accuracy of different DNN models. To demonstrate primarily hardware innovations, it would be desirable to report results for widely used DNN models (e.g., AlexNet, GoogLeNet) whose accuracy and robustness have been well studied and tested.

The power and energy consumption of the hardware design should be reported for various DNN models. In fact, energy and power are important in processing DNNs in order to evaluate the impact in terms of resources. Having a limited amount of power and memory usage can be important in embedded devices with limited battery capacity (e.g., smartphones, smart sensors and wearables), or in cloud in data centres with stringent power ceilings due to cooling costs. So, in the evaluation of **power, energy and memory consumption** it is important to account for all aspects of the system.

High throughput is necessary to deliver real-time performance for interactive applications such as navigation and robotics. For data analytics, high throughput means that more data can be analysed in a given amount of time. As the amount of visual data is growing exponentially, high-throughput in big data analytics becomes important. **Low latency** is necessary for real-time interactive applications. Latency measures the time between the arrival of a fragment of data in a system and when the result is generated. It is measured in terms of seconds, while throughput is measured in operations/second. Achieving low latency and high throughput at the same time can be a challenge but real-time applications (such as high-speed navigation where it would reduce the time available for course correction) need to be efficient in these tasks. The latency and throughput should be reported in terms of the batch size and the actual run time for various DNN models, which accounts for mapping and memory bandwidth effects.

Hardware cost is in large part due to the amount of **storage**, the number of CPU **cores** and all the **GPU specifications** (memory, power and speed). In terms of cost, different platforms will have different implementation-specific metrics. To mention that there is a correlation between the number of cores and the throughput. In addition, while many cores can be built on a system, the number of cores that can be used at a given time should be reported.

The **scalability of training** is also critical for a Deep Learning framework since a single GPU does limit the performance during training process.

All these **metrics** should be taken in consideration alongside with the main properties of a given DNN model study:

- the **accuracy** of the model in terms of the datasets;
- the **network architecture** of the model, including number of layers, filter sizes, number of filters and number of channels;
- the number of weights, that can impact the **storage requirements**;

As stated in the paper *Efficient processing of deep neural networks a tutorial and survey* [3], it is important that all these metrics and specifications are analysed in order to evaluate all the design trade-offs. For instance, without the accuracy given for a specific dataset and task, it can be run a simple DNN with low power, high throughput, and low cost but the processor is not used as much as it should; the test setup should also be reported, including whether the results are measured from simulation and the type of data that were used for the tests.

3.2 Pre-existent Deep Learning benchmarks

There are already some available benchmarking projects which have the goal to evaluate and analyse Deep Learning frameworks and/or models' performance. It is fair to point out that most of these benchmark tools only focus on a specific set of metrics, which are considered the most relevant and significant for the study by the authors in that specific context. In fact, a benchmark tool interested in evaluating the performance of different models will take into account some parameters such as accuracy and loss; on the other hand, projects created to run same models on different architectures and frameworks will probably give more importance to the speed of training and the speedup and the scalability with various combinations of architectures.

In the following lines the most influential and popular benchmark tools are reported and some of them have also been source of inspiration for this work.

DLBENCH (Benchmarking State-of-the-Art Deep Learning Software Tools) is a benchmark project developed by the Hong Kong Baptist University and is created to execute several experiments with different neural networks (fully connected, convolutional and recurrent). It supports directly Caffe, TensorFlow, CNTK, Torch and MXNet and is mainly focused on processing time and convergence rate, which are two main factors that concern users when training a Deep Learning model. Their source code to do benchmarks is easy but very useful: to run a set of experiments it must be passed a config file as argument of a command. This file, which must respect a rigorous syntax, can be filled with the models to benchmark, each one with its own specifications (batch size, epochs, learning rate, number of GPUs or CPU threads and other parameters), and the tools to use for all the models added. There is also a utility to collect GPU information during the training. To evaluate the running performance, they measure the time duration of an iteration that processes a small batch of input data. In practice, after a certain round of iterations or the convergence of learning, the training progress will be terminated. Therefore, they benchmark these tools by using a range of mini-batch sizes for different types of network. The data provided in the project website are reported for different batch sizes and several hardware configurations [5].

DAWNBench is a benchmark suite for end-to-end Deep Learning training and inference. As stated in the official site: «Computation time and cost are critical resources in

building deep models, yet many existing benchmarks focus solely on model accuracy. DAWNBench provides a reference set of common Deep Learning workloads for quantifying training and inference time […] across different optimization strategies, model architectures, software frameworks, clouds, and hardware» [6]. All these metrics are evaluated on image classification tasks on ImageNet and Cifar10 datasets. This is actually the first benchmark to compare end-to-end training and inference across multiple Deep Learning frameworks and tasks. There are leader boards with submitted benchmarks results for each category and this can lead to a continuous and developing improvement from all the users who want to participate.

DeepBench is another benchmark on different hardware platforms. It uses some neural network libraries (like cuDNN or MKL) to benchmark the performance of basic operations on different hardware. As the authors say, their goal is quite different from other benchmarks: «The performance characteristics of models built for different applications are very different from each other. Therefore» the main goal is «benchmarking the underlying operations involved in a deep learning model. Benchmarking these operations will help raise awareness amongst hardware vendors and software developers about the bottlenecks in deep learning training and inference» [16].

HPE Deep Learning Benchmarking Suite is an automated benchmarking tool, which makes it easy to run performance tests with most popular Deep Learning frameworks. It enables consistent and reproducible benchmark experiments on various hardware/software combinations. This suite makes use of Docker containers as a primary mechanism to run benchmarks.

> *It includes a collection of command line tools for running consistent and reproducible benchmark experiments on various hardware/software combinations. In particular, DLBS implements internally various deep models with the same implementations for all supported frameworks. Deep models that are supported include various VGGs, ResNets, AlexNet and GoogleNet models. Benchmarks target single node multi GPU configurations, with support of real (ImageNet dataset) and synthetic data, single and half precision, inference and training phases* [26].

The set of frameworks that are now supported includes mainly the most known tools, like Caffe, NVIDIA Caffe, Intel Caffe, Caffe2, TensorFlow, MXNet, PyTorch. However, only ImageNet dataset is included in the suite.

Convnet-benchmarks is a GitHub project which deals with the benchmarking of all public open-source implementations of convolutional networks. This simple research was made by one of the creators of Torch/PyTorch, who picked some popular ImageNet models (AlexNet, GoogleNet, Overfeat) and clocked the time for a full forward and backward pass. Although it is not a complete suite, it is very well known by the community

and it is one of the main points of reference of several users of Deep Learning frameworks [4]

CNN-benchmarks is a project of benchmarks for popular convolutional neural network models on CPU and different GPUs, with and without cuDNN. All benchmarks were run in Torch by using famous models with a minibatch size of 16 in order to do a direct comparison among different GPUs. In fact, the main goal of this tool is to stress some important points about the performance of models, architectures and the GPU's importance in the process of training [11].

Apart from these works, other significant benchmark results (without transparent implementation) can be found on the major Deep Learning websites. For example, TensorFlow benchmarks report the performance of the main convolutional neural networks on several architectures with the most known models in detail, analysing synthetic and real data, reporting graphs and numbers in terms of speed (images per second) and speedup with the increase of the number of GPUs used. Moreover, the NVIDIA website shows some few interesting results for almost all the most important frameworks. Even though there is no possibility to evaluate and validate the performance reported and shown in these webpages, these results have been significantly used as a point of reference, given the importance of the source.

3.3 Critique and limits of existent projects

All the suites and projects try to evaluate the performance of Deep Learning frameworks and models from different points of view, analysing in depth all the scenarios and getting results by making comparisons with several factors. However, it is worth to mention the presence of different and various approaches, often due to certain requirements or objectives.

Most of the Deep Learning benchmarks with the goal to compare networks to do a specific task inevitably focus on accuracy, which is the most considered parameter if the intention is to deploy in a real application a new model trained on a personal dataset: in this case, what really matters is the reliability and the precision in solving a problem in the best way possible. It is clear, though, that this approach requires high experience and capability in Deep Learning, especially with the framework used: it is necessary to modify several parameters and create a brand-new network that fits the problem, task that can be very challenging for a beginner.

Alternatively, other performance evaluation projects deal with the speed in processing images for more than one architecture but usually using the same framework for the experiments. This is particularly useful to test new GPUs in a pre-set-up environment in order to characterize them in the market and to study the scalability. Benchmarking this way is quite common. Nevertheless, using a unique tool limits the implementation unless it is required for some reasons. Several Deep Learning tools are present and free

to use on the web. There is not a clear specialization for each of them and the community is still divided on using different tools for the most common task, like image recognition.

Certainly, these benchmark methods are not the only ones: it is possible to face the problem in multiple ways and there is not a solution better than the others but it only depends on the research needs.

3.4 The approach proposed in this work

As shown in the previous section, there is no popular benchmark that specifically studies the behaviour among different models, architectures, datasets and frameworks at the same time. From this point of view, scalability is not considered as much as it should. In fact, it could be very interesting to see if a tool manages to work better with more cores, how it performs by increasing the number of GPUs or also why there are significant gaps with some models. Furthermore, power and memory usage is often neglected because it is always assumed to have enough resources to deploy neural networks: indeed, not all the users can access powerful machines and nowadays the use of Deep Learning is spreading on mobile platforms and in this case the consumption of memory is a bottleneck, i.e. it must be limited in order to fit the hardware requirements.

Performing a comparison among Deep Learning frameworks and hardware architectures in a holistic way is an innovative approach to this field because there are neither previous academic works in this institution nor well-known public projects with this goal. There is no doubt that a benchmark focused on all these features adds valuable and significant information for the research community. As we will show in the next chapter, this work aims to carry out such a holistic study.

CHAPTER 4
Project design and development

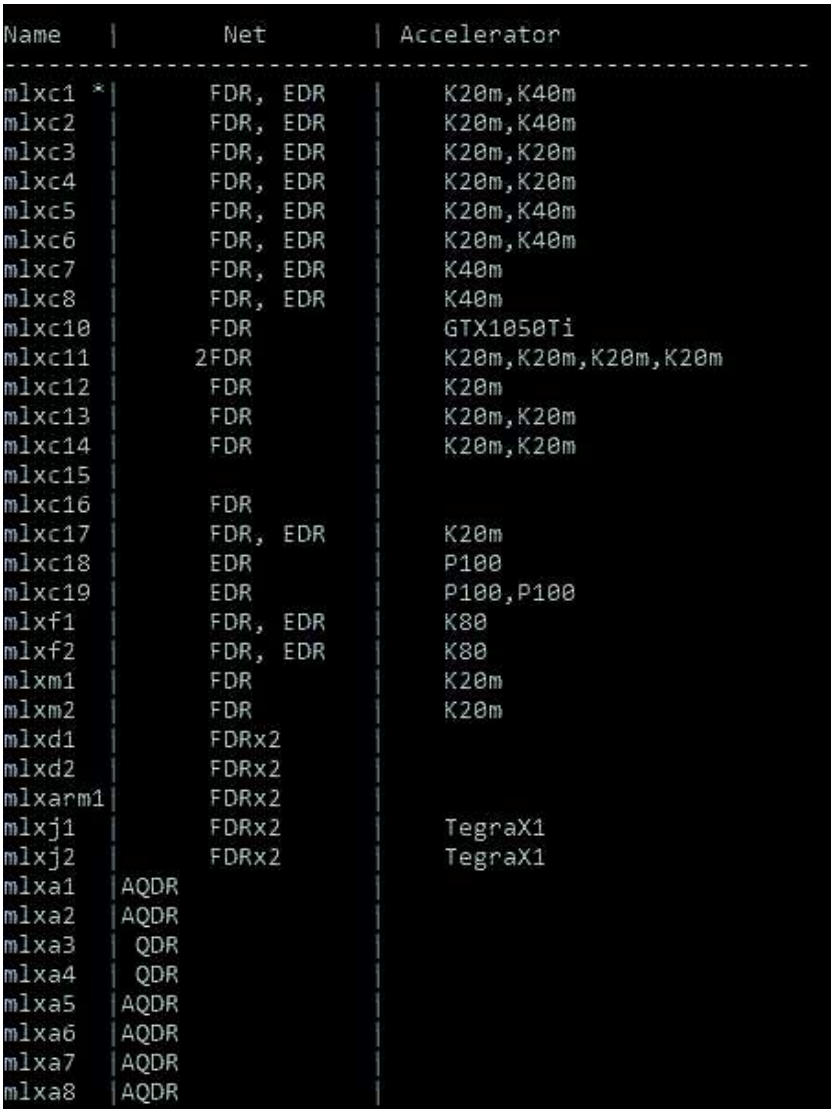
After describing all the background context of Deep Learning and introducing the problem of benchmarking Deep Learning environments, this chapter explains how the entire environment was set up in order to run benchmarks on several configurations.

4.1 Hardware specifications

The environment where the performance test has been executed is a cluster composed of 36 nodes, whose configuration is shown in Figure 4.1.a. Only a small set of these nodes have been used for the experiments: the choice of the ones used has been carried by the significance and the goal to stress as much as possible the scalability from one architecture to another. In fact, the networks will run by using CPU and an increasing number of GPUs for a total of six hardware scenarios.

CPU performance is important nowadays since it is still used for training deep neural networks in some applications. However, in this specific work it will be used to show how GPUs are much more convenient and suitable to this kind of tasks for the reasons explained in Chapter 2. The operating system is a CentOS Linux, version 7.3, for all the nodes and the CPUs used are Intel E5-2620, 12 cores and 2 threads per core, a E5-2630, 20 cores and also 2 threads per core.

Scalability plays a fundamental role in the entire deep learning field, that is why two quite dissimilar GPU models, both in terms of price and power, have been picked for the benchmark: a basic NVIDIA Tesla K20m and a more performant NVIDIA Tesla P100. It will be shown the behaviour with one and two GPUs for both K20m and P100. The main hardware specifications, with detail of the configurations used and the architectures for the tests can be found in Figure 4.1.b.



Name	Net	Accelerator
mlxc1 *	FDR, EDR	K20m,K40m
mlxc2	FDR, EDR	K20m,K40m
mlxc3	FDR, EDR	K20m,K20m
mlxc4	FDR, EDR	K20m,K20m
mlxc5	FDR, EDR	K20m,K40m
mlxc6	FDR, EDR	K20m,K40m
mlxc7	FDR, EDR	K40m
mlxc8	FDR, EDR	K40m
mlxc10	FDR	GTX1050Ti
mlxc11	2FDR	K20m,K20m,K20m,K20m
mlxc12	FDR	K20m
mlxc13	FDR	K20m,K20m
mlxc14	FDR	K20m,K20m
mlxc15		
mlxc16	FDR	
mlxc17	FDR, EDR	K20m
mlxc18	EDR	P100
mlxc19	EDR	P100,P100
mlxf1	FDR, EDR	K80
mlxf2	FDR, EDR	K80
mlxm1	FDR	K20m
mlxm2	FDR	K20m
mlxd1	FDRx2	
mlxd2	FDRx2	
mlxarm1	FDRx2	
mlxj1	FDRx2	TegraX1
mlxj2	FDRx2	TegraX1
mlxa1	AQDR	
mlxa2	AQDR	
mlxa3	QDR	
mlxa4	QDR	
mlxa5	AQDR	
mlxa6	AQDR	
mlxa7	AQDR	
mlxa8	AQDR	

Figure 4.1.a - Cluster configuration in the test environment

CPU	Cores	OS	Memory
dual Intel E5-2620 v2 @ 2.10GHz	12	CentOS 7.3	32 GB
dual Intel E5-2630 v4 @ 2.20GHz	20	CentOS 7.4	64 GB

GPU	Cores	Memory	Power
NVIDIA Tesla K20m	2496	5 GB	225 W
NVIDIA Tesla P100	3584	16 GB	250 W

Figure 4.1.b – CPU and GPU specifications

4.2 Selection of Deep Learning frameworks

The core part of this work is to offer an evaluation of the most used deep learning tools, compare them, show differences in implementation and, most of all, in performance. The work has been progressively expanded to a wide range of frameworks and, for each one, all the features have been studied in depth, using examples and more complex cases to that end. The goal was to acquire the necessary mastery of each tool in order to evaluate the advantages and disadvantages of the whole system. After evaluating several deep learning tools, only a subset of them has been selected for the tests for several factors and the most important considerations for the choice are provided below:

- As stated before, it was necessary to offer eminent software among the community by **looking at statistics and polls**. The decision of not including TensorFlow, the most used of all, could not have been understood. Old deep learning frameworks like Theano, even though still useful for some computational tasks, have been excluded for the decreasing popularity and use by programmers who redirected their attention to more recent libraries.
- An essential requirement was the **availability of official models** for the chosen datasets. Not every tool offers organized samples for the ImageNet and/or for Cifar10 datasets (MNIST is almost always present) and in this context "organized" means complete and manageable. For instance, samples that do not provide the opportunity to test scalability were not taken into account, as well as those which did not provide ways to set the crucial parameters for the experiments (even though some modifications have been adopted in some models). Moreover, unofficial examples have been rejected unless the source was absolutely reliable (and this happened only in one case). This decision aimed to preserve the accuracy of data and to avoid misunderstanding with the comparisons. That is the reason why Caffe2, the new main deep learning tool from Facebook, was not considered in this work, though in a future work it could be used since more models will be available soon. Instead, the old and not-more-maintained Caffe was picked because it is still very used by the majority of new users who decide to face deep learning and computer vision problems.
- A less important discriminant was **the programming language**. Since **Python** is a very common language for almost all the deep learning programmers thanks to its modularity and features, deep learning tools which implement a Python interface have been considered and analysed. The central point is the concept of likelihood: the more the tools are similar to each other the more the comparison can be accurate. Torch is a main deep learning asset now but its Python counterpart, PyTorch, is much more studied and exploited for the simple difference that it is written in Python.

- Another not significant but interesting pattern followed was **diversity**. Diverse tools from different sources can guarantee the coverage of all the approaches and notable methods used in the state of the art. All the leader companies in technology are focusing on deep learning, which is one of the most attractive topic of computer science and, likely, the future of Artificial Intelligence. It is not a case that Amazon, Google, Facebook and Microsoft invented or adopted their "own" frameworks following this trend as shown in Figure 4.2.

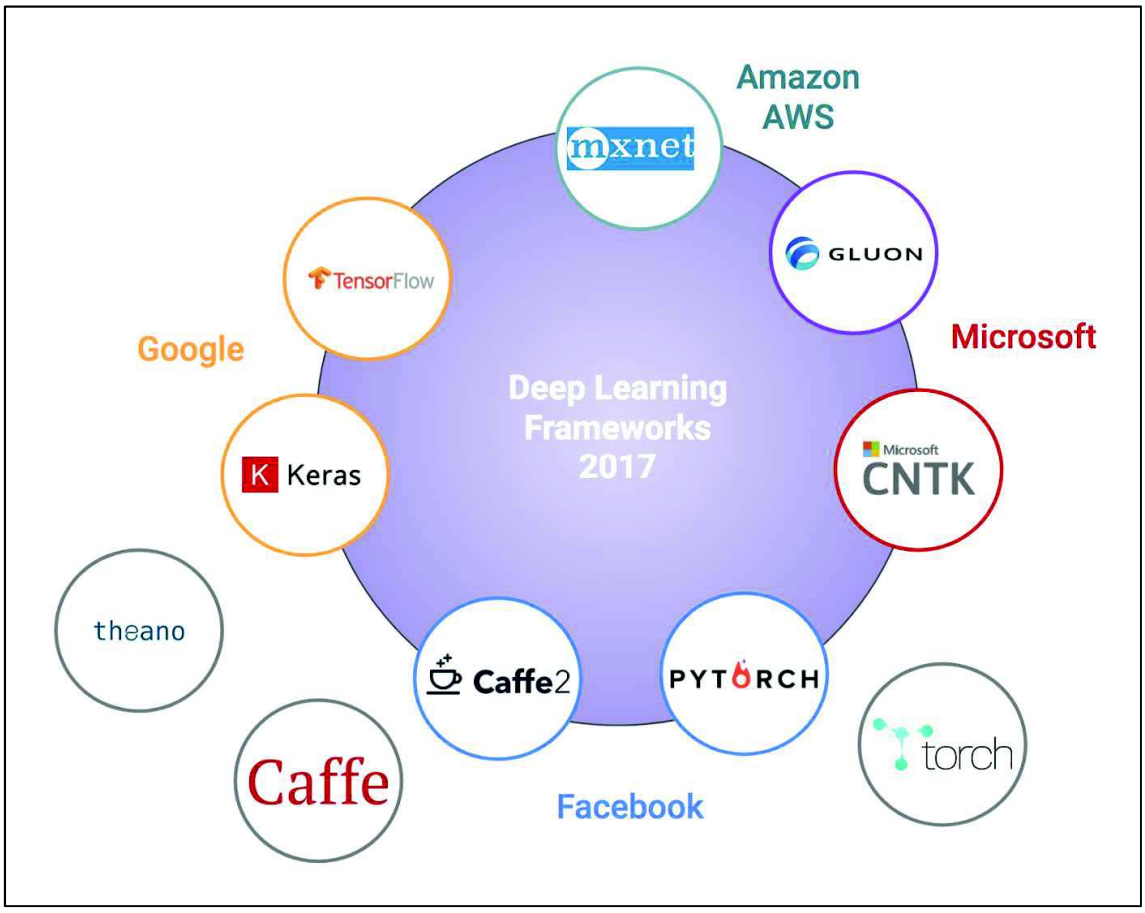


Figure 4.2 - Technology leader companies and the most popular Deep Learning frameworks in 2017 [8]

Facebook, through different research groups, has "under its control" PyTorch and Caffe2 (until 2017 even Caffe; now Caffe inventors work for Facebook and they created a new second version of this tool). Google came up with TensorFlow, an absolute innovation in terms of modularity, expandability and portability, while Microsoft came on stage with CNTK (now called Cognitive Tools). Amazon did not decide to just stare at their opponents and "adopted" MXNet, choosing this framework for some applications related to Amazon Web Services. From this brief discussion it is clear that a heterogeneous set of deep learning tools provide a complete excursion throughout the software that surround us and we use almost every day not only for programming tasks.

To repeat the concept, the above points are not the unique considerations taken but they have been determinant for the choice of the analysis. It should be also mentioned that there are several "variants" of the official frameworks: a relevant example is the NVIDIA fork of the Caffe GitHub project, which is still maintained and more popular for better performance with GPUs than the official Caffe. In this case, it has been decided to stick with the standard code of these tools for simplicity but a future development of this work could take into considerations these aspects.

Thus, **Caffe**, **TensorFlow**, **PyTorch**, **MXNet** and **CNTK** have been chosen for the benchmark tests in order to give a wide scope for the future spread of the results obtained.

4.3 Preparation of Deep Learning frameworks

The installation of every tool is very straight-forward by following the instructions in the official websites but if the environment is a cluster and root permissions are not granted, everything is more complex; not only because it is not possible to run some commands to freely install packages but it is essential to pay attention on things that we usually ignore, like the location where dependencies are installed, make libraries visible and so on.

Therefore, it has been decided to use a well-known **package manager**: **Anaconda**. Anaconda is an open source package and environment management system. It basically allows to find and install packages and setup a separate environment to run with a different Python version [7]. In this way, installing new packages has not been a problem anymore. Just to clarify: this is not the best solution because package managers have to be used with caution and they often will give programmers trouble. Nevertheless, every framework (with the exception of Caffe which has to be installed from source) supports an installation through pre-compiled binaries from Anaconda and this was optimal for the case.

Another trivial point: installing from source is always better than installing from pre-compiled binaries but this is also much more annoying and time-wasting; additionally, it often requires root permissions. So, to keep things simpler, installation from binaries has been chosen for TensorFlow, PyTorch, MXNet and CNTK. For each tool, an Anaconda environment was created and all the necessary packages for that tool were installed in that precise environment: this solution makes the dependencies separated with low coupling. For using a tool, the command "source activate [environment name]" is required to activate the environment where the framework has been installed. In Figure 4.3 the used versions of each framework are shown.

Framework	Version
Caffe	1
TensorFlow	1.9
PyTorch	0.4.0
MXNet	1.2.0
CNTK	2.5.1

Figure 4.3 – Used versions of each Deep Learning frameworks

For running the frameworks with GPU support, it was necessary to install NVIDIA CUDA and also NCCL and cuDNN for high-performance GPU acceleration.

For the evaluation, CUDA 9 was used along with NCCL 2.1 and cuDNN 7.1.1: the decision of the specific versions is due to compatibility issues with all the frameworks (especially for TensorFlow, which required CUDA 9 at least).

4.4 Selection of neural networks and datasets

Selecting the right neural networks for a performance evaluation could seem a simple task because of the high number of possibilities available on the web. However, this has been object of a very deep investigation for its correlation with the selection of the datasets.

The selection of datasets and models should consider all the parameters described in chapter 2 (like batch size and number of epochs) because they are going to affect the tests and the time required for doing every experiment.

Following the purpose to use standard models for benchmarking, it was clear that ImageNet winners or the most popular networks which were submitted for the competition were the best options, considering the fact that also other benchmarks picked up these ones. The **chosen** networks were: **AlexNet**, **GoogLeNet**, **ResNet50** and **VGG16**.

Initially, the intention was to train these models with the CIFAR10 dataset in order to make a simple but efficient deep learning benchmark. Furthermore, large datasets composed of more than a million of images (like ImageNet) have limitations in learning time, so training is longer than in smaller datasets and this could take much more time, above all for CPU training and a full training of several epochs. The issue encountered with models for CIFAR10 was the following: a certain official model of a famous network can be available on a framework but not in another; this is frequent for datasets which are different from ImageNet. For instance, the official TensorFlow models for ImageNet were AlexNet and ResNet. At that time, when a part of the work in this report was already

done, it was decided to keep the tests for CIFAR10 because they are still useful for evaluation, even though there are some missing configurations.

This led to download also the ImageNet dataset for object recognition (ILSVRC12 dataset) to train the networks and obtain full data for each model and tool. Moreover, the small MNIST dataset of handwritten images was added with the well-famous LeNet, which is very easy to train.

4.5 Preparation of neural networks and datasets

The first thing to do is the **preparation of the datasets**. This requires downloading the data from the official sites: CIFAR10 and MNIST dataset have size respectively of 163 MB and 11 MB approximately; instead, the ImageNet dataset (precisely, ILSVRC12 dataset) is much bigger and needs more space in disk in order to get both the training and validation images. In this case, the total size of all the tar files required is approximately 145 GB. In addition to these packages, the annotations of object bounding boxes are needed for the classification, especially for MXNet training scripts.

After getting all the images, the actual format of a dataset is a framework specific and data must be converted in order to make everything work in the correct way (this will take much other space in disk).

Caffe and PyTorch use standard Caffe's LMDB datasets (lmdb). MXNET uses data in standard RecordIO format (recordio). TensorFlow backend (tf_cnn_benchmark) uses TFRecord files (tfrecord). CNTK needs to create a train and a test folder that store respectively train and test images in PNG format and mapping files (train_map.txt and test_map.txt) for the CNTK ImageReader as well as the mean file. To note that only few conversion scripts were already included in the training files but all the other conversions have been done manually by creating ad-hoc scripts.

As data is ready to be used for training, it remains to **prepare the models' implementation**, which is the hardest part to complete. The original idea was to have all models in one format (Caffe's prototxt) and then convert those models into other formats since there are different ways to convert models from one framework to another. For instance, Caffe to TensorFlow converter exists, as well as a PyTorch one. It turned out that this was not the best option: apart from the lack of official converters for MXNet and CNTK (only files provided by the community), this approach seems to be affected by performance issues.

Thus, it was decided to go with framework specific implementations. Among the different and various options inside the same official repository for each tool, the following choices were used:

- **TensorFlow**: tf_cnn_benchmarks from tensorflow/benchmarks repository contains implementations of several popular convolutional models with four datasets

(ImageNet, CIFAR10, MNIST and Flowers), and is designed to be as fast as possible: in fact, many high-performance strategies have been utilized. This was preferred respect to clean and easy-to-read implementations for giving priority to the training speed parameter of the benchmarks;

- **Caffe**: models for ImageNet are defined as training prototxt files in the "models" folder of the NVIDIA/caffe repository. The LeNet model is in the "examples" folder. These folders contain multiple subfolders - one subfolder for one model. Although the standard Caffe installation was used for the benchmarking, network definitions were taken from the maintained NVIDIA fork because of up-to-date files and GPU performance improvements. Some variations were added to adapt the models to the BVLC version. For the CIFAR10 models' implementations, it was picked an unofficial project which was suggested by several users (Classification_Nets);
- **MXNet**: the image-classification folder contains several models and scripts for running training with all the datasets in different modalities;
- **PyTorch**: the "examples" folder was used which contains scripts for using datasets, the models are included in the torchvision package;
- **CNTK**: in the main repository, the CNTK/Examples/Image/Classification path has several network definitions for the famous datasets.

Almost all these projects lacked several features which are essential for our benchmark, such as CPU or multi-GPU training support, missing training and network parameters and other important values. This is the reason why great attention was reserved in this part of the work in order to make the benchmarks work in the right way, without problems or missing features: this refining and polishing phase required to write several crucial lines of code in every single network file and a basic knowledge of each deep learning framework was essential, otherwise the modification of these tool specific configuration files can lead to frequent mistakes and errors.

The main modifications made to some of the benchmark projects used are about providing:

- organization and structure to the great amount of files required for the entire project;
- support of CPU and/or multi-GPU training through some framework-specific commands and libraries;
- support of fundamental parameters (such as the number of epochs, epoch's size, batch size etc…) for making the benchmarks uniform and customizable for all the configurations;

- adaptability to the system used by installing missing packages and APIs required for the learning process;
- high-performance settings for making the whole training preparation and training itself as fast as possible.

After this long preparation stage, it is necessary to **choose the parameters** to run the training with. A full training session composed of several epochs for each combination of architecture, model and framework could be very expensive in terms of resources and time. Since the goal is to acquire data about training speed and power and memory usage, it was decided to train the networks only for some few iterations. This hypothesis is acceptable because accuracy is not being considered according to the reasons explained in the previous chapter.

Regarding the parameters of each network (in primis, batch size and learning rate), the idea was to stick with the standard values. However, there have been problems due to GPU memory limit of the K20m. To solve the issue, models trained with the ImageNet dataset have a lower batch size than the one suggested in the official implementations. So, it must be considered that a low batch size could not stress the scalability in some cases because this value is divided by the number of GPUs used for training and the lower the batch size, the lower the power of parallel training.

The exact values are reported in detail in Figure 4.5.

Dataset	Network	Effective Batch size	Epochs	Iterations or steps	Learning rate
MNIST	LeNet	64	1	938	0.01
CIFAR10	AlexNet	128	1	390	0.01
	ResNet50	128	1	390	0.1
	GoogLeNet	128	1	390	0.1
	VGG16	128	1	390	0.01
ImageNet	AlexNet	128	1	300	0.01
	ResNet50	8	1	300	0.1
	GoogLeNet	8	1	300	0.1
	VGG16	8	1	300	0.01

Figure 4.5 – Training parameters applied for each Dataset and Neural Network

Another important parameter which can limit the training process is the number of threads used for input data loading. This is a sort of bottleneck and, if it is not specified as a parameter in the training command, the performance could not be as good as expected. For this reason, it is important to set always the number of data threads equal to the number of cores of the CPU (in this case, 12 or 20).

Obviously, all these models' parameters have been the same for each tool but their actual realization could differ for other hidden variables or have slightly variations from one framework to another. As a side note: the project could support training with different data types (precision) like float32 (single precision) and float16 (half precision) but since data types slightly varies from one framework to another, it has been decided to stick with the standard single precision (for instance, Caffe does not support half precision).

4.6 Project structure

In this section the "project interface" created from scratch will be discussed along with the organization of the whole work: the goal of the interface is to acquire performance evaluations from all the models prepared and modified, as explained in the previous section, with the chosen metrics in the easiest way.

The structure was inspired by other famous projects and will try to examine the most significant data following the already mentioned example projects and the considerations described before. The personal files were created to benchmark in the best way the modified models' configurations files and provide a useful and structured interface to use the tool simply and efficiently.

First, **GPU usage** will be registered in order to understand which models, GPUs and frameworks perform better in terms of memory and power usage: this is particularly valuable because it is a requirement and a feature very demanded from most part of users who want to know the behaviour of a certain hardware and software conformation.

Second, **training speed** will be measured to examine the effective performance during the process of learning: this is the typical way to benchmark the performance of a neural network and it was decided to follow the trend for providing a common unit in comparison with other benchmarks.

The structure of the project is very linear and takes inspiration from the DLBENCH benchmarking project with several modifications to make everything simpler as shown in Figure 4.6.a and every component will be explained in detail in the following paragraphs.

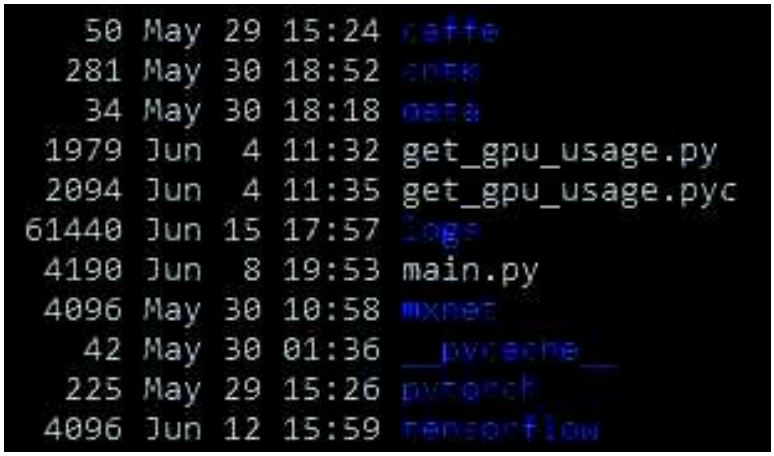


Figure 4.6.a – The main project directory structure

A Python script **main.py** is used for setting-up the training command to run according to the arguments provided. The other script **get_gpu_usage.py** is a set of utilities that are needed to acquire memory and power usage.

There is a folder for each framework (Figure 4.6.b) which contains subfolders organized by dataset (Figure 4.6.c) and model (Figure 4.6.d): all the code to train every network is inside them.

Figure 4.6.b – The datasets' subfolder structure for each framework

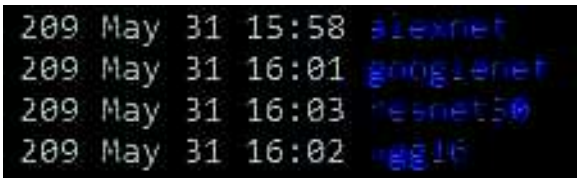


Figure 4.6.c – The models' subfolder structure for both CIFAR10 and ImageNet datasets

Figure 4.6.d – The models' subfolder structure for MNIST dataset

It is present also a folder called **logs** where two files are saved for each run:

- The first is used to memorize all the significant information about the training (time, memory and power usage);

- the second is necessary to register GPU information, as it will be explained later.

The Python file **main.py** is very simple and plain as it accepts 5 arguments which are:

- **num_gpus**: number of GPUs to use for training, -1 to use CPU (every core will be used in this case);
- **gpu**: name of the GPU, necessary to acquire GPU information about memory and power usage;
- **model**: it can be chosen among LeNet, AlexNet, GoogLeNet, ResNet50 and VGG16;
- **dataset**: it can be chosen among MNIST, CIFAR10 and ImageNet;
- **tool**: it can be chosen among Caffe, TensorFlow, PyTorch, MXNet and CNTK.

From the arguments given, it is created the right command to run the training. Before launching the command for the tool selected, a background thread is created, whose life is strictly connected to the execution of the training: in fact, the thread calls a function in get_gpu_usage.py which prints in a file (we can call it as power-log file) every second the output of the command "nvidia-smi" until the training process is still alive.

The NVIDIA System Management Interface (nvidia-smi) is a command line utility intended to aid in the management and monitoring of NVIDIA GPU devices. It can basically query the GPU device state and information, providing also information about running processes on GPUs.

When the training is completed, another function of get_gpu_usage.py is called to extract GPU usage from the power-log file created: it simply parses the output of nvidia-smi printed every second in the file in order to do the GPU average power and memory according to the GPU name. Finally, the time spent by the command, the GPU average power and memory will be printed. This information will also be saved in a log file and both the power-log and the log file will be moved to the "logs" directory.

It is worth to say that measuring the training speed through the time spent by the command is not a good option at all because it is not absolutely accurate. That is why it was decided to look directly at the output of the command used to launch training: every framework reports training information in different ways (number of batch size per second, seconds for a batch size, iterations per second and so on) and from them it is possible to get indirectly the precise images per second measure.

Each tool subfolder contains all the official models for each one of the three datasets and most part of the work (around 75/80 % of the total project) has been done to organize, adjust and create all the conditions to run the benchmarks as explained earlier.

CHAPTER 5
Evaluation and comparison of benchmark results

In this chapter, the results of the experiments will be shown and discussed deeply. The following pages will focus on the most significant outcomes, trying to obtain useful ideas and considerations.

5.1 The benchmark overview

Before starting with the evaluation, some recapitulation about the benchmarking process will be given. As it was said in the previous chapter, we have **5 networks** (LeNet, AlexNet, ResNet50, GoogLeNet, VGG16) and **3 datasets** (MNIST, CIFAR10, ImageNet): the first model will run exclusively on MNIST, the other four will run on both ImageNet and CIFAR10. So, we have a total of **9 different test cases**.

The **frameworks are 5** (Caffe, TensorFlow, PyTorch, MXNet, CNTK) and the **hardware configurations** used with each framework are **6** (CPU with 12 cores, CPU with 20 cores, 1 K20m, 2 K20m, 1 P100, 2 P100) but they are actually 5 by considering that the CPU with 12 cores will be used for training only with MNIST and CIFAR10 and the CPU with 20 cores only with ImageNet. This decision was taken in order to accelerate the running benchmark, since training a network on ImageNet takes much more time than on the other datasets. Figure 5.1 shows all the available configurations used.

Dataset	MNIST	CIFAR10				IMAGENET			
Network / Framework	LeNet	AlexNet	ResNet	GoogLeNet	VGGNet	AlexNet	ResNet	GoogLeNet	VGGNet
Caffe	✓	✓	✓	✓	✓	✓	✓	✓	✓
TensorFlow	✓	✓	✓	not available	not available	✓	✓	✓	✓
Pytorch	✓	✓	✓	not available	✓	✓	✓	not available	✓
MXNet	✓	✓	✓	✓	✓	✓	✓	✓	✓
CNTK	✓	not available	✓	✓	not available	✓	✓	✓	✓

Figure 5.1 – Available configurations used in the project

The results can be considered quite valid since all the **data have been averaged over 3 runs**.

By a quick calculus, it can be derived that the benchmark runs have been 270 in order to get the **three parameters**:

- average speed for training (images per second),
- average GPU power usage (in Watt)
- average GPU memory usage (in Mebibyte).

The time required to carry out accurately all the experiments is approximately two weeks, considering a medium commitment and no unexpected issues to solve.

The chosen batch sizes could appear to be relatively small. But, as it was explained in the previous chapter, they were decreased after realizing the insufficient K20m memory for several tasks. Unfortunately, this has a contraindication: the lower the batch size, the lower the scalability. Consequently, some configurations seem to not respect the expected values: for example, a model run on 1 GPU could have a better performance than the same model on 2 GPUs. Luckily, the problem does not affect many configurations but it is worth to mention it.

In the next sections, each of the nine network configurations will be briefly analysed and discussed on all the metrics. Then, overall comparisons and considerations will be made among all experiments and the results will be tested and evaluated with other benchmarks. A table with all the results is available in Appendix A.

5.2 Data analysis for MNIST

LeNet is a very small and simple network to train on MNIST dataset. However, the goal to privilege the accordance with other experiments and with the whole work itself made us choose to train the network for only one epoch. Because of the small dimension of the network, it is not properly correct to analyse performance in terms of training speed: in fact, the scalability could not be appreciated as much as it should in a more complex and deeper model.

The results for training speed are shown in Figure 5.2.a. By comparing the frameworks, there is a notable difference between the set composed by Caffe and MXNet and the other composed by TensorFlow, PyTorch and CNTK. The former, which evidently outperforms the latter, presents very high values for GPU training, with a 20x – 30x speedup respect to CPU training, even though this speedup is much less evident when 2 GPUs are used. Moreover, MXNet seems to have a particular implementation that does not give significant values when 2 P100 are used for the reasons explained above. On the other hand, the second group (TensorFlow, PyTorch and CNTK) does not scale bad with the help of GPUs but the levels are lower than the levels of Caffe and MXNet, even though PyTorch and CNTK show the best results among the five tools for CPU.

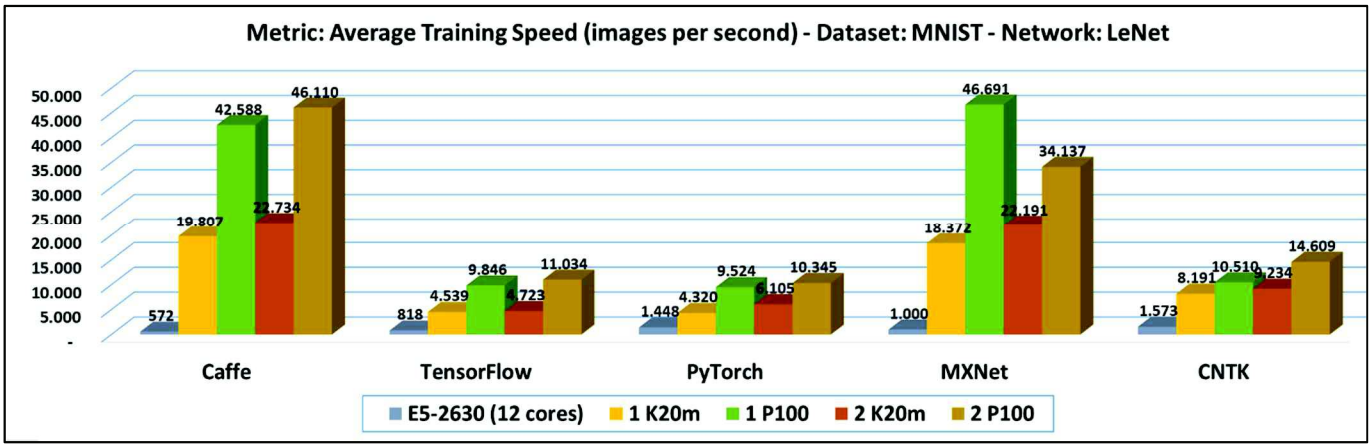


Figure 5.2.a – Benchmarks for training speed for Dataset MINST and Network LeNet

About memory usage, we can see from Figure 5.2.b how TensorFlow is clearly the most expensive from this point of view and strangely the performances don't follow the same trend. The other frameworks are almost equivalent with some few exceptions:

- PyTorch and CNTK have a near-to-zero level of average memory usage when it comes to CPU;
- PyTorch again offers very similar values for 1 and 2 GPUs both for P100 and K20m;
- 2 K20m present more memory consumption than 1 P100 on MXNet;
- CNTK shows optimum results.

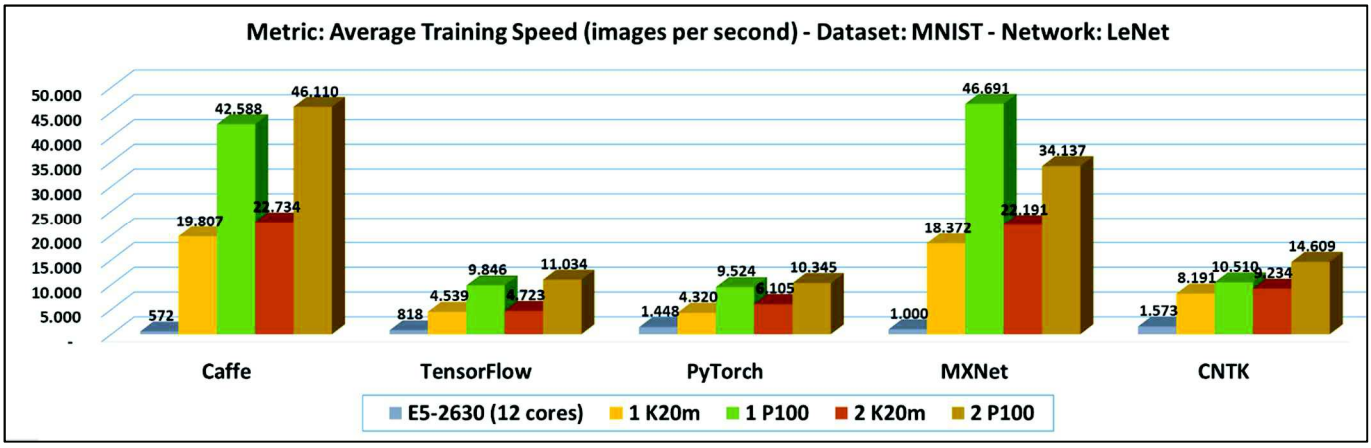


Figure 5.2.b – Benchmarks for GPU memory usage for Network LeNet and Dataset MINST

Numbers are different in terms of power usage, where there is a general equivalence as shown in Figure 5.2.c. P100 consumes very low power and this is evident for all the frameworks. Surprisingly, 1 P100 is more expensive than 2 P100 on Caffe and this is also true for the K20m in every tool. It is remarkable that there are high values with CPU

training for TensorFlow and CNTK. In this metric, PyTorch slightly stands out for low levels in each configuration.

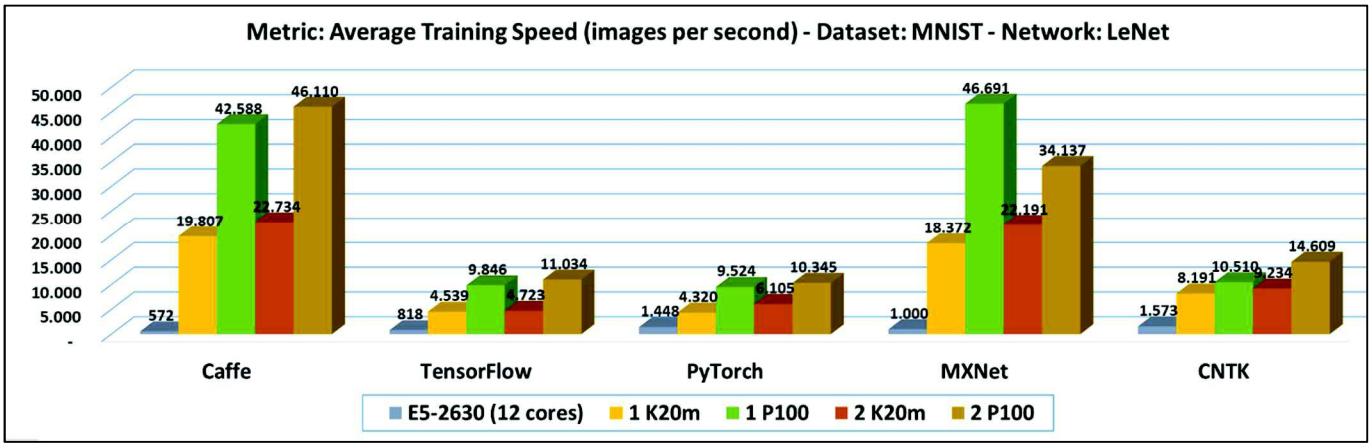


Figure 5.2.c – Benchmarks for GPU power usage for Dataset MINST and Network LeNet

5.3 Data analysis for CIFAR10

CIFAR10's models are not the same networks as the ImageNet ones but the same concepts and features have been applied in order to fit a smaller dataset. The same discussion for LeNet is still valid since the models are not so deep and complex for truly understanding the scalability but is still useful for the purpose of benchmarking.

5.3.1. AlexNet

CNTK is not present in the metrics due to the lack of availability of an official model.

Average speeds are shown in Figure 5.3.1.a. TensorFlow and Pytorch share almost the same data: the second one is a little more performant, but they do not present enough scalability as well as MXNet apart from the P100 data which are better than the others two. On the contrary, Caffe shows the best performance and scalability in all the configurations (1.5x from 1 P100 to 2 P100, more than 200x from CPU to 1 P100).

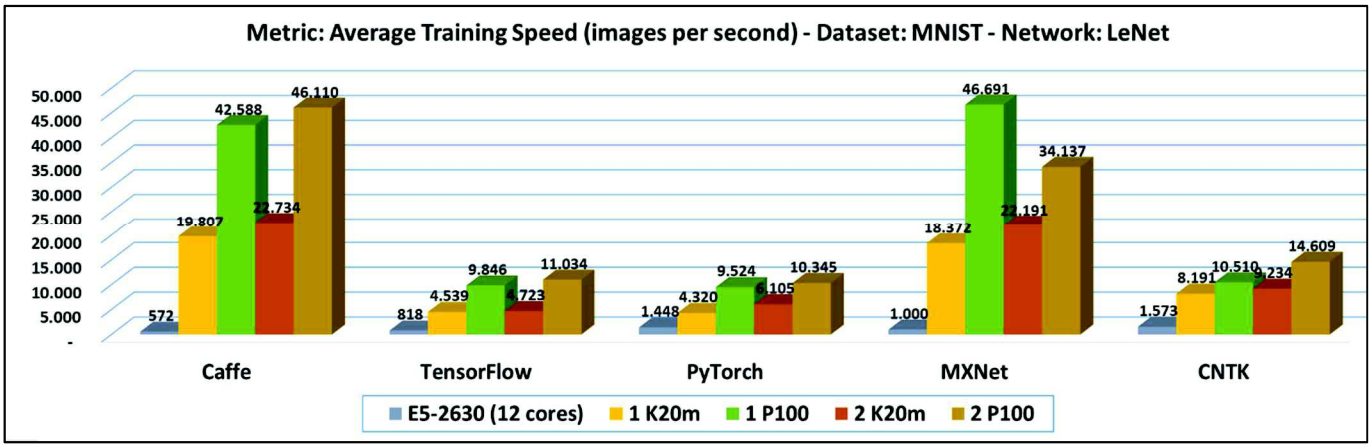


Figure 5.3.1.a – Benchmarks for training speed for Dataset CIFAR10 and Network AlexNet

Even in the memory metric (see Figure 5.3.1.b), Caffe is the best, confirming the good implementation of the model over all. To note that for this tool 2 P100 statistically consume less than 1. Just a bit better PyTorch, followed by MXNet. At the same time, TensorFlow confirms itself as the most memory expensive without good results in speed for one epoch.

Although the framework of Google does not particularly excel in the last two metrics for CIFAR10's AlexNet, it does well for GPU power usage but the level for CPU is too much high. Very similar PyTorch and MXNet. The results for Caffe in this case are excessive as it exceeds the average for 1 and 2 K20m.

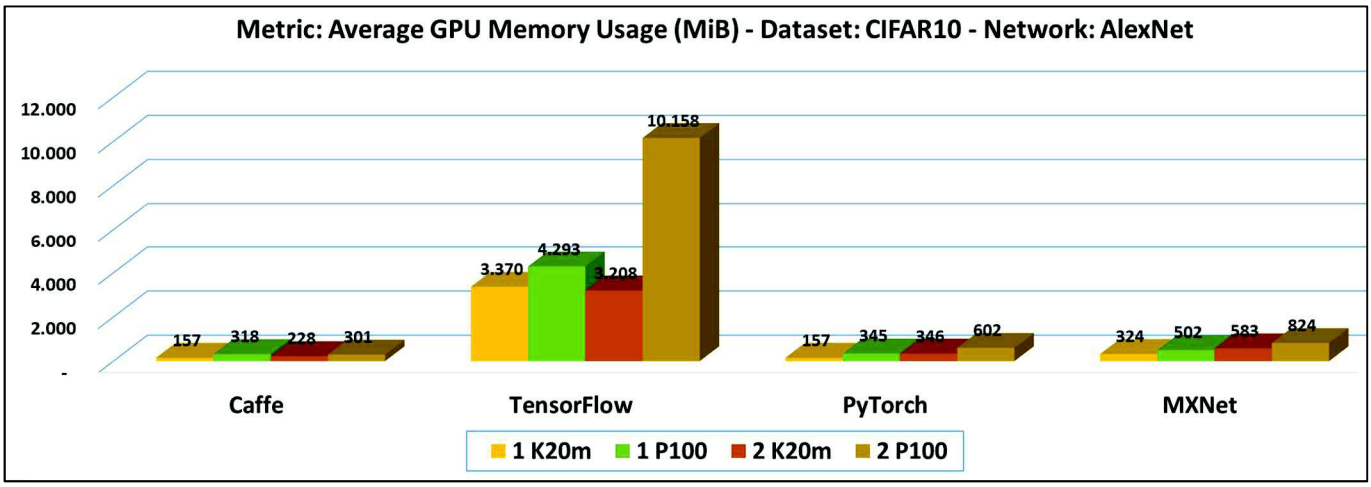


Figure 5.3.1.b – Benchmarks for GPU memory usage for Dataset CIFAR10 and Network AlexNet

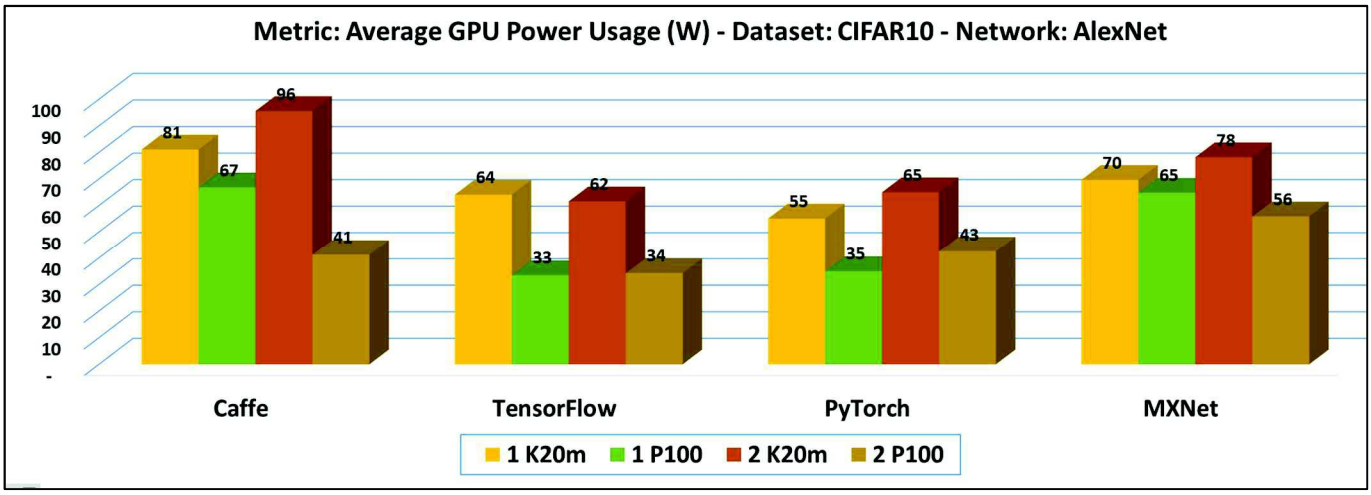


Figure 5.3.1.c – Benchmarks for GPU power usage for Dataset CIFAR10 and Network AlexNet

5.3.2. Resnet50

This model is the only one for CIFAR10 which has an official implementation for every tool analysed, though the implementations are dissimilar and this could have brought little inaccuracy in the results.

Average speeds (see Figure 5.3.2.a) are reported and the results are totally different from the previous tests. Even if there are no excessive high or low values, it is surprising that Caffe loses many points in terms of speed but the scalability is still acceptable. From the scalability point of view, the same can be said for PyTorch but the 2 P100 speed is lower than the 1 P100 one for the recurrent implementation issues. MXNet does not scale efficiently, however it gives similar good performance like TensorFlow that scales very well. CNTK overwhelms every framework with 2 P100 and offers more than average speed in most of the configurations.

Memory usage (see Figure 5.3.2.b) is still a problem for TensoFlow which is the worse. PyTorch, CNTK and MXNet have very similar data each other, while Caffe is a bit higher than the others three.

Power usage results (see Figure 5.3.2.c) are instead quite the same for each tool and there are no further comments to add apart from the fact that CNTK unusually manages to achieve optimum numbers with K20m configurations but not sufficient records with CPU and 2 P100.

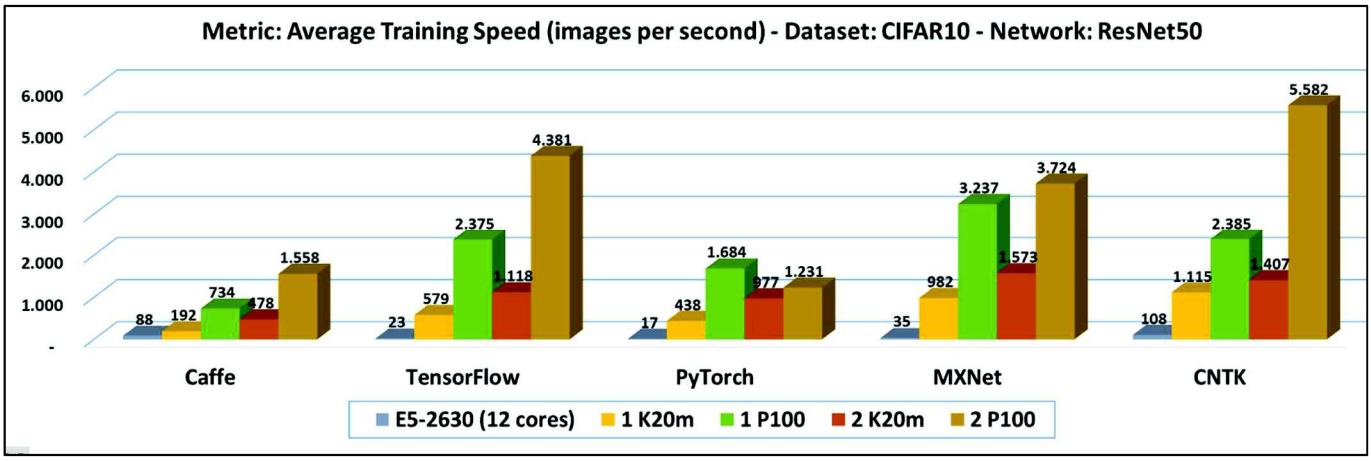


Figure 5.3.2.a – Benchmarks for training speed for Dataset CIFAR10 and Network ResNet50

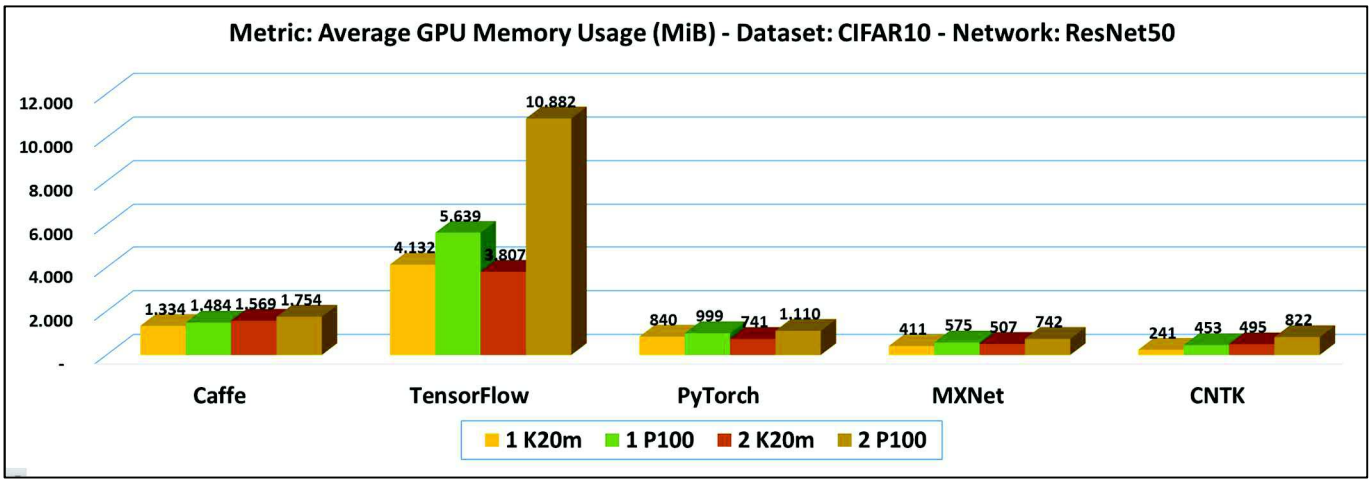


Figure 5.3.2.b – Benchmarks for GPU memory usage for Dataset CIFAR10 and Network ResNet50

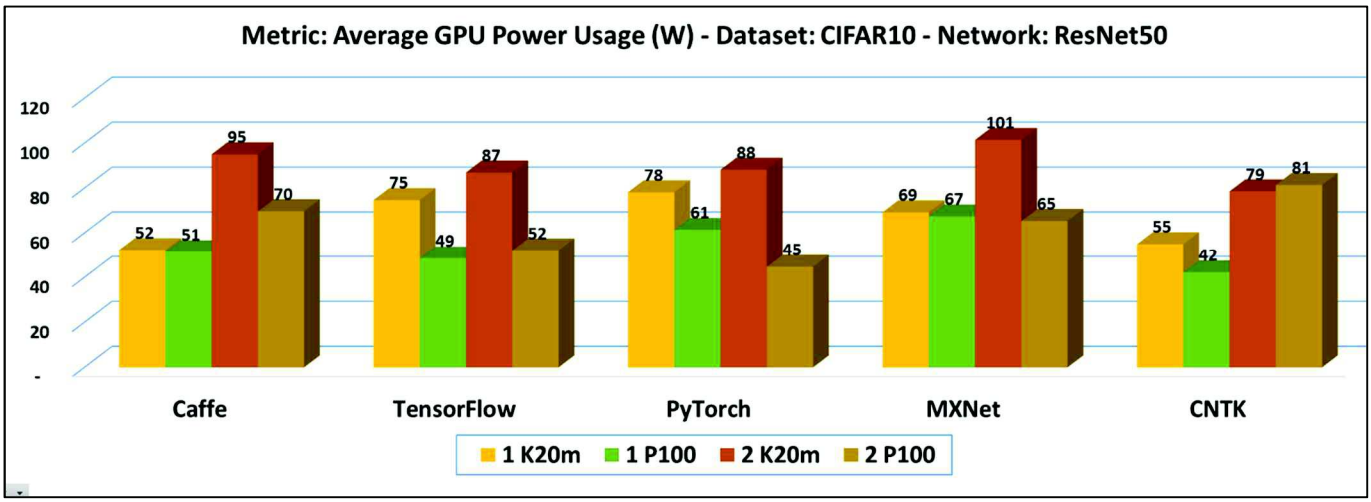


Figure 5.3.2.c – Benchmarks for GPU power usage for Dataset CIFAR10 and Network ResNet50

5.3.3. GoogLeNet

TensorFlow and PyTorch are not present in the metrics due to the lack of availability of an official model.

Figure 5.3.3.a shows as MXNet achieves the best results in each hardware configuration but the 24.7 speedup from CPU to 1 K20m is the unique positive number about scalability: the difference between 1 and 2 GPUS are not so relevant. Instead, Caffe and CNTK above all can scale remarkably. As it can be easily noticed, Caffe presents the known implementation limits for 2 P100.

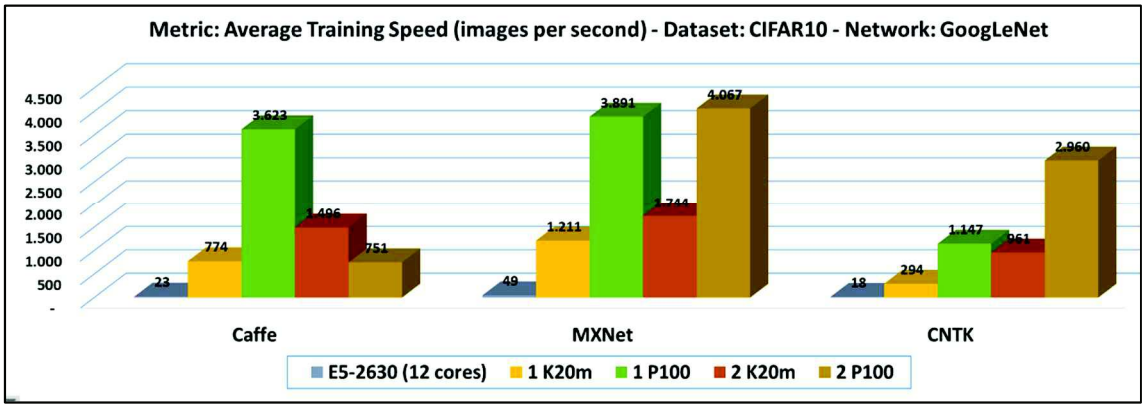


Figure 5.3.3.a – Benchmarks for training speed for Dataset CIFAR10 and Network GoogLeNet

The memory metric (Figure 5.3.3.b) resembles the speed results already shown: MXNet has the best results, Caffe has an average and linear usage, CNTK has very high results especially for K20m.

Also for power (Figure 5.3.3.c) there is not much more to say: MXNet and Caffe are equal on almost all the data, CNTK shows evidently the worst numbers of the group.

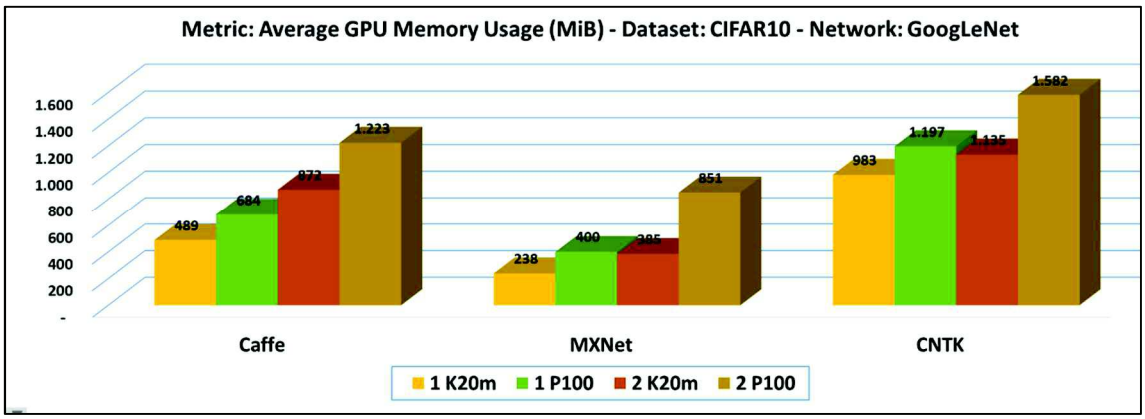


Figure 5.3.3.b – Benchmarks for GPU memory usage for Dataset CIFAR10 and Network GoogLeNet

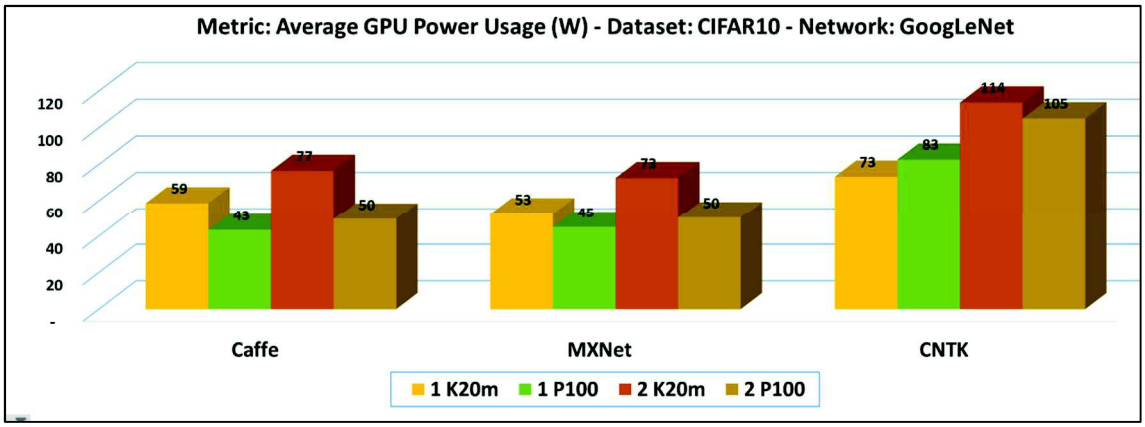


Figure 5.3.3.c – Benchmarks for GPU power usage for Dataset CIFAR10 and Network GoogLeNet

5.3.4. VGG16

TensorFlow and CNTK are not present in the metrics due to the lack of availability of an official model.

In the case of this network, the results are very similar for all the three frameworks. In general, we have the same and not so evident scalability (Figure 5.3.4.a) for each GPU, with MXNet and PyTorch doing slightly better than Caffe.

Caffe and MXNet have now the same profile regarding memory usage (Figure 5.3.4.b): memory follows the increasing number of GPUs with an almost linear trend. PyTorch presents low levels for each configuration but for the 2 P100 spends more memory than the others.

It is the same for power consumption (Figure 5.3.4.c): PyTorch remains the least expensive; Caffe and MXNet are very similar but the second one has higher values for P100 configurations.

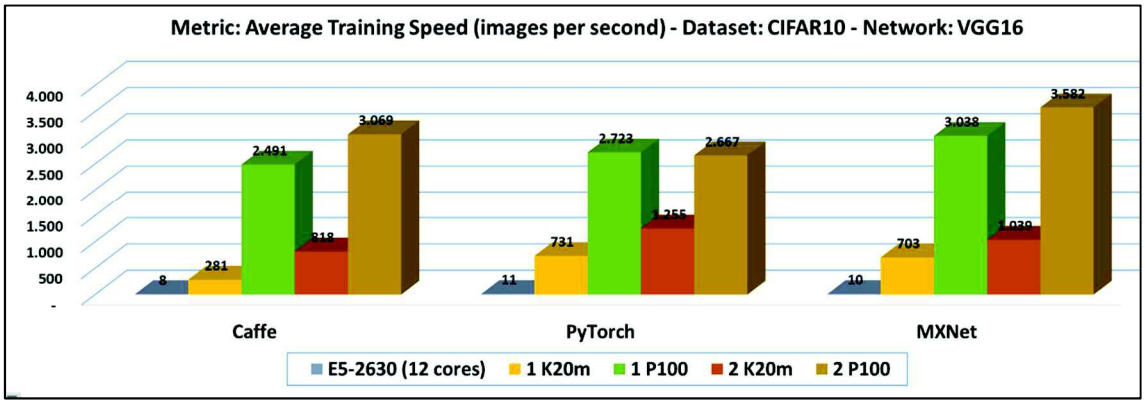


Figure 5.3.4.a – Benchmarks for training speed for Dataset CIFAR10 and Network VGG16

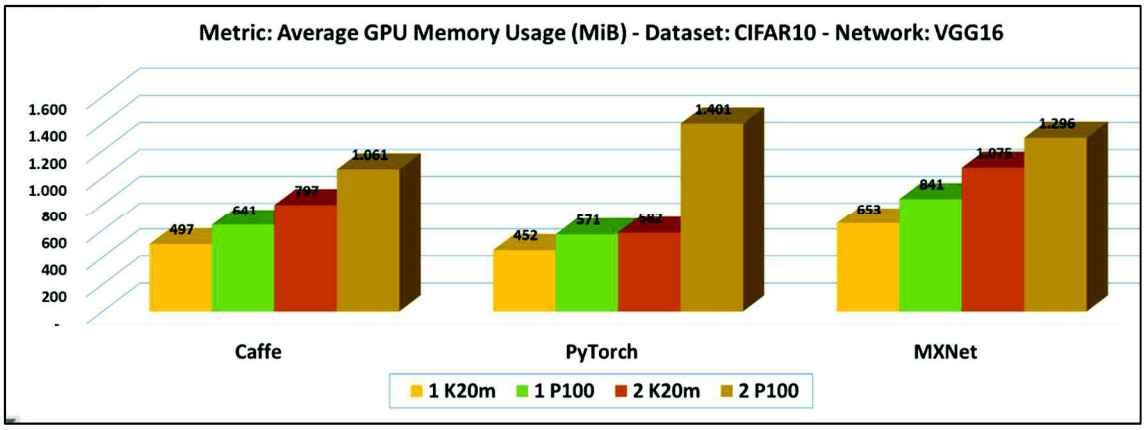


Figure 5.3.4.b – Benchmarks for GPU memory usage for Dataset CIFAR10 and Network VGG16

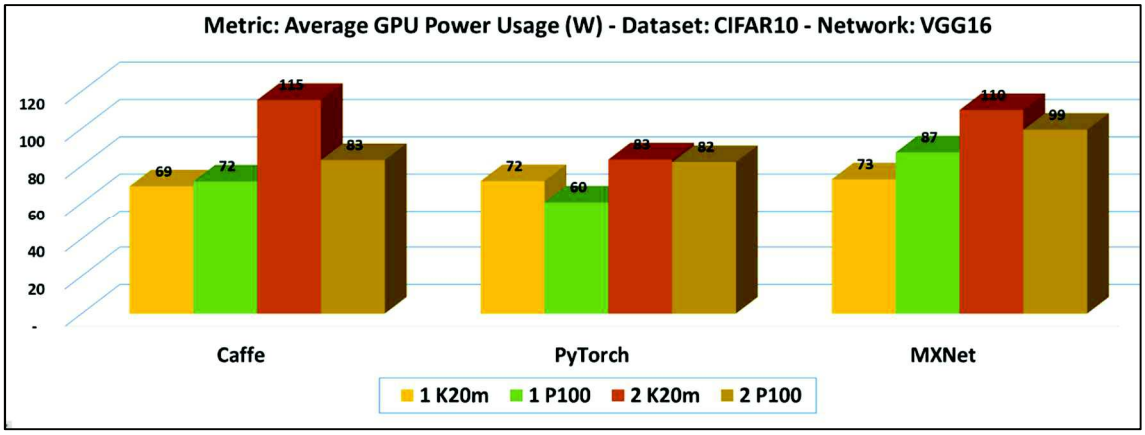


Figure 5.3.4.c – Benchmarks for GPU power usage for Dataset CIFAR10 and Network VGG16

5.4 Data analysis for ImageNet

Here we have the most interesting results since the models are enough relevant and all the benchmarks typically use these networks for evaluation metrics. Furthermore, the network implementation for each framework is much more reliable and significant in terms of performance. In this case, the opportunity to compare data with other benchmark projects is a real possibility and it will give a considerable feedback for the work done.

5.4.1. AlexNet

The training speed reported in Figure 5.4.1.a demonstrates variable trends for all the configurations and each framework behaves in a different way. Caffe has a discrete scalability for K20m and a better one for P100 with low-average data. CNTK manages to obtain very similar data but with less scalability on 2 P100 and a better one on K20m.

TensorFlow follows the same values of Caffe with lower ones on 2 P100 and higher ones on K20m. PyTorch has excellent results with only 1 GPU but the performance does not equally increase with 2 GPUs and MXNet does the same with a higher throughput for 2 K20m and a lower one for 1 K20m.

The values for memory usage are more similar and standard (Figure 4.4.1.b) with Caffe and PyTorch that have the lowest values, MXNet and CNTK are slightly higher. TensorFlow confirms to have a significant GPU memory expensiveness by using a great amount of memory every time.

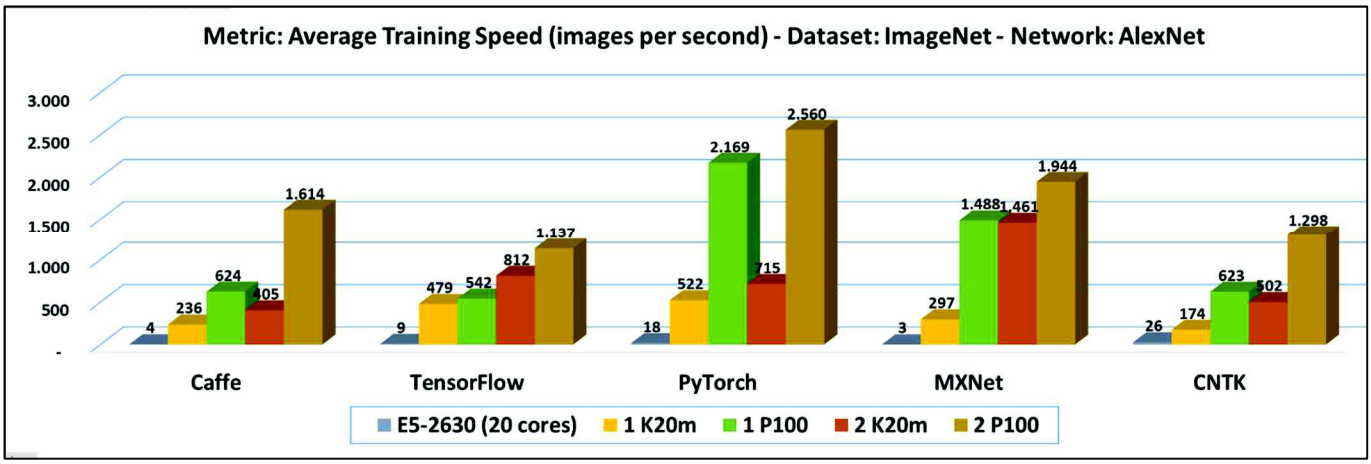


Figure 5.4.1.a – Benchmarks for training speed for Dataset ImageNet and Network AlexNet

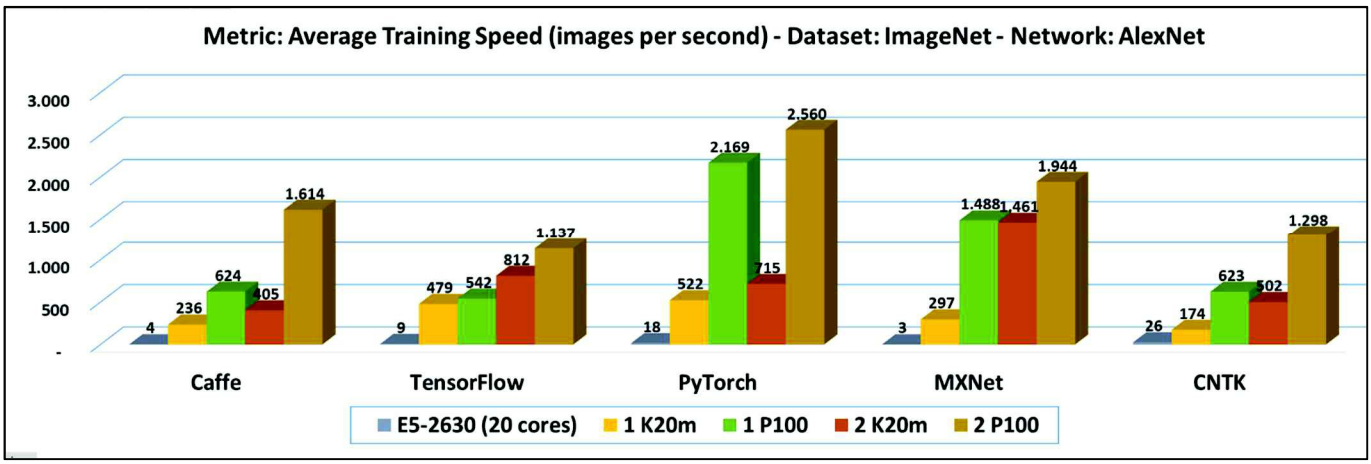


Figure 5.4.1.b – Benchmarks for GPU memory usage for Dataset ImageNet and Network AlexNet

Compared to the power consumption (Figure 5.4.1.c), Caffe, TensorFlow and CNTK share an equal and expected profile. In contrast, PyTorch can obtain good results also in this metric with low levels especially for 2 GPUs (both K20m and P100). Instead,

MXNet values are very much the same for each hardware configuration which is a novelty.

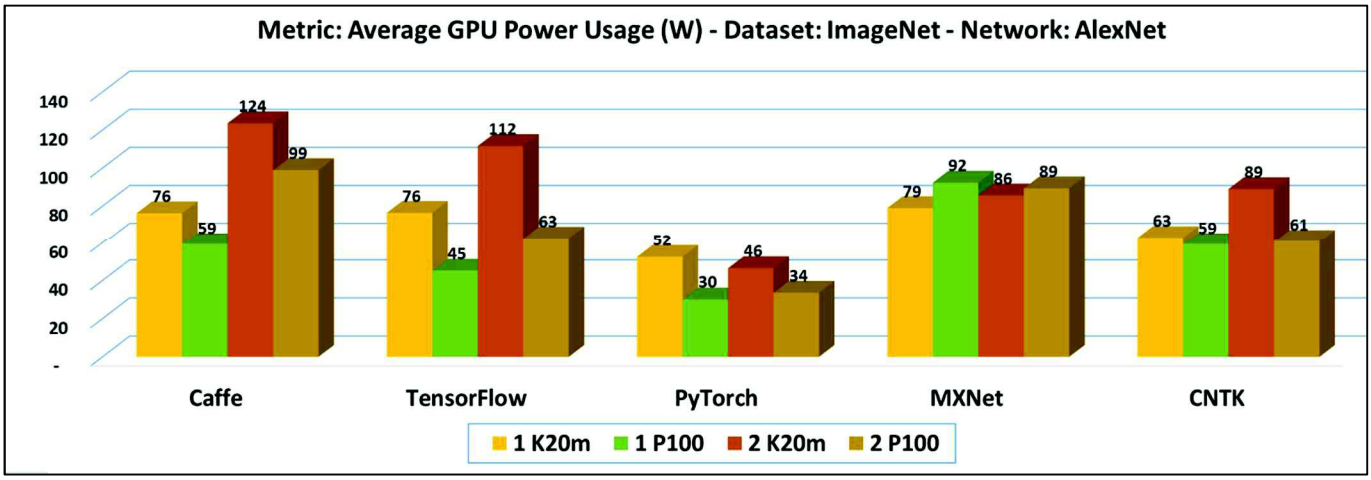


Figure 5.4.1.c – Benchmarks for GPU power usage for Dataset ImageNet and Network AlexNet

5.4.2. ResNet50

As shown in Figure 5.4.2.a, in this case the trend of speed is dissimilar from AlexNet and we have more average data. The only few things to remark are:

- the optimum result of TensorFlow on 2 P100;
- the good scalability of CNTK and TensorFlow;
- the worst throughput for 1 P100 of CNTK, much lower than the average
- the mediocre data obtained by Caffe;
- the recurrent implementation issue of Caffe and PyTorch regarding the 2 P100 configuration.

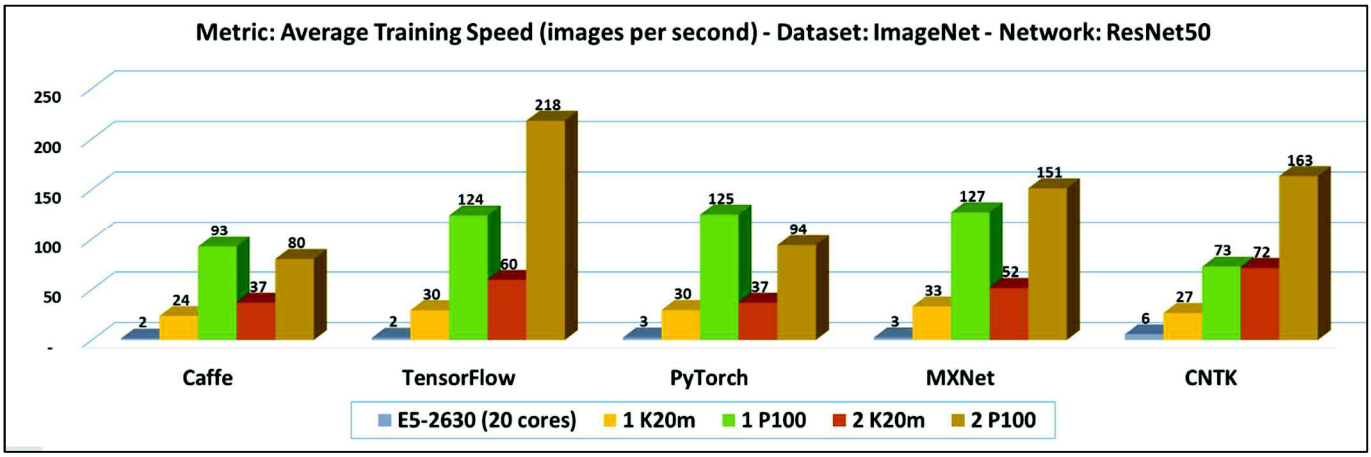


Figure 5.4.2.a – Benchmarks for training speed for Dataset ImageNet and Network ResNet50

As usual, TensorFlow models' implementations reveal themselves to be very memory-consuming (Figure 5.4.2.b). PyTorch, MXNet and Caffe have the same columns but in a different scale as shown in the figures below: the first one is better than the others, the third one is worse. CNTK achieves almost the same result of Caffe but with more consumption proportionally for P100.

Again, PyTorch is a little bit better than the remaining frameworks in the power metric (Figure 5.4.2.c). Generally, all the tools have the usual trend, only CNTK that presents a particularly high value for the P100 and a lower one for the 2 K20m.

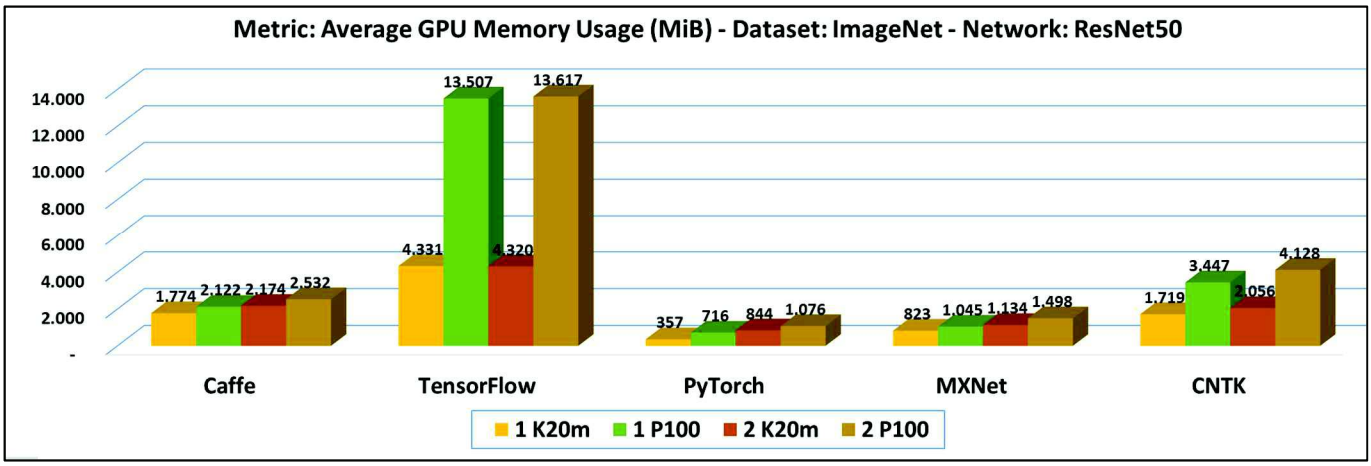


Figure 5.4.2.b – Benchmarks for GPU memory usage for Dataset ImageNet and Network ResNet50

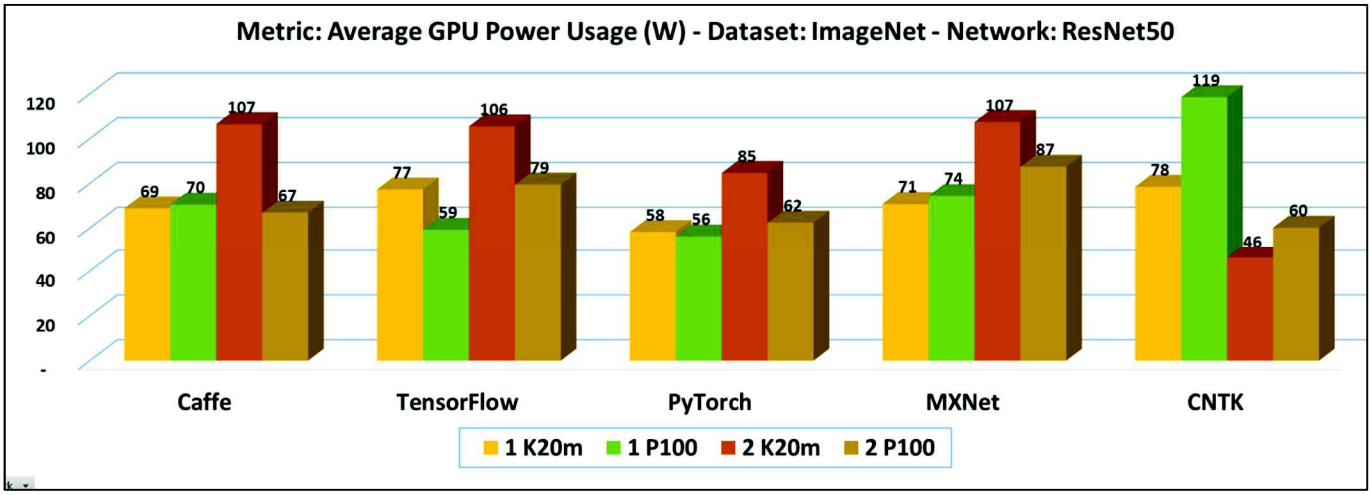


Figure 5.4.2.c – Benchmarks for GPU power usage for Dataset ImageNet and Network ResNet50

5.4.3. GoogLeNet

PyTorch is not present in the metrics due to the lack of availability of an official model.

Even with GoogLeNet for ImageNet, TensorFlow provides the best performance in terms of throughput scalability (Figure 5.4.3.a). Caffe and MXNet give approximately equals results, while CNTK has the lowest values.

TensorFlow is still the primary memory consumer (Figure 5.4.3.b). MXNet presents the best values, along with CNTK. Caffe is just a bit more expensive than these two but much lower than TensorFlow.

Almost nothing to add for power usage (Figure 5.4.3.c): CNTK is a little better than the rest; in addition, the values for 1 P100 and 2 P100 are particularly similar in all the frameworks taken in considerations.

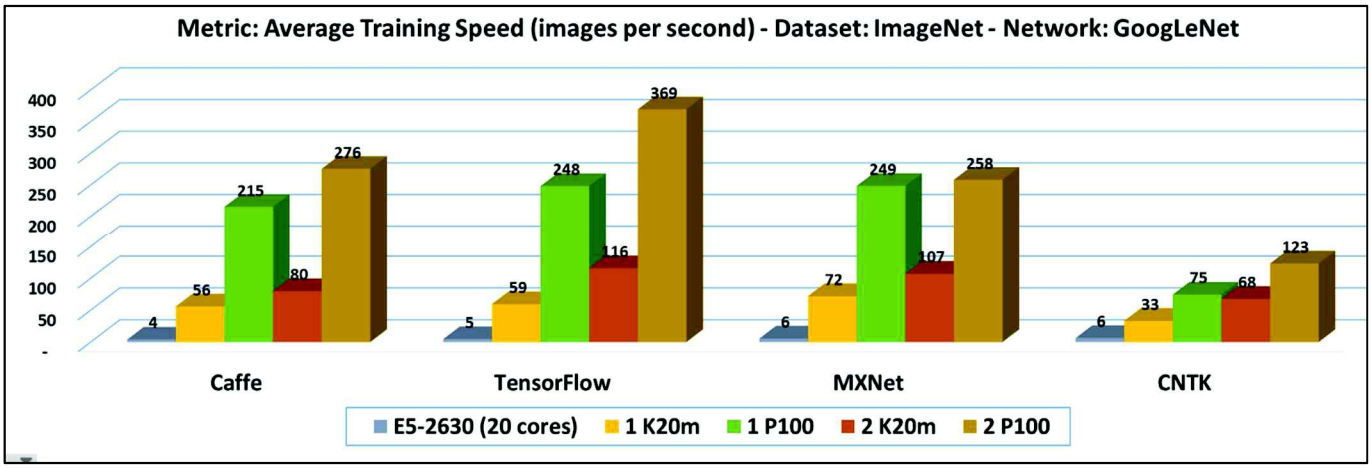


Figure 5.4.3.a – Benchmarks for training speed for Dataset ImageNet and Network GoogLeNet

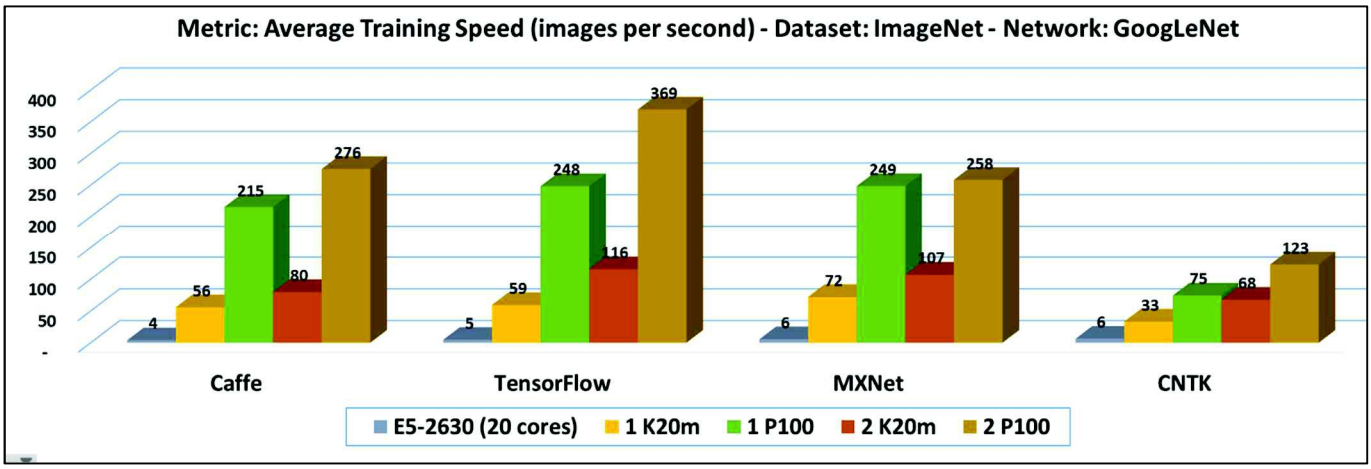


Figure 5.4.3.b – Benchmarks for GPU memory usage for Dataset ImageNet and Network GoogLeNet

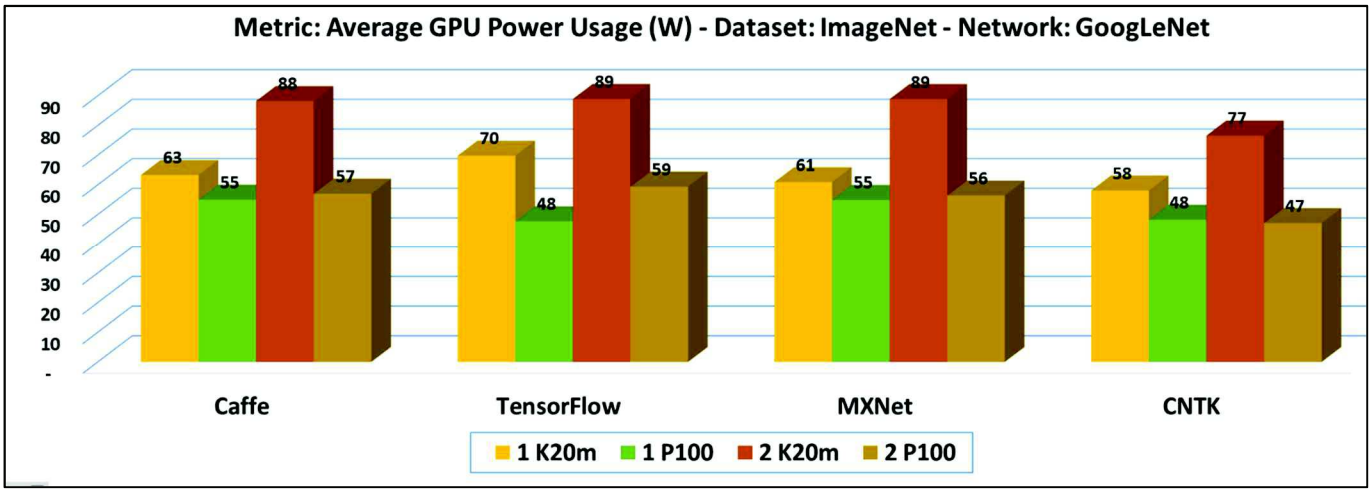


Figure 5.4.3.c – Benchmarks for GPU power usage for Dataset ImageNet and Network GoogLeNet

5.4.4. VGG16

In our final model we can notice that the data shape is analogous in the five frameworks. The figure with the average speed (Figure 5.4.4.a) shows that in every tool the scalability is not remarkable but still appreciable even though Caffe and MXNet implementations does not scale adequately with the P100. TensorFlow manages to achieve the highest values, followed by PyTorch, MXNet, CNTK and Caffe.

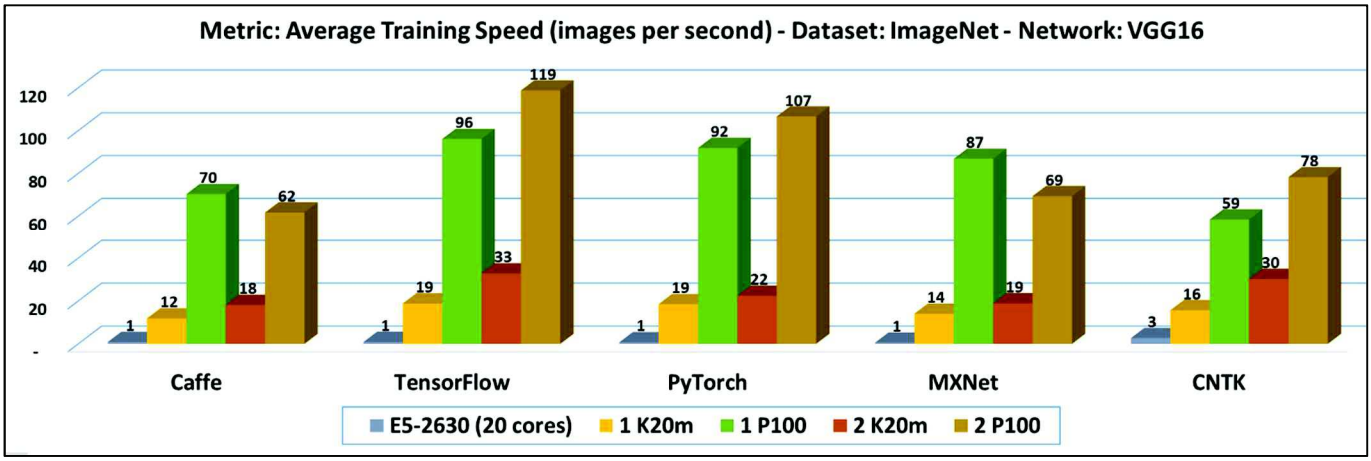


Figure 5.4.4.a – Benchmarks for training speed for Dataset ImageNet and Network VGG16

The highest values are still for TensorFlow regarding the memory usage (Figure 5.4.4.b). The other frameworks share approximately the same numbers:

PyTorch has more equal values among the all architectures, instead MXNet has nearly higher values than the average.

Even the power usage (Figure 5.4.4.c) follows a similar profile in each framework. The main aspects to mention are the lowest levels of CNTK.

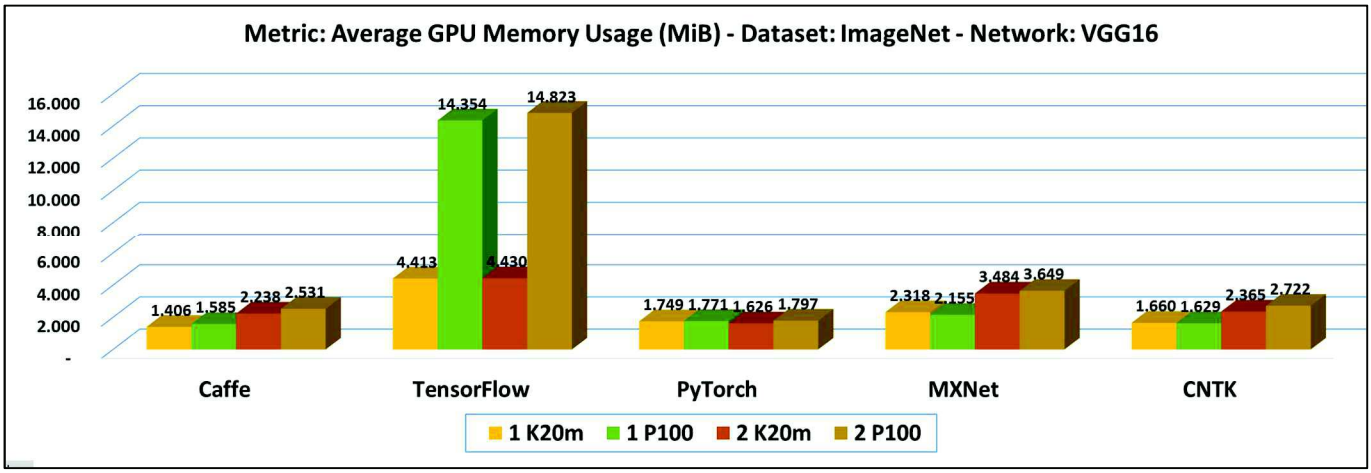


Figure 5.4.4.b – Benchmarks for GPU memory usage for Dataset ImageNet and Network VGG16

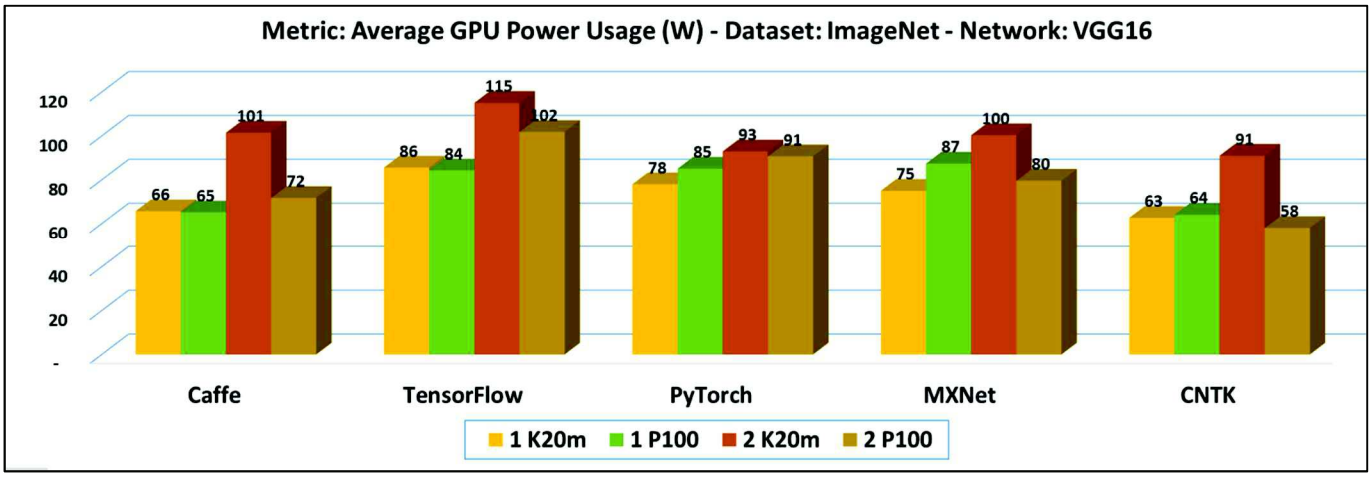


Figure 5.4.4.c – Benchmarks for GPU power usage for Dataset ImageNet and Network VGG16

5.5 Overall considerations

The results offer a wide variety of considerations from numerous points of view. The purpose of this discussion is not to merely find the best network or a winner among the frameworks used because it would not be appropriate. The main intention is to offer data to be interpreted and analysed for research goals and obtain more specific and interesting considerations about hardware architectures, deep learning software and neural networks.

Some **preliminary conclusions** which were practically expected for the study:

- using a GPU gives better performances than using a CPU;

- by increasing the number of GPUs, we have an improvement but the speedup is not as much as moving from CPU to GPU in most cases. This parameter is actually going to increase with a longer training session and a higher batch size;
- the P100 evidently outperforms the K20m in both speed and resources consumption;
- the more the GPUs are, the more the memory consumption and, almost always, the power usage are;
- scalability varies from one configuration to another.

As it was evident from the graphs above, performance is extremely variable among networks, datasets and GPUs. By comparing and making a logical evaluation, it is possible to discover recurrent trends and behaviour among the great amount of collected data.

For example, one of the clearest aspects was the huge expensiveness of TensorFlow in terms of GPU memory usage. In fact, the Deep Learning framework from Google was almost always far more expensive than the other competitors. There is actually a real explanation behind this anomaly: the official models of TensorFlow which were picked up for this benchmarking are optimized for high performance, as stated in the previous chapter, and this is the reason why they consumed lots of GPU memory. Nevertheless, TensorFlow does not overwhelm the other frameworks for training speed (even though it is still one of the best) and this is something to take into account because the massive expensiveness of resources is not justified by the performance in speed.

Another notable fact is the significantly lack of speed of Caffe, which in several configurations does not perform as much as the remaining tools. This statistic can be clarified by remembering that Caffe is an old tool and not maintained anymore. Certainly, the newest versions of the competitors are up-to-date with the latest improvements in performance. However, the NVIDIA Caffe variant could keep up with the latest versions of the other frameworks and it is our intention to develop this work by adding performance benchmarks also for this modified version.

By comparing the power usage, we can see that there are no significant differences in the configurations analysed. In fact, the amount consumed of power does not change in a very substantial way for the same network apart from some exceptions (like in the case of AlexNet trained with ImageNet). On the other hand, each network has its own requirements and specifications and the amount of power reflects that own structure. The same discussion could be made for memory usage because, without considering the atypical values of TensorFlow, all the other tools present a similar profile and trend for the same network.

Talking about scalability, the discussion is much more complex because the trend is not so evident and more variable. Clearly, TensorFlow and CNTK have shown a better predisposition to scale with more GPUs in quite a few networks, especially on ImageNet.

But Caffe, even though with low numbers, also demonstrated that it can improve its performance better than others in some occasions.

A note must be done for MXNet which was average in the performance benchmarks but gave us the opportunity to test the models in the easiest way, with few code to modify and a documentation which was absolutely rich and clear. This aspect is not pertinent with the data, but in this evaluation context, where all the frameworks have advantages and disadvantages and it is difficult to see a leader, some minor points must be considered. Therefore, if the reader is looking for a very easy and complete benchmark, with explanations and a great number of parameters to personalize the training session, MXNet should have a try.

Our benchmark took into the analysis the most popular models in the image recognition task, which are already well studied and compared. For this reason, we are not going to talk deeply about their comparison for obvious reasons. However, our data can confirm that a deeper and more elaborate network will consume more resources and take more time to be trained. For instance, VGG16 and ResNet50 have shown to be worst in power, memory and speed results than AlexNet and GoogLeNet and these numbers were certainly expected given their complexity.

It is obvious that hundreds of additional considerations could be made for all the factors but we decided to make things simple and understandable by everyone, focusing only on the main points which were more interesting and relevant. A future goal will be to expand the data and the evaluations on a deeper level of abstractions in order to fully please the most exigent community.

5.6 Result validation

Usually it is common to compare the results of a project for assessing the correctness of the data obtained. However, this work is not a replication of another benchmark system: the development and the general structure are absolutely new and personal, maybe also not perfect but the progress of the work was based only on a personal study of the technology and on individual decisions, though it took inspiration from the projects described in chapter 2.

In order to offer some examples, it can be useful to consider all the efforts spent on the analysis, the design and the planning of the problem: the single parameters, the specific architecture configurations, the models picked and the modifications made are unique.

For this reason, comparing in a very accurate and precise way the results obtained with other benchmarks that used other parameters and configurations does not have any sense. It would be only deceiving and confusing. For instance, a benchmark that uses synthetic data with a different batch size, number of epochs and iterations will be never comparable with our data even if the dataset, the network and the GPU are the same.

Anyway, other famous benchmarks must be interpreted in relation with our work not specifically but in a more abstract way. The results have been confronted with the benchmarks taken from the DLBENCH project page, the TensorFlow official performance guide and the NVIDIA Data Center page and the scalability in each of them resembles the trend of our data and graphs for CPU and GPU training.

Nevertheless, we can also attest the likelihood of our results and observe if there are any peculiarities and singularities that do not fit the situation and the context. Certainly, having had a 2 P100 training that runs slower than a CPU training evidently shows a present issue and explanations must be found to correct or to explain this problem.

But this is not the case for these data: almost all the values respect an expected and standard behaviour and logic. Unusual numbers have been remarked during the previous sections and they were caused by the low batch size and the model implementation (remember that a low batch size was chosen for the GPU memory limit of the K20m). Apart from these exceptions, the rest of our results seems to be in accordance with our common sense and this certify the quality of the work done.

CHAPTER 6
Conclusion

There is no doubt that this work has developed topics and themes which are relevant to understand the new opportunities and the future developments of Deep Learning. The precise analysis and methodology allowed to elaborate a simple but useful benchmarking project that aims to satisfy some requirements and, after the deployment and the verification, we can say that the final results can absolutely attest the value of the efforts made.

The followed path has been helpful for achieving the desired outcomes: the general study of Deep Learning was intended to acquire the necessary knowledge for experimenting and doing research in this field; then, a scrupulous and attentive analysis of benchmarking approaches and concepts was carried out in order to focus the problem and face it in the best way possible with personal and original methods; after that, the project was realised according to the previous study and every choice was taken for refining every detail; after all this preparation, the experiments were carried out and the obtained data were evaluated with interesting discussions.

The work has fulfilled all the initially proposed objectives. First of all, the technology and the primary Deep Learning notions have been fully explained. Most of all, the created model for the evaluation of Deep Learning environments was able not only to provide useful and completed results for a wide range of hardware and software configurations but it also efficiently analysed and compared them from the points of interests which were considered more valuable in the specific context. Moreover, the outcomes could be used to assess the performance of a system and optimise it with the improvements shown.

Although we can be gratified with the final product, there is still work to do to perfect and to expand the whole project. The problems encountered during the progress have made us take some decisions that can affect the effectiveness of the evaluation. For example, some models' implementations do not reflect the maximum efficiency needed to have the best performance, as well as several frameworks' parameters.

Therefore, there are some enhancements that can be generally applied to enrich the entire system but that is not all. On one hand, the intention is to test the benchmark on multiple hardware: this means to examine the models with more and different GPUs in order to provide a more interesting overview of the scalability. On the other hand, we feel the need to analyse other frameworks like Caffe2 and the NVIDIA Caffe project which are very popular among the Deep Learning users and are believed to be very efficient and

promising for the data they could offer. Moreover, different metrics could be measured and a more complete vision of the tests could be made in this way.

Finally, the most ambitious goal is distributed training: recently, almost every framework has offered the possibility to train the network by using multiple nodes of a cluster. This is actually the best way to train quickly a network and reach a huge and amazing performance. Since there are very few projects that deal with this task, the proposed extension could be much more innovative and helpful for a wider range of users and could bring an additional contribution to the area of Deep Learning benchmarking.

Bibliography

[1] Agatonovic-Krustin S., Beresford R., *Basic concepts of artificial neural network (ANN) modelling and its application in pharmaceutical research*, Journal of Pharmaceutical and Biomedical Analysis, volume 22 – issue 5, 2000. Available at: https://www.sciencedirect.com/science/article/pii/S0731708599002721 (Accessed: 28 June 2018)

[2] Bengio Y., Hinton G., Lecun Y., *Deep Learning*, Nature, vol. 521, no. 7553, 27 May 2015. Available at: https://www.nature.com/articles/nature14539 (Accessed: 28 June 2018)

[3] Chen Y., Yang T., Sze V., *Efficient Processing of Deep Neural Networks: a Tutorial and Survey*, 13 August 2017. Available at: https://arxiv.org/pdf/1703.09039.pdf (Accessed: 28 June 2018)

[4] Chintala S., Convnet-benchmarks, GitHub, Inc., 9 June 2017, Available at: https://github.com/soumith/convnet-benchmarks (Accessed: 29 June 2018)

[5] Chu X., Shi S., Xu P., Wang Q., *Benchmarking State-of-the-Art Deep Learning Software Tools*, Department of Computer Science, Hong Kong Baptist University, 17 February 2017. Available at: https://arxiv.org/pdf/1608.07249.pdf (Accessed: 29 June 2018)

[6] Coleman C., Chris Ré, Matei Z., Narayanan D., Nardi L., Kang D., Kunle O., Peter B, Zhang J., Zhao T., *DAWNBench. An End-to-End Deep Learning Benchmark and Competition*, NIPS ML Systems Workshop, 2017. Available at: https://dawn.cs.stanford.edu/benchmark/about.html (Accessed: 28 June 2018)

[7] Conda, *Conda documentation.* Available at: https://conda.io/docs/index.html (Accessed: 29 June 2018)

[8] Den Bakker I., *Battle of the Deep Learning frameworks—Part I: 2017, even more frameworks and interfaces*, 19 December 2017. Available at: https://towardsdatascience.com/battle-of-the-deep-learning-frameworks-part-i-cff0e3841750 (Accessed: 29 June 2018)

[9] Gupta R., *Getting started with Neural Network for regression and Tensorflow*, Medium Corporation, 26 June 2017. Available at: https://medium.com/@rajatgupta310198/getting-started-with-neural-network-for-regression-and-tensorflow-58ad3bd75223 (Accessed: 28 June 2018)

[10] Hinton G.E., Krizhevsky A., Sutskever I., *ImageNet Classification with Deep Convolutional Neural Networks*, 18 May 2015. Available at: http://vision.stanford.edu/teaching/cs231b_spring1415/slides/alexnet_tugce_kyunghee.pdf (Accessed: 28 June 2018)

[11] Johnson J., *Benchmarks for popular CNN models*, GitHub, Inc., 25 September 2017. Available at: https://github.com/jcjohnson/cnn-benchmarks (Accessed: 29 June 2018)

[12] Krizhevsky A., Hinton G.E., Sutskever I., *ImageNet Classification with Deep Convolutional Neural Networks*, 2012, Available at: https://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf (Accessed: 29 June 2018)

[13] Lawrence J., Malmsten J., Rybka A., Sabol D.A., Triplin K., *Comparing TensorFlow Deep Learning Performance Using CPUs, GPUs, Local PCs and Cloud,* Research Day, Pace University, 5 May 2017. Available at: https://www.semanticscholar.org/paper/Comparing-TensorFlow-Deep-Learning-Performance-PCs-Lawrence-Malmsten/42ceccb61c35613bc262c47b35e392ec79ac247d (Accessed: 29 June 2018)

[14] Le J., *The 10 Deep Learning Methods AI Practitioners Need to Apply*, Towards Data Science, 17 November 2017. Available at: https://towardsdatascience.com/the-10-deep-learning-methods-ai-practitioners-need-to-apply-885259f402c1 (Accessed: 28 June 2018)

[15] McCarthy J., *What is AI? / Basic Questions*. Available at: http://jmc.stanford.edu/artificial-intelligence/what-is-ai/index.html (Accessed: 28 June 2018)

[16] Narang S., *DeepBench*, Baidu Research, 26 September 2016. Available at: https://svail.github.io/DeepBench/ (Accessed: 28 June 2018)

[17] NVIDIA Developer, *CUDA Zone.* Available at: https://developer.nvidia.com/cuda-zone (Accessed: 28 June 2018)

[18] NVIDIA Developer, *NVIDIA Collective Communications Library (NCCL). Multi-GPU and multi-node collective communication primitives.* Available at: https://developer.nvidia.com/nccl (Accessed: 28 June 2018)

[19] NVIDIA Developer, *NVIDIA cuDNN. GPU Accelerated Deep Learning*. Available at: https://developer.nvidia.com/cudnn (Accessed: 28 June 2018)

[20] Roell J., *Understanding Recurrent Neural Networks: The Preferred Neural Network for Time-Series Data*, 26 June 2017. Available at: https://towardsdatascience.com/understanding-recurrent-neural-networks-the-preferred-neural-network-for-time-series-data-7d856c21b759 (Accessed: 28 June 2018)

[21] Shaikh F., *Why are GPUs necessary for training Deep Learning models?*, 18 May 2017. Available at: https://www.analyticsvidhya.com/blog/2017/05/gpus-necessary-for-deep-learning/ (Accessed: 29 June 2018)

[22] Simonyan K., Zisserman A., *Very Deep Convolutional Networks for Large-Scale Image Recognition*, 10 April 2015. Available at: https://arxiv.org/pdf/1409.1556.pdf (Accessed: 29 June 2018)

[23] Stanford University Infolab, *Arthur Samuel: Pioneer in Machine Learning*. Available at: http://infolab.stanford.edu/pub/voy/museum/samuel.html (Accessed: 28 June 2018)

[24] The Microsoft Cognitive Toolkit, *CNTK 102: Feed Forward Network with Simulated Data*. Available at: https://cntk.ai/pythondocs/CNTK_102_FeedForward.html (Accessed: 28 June 2018)

[25] University of Cincinnati, *Artificial Neural Network Fundamentals*. Available at: http://uc-r.github.io/ann_fundamentals (Accessed: 28 June 2018)

[26] Vassilieva N., Serebryakov S., *Deep Learning Benchmarking Suit*e. Available at: https://github.com/HewlettPackard/dlcookbook-dlbs (Accessed: 28 June 2018)

[27] Wikipedia, *Artificial intelligence*. Available at: https://en.wikipedia.org/wiki/Artificial_intelligence (Accessed: 29 June 2018)

[28] Wikipedia, *Graphic processing unit*. Available at: https://en.wikipedia.org/wiki/Graphics_processing_unit (Accessed: 29 June 2018)

[29] Wikipedia, *Machine learning*. Available at: https://en.wikipedia.org/wiki/Machine_learning (Accessed: 29 June 2018)

APPENDIX A
Table of results

Metric	Dataset	Network	Frame-work	E5-2630 (12/20 cores)	1 GPU K20m	1 GPU P100	2 GPU K20m	2 GPU P100
Average Training Speed (images per second)	MNIST	LeNet	Caffe	572	19.807	42.588	22.734	46.110
Average Training Speed (images per second)	MNIST	LeNet	TensorFlow	818	4.539	9.846	4.723	11.034
Average Training Speed (images per second)	MNIST	LeNet	PyTorch	1.448	4.320	9.524	6.105	10.345
Average Training Speed (images per second)	MNIST	LeNet	MXNet	1.000	18.372	46.691	22.191	34.137
Average Training Speed (images per second)	MNIST	LeNet	CNTK	1.573	8.191	10.510	9.234	14.609
Average Training Speed (images per second)	CIFAR10	AlexNet	Caffe	88	4.437	17.903	7.668	28.179
Average Training Speed (images per second)	CIFAR10	AlexNet	TensorFlow	405	3.875	9.029	6.137	9.158
Average Training Speed (images per second)	CIFAR10	AlexNet	PyTorch	84	3.556	7.729	5.120	8.533
Average Training Speed (images per second)	CIFAR10	AlexNet	MXNet	55	3.119	12.154	2.643	7.708
Average Training Speed (images per second)	CIFAR10	AlexNet	CNTK	0	0	0	0	0
Average Training Speed (images per second)	CIFAR10	ResNet50	Caffe	88	192	734	478	1.558
Average Training Speed (images per second)	CIFAR10	ResNet50	TensorFlow	23	579	2.375	1.118	4.381
Average Training Speed (images per second)	CIFAR10	ResNet50	PyTorch	17	438	1.684	977	1.231

Metric	Dataset	Network	Frame-work	E5-2630 (12/20 cores)	1 GPU K20m	1 GPU P100	2 GPU K20m	2 GPU P100
Average Training Speed (images per second)	CIFAR10	ResNet50	MXNet	35	982	3.237	1.573	3.724
Average Training Speed (images per second)	CIFAR10	ResNet50	CNTK	108	1.115	2.385	1.407	5.582
Average Training Speed (images per second)	CIFAR10	GoogLeNet	Caffe	23	774	3.623	1.496	751
Average Training Speed (images per second)	CIFAR10	GoogLeNet	TensorFlow	0	0	0	0	0
Average Training Speed (images per second)	CIFAR10	GoogLeNet	PyTorch	0	0	0	0	0
Average Training Speed (images per second)	CIFAR10	GoogLeNet	MXNet	49	1.211	3.891	1.744	4.067
Average Training Speed (images per second)	CIFAR10	GoogLeNet	CNTK	18	294	1.147	961	2.960
Average Training Speed (images per second)	CIFAR10	VGG16	Caffe	8	281	2.491	818	3.069
Average Training Speed (images per second)	CIFAR10	VGG16	TensorFlow	0	0	0	0	0
Average Training Speed (images per second)	CIFAR10	VGG16	PyTorch	11	731	2.723	1.255	2.667
Average Training Speed (images per second)	CIFAR10	VGG16	MXNet	10	703	3.038	1.039	3.582
Average Training Speed (images per second)	CIFAR10	VGG16	CNTK	0	0	0	0	0
Average Training Speed (images per second)	ImageNet	AlexNet	Caffe	4	236	624	405	1.614
Average Training Speed (images per second)	ImageNet	AlexNet	TensorFlow	9	479	542	812	1.137
Average Training Speed (images per second)	ImageNet	AlexNet	PyTorch	18	522	2.169	715	2.560
Average Training Speed (images per second)	ImageNet	AlexNet	MXNet	3	297	1.488	1.461	1.944
Average Training Speed (images per second)	ImageNet	AlexNet	CNTK	26	174	623	502	1.298
Average Training Speed (images per second)	ImageNet	ResNet50	Caffe	2	24	93	37	80

Metric	Dataset	Network	Frame-work	E5-2630 (12/20 cores)	1 GPU K20m	1 GPU P100	2 GPU K20m	2 GPU P100
Average Training Speed (images per second)	ImageNet	ResNet50	TensorFlow	2	30	124	60	218
Average Training Speed (images per second)	ImageNet	ResNet50	PyTorch	3	30	125	37	94
Average Training Speed (images per second)	ImageNet	ResNet50	MXNet	3	33	127	52	151
Average Training Speed (images per second)	ImageNet	ResNet50	CNTK	6	27	73	72	163
Average Training Speed (images per second)	ImageNet	GoogLeNet	Caffe	4	56	215	80	276
Average Training Speed (images per second)	ImageNet	GoogLeNet	TensorFlow	5	59	248	116	369
Average Training Speed (images per second)	ImageNet	GoogLeNet	PyTorch	0	0	0	0	0
Average Training Speed (images per second)	ImageNet	GoogLeNet	MXNet	6	72	249	107	258
Average Training Speed (images per second)	ImageNet	GoogLeNet	CNTK	6	33	75	68	123
Average Training Speed (images per second)	ImageNet	VGG16	Caffe	1	12	70	18	62
Average Training Speed (images per second)	ImageNet	VGG16	TensorFlow	1	19	96	33	119
Average Training Speed (images per second)	ImageNet	VGG16	PyTorch	1	19	92	22	107
Average Training Speed (images per second)	ImageNet	VGG16	MXNet	1	14	87	19	69
Average Training Speed (images per second)	ImageNet	VGG16	CNTK	3	16	59	30	78
Average GPU Memory Usage (MiB)	MNIST	LeNet	Caffe	0	140	291	190	405
Average GPU Memory Usage (MiB)	MNIST	LeNet	TensorFlow	0	2.462	2.828	4.481	12.831
Average GPU Memory Usage (MiB)	MNIST	LeNet	PyTorch	0	167	340	175	354
Average GPU Memory Usage (MiB)	MNIST	LeNet	MXNet	0	134	180	188	395

Metric	Dataset	Network	Frame-work	E5-2630 (12/20 cores)	1 GPU K20m	1 GPU P100	2 GPU K20m	2 GPU P100
Average GPU Memory Usage (MiB)	MNIST	LeNet	CNTK	0	75	193	101	257
Average GPU Memory Usage (MiB)	CIFAR10	AlexNet	Caffe	0	157	318	228	301
Average GPU Memory Usage (MiB)	CIFAR10	AlexNet	TensorFlow	0	3.370	4.293	3.208	10.158
Average GPU Memory Usage (MiB)	CIFAR10	AlexNet	PyTorch	0	157	345	346	602
Average GPU Memory Usage (MiB)	CIFAR10	AlexNet	MXNet	0	324	502	583	824
Average GPU Memory Usage (MiB)	CIFAR10	AlexNet	CNTK	0	0	0	0	0
Average GPU Memory Usage (MiB)	CIFAR10	ResNet50	Caffe	0	1.334	1.484	1.569	1.754
Average GPU Memory Usage (MiB)	CIFAR10	ResNet50	TensorFlow	0	4.132	5.639	3.807	10.882
Average GPU Memory Usage (MiB)	CIFAR10	ResNet50	PyTorch	0	840	999	741	1.110
Average GPU Memory Usage (MiB)	CIFAR10	ResNet50	MXNet	0	411	575	507	742
Average GPU Memory Usage (MiB)	CIFAR10	ResNet50	CNTK	0	241	453	495	822
Average GPU Memory Usage (MiB)	CIFAR10	GoogLeNet	Caffe	0	489	684	872	1.223
Average GPU Memory Usage (MiB)	CIFAR10	GoogLeNet	TensorFlow	0	0	0	0	0
Average GPU Memory Usage (MiB)	CIFAR10	GoogLeNet	PyTorch	0	0	0	0	0
Average GPU Memory Usage (MiB)	CIFAR10	GoogLeNet	MXNet	0	238	400	385	851
Average GPU Memory Usage (MiB)	CIFAR10	GoogLeNet	CNTK	0	983	1.197	1.135	1.582
Average GPU Memory Usage (MiB)	CIFAR10	VGG16	Caffe	0	497	641	797	1.061
Average GPU Memory Usage (MiB)	CIFAR10	VGG16	TensorFlow	0	0	0	0	0

Metric	Dataset	Network	Frame-work	E5-2630 (12/20 cores)	1 GPU K20m	1 GPU P100	2 GPU K20m	2 GPU P100
Average GPU Memory Usage (MiB)	CIFAR10	VGG16	PyTorch	0	452	571	582	1.401
Average GPU Memory Usage (MiB)	CIFAR10	VGG16	MXNet	0	653	841	1.075	1.296
Average GPU Memory Usage (MiB)	CIFAR10	VGG16	CNTK	0	0	0	0	0
Average GPU Memory Usage (MiB)	ImageNet	AlexNet	Caffe	0	1.288	1.472	1.726	1.942
Average GPU Memory Usage (MiB)	ImageNet	AlexNet	TensorFlow	0	4.203	14.230	4.430	15.201
Average GPU Memory Usage (MiB)	ImageNet	AlexNet	PyTorch	0	1.036	1.081	902	1.949
Average GPU Memory Usage (MiB)	ImageNet	AlexNet	MXNet	0	1.561	1.680	2.628	2.629
Average GPU Memory Usage (MiB)	ImageNet	AlexNet	CNTK	0	1.798	2.030	2.342	2.986
Average GPU Memory Usage (MiB)	ImageNet	ResNet50	Caffe	0	1.774	2.122	2.174	2.532
Average GPU Memory Usage (MiB)	ImageNet	ResNet50	TensorFlow	0	4.331	13.507	4.320	13.617
Average GPU Memory Usage (MiB)	ImageNet	ResNet50	PyTorch	0	357	716	844	1.076
Average GPU Memory Usage (MiB)	ImageNet	ResNet50	MXNet	0	823	1.045	1.134	1.498
Average GPU Memory Usage (MiB)	ImageNet	ResNet50	CNTK	0	1.719	3.447	2.056	4.128
Average GPU Memory Usage (MiB)	ImageNet	GoogLeNet	Caffe	0	755	945	1.135	1.340
Average GPU Memory Usage (MiB)	ImageNet	GoogLeNet	TensorFlow	0	4.296	12.752	4.339	13.485
Average GPU Memory Usage (MiB)	ImageNet	GoogLeNet	PyTorch	0	0	0	0	0
Average GPU Memory Usage (MiB)	ImageNet	GoogLeNet	MXNet	0	383	493	509	940
Average GPU Memory Usage (MiB)	ImageNet	GoogLeNet	CNTK	0	406	604	562	984

Metric	Dataset	Network	Frame-work	E5-2630 (12/20 cores)	1 GPU K20m	1 GPU P100	2 GPU K20m	2 GPU P100
Average GPU Memory Usage (MiB)	ImageNet	VGG16	Caffe	0	1.406	1.585	2.238	2.531
Average GPU Memory Usage (MiB)	ImageNet	VGG16	TensorFlow	0	4.413	14.354	4.430	14.823
Average GPU Memory Usage (MiB)	ImageNet	VGG16	PyTorch	0	1.749	1.771	1.626	1.797
Average GPU Memory Usage (MiB)	ImageNet	VGG16	MXNet	0	2.318	2.155	3.484	3.649
Average GPU Memory Usage (MiB)	ImageNet	VGG16	CNTK	0	1.660	1.629	2.365	2.722
Average GPU Power Usage (W)	MNIST	LeNet	Caffe	33	71	46	64	36
Average GPU Power Usage (W)	MNIST	LeNet	TensorFlow	32	72	35	71	42
Average GPU Power Usage (W)	MNIST	LeNet	PyTorch	31	51	30	48	32
Average GPU Power Usage (W)	MNIST	LeNet	MXNet	32	68	32	65	35
Average GPU Power Usage (W)	MNIST	LeNet	CNTK	32	58	35	65	43
Average GPU Power Usage (W)	CIFAR10	AlexNet	Caffe	32	81	67	96	41
Average GPU Power Usage (W)	CIFAR10	AlexNet	TensorFlow	32	64	33	62	34
Average GPU Power Usage (W)	CIFAR10	AlexNet	PyTorch	31	55	35	65	43
Average GPU Power Usage (W)	CIFAR10	AlexNet	MXNet	32	70	65	78	56
Average GPU Power Usage (W)	CIFAR10	AlexNet	CNTK	0	0	0	0	0
Average GPU Power Usage (W)	CIFAR10	ResNet50	Caffe	32	52	51	95	70
Average GPU Power Usage (W)	CIFAR10	ResNet50	TensorFlow	31	75	49	87	52
Average GPU Power Usage (W)	CIFAR10	ResNet50	PyTorch	31	78	61	88	45

Metric	Dataset	Network	Frame-work	E5-2630 (12/20 cores)	1 GPU K20m	1 GPU P100	2 GPU K20m	2 GPU P100
Average GPU Power Usage (W)	CIFAR10	ResNet50	MXNet	32	69	67	101	65
Average GPU Power Usage (W)	CIFAR10	ResNet50	CNTK	33	55	42	79	81
Average GPU Power Usage (W)	CIFAR10	GoogLeNet	Caffe	32	59	43	77	50
Average GPU Power Usage (W)	CIFAR10	GoogLeNet	TensorFlow	0	0	0	0	0
Average GPU Power Usage (W)	CIFAR10	GoogLeNet	PyTorch	0	0	0	0	0
Average GPU Power Usage (W)	CIFAR10	GoogLeNet	MXNet	32	53	45	73	50
Average GPU Power Usage (W)	CIFAR10	GoogLeNet	CNTK	32	73	83	114	105
Average GPU Power Usage (W)	CIFAR10	VGG16	Caffe	32	69	72	115	83
Average GPU Power Usage (W)	CIFAR10	VGG16	TensorFlow	0	0	0	0	0
Average GPU Power Usage (W)	CIFAR10	VGG16	PyTorch	34	72	60	83	82
Average GPU Power Usage (W)	CIFAR10	VGG16	MXNet	32	73	87	110	99
Average GPU Power Usage (W)	CIFAR10	VGG16	CNTK	0	0	0	0	0
Average GPU Power Usage (W)	ImageNet	AlexNet	Caffe	28	76	59	124	99
Average GPU Power Usage (W)	ImageNet	AlexNet	TensorFlow	28	76	45	112	63
Average GPU Power Usage (W)	ImageNet	AlexNet	PyTorch	28	52	30	46	34
Average GPU Power Usage (W)	ImageNet	AlexNet	MXNet	28	79	92	86	89
Average GPU Power Usage (W)	ImageNet	AlexNet	CNTK	29	63	59	89	61
Average GPU Power Usage (W)	ImageNet	ResNet50	Caffe	28	69	70	107	67

Metric	Dataset	Network	Frame-work	E5-2630 (12/20 cores)	1 GPU K20m	1 GPU P100	2 GPU K20m	2 GPU P100
Average GPU Power Usage (W)	ImageNet	ResNet50	TensorFlow	28	77	59	106	79
Average GPU Power Usage (W)	ImageNet	ResNet50	PyTorch	28	58	56	85	62
Average GPU Power Usage (W)	ImageNet	ResNet50	MXNet	28	71	74	107	87
Average GPU Power Usage (W)	ImageNet	ResNet50	CNTK	29	78	119	46	60
Average GPU Power Usage (W)	ImageNet	GoogLeNet	Caffe	28	63	55	88	57
Average GPU Power Usage (W)	ImageNet	GoogLeNet	TensorFlow	29	70	48	89	59
Average GPU Power Usage (W)	ImageNet	GoogLeNet	PyTorch	0	0	0	0	0
Average GPU Power Usage (W)	ImageNet	GoogLeNet	MXNet	28	61	55	89	56
Average GPU Power Usage (W)	ImageNet	GoogLeNet	CNTK	29	58	48	77	47
Average GPU Power Usage (W)	ImageNet	VGG16	Caffe	28	66	65	101	72
Average GPU Power Usage (W)	ImageNet	VGG16	TensorFlow	29	86	84	115	102
Average GPU Power Usage (W)	ImageNet	VGG16	PyTorch	28	78	85	93	91
Average GPU Power Usage (W)	ImageNet	VGG16	MXNet	28	75	87	100	80
Average GPU Power Usage (W)	ImageNet	VGG16	CNTK	29	63	64	91	58