UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
INSTITUTO DE INFORMÁTICA
PROGRAMA DE PÓS-GRADUAÇÃO EM COMPUTAÇÃO

CRISTIANO ALEX KÜNAS

Optimizing Machine Learning Models Training in the Cloud

Thesis presented in partial fulfillment of the requirements for the degree of Master of Computer Science

Advisor: Prof. Dr. Philippe Olivier Alexandre Navaux

Porto Alegre
May 2023

CIP — CATALOGING-IN-PUBLICATION

Künas, Cristiano Alex

Optimizing Machine Learning Models Training in the Cloud / Cristiano Alex Künas. – Porto Alegre: PPGC da UFRGS, 2023.

78 f.: il.

Thesis (Master) – Universidade Federal do Rio Grande do Sul. Programa de Pós-Graduação em Computação, Porto Alegre, BR–RS, 2023. Advisor: Philippe Olivier Alexandre Navaux.

1. Cloud computing. 2. TPU. 3. Machine learning. 4. Performance evaluation. 5. High performance computing. I. Navaux, Philippe Olivier Alexandre. II. Título.

“Sometimes it is the people no one imagines anything of
who do the things that no one can imagine.”

— Alan Turing

AGRADECIMENTOS

Nesses anos de mestrado, de muito estudo, esforço e empenho, gostaria de agradecer a algumas pessoas que me acompanharam nessa jornada e foram fundamentais para sua conclusão. Portanto, expresso aqui, a minha sincera gratidão a todas elas.

Agradeço primeiramente à minha companheira Gabriela, pelo apoio, amor, carrinho e dedicação que foram fundamentais desde o início.

À minha família pelo apoio que deram em toda minha caminhada acadêmica, incentivando e ajudando-me a superar as dificuldades e os obstáculos encontrados.

Minha gratidão especial ao meu orientador Prof. Dr. Philippe Olivier Alexandre Navaux, pelo apoio durante os últimos dois anos. Sou grato pelos ensinamentos e oportunidades que contribuiram para o meu crescimento pessoal e profissional. Obrigado pela confiança no Mestrado.

Ao corpo docente e quadro de funcionários do Instituto de Informática da UFRGS, em especial: Prof. Carissimi e Prof. Lucas. Também gostaria de agradecer à CAPES pelo apoio financeiro.

Agradeço à meus amigos e colegas do Grupo de Processamento Paralelo e Distribuído (GPPD) que me deram suporte, tornando estes anos mais suportáveis e proporcionando boas lembranças. Em especial: Matheus Serpa.

E a todos que de alguma maneira fizeram parte da minha formação, fica aqui minha gratidão.

ABSTRACT

Driven by the development of new technologies such as personal assistants or self-driving cars, machine learning has quickly become one of the most active fields in computer science. From Big Data to Deep Learning applications, new workloads are resource-demanding, driving high-performance computing (HPC) demand. Therefore, it is essential to optimize the execution of these workloads on modern processors. Several approaches have been proposed to accelerate machine learning on GPUs, massively parallel computers, and dedicated ASICs. On the other hand, there is a growth in the processing of computer programs in the cloud. It is a way to reduce the cost of acquiring computers to run programs locally.

In this master's thesis, we adapted and migrated three DL applications to exploit Cloud Computing resources. This approach helps alleviate contention for high-demand local HPC resources, allowing them to focus on running applications. We optimize the execution of these applications using Tensor Processing Units (TPUs). The objective is to evaluate the performance, accuracy, and cost of using such devices for ML/DL training. In our experiments, we showed that the size of the application could influence performance and execution costs. Small applications, which end up not using all the hardware, may have little or no cost efficiency, i.e., the execution cost is higher and is not offset by the performance obtained. In these cases, the choice to run in the cloud must be analyzed with caution, noting if there are additional benefits. The performance achieved is improved in larger applications, which use computational resources more effectively. Also, it is about 50% cost efficient in running the same amount of work compared to the local cluster. The cost per training can be further reduced through the use of preemptive TPUs, costing about 70% less compared to on-demand TPUs.

Keywords: Cloud computing. TPU. Machine learning. Performance evaluation. High performance computing.

Otimizando o Treinamento de Modelos de Aprendizado de Máquina na Nuvem

RESUMO

Impulsionado pelo desenvolvimento de novas tecnologias, como assistentes pessoais ou carros autônomos, o aprendizado de máquina tornou-se rapidamente um dos campos mais ativos da ciência da computação. De Big Data a aplicativos de Aprendizado Profundo, as novas cargas de trabalho são notoriamente exigentes em termos de recursos intensificando a demanda por computação de alto desempenho (HPC). Portanto, é de suma importância otimizar a execução destas cargas de trabalho em processadores modernos. Várias abordagens foram propostas para acelerar o aprendizado de máquina em GPUs e computadores massivamente paralelos, bem como ASICs dedicados. Por outro lado, há um crescimento no processamento de programas de computador na nuvem. É uma forma de reduzir o custo de aquisição de computadores para executar programas localmente.

Nesta dissertação de mestrado, nós adaptamos e migramos três aplicações DL para explorar recursos de Cloud Computing. Essa abordagem ajuda a aliviar a contenção de recursos locais de alta demanda de HPC, permitindo que eles se concentrem na execução de aplicativos. Otimizamos a execução destas aplicações utilizando Tensor Processing Units (TPUs). O objetivo é avaliar o desempenho, a acurácia e custo do uso de tais dispositivos para treinamento de ML/DL. Em nossos experimentos, mostramos que o tamanho da aplicação pode influenciar no desempenho e no custo de execução. Pequenas aplicações, que acabam não utilizando todo o hardware podem ter pouca ou nenhuma eficiência de custo, ou seja, o custo de execução é maior e não é compensado pelo desempenho obtido. Nestes casos, a escolha de executar na nuvem deve ser analisada com cautela, observando se há beneficios adicionais. Em aplicações maiores, que usam de forma mais efetiva o recurso computacional, o desempenho alcançado é melhorado. Além disso, apresenta eficiência de custo de cerca de 50% na execução da mesma quantidade de trabalho comparado ao cluster local. O custo por treinamento pode ser ainda mais reduzido através do uso de TPUs preemptivas, custando cerca de 70% menos comparado com TPUs sob demanda.

Palavras-chave: Computação em nuvem. TPU. Aprendizado de máquina. Avaliação de desempenho. Computação de alto desempenho.

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVIATIONS AND ACRONYMS

ANN	Artificial Neural Network
API	Application Programming Interface
ASIC	Application-Specific Integrated Circuit
AWS	Amazon Web Services
Azure	Microsoft Azure
CPU	Central Processing Unit
CNN	Convolutional Neural Network
DL	Deep Learning
DNN	Deep Neural Network
EC2	Elastic Compute Cloud
FLOPS	Floating-point Operations Per Second
FPGA	Field-Programmable Gate Array
GB	GigaByte
GCP	Google Cloud Platform
GCS	Google Cloud Storage
GPU	Graphic Processing Unit
GRPC	Google Remote Procedure Call
HBM	High Bandwidth Memory
HPC	High Performance Computing
IaaS	Infrastructure as a Service
I/O	Input/Output
LNCC	National Laboratory for Scientific Computing
LSTM	Long Short-Term Memory
ML	Machine Learning

MXU	Matrix Unit
NIST	National Institute of Standards and Technology
NN	Neural Network
PaaS	Platform as a Service
PFS	Parallel File System
ReLU	Rectified Linear Unit
RNN	Recurent Neural Network
SA	Sentiment Analisys
SaaS	Software as Service
SLA	Service Level Agreement
TB	TeraByte
TFlops	TeraFlops per second
TPU	Tensor Processing Unit
TWINS	Time WINdows Scheduler
UFRGS	Federal University of Rio Grande do Sul
vCPU	Virtual Central Processing Unit
VM	Virtual Machine

CONTENTS

1 INTRODUCTION

Scientific applications from various research domains (*e.g.*, health sciences, chemistry, physics, petroleum, climate) impose ever-increasing performance requirements on high-performance computing (HPC) (MALISZEWSKI et al., 2020). In addition, new workloads are entering HPC installations, from Big Data to Machine Learning (ML) applications, making systems increasingly complex. These requirements warrant permanent upgrades and deployment of new large-scale platforms. As the complexity of these systems tends to grow, so does the number of parameters and factors that can directly or indirectly affect performance.

Deep Learning (DL) techniques fuel recent advances in artificial intelligence. This success is due to three main advancements: advances in neural network models and programming frameworks, the availability of massive amounts of data, and hardware accelerators that can process that data faster, such as FPGAs, GPUs, and TPUs (CHO et al., 2017; LIM, 2021; PINTO et al., 2018). DL models have been increasingly used to solve complex problems. However, the complexity imposed on applications generates overhead for the local system, which has its computational resources disputed by scientific applications and by training or updating these models. With the increasing complexity and the amount of data, training these models has required increasingly powerful computing systems with high acquisition and maintenance costs.

An alternative to reduce these costs is using services and computing resources in the Cloud. This business model allows access to a wide range of computer systems, including high-performance systems, only pay for usage, without the user having to bear the cost of purchasing the equipment (ROLOFF, 2013). However, choosing the most suitable computer system to train a DL model in the Cloud is complex. The choice should consider factors such as execution time and cost (MALTA, 2021; MALTA; AVILA; BORIN, 2019; PAKDEL; HERBERT, 2017).

Unlike traditional systems, Cloud Computing does not require initial investments in infrastructure and software licenses. Due to the elasticity and pay-per-use billing model, the total maintenance cost can be close to zero when resources are not in use. Furthermore, installation costs, hardware depreciation, power consumption, and cooling are eliminated or significantly reduced. Also, the amount of resources available is virtually unlimited. Therefore, the motivation for moving applications to the Cloud is to minimize these costs while increasing scalability and availability.

In addition, Cloud Computing service providers frequently update their computational centers with new accelerators, such as GPUs and TPUs, which makes it possible to train ever larger models at lower costs as technology advances. This type of benefit is difficult to obtain with local computational centers since this equipment is costly, and the renewal of the computational center in small companies and universities does not occur with the frequency that new versions of this equipment are released. Thus, our work aims to offload ML/DL algorithms training to the Cloud, one of the tasks that consume the most computational resources and can take days, avoiding overloading the local system, which is often highly overloaded by many scientific applications running[1].

1.1 Contributions of this research

The main objective of our research is to optimize the training of DL algorithms using high performance computing resources on the Cloud. To this end, our main contributions are:

- We adapt and migrate DL applications to exploit Cloud Computing features;
- We developed an application to optimize the data preprocessing to transfer to the cloud;
- We optimize the execution of these applications on the Cloud using Tensor Processing Units (TPUs);
- We study the performance, accuracy, and cost of using TPUs on the Cloud.

1.2 Document Organization

The document is organized as follows. Chapter 2 presents a background on the topics of this dissertation and discusses related work on cloud-based training of large neural networks using TPUs and TPU Pods. Chapter 4 presents the methodology of our work, details the applications, and outlines the hardware and software setup used. In chapter 6, we showcase the results achieved, covering the performance, accuracy, and cost evaluation. Finally, chapter 7 draws conclusions based on our findings and presents some future work insights.

[1] In progress projects on the SDumont Supercomputer: https://sdumont.lncc.br/projects_statistics.php

2 BACKGROUND

The following sections explain some concepts that serve as a base for this dissertation. It starts by reviewing Cloud Computing, including its definition, essential characteristics, service models, and features that can affect application execution. Then, we present the hardware accelerators available, especially TPUs. Later, we describe the main components involved in training a deep model and present strategies for training large models.

2.1 Cloud Computing

Currently, cloud computing is a well-established method of providing computing resources on-demand. It combines various features from distributed, grid, and parallel computing, as well as technologies like virtualization, which dynamically abstract hardware resources (BUYYA et al., 2009).

The National Institute of Standards and Technology (NIST) provides a widely accepted definition of cloud computing that serves as the basis for this document. According to NIST, cloud computing is a model that enables access to a shared pool of computing resources (such as networks, servers, storage, applications, and services) on-demand via the network. These resources can be quickly provisioned and released with minimal effort or interaction from the service provider (MELL; GRANCE, 2011).

This Section describes the definition of Cloud Computing, presents the service models, the features of Cloud Computing, and the hardware accelerators in cloud environments.

2.1.1 Essential Characteristics

For a cloud computing service to be considered to adhere to the NIST definition (MELL; GRANCE, 2011), it needs to exhibit five essential characteristics.

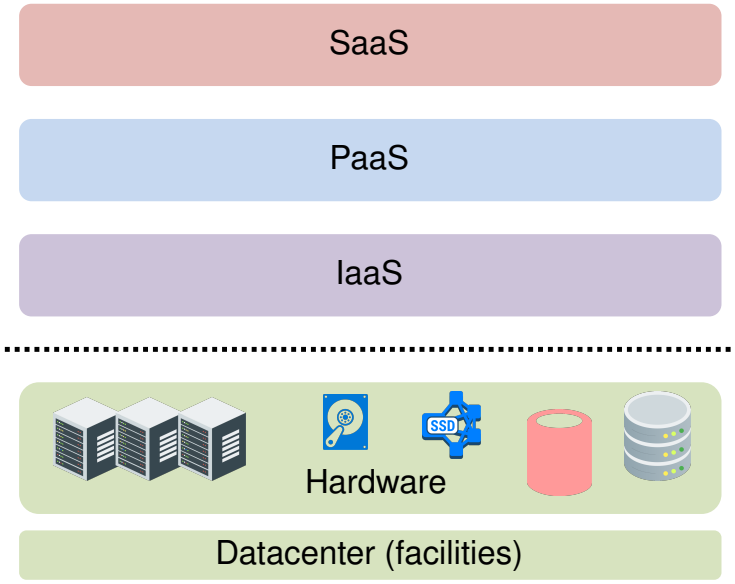
- **On-demand self-service:** This feature allows you to allocate and deallocate computational resources without requiring human interaction with the service provider.
- **Broad network access:** All services offered by the provider must be accessible over a network, and access must be available through standard mechanisms.

- **Resource pooling:** The provider manages a pool of computing resources using a multi-tenant model with different physical and virtual resources and serves multiple users per their demand.
- **Rapid elasticity:** The provider allows the user to elastically provision resources of any amount at any time. To the user, the available resources seem limitless.
- **Measured Service:** The provider is responsible for controlling resource usage and availability. This information must be transparent to the user to guarantee the Service Level Agreement (SLA). They are also used for billing purposes.

2.1.2 Service Models

The National Institute of Standards and Technology (NIST) defines three service models in their proposal: Software as a Service (SaaS), Platform as a Service (PaaS), and Infrastructure as a Service (IaaS). These models determine what the user will receive and how much control they will have over the features and services. Figure 2.1 shows the abstraction levels for each service model. The hardware layer is placed at the bottom to emphasize that the higher the service is from this layer, the more abstract it becomes and the less control the user will have. The following is a brief explanation of each service model.

Figure 2.1 – Cloud Computing service models.



Source: The Author.

- **Software as a Service (SaaS):** This is the most common model. In this model,

users have access to ready-to-use applications. The customer only needs to register to use the system and doesn't have to worry about installation, maintenance, and upgrades. However, the control over the service is limited, with only certain aspects of configuration and administration available to the user. The provider is responsible for all technical aspects, including servers, security, and data center issues.

- **Platform as a Service (PaaS):** This model provides a platform for end-users to build their applications. It includes all the necessary infrastructure, servers, tools, libraries, and databases, so companies can focus solely on their systems. The user is responsible for their data and applications. With PaaS, there's no need to buy, set up, or maintain hardware and software. The developer doesn't see the underlying infrastructure but can configure the apps and their environment. This supports agile software development and is often adopted by startups that lack resources. Having a cloud-based development platform is important for remote work and is a key strategy for new or pandemic-affected organizations.
- **Infrastructure as service (IaaS):** It is the model that provides a higher level of control to users by offering virtual machines over which they have administrative control. The user has complete authority over their virtual machine and can decide the processing capacity (*e.g.*, number of processing cores) and the software to be used, including the operating system. This model allows users to customize their virtual machine environment to meet their specific needs. IaaS abstracts physical components such as servers, storage, space, and networks, eliminating the need for companies to buy hardware.

2.1.3 Features of Cloud Computing

Cloud computing service providers such as Amazon Web Services (AWS), Microsoft Azure (Azure), and Google Cloud Platform (GCP) offer many types of computing services. In the most basic model, infrastructure as a service (IaaS), users can hire virtual machines (VMs), storage devices, network connectors, etc., and pay only for their use. Virtual machines are usually charged by the time they remain powered on (*e.g.*, US$/h), while storage devices are charged by the space occupied and the time of use (*e.g.*, US$/GB/h). Although the IaaS model is similar to traditional high-performance clusters, the Cloud has several characteristics that differentiate it from these systems.

Variety of computational resources: Cloud computing service providers offer diverse computing resources, including several virtual machine options, with different combinations of CPU and memory and storage and network devices. For example, AWS offers hundreds of virtual machine[1], with prices ranging from 0.0042 US$/h (with 2 vCPUs and 0.5 GB of memory) to 109.20 US$/h (with 448 vCPUs and 12TB of memory). Additionally, many virtual machine configurations have specialized hardware such as GPUs, TPUs, and FPGAs that are useful for accelerating the execution of many workloads requiring high computational performance. AWS, for example, offers a category of virtual machines with 8 NVIDIA Ampere A100 GPUs with prices of up to 11.57 US$/h. The types of computing resources hired can affect both the cost and the total processing time of a high-performance application in the Cloud. In this context, the computational resources with the highest price (in US$/h) have better configurations and tend to offer the best performance. On the other hand, as the total cost of processing depends on price and processing time, the resource with the lowest price does not always offer the lowest processing cost. The variety of resources offered in Cloud Computing allows users to adjust both the hardware of the computational system and the software of the application to optimize the cost or performance of processing. On the other hand, to maximize the performance of an application in a high-performance cluster, the user is generally limited to adjusting the application's software to make good use of the resources available in the computational system.

Performance volatility: Cloud computing service providers use virtualization technologies to flexibly allocate computing resources to customers. With this, it is possible to divide the computational resources of a single physical server among multiple virtual machines so that different clients hire fractions of the system for exclusive use. Although the virtualization layer provides a reasonable level of resource isolation, some parts of the hardware, such as the processor's cache and the communication channel with main memory, cannot be divided and are shared by virtual machines running on the same physical machine. In this way, the intense demand for these parts in a virtual machine (*e.g.*, high memory access) can affect the performance of applications in other virtual machines allocate in the same physical machine. As the user has no control over the allocation of other virtual machines on the same physical machine, the performance of virtual machines may vary throughout their use. Some Cloud Computing service providers allow the user to hire the physical machine for exclusive use, which would avoid performance

[1] https://aws.amazon.com/ec2/instance-types/

volatility. On the other hand, the price of this type of service can be much higher than hiring VMs that share the physical machine.

Preemptible Virtual Machines: Cloud Computing service providers typically offer different models for contracting VMs with different prices and guarantees regarding availability and volatility. Most providers offer three main models: (i) the on-demand model, where the user hires VMs as needed, *i.e.*, on demand. In this case, the user is guaranteed VM availability, but the price is higher; (ii) the reserved model, where the user commits to using a minimum amount of resources in a given period (*e.g.*, one year). In exchange, the provider offers a reduced price and guaranteed resource availability; and (iii) the spot model (or preemptive), where the user can contract VMs on demand at a reduced price. However, the provider reserves the right to resume these VMs anytime, *i.e.*, no availability guarantees exist. The availability of preemptible VMs fluctuates based on users' demand. If there are not enough instances to meet customers' demands, the VM can be stopped by the cloud provider (temporarily or permanently). In this case, the provider hibernates the VM, saving its memory and context. If the demand decreases, the instance can be resumed from the hibernation point. Despite the risk of unavailability, the main advantage of preemptible VMs is that their prices are much lower than on-demand VMs.

2.2 Hardware Accelerators in Cloud Environments

Cloud providers offer several accelerators such as FPGAs, GPUs and TPUs. This section aims to introduce the instance types and discuss the main characteristics of each computer architecture.

The cloud providers offer a wide selection of optimized instance types to cater to different use cases. The instance types consist of various combinations of CPU, memory, storage, and network capabilities and offer the flexibility of choosing the suitable composition of resources for their applications. Each type of instance includes one or more instance sizes, allowing the scalability of its resources according to the workload's requirements to be executed. Accelerated computing instances use hardware accelerators, or coprocessors, to execute functions, such as floating-point number calculations, graphic processing, or data standard matching, more efficiently than is possible with software executing on CPUs. They have several instances for accelerated computing, ranging from GPUs to FPGAs, besides cloud providers' accelerators own, such as AWS Inferentia and

Google Cloud TPU.

2.2.1 FPGAs - Field-Programmable Gate Arrays

We introduce the field-programmable gate arrays (FPGAs) available on the cloud providers, mainly used for machine learning inference.

Amazon EC2 F1 instances offer customizable hardware acceleration with programmable field port arrays (FPGAs). They use FPGAs to enable the delivery of custom hardware accelerations. F1 instances are easy to program and come with everything needed to develop, simulate, debug, and compile hardware-accelerated code.Using F1 instances to deploy hardware accelerations can be helpful in a variety of applications to solve complex problems in science, engineering, and business that require high bandwidth, enhanced networks, and very high computing power. Examples of target applications that can benefit from F1 instance acceleration are genomics, search/analysis, image and video processing, network security, electronic design automation (EDA), image and file compression, and big data analysis. The instance has Intel Xeon Scalable processors and Xilinx Virtex UltraScale+ VU9P FPGAs with up to 64 GB of memory.

Amazon EC2 VT1 instances are designed to accelerate real-time video transcoding and deliver low-cost transcoding for live video streams. They can deliver up to 30% lower cost per stream compared to Amazon EC2 G4dn GPU-based instances and up to 60% lower cost per stream compared to Amazon EC2 C5 CPU-based instances for transcoding live video streams. VT1 instances can support streams up to 4K UHD resolution at 60 frames per second (FPS) and can transcode up to 64 simultaneous 1080p60 streams in real-time. They are powered by up to 8 Xilinx® Alveo™ U30 media accelerator cards and support up to 96 vCPUs, 192GB of memory, 25 Gbps of enhanced networking, and 19 Gbps of EBS bandwidth. They are optimized for workloads such as live broadcast, video conferencing, and just-in-time transcoding. It has up to 8 Xilinx U30 media accelerator cards with accelerated H.264/AVC and H.265/HEVC decoders.

2.2.2 GPUs - Graphic Processing Units

Usually, computational clouds like Amazon AWS have instances with the most modern GPUs and some low-cost ones with older GPUs. As of June 2022, AWS has

8 GPU instances. Amazon EC2 P4 instances are the latest generation of GPU-based instances and provide the highest performance for machine learning training and high-performance cloud computing. The instance has 8 NVIDIA A100 Tensor Core GPUs, each with 6,912 CUDA Cores and 432 Tensor cores.

These instances provide the highest performance for training machine learning (ML) and high-performance computing (HPC) applications in the cloud. P4d instances are powered by the latest NVIDIA A100 Tensor Core GPUs and deliver industry-leading high throughput and low latency networking. These instances are the first in the cloud to support 400 Gbps instance networks. P4d instances offer up to 60% lower cost to train ML models, with an average 2.5$\times$ better performance for deep learning models than previous generation P3 and P3dn instances.

Researchers, data scientists, and developers can use the P4 instances to train machine learning (ML) models for different use cases such as Natural Language Processing (NLP), object detection and classification, and recommendation systems, as well as run High-Performance Computing (HPC) applications such as oil and gas simulation, financial modeling and pharmaceutical discovery. Unlike on-premises systems, customers can access "unlimited" compute and storage resources, scaling their infrastructure based on their business needs without any setup or maintenance cost.

Like P4 instances, AWS EC2 P3 instances provide up to 8 NVIDIA V100 Tensor Core GPUs and up to 100 Gbps of network throughput for ML and HPC applications. The main difference between the P4 and P3 instances is that P4 has NVIDIA A100 GPUs while P3 has NVIDIA V100. Another similar instance is the AWS EC2 P2, which is intended for general-purpose GPU computing applications. The GPU of this version is the NVIDIA K80.

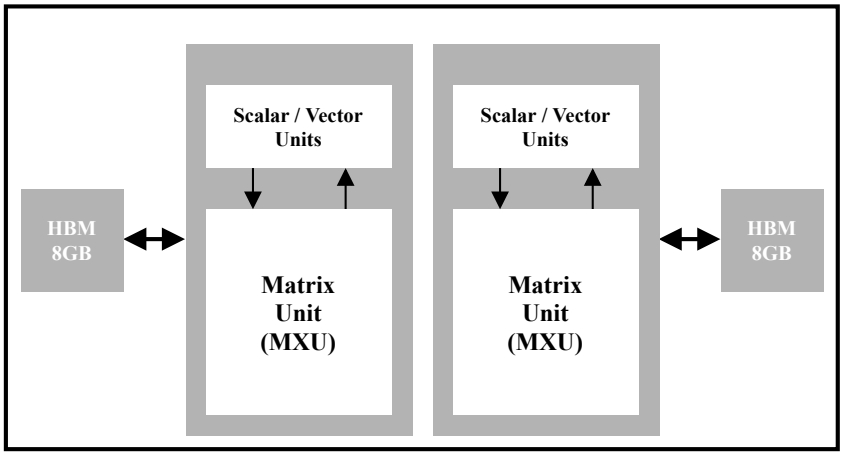
2.2.3 TPUs - Tensor Processing Units

This section discusses the Tensor Processing Unit (TPU), an AI accelerator developed by Google and available on the Google Cloud Platform (GCP). It is deeply integrated with the TensorFlow software.

TPUs are specialized application-specific integrated circuits (ASICs) to support large-scale machine-learning tasks. The performance of linear algebraic computing, intensely used in machine learning, is accelerated with the TPU resources, minimizing the time it takes to generate precision when training large and complex neural network mod-

els. Figure 2.2 shows the architecture of a TPUv2 chip. Each TPUv2 device has four internal chips, and each is made up of two cores. Each core has scalar, vector, and matrix units (MXU) connected with the on-chip high bandwidth memory (HBM) of the 8 GB for each TPUv2 core. The performance of each TPU chip is 180 TFlops (32-bit and 16-bit mixed precision) (YOU et al., 2019). Compared to TPUv2, TPUv3 doubles the number of MXUs and HBM capacity per core and has a peak of 420 TFlops, 2.3× greater than TPUv2 (WANG; WEI; BROOKS, 2019). Furthermore, each core of the TPU device performs calculations independently, and the high bandwidth interconnections allow the chips to communicate directly with each other in the TPU device (GOOGLE, 2022b).

Figure 2.2 – The architecture of the TPUv2 chip with 16GB of HBM. Each TPU v2 device has four internal chips comprising two cores and can achieve up to 180 TFlops.



Source: The Author.

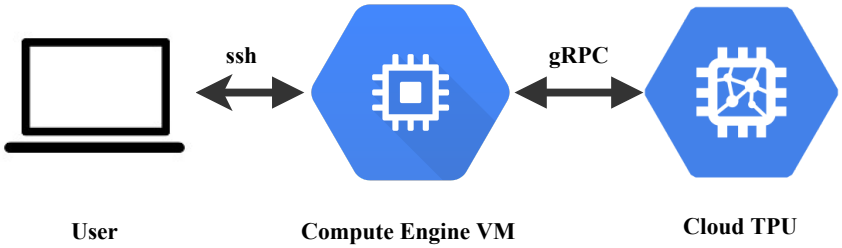
Additionally, you can run machine learning workloads on a TPU pod that increases workloads with little or no code changes. A TPU pod is a set of TPU devices connected by dedicated high-speed network interfaces. A TPU pod allows you to distribute the processing load among multiple TPUs, which can have up to 512 cores and provide 11.5 petaFLOPS performance for TPUv2 and up to 2048 cores with 100+ petaFLOPS performance for TPUv3. A high-performance CPU-based host machine is connected to each TPU board for data loading and preprocessing (YING et al., 2018).

Figure 2.3 illustrates how a user can access a Cloud TPU. Cloud TPUs are network-attached. The user must create a virtual computing engine machine and apply a cloud TPU. The virtual machine connects to the cloud TPU through *grpc*[2], Google's open-source high-performance Remote Procedure Call that can run in any environment. Users do not need to install any driver and can use the machine images provided by Google

[2]https://grpc.io/

Cloud. However, the users still need to design the algorithm and write the code for their applications (YOU et al., 2019).

Figure 2.3 – Illustration on how to offload to a cloud TPU.



Source: The Author.

When working with the Cloud TPU model, it is important to configure it correctly in order to take advantage of the distributed training capabilities of the device. One strategy for doing this is to scale the batch size by the number of TPU cores that are available. For example, if the batch size is 32, the global batch size will be 256 (8 cores x 32 = 256) (KÜNAS et al., 2021). This means that each core will process a batch of 32 examples, and the results will be combined across all cores to form the final output. The global batch size is then automatically fragmented across all replicas, which allows for the efficient processing of large data sets. This approach allows for the parallel processing of multiple examples simultaneously, greatly speeding up the training process.

2.3 Main deep learning concepts

An ML algorithm is capable of learning how to perform a task from a dataset (GOODFELLOW; BENGIO; COURVILLE, 2016). Typically, a machine learning system consists of four key components: a dataset, a model that generalizes a function f based on the dataset, a cost function that measures the deviation of the model from a correct solution, and an optimization algorithm that aims to optimize (minimize or maximize) the value of the cost function.

Machine learning algorithms are generally categorized into three types: supervised learning, unsupervised learning, and reinforcement learning algorithms. In supervised learning algorithms, a set of labels Y is included in addition to the dataset X, which describes each element of X. The objective is to optimize the parameters of a parametric function f, also known as the model, so that for each sample in X, f returns a value that closely matches its respective label (BEN-NUN; HOEFLER, 2019). This process is

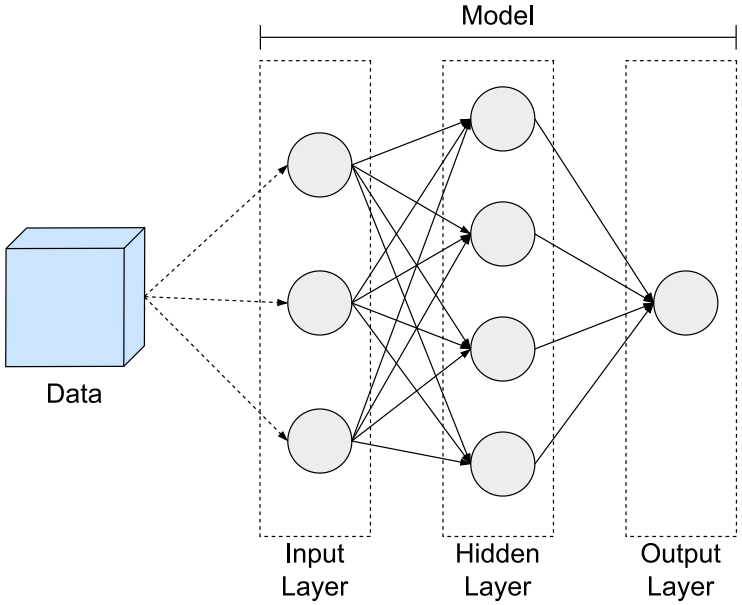
referred to as training (MAYER; JACOBSEN, 2020), and the aim is to minimize the training error (also known as loss), which is the error observed when generating a prediction based on an input. In unsupervised learning, the dataset is not labeled, and the model is designed to uncover insights about the data by identifying patterns and hidden structures within it. Finally, in reinforcement learning, observations are made at specific time points within an environment, and the training process involves optimizing a policy function that maximizes the observer's reward (BEN-NUN; HOEFLER, 2019).

Artificial neural networks (ANNs) are one of the most powerful machine learning algorithms among the various types available. These algorithms are inspired by biological nervous systems (MCCULLOCH; PITTS, 1943) and consist of artificial neurons or perceptrons arranged in layers, as illustrated in Figure 2.4. The layers are formed by multiple sequentially connected neurons, and the connections between neurons are called synapses, which have associated weights that are part of the model's parameters. In the input layer, various parts of the input data are mapped to the input layer's perceptrons. The subsequent layers receive the previous layer's output, which is calculated using a mathematical function (activation function) of the previous activation functions and the weights associated with the corresponding synapses. Ultimately, the output layer contains the prediction produced by the model. The training process consists of finding values for the weights (*i.e.*, the parameters) that optimize the accuracy of the result produced by the model (*i.e.*, the model accuracy).

Artificial neural networks with many hidden layers are called Deep Neural Networks (DNNs). Chaining in multiple layers allows DNNs to learn increasingly complex functions and increase input data abstraction levels at each layer (WANG et al., 2015). Different architectures were proposed as these deep neural networks evolved. Convolutional Neural Networks, for example, introduced new layers, such as convolutional and pooling layers, and were inspired by the visual cortex of animal brains (GOODFELLOW; BENGIO; COURVILLE, 2016). This category of models is called Deep Learning (DL).

Deep learning techniques have grown rapidly following their success in image classification. Currently, DL models have been used for various purposes, such as financial forecasting, and driving assistance, among others (KIM; CHUN; DEY, 2015; HATCHER; YU, 2018; ABIODUN et al., 2018). As these models got more complex with more parameters, trained with more data, and high-resolution data, training them started demanding more computational resources and the development of strategies to improve training efficiency.

Figure 2.4 – An example of an artificial neural network, where circles denote perceptrons, and arrows denote connections between perceptrons (synapses). Each arrow (excluding dashed ones) has an associated weight. The output of each perceptron is calculated using an activation function.



Source: The Author.

2.3.1 Training deep learning models

Training a deep learning model involves an iterative forward, and backward propagation process performed multiple times over the entire dataset. The dataset is typically partitioned into three subsets: the training subset, used to tune the model parameters; the validation dataset used to verify the model's performance on new data during training; and the test dataset, used to evaluate the model at the end of training.

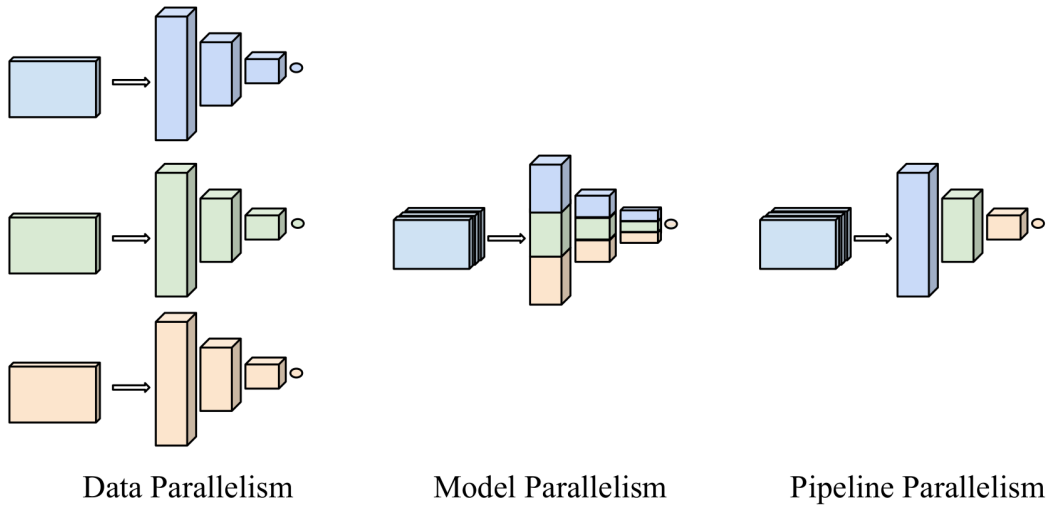
Each iteration over the training dataset is called an epoch, and each consists of multiple steps, including computing gradients and updating the model parameters using a batch of data. The training process continues until a stop criterion is met, including reaching a target accuracy or loss or training for a fixed number of epochs.

Hyperparameters, such as batch size, also affect the training process and the quality of the resulting model. Choosing the correct hyperparameters is critical for optimizing the model's quality. The process of tuning hyperparameters is usually done during the model's development. The choice of hyperparameters must consider the computing infrastructure's specifications. For example, the batch size can affect the model's accuracy and training time, with its maximum value being limited by the processing device's volatile memory.

2.3.1.1 Model partitioning strategies for distributed training

The time required to train a deep learning model can be significant, depending on various factors such as the model size (number of parameters), operations performed, the number of samples in the dataset, the batch size, the data type, and other hyperparameters. However, a practical approach to reducing training time while meeting the growing demand for computational power is to distribute the training process across multiple computing devices, such as GPUs or TPUs (BEN-NUN; HOEFLER, 2019). This can be achieved by partitioning different components of the model, such as the input data (using data parallelism), the model itself (using model parallelism), or the layers of the model (using pipeline parallelism), as presented in Figure 2.5.

Figure 2.5 – Strategies for model partitioning. Different colors represent different computing devices.



Source: The Author.

Data parallelism: Data parallelism involves partitioning the dataset across multiple computing devices, each possessing a copy of the model parameters and performing independent propagation. AllReduce synchronizes gradients, but overhead may limit scalability. Asynchronous training is an alternative to reduce synchronization overhead, but periodic synchronization is required. The batch size may need to be reduced as the number of devices increases (GUPTA; ZHANG; WANG, 2016).

Model parallelism: Model parallelism distributes neurons of each layer across devices for training using the same data batch. This allows the training of large models not fitting device memory but incurs additional communication after each layer finishes processing. Forward propagation and backward propagation are synchronous operations

and may cause device underutilization issues.

Pipeline parallelism: Pipeline parallelism refers to overlapping computations, such as between layers (as data becomes ready) or by partitioning layers and assigning each to a specific worker (BEN-NUN; HOEFLER, 2019). Layer parallelism allows training large models without workers storing all layer parameters. Communication occurs at layer boundaries, avoiding all-to-all operations. To fully use the computing system, the pipeline must be partitioned among devices to all process the same amount of samples per second.

2.3.2 Training deep learning models in the cloud

A typical machine-learning workflow can be divided into the following stages:

1. *Data preparation:* In this stage, the relevant data to train the model is extracted and pre-processed. Feature extraction or selection is often performed to enhance model accuracy;
2. *Training:* The model is trained using the prepared data and techniques described in previous sections. The output of the training process is usually stored in a model registry, which offers a range of APIs to manage the lifecycle of trained models.
3. *Serving:* The trained model is retrieved from the model registry and served through a prediction service that can be used for inference.

Cloud computing provides multiple options for executing the stages of the machine learning workflow. Based on their requirements and technical capabilities, users can choose between these three types of services: Infrastructure as a Service (IaaS), Platform as a Service (PaaS), and Software as a Service (SaaS).

Several cloud operators have added machine learning as a service to their cloud offerings. Most providers offer multiple options for running machine learning tasks on their clouds, ranging from IaaS-level services (VM instances with prepackaged ML software) to SaaS-level solutions (Machine Learning as a Service). Much of the technology offered is standard distributed machine learning systems and libraries. Among other things, Google's Cloud Machine Learning Engine supports TensorFlow and even provides TPU instances (GOOGLE, 2022a). As a competitor to Google's TPUs, Microsoft Azure Machine Learning supports ML application acceleration through FPGAs (OVTCHAROV et al., 2015). Amazon AWS introduced SageMaker, a hosted service for building and train-

ing machine learning models in the cloud. The service includes Jupyter notebooks and support for TensorFlow, MXNet, and Spark (SERVICES, 2022).

2.4 Concluding Remarks

This chapter has introduced various concepts related to cloud computing based on the NIST definition. The pay-per-use is the billing model used in cloud computing, where the user pays just for the real usage of the resources. This characteristic is interesting for DL users because this group of users sometimes does not have a permanent resource demand since they can work on temporary projects.

3 RELATED WORK

This section discusses the most representative works that evaluate the application's performance. They cover existing work on cloud-based training of large neural networks using primarily TPUs and TPU Pods.

You et al. (2019) investigate supercomputers' capability of speeding up DNN training. The approach is to use a large batch size powered by the Layerwise Adaptive Rate Scaling (LARS) algorithm for efficient usage of massive computing resources. They empirically evaluate the effectiveness on five neural networks: AlexNet, AlexNet-BN, GNMT, ResNet-50, and ResNet-50-v2 trained with large datasets while preserving the state-of-the-art test accuracy. Using 2,048 Intel Xeon Phi 7250 Processors, they reduced the 90-epoch ResNet-50 training time from hours to 20 minutes. They implemented an approach on Google's cloud Tensor Processing Unit (TPU) platform, which verifies your previous success on CPUs and GPUs (YOU et al., 2018). They scaled the batch size of ResNet-50-v2 to 32K and achieved 76.3 percent accuracy. They applied the approach to Google's Neural Machine Translation (GNMT) application, which helps to achieve a 4x speedup on the cloud TPUs.

Wongpanich et al. (2021) explore techniques to scale up the training of EfficientNets on TPU-v3 Pods with 2048 cores, motivated by speedups that can be achieved when training at such scales. Currently, EfficientNets can take on the order of days to train. EfficientNets are a family of state-of-the-art image classification models based on efficiently scaled convolutional neural networks. They discuss optimizations required to scale training to a batch size of 65536 on 1024 TPU-v3 cores, such as selecting large batch optimizers and learning rate schedules and utilizing distributed evaluation and batch normalization techniques. Additionally, they presented timing and performance benchmarks for EfficientNet models trained on the ImageNet dataset to analyze the behavior of EfficientNets at scale. With the optimizations, they could train EfficientNet on ImageNet to an accuracy of 83% in 1 hour and 4 minutes.

Deep learning is computationally intensive, and hardware vendors have responded by building faster accelerators in large clusters. Training deep learning models requires overcoming both algorithmic and systems software challenges. Ying et al. (2018), discuss three systems-related optimizations: (1) distributed batch normalization to control per-replica batch sizes, (2) input pipeline optimizations to sustain model throughput, and (3) 2-D torus all-reduce to speed up gradient summation. They combined these optimizations

to train ResNet-50 on ImageNet to 76.3% accuracy in 2.2 minutes on a 1024-chip TPU v3 Pod with a training throughput of over 1.05 million images/second and no accuracy drop.

The paper of Jouppi et al. (2017) evaluates a Tensor Processing Unit (TPU) in a data center environment. They compared the TPU to a server-class Intel Haswell CPU and an Nvidia K80 GPU and found that the TPU was significantly faster and more power-efficient. The workload, written in the high-level TensorFlow framework, uses production neural networks (NN) applications (MLPs, CNNs, and LSTMs) that represent 95% of NN inference demand. Despite low utilization for some applications, the TPU is, on average, about 15X to 30X faster than the GPU or CPU. The authors also examined the impact of different hardware and software configurations on the TPU's performance and found that specific optimizations could further improve its speed and efficiency.

There is an industry-wide trend toward hardware specialization to improve performance, principally, compute-intensive deep learning models. To systematically benchmark deep learning platforms, Wang, Wei and Brooks (2019) introduce ParaDnn, a benchmark suite for deep learning that generates models for fully connected (FC), convolutional (CNN), and recurrent (RNN) neural networks. Along with six real-world models, they benchmarked Google's Cloud TPU v2/v3, NVIDIA's V100 GPU, and an Intel Skylake CPU platform. They deeply dive into TPU architecture, reveal its bottlenecks, and highlight valuable lessons learned for future specialized system design. They also provide a thorough comparison of the platforms and find that each has unique strengths for some types of models.

You et al. (2019) studied a principled layerwise adaptation strategy to accelerate the training of deep neural networks using large mini-batches. Using this strategy, they developed a new layerwise adaptive large batch optimization technique called LAMB. The empirical results demonstrate the superior performance of LAMB across various tasks, such as BERT and ResNet-50 training, with very little hyperparameter tuning. In particular, for BERT training, their optimizer enables the use of huge batch sizes of 32,868 without any degradation of performance. By increasing the batch size to the memory limit of a TPUv3 Pod, BERT training time can be reduced from 3 days to just 76 minutes.

A performance investigation of graphics processing units (GPUs) and tensor processing units (TPUs) on Google Colaboratory for training convolutional neural networks (CNNs) was approached by Sharma, Gupta and Gupta (2021). The authors conducted experiments using popular CNN architectures, including AlexNet, VGG16, and Incep-

tionV3, and measured training time and accuracy on both GPUs and TPUs. They also compare the results to those obtained on a local machine with a high-end GPU. The authors find that TPUs outperform GPUs regarding training speed and achieve similar accuracies. They also note that Google Colaboratory provides a cost-effective way to train deep learning models using powerful hardware resources.

Other works evaluate the performance of Google Colaboratory, a cloud-based platform for running Jupyter notebooks with free access to GPUs, as a tool for accelerating deep learning applications. For example, Kimm, Paik and Kimm (2021) compares the performance of TPUs, GPUs, and CPUs by conducting experiments with the ResNet-50 and Inception-V3 models and measuring training time and accuracy. They also investigate the impact of different batch sizes and the number of workers on training times and scalability. The authors find that TPUs outperform GPUs and CPUs in terms of both training speed and scalability, with up to a 24x speedup over GPUs and up to a 228x speedup over CPUs. They also note that, for TPUs and GPUs, the performance is highly dependent on the selected batch size and the number of workers, while the performance of CPUs is little affected by these factors. On the other hand, although Carneiro et al. (2018) do not use the TPU in their study, they compare their results to those obtained on a local machine with a high-end GPU, investigating the impact of different hardware configurations and batch sizes on training times. For authors, the Google Colaboratory provides a way to accelerate deep learning applications with GPU performance comparable to that of a local machine with a high-end GPU.

3.1 Concluding Remarks

In Table 3.1, we present a comprehensive description of the coverage of the state-of-the-art in comparison with this work. Several studies have performed evaluations with TPUs on cloud computing. Some of the related works address training large deep-learning models using TPU Pods. Differently from these related works, we used a single TPU with eight cores, and although we didn't explore optimization techniques, we achieved exciting results. In addition, as with most related works, we also evaluate the accuracy achieved by our models. On the other hand, none of them performs a cost-efficiency analysis when using TPUs compared to a local cluster. This aspect is analyzed in this work.

Table 3.1 – Summary of related work. Each line represents a related work. Each column represents a desired property.

	Performance	Accuracy	Cost-Efficiency	Use TPU	Use Pods TPU
You et al. (2019)	✓	✓		✓	
Wongpanich et al. (2021)	✓	✓		✓	✓
Ying et al. (2018)	✓	✓		✓	✓
Jouppi et al. (2017)	✓			✓	
Wang, Wei and Brooks (2019)	✓			✓	
You et al. (2019)	✓	✓		✓	✓
Sharma, Gupta and Gupta (2021)	✓	✓		✓	
Kimm, Paik and Kimm (2021)	✓	✓		✓	
Carneiro et al. (2018)	✓				
This Thesis	✓	✓	✓	✓	

Source: The Author.

4 METHODOLOGY

After the great success in solving image classification problems, the application of machine learning techniques gained speed and spread to other areas. Training more extensive and complex models make hardware accelerators essential tools. With high acquisition costs for high-performance platforms, an alternative is using cloud computing resources. In this paradigm, the user pays only for the use, eliminating equipment acquisition costs. In this context, our work aims to offload ML/DL algorithms training to the Cloud to avoid overloading the local system.

For this, we adapted and migrated three DL models to the Cloud. We studied each model carefully and identified the sections of code that needed to be modified to run on the TPU. The intention is to adapt the model as much as possible but keep its original structure, *i.e.*, without adding or removing layers or modifying hyperparameters. Hence, changes must be timely and specific so the algorithm can connect to the device and allow parallel training on all available TPU cores.

We added code to detect and connect to the TPU device. For the model structure, it is important to use a deep learning library that provides direct support for TPU training, such as TensorFlow. The model needs to be built using TPU-specific APIs, such as those available in TensorFlow's *tf.distribute.TPUStrategy*. This strategy allows leveraging the parallel processing power of TPUs. Additionally, it is recommended to use the *tf.data* API to create an input pipeline optimized for TPUs. This enables asynchronous and parallel execution of data reading and preprocessing alongside model training, maximizing the TPU's speed. When training the model on a TPU, it is necessary to adjust the training function to be executed in a distributed manner across multiple processing units. This involves splitting the data batch among the TPU replicas, performing parallel training, and synchronizing gradient updates between replicas using the reduction operations (all-reduce) available in the TPU APIs.

The following sections detail each of the three applications with its architecture. We also present the used hardware and software configuration. For each application, we collect performance metrics and compare them to the baseline. Performance is measured by total training time in seconds. Accuracy metrics were also analyzed to validate the application migration, *i.e.*, if the model presents accuracy rates similar to those found in the baseline or even in the literature, the model migration was successful.

We run and collect metrics for four architectures: GPUs P100 and V100, TPUv2,

and TPUv3. However, we do not experiment with all the architectures mentioned in some cases due to difficulties with availability and access to such architectures. The results presented are an average of at least 10 runs, with a relative error of less than 5% and a 95% confidence level using the t-Student distribution.

To analyze the cost-efficiency of using the Cloud TPU for model training, we scale the performance value with the price per hour. Table 4.1 shows the cost per hour for a local cluster and the Google Cloud TPUs[1]. The value for the cluster was calculated as follows. We consider the hardware cost for a machine of $\$25,000$ and that the machine will be used for one year, so we arrive at a hardware cost per hour of $\$2.85$.

$$\frac{\$25,000}{1 \times 365 \times 24} = \$2.85$$

This is a simplified cost calculation. We do not measure the price for facilities, personnel, and power consumption and do not include them in the total price.

Table 4.1 – Cost (in Dollar/hour) of each solution.

Device	**Cost**
TPUv3-8	$8.00
TPUv2-8	$6.00
Local cluster	$2.85

Source: The Author.

4.1 I/O Access Pattern Detector

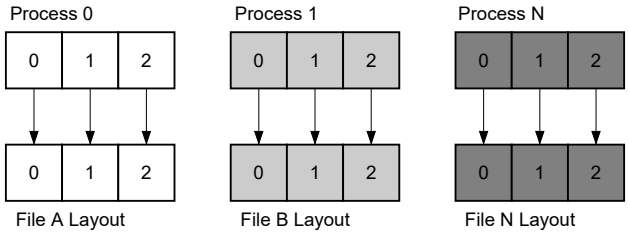
A Neural Network (NN) model was developed on Bez et al. (2019) to automatically detect the I/O access pattern of HPC applications at runtime and was integrated on the TWINS (BEZ et al., 2017) scheduler. The dataset used to train this model contains metrics collected at runtime on access patterns frequently used by the HPC I/O community (they cover $\approx 39{,}000$ observations)[2]. The patterns are classified according to the file approach (single file vs. file-per-process) and the spatiality (contiguous or 1D-strided) into three classes. We do not consider the 1D-strided approach for the file-per-process as it is not representative.

[1] https://cloud.google.com/tpu/pricing

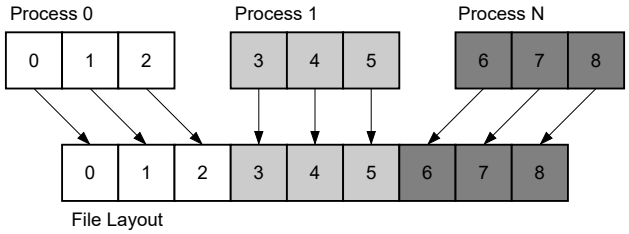
[2] https://cristianokunas.gitlab.io/wcc-sbac-pad-2021/

The first class is the **file-per-process** with **contiguous** access (Fig. 4.2(a)), where each process of an application issues its operations in its own file, accessing one offset after the other. The second class is the **shared file** with **contiguous** access (Fig. 4.2(b)), where all the processes read and write data to a common file by accessing contiguous chunks of the file. Finally, the third class is the **shared file** with **1D-strided** access (Fig. 4.2(c)), where each process accesses portions of the file with a fixed-size gap between them.

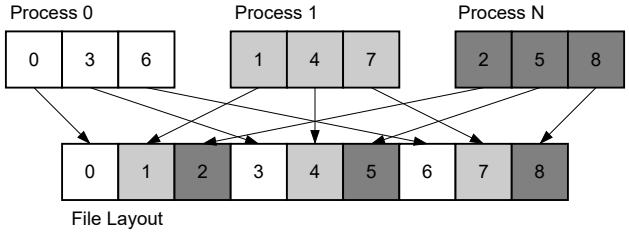
Figure 4.1 – Visual representation of three different I/O access patterns commonly observed in scientific HPC applications.



(a) File-per-process



(b) Shared file with contiguous accesses



(c) Shared file with with 1D-strided accesses

Source: The Author.

This model was initially proposed to work at the I/O forwarding layer of HPC systems (though not restricted to it). Its access pattern detection engine receives input information collected at each I/O node, which stands between the compute nodes and the PFS. This information, referring to the previous observation period, consists of the request size (minimum, average, maximum), the number of file identifiers, and the average distance between consecutive requests for the same file identifier (to represent the spatial-

ity of accesses). After applying the Spearman non-parametric correlation (SPEARMAN, 1904) to identify those most related to the access pattern class, the authors selected these parameters.

This application will be referenced throughout the text as NN-TWINS. For the evaluation, we considered the following metrics: Train Time, Train Accuracy, Test Time, and Test Accuracy.

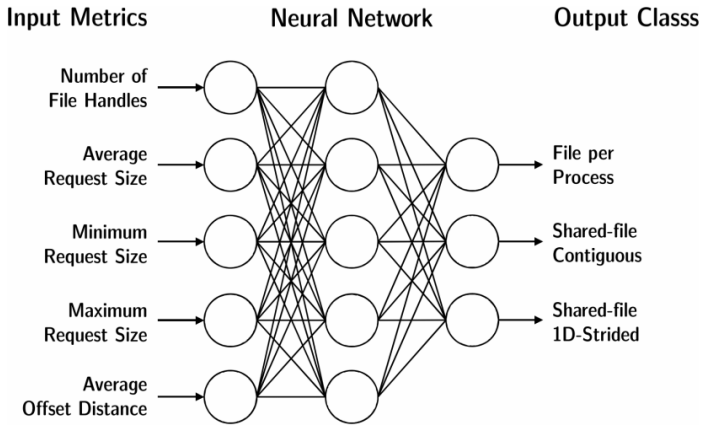
4.1.1 NN-TWINS Architecture

The classifier implementation uses Keras (MOOLAYIL, 2019), a high-level neural network API with TensorFlow (ABADI et al., 2016a) as the back-end. Before feeding the metrics to the Neural Network, we applied Yeo-Johnson (YEO; JOHNSON, 2000), scaling, and data transformations to make the data more suitable for the network and to speed up training. Yeo-Johnson is a power-transform similar to Box-Cox (BOX; COX, 1964) but supports zero or negative value features. In this particular case, some metrics can have zero values. The scale transformation calculates the standard deviation for a characteristic and divides each value by that deviation. Finally, the data transform computes and subtracts the mean from each value.

The model is built with three layers, as illustrated in Fig. 4.2: an input layer with all five features (metrics), a hidden layer with the same number of neurons, and an output layer with three units, one to represent each access pattern class. The first two layers use a Rectified Linear Unit (ReLU) (HAHNLOSER et al., 2000) activation function with a normal kernel initialization function. The ReLU activation introduces non-linearity into the network. The output layer uses a *softmax* activation function to crunch the outputs of each unit in the range $[0, 1]$ and ensure that the sum of the outputs equals 1.

The model uses the optimizer RMSProp, with a learning rate of 0.001 and momentum of 0.9. The loss function was *categorical cross-entropy*. The result has an n-dimensional vector that is all-zeros except for a one at the index corresponding to the sample class. In our case, we have a three-dimensional vector for each sample. We have followed the same approach used in previous work, where we split the dataset in two using Scikit-Learn (PEDREGOSA et al., 2011): 70% for training and 30% for testing.

Figure 4.2 – Neural network architecture to classify metrics into three classes regarding file layout and access spatiality.



Source: Bez et al. (2019).

4.2 Text Classification

This application is a Recurrent Neural Network (RNN) for text Sentiment Analysis (SA). The approach focuses on using RNN with Long Short-Term Memory (LSTM). RNN analyzes an input text (review) and predicts how positive or negative the sentiment expressed is. The implementation of RNN of this work is done using Python (ROSSUM; DRAKE, 2011) programming language, and the modules Keras (MOOLAYIL, 2019), TensorFlow (ABADI et al., 2016b), and Scikit-learn (PEDREGOSA et al., 2011).

The database used was elaborated in the work of Maas et al. (2011). This base has a collection of $50,000$ IMDb[3] ratings. The built data set is balanced, containing an even number of analyses in 2 possible classes: positive (pos) and negative (neg). This means that the sentences are well distributed among the classes. No class has a much larger number of sentences than the other. Thus, the sentiment classification in this dataset can be interpreted as a binary classification problem. The set consists of sentences written in English and is used in several works dealing with sentiment analysis (GANDHI et al., 2021; HAQUE; LIMA; MISHU, 2019; QAISAR, 2020; SUBRAMANIAN et al., 2021; VIELMA; VERMA; BEIN, 2020). This data can be easily found on the Internet[4].

The pre-processing is done in stages. These aim at removing noise present in the sentences and coding the texts so that they can be used in LSTM-based model training:

- Special characters and punctuation marks are removed;

[3]https://www.imdb.com/
[4]https://ai.stanford.edu/ amaas/data/sentiment/

- All words are written in lowercase;
- Stopwords are removed. These are words that do not add much meaning to the text, usually articles, conjunctions, and prepositions. Table 4.2 shows examples of stopwords. The NLTK library was used to remove stop words;
- Sequences are limited to a fixed size (300 words).

Table 4.2 – Examples of *stop words*.

a	all	an	and
at	as	be	but
do	even	from	go
hello	just	look	too
this	who	with	you

Source: The Author.

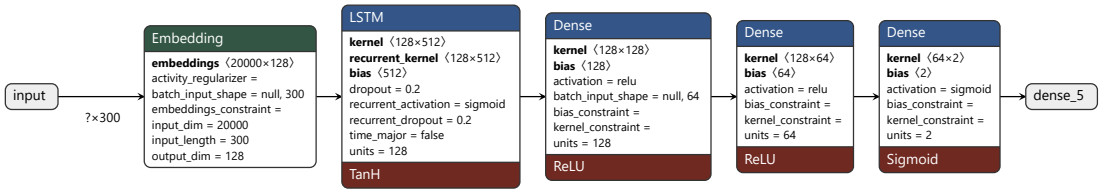
The dataset was split into two following the approach found in the literature: 80% for training and 20% for testing (QAISAR, 2020; MUTINDA; MWANGI; OKEYO, 2023). This application will be referenced throughout the text as RNN-SA. For the evaluation, we considered the following metrics: Train Time, Train Accuracy, Test Time, and Test Accuracy.

4.2.1 RNN-SA Architecture

The model consists of six layers: an input layer, an embedding layer, an LSTM layer, and three dense layers, the last being the model's output. Figure 4.3 shows the RNN architecture. The input layer receives as a parameter the maximum size of the sequences. That is, each review present in the dataset is reduced to a fixed size in order to standardize the input. In our experiments, we defined this size as 300 words. For the embedding layer, we defined the maximum amount of words to keep in the vocabulary (20,000), the dimension of word embedding (128), and the size of the sequences defined in the first layer. Word embedding gives us a way to use a representation in which similar words have similar coding. The embedding dimension is defined according to the size of the data set. A higher dimensional embedding can capture refined relationships between words but requires more data to learn.

The dimensionality of the output space of the LSTM layer has 128 internal units. The dropout configures the fraction of units to be discarded for the linear transformation of the inputs. The dropout value varies in the interval [0, 1], and the higher the rate, the

Figure 4.3 – Sentiment Analysis RNN Architecture.



Source: The Author.

fewer neurons are adjusted during net training. This causes the remaining neurons to learn ways to suppress the absence of the others. This creates a system of redundancy where one neuron can respond to another. The dropout value used is 0.2. For the first dense layer, the output space dimensionality was the same as the LSTM layer, decreasing to 64 in the second and 2 in the last. *Uniform* is the initializer for the kernel weight matrix. A *Rectified Linear Unit* (ReLU) activation function was applied for the first two Dense layers. The last dense layer uses a simple *Sigmoid* activation function, to adjust the output between 0 and 1. The final product is a single output.

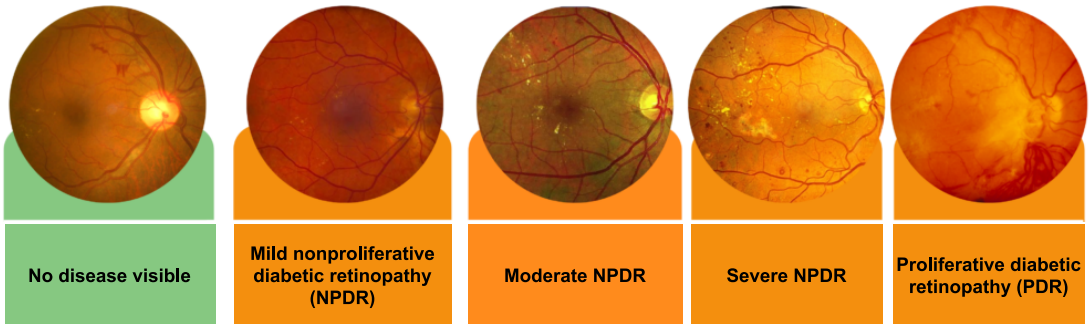
4.3 Image classification

This application is an image classification to detect Diabetic Retinopathy. Our inspiration codebase is the Voets reproduction codebase[5] (VOETS; MØLLERSEN; BONGO, 2019). As input, we used Kaggle's EyePACS dataset[6]. This database is commonly used for deep learning applications in DR detection and is divided into two subsets, train and test. The training dataset contains 35,126 images, where 25,810 have no signs of disease; 2,443 present mild retinopathy; 5,292 present moderate retinopathy; 873 present grave retinopathy; and 708 present proliferative retinopathy. We split these images into training and validating datasets. Thus, we perform a 5-class classification.

First, we process all images by locating the center and radius of the eye fundus and redimensioning every picture to 256x256 pixels. The images dataset was converted to TFRecord format and then uploaded to a bucket on Google Storage for the training in the TPU device. Each TFRecord file contains 2,000 images (except the last one, which has 1,126). We split the TFRecord files into training (80%) and validating (20%) datasets. Therefore, the training dataset[7] consists of 28,000 images, and the validating dataset con-

[5]https://github.com/mikevoets/jama16-retina-replication
[6]https://www.kaggle.com/competitions/diabetic-retinopathy-detection
[7]https://kaggle.com/datasets/cristianokunas/diabetic-tfrecords256

Figure 4.4 – Five-stage classification for Diabetic Retinopathy.



Source: The Author.

sists of 7,126 images.

TFRecord, TensorFlow's custom data format, is a powerful tool. It's natively supported by the high-performance tf.data API, can handle distributed datasets, and takes advantage of parallel I/O. Working with large datasets can greatly benefit from using a binary file format for storage. Binary data consumes less space on disk, is faster to transfer, and can be read more efficiently. Using a binary file format can lead to a faster import pipeline and ultimately reduce the training time for your model.

In addition to performance benefits, the TFRecord file format is also optimized for use with TensorFlow. It simplifies combining multiple datasets and seamlessly integrates with the library's data import and preprocessing features. This is particularly useful for datasets that are too large to fit in memory (88.29 GB for the EyePACS dataset without preprocessing), as only the necessary data (*e.g.* a batch) is loaded from the disk and processed at a time. Overall, the TFRecord file format provides a convenient and efficient way to work with large datasets in TensorFlow.

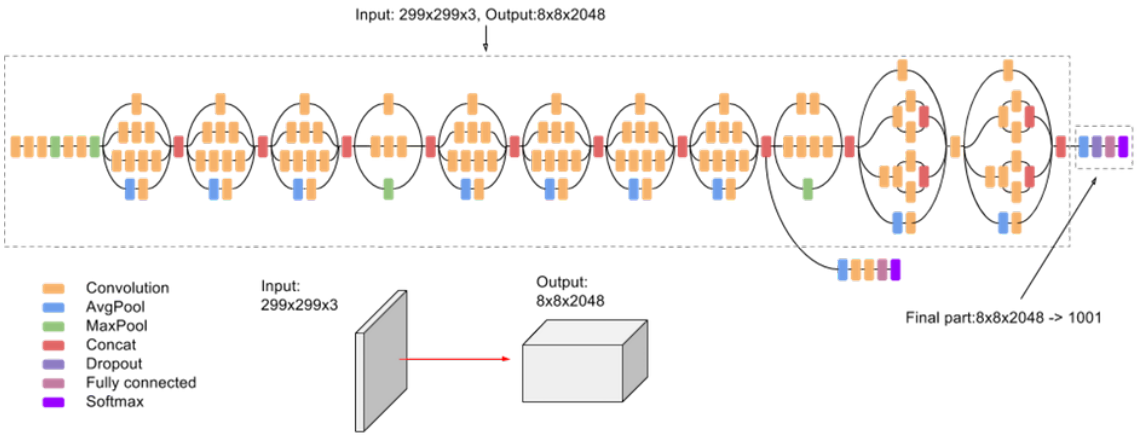
This application will be referenced throughout the text as IC-DR. We considered the following metrics for the evaluation: Train Time and Train Accuracy.

4.3.1 IC-DR Architecture

Our model uses the Inception v3 architecture to transfer learning. We initialized the network with imagenet weights for all layers except the fully connected layer on top, which received random weights. In Figure 4.5, the layers and operation of the Inception v3 architecture are detailed.

After loading the imagenet weights, we add a *Global Average Pooling 2D* layer and two *Dense* layers, the first fully connected with 1024 units using *Rectified Linear*

Figure 4.5 – Inception V3 architecture.



Source: https://cloud.google.com/tpu/docs/inception-v3-advanced.

Unit (ReLU) activation function, and the second using *Softmax* activation function and five units, one to each of 5 classes. Figure 4.6 shows the architecture.

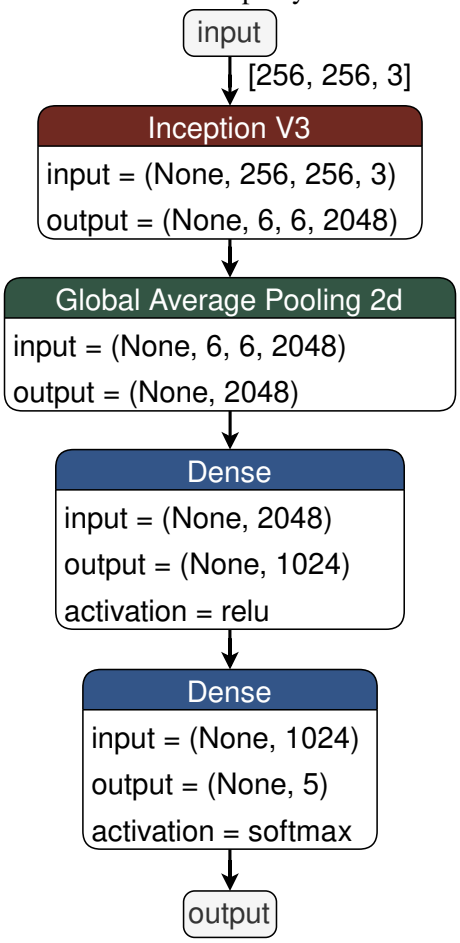
The model uses Adam optimizer, a gradient descendent algorithm based on the adaptive estimation of first and second-order moments. The learning rate value was 0.0014. The accuracy was collected to judge the model. The loss function calculates the logarithmic loss between actual and predicted labels. In this paper, we use the Sparse Categorical Crossentropy function. In this model, we used callbacks to collect the model metrics, monitor the `val_loss` metric to reduce the learning rate, and checkpoints to save the best model.

4.4 Software Setting

A recent survey showed that Python remains the top language for deploying, executing, and integrating ML/DL algorithms and related tasks like data transformation (WANG et al., 2019). Its popularity stems from its ease of learning, fast implementation, and rich environment, including popular ML/DL frameworks like Caffe, Tensorflow, Torch, and MXNet.

Our applications were deployed using Python 3.7.3 and the embedded frameworks Tensorflow (2.6.0) and Keras (2.6.0). We used CUDA Toolkit 11.8 and cuDNN 8.7 for GPU versions, following each developer's recommended installation procedures. This software configuration was chosen due to compatibility.

Figure 4.6 – Diabetic Retinopathy Model Architecture.



Source: The Author.

4.5 Experimental Platforms

The experiments described in this paper were conducted on the computational resources available at the Google Cloud[8], in the PCAD infrastructure at INF/UFRGS[9], and Santos Dumont Supercomputer (SDumont)[10] at the National Laboratory for Scientific Computing (LNCC)[11]. For the selection of the platforms, we adopt as a criterion the access availability.

- **Cloud TPUv2:** We use a single TPUv2 with 8 cores and 64 GB memory. The TPU device provides 180 Teraflops performance. This environment is named **TPUv2** throughout the rest of this work (Google Cloud - Iowa us-central1).
- **Cloud TPUv3:** We use a single TPUv3 with 8 cores and 128 GB memory. The TPU

[8]https://cloud.google.com/
[9]http://gppd-hpc.inf.ufrgs.br/
[10]https://sdumont.lncc.br
[11]https://www.lncc.br

device provides 420 Teraflops performance. This environment is named **TPUv3** throughout the rest of this work (Google Cloud - Iowa us-central1).

- **Blaise:** A single compute node composed of two Intel Xeon E5-2699 v4 Broadwell (2.2GHz) CPU, 44 physical cores (22 per socket), 256 GB of RAM, and four NVIDIA Tesla P100-SXM2-16GB. All the experiments conducted used only one GPU. This environment is named **P100** throughout the rest of this work (PCAD - Porto Alegre).
- **Bull Sequana X1120 (GPU):** A single compute node composed of two Intel Xeon Cascade Lake Gold 6252 (2.1GHz) CPU, 48 physical cores (24 per socket), 384 GB of RAM, and four NVIDIA Tesla V100-SXM2-32GB. All the experiments conducted also used only one GPU. This environment is named **V100** throughout the rest of this work (Santos Dumont - Petrópolis).

5 CHALLENGES OF EMPLOYING CLOUD HARDWARE ACCELERATORS

With the emergence of new intelligent applications, there has been a revolution in information management, mainly including processing routines, storage, and computing capacity. We have seen a significant evolution of computing paradigms in the last decade. Cloud computing is one of the most well-known and popular paradigms, and its adoption continues to accelerate as companies seek agility, flexibility, and new sources of competitive advantage. Hardware accelerators, such as GPUs, FPGAs, and ASICs, have become increasingly popular in the cloud due to their ability to accelerate compute-intensive workloads. However, employing hardware accelerators in the cloud also presents several challenges that need to be considered.

One of the main challenges is related to data when employing hardware accelerators in the Cloud. For example, TPUs are designed to work in conjunction with Google Cloud Storage (GCS) to access data efficiently. However, transferring large volumes of data to the Cloud can be time-consuming and costly, especially if the data is stored in a different cloud or on-premise. Storing large volumes of data on the Cloud can be expensive, especially if the data needs to be accessed frequently. Also, preprocessing the data for use with TPUs can be time-consuming and computationally expensive.

These challenges can be mitigated by automating data preprocessing and optimizing data transfer and storage. With that in mind, we developed a pre-processing application to optimize data storage and transfer and investigated the impact that pre-processing causes on network interconnection (KÜNAS et al., 2022). This pre-processing application also was used to prepare the EyePACS dataset for the IC-DR application. Then, its performance was evaluated and verified in Cloud and Local. The application was implemented in Python and is composed of two modules. The first module performs all pre-processing in a local environment and then sends the data to the storage. Differently, the second module does the opposite, sending the original data to the storage and then pre-processing it on the Cloud.

The dataset used in this experiment is the *APTOS 2019 Blindness Detection* available in the Kaggle competition for diabetic retinopathy[1], which was chosen because it has a smaller size compared to the EyePACS dataset used in this dissertation to train the IC-DR application, facilitating our experiment. This dataset is commonly used for machine learning applications in DR detection and is divided into two subsets. We selected

[1]https://www.kaggle.com/c/aptos2019-blindness-detection/data

the training subset containing $3,662$ retinal images. The images are in PNG (Portable Network Graphics) format and have varying sizes (Figure 5.3).

The Cloud environment consists of a virtual machine type N1 Standard (16 cores, 60 GB RAM), Linux Ubuntu 20.04 LTS operating system, located in eastern South America. The Storage is a Bucket on Google Cloud Storage provided through an instance of Firebase. This storage is an object storage solution with large capacity, high availability, and redundancy. The Local environment comprises a device with an Intel Core i7-9750 processor with 6 physical cores (2.60 GHz). This equipment has 16 GB of DDR4 RAM Memory. We used the Linux Ubuntu 20.04.4 LTS operating system with *kernel* version 5.13.0-41.

To evaluate the implementation, we first evaluated the network connection. This evaluation measured the maximum amount of data that could be sent from the local node to the cloud provider. The measurements started with throughput analysis using the *Iperf* tool (TIRUMALA et al., 2005). The results obtained from this first step are described in Table 5.1.

Table 5.1 – Network measurements with the *Iperf* tool.

Parameter	**Local to Cloud**
Interval	60 seconds
Total transferred	406 MBytes
Bandwidth	56.8 Mbits/second

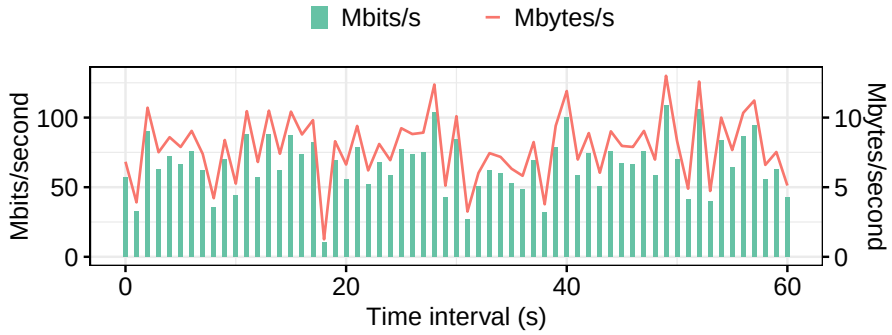
Source: The Author.

Figure 5.1 shows the bandwidth and the amount of bytes transferred every second. Comparing with the measurement performed with the *Iperf* tool, it is noticeable that when sending the original data, we used all the available bandwidth, with an average of ≈ 66 Mbps (Figure 5.1 (a)), which can overload the network. The opposite occurs when sending pre-processed data, in which we reduce the bandwidth used by about $\approx 11.5\times$, with an average of ≈ 5.7 Mbps (Figure 5.1 (b)). We also observed a variation in the original data's minimum and maximum transfer peaks (Figure 5.1 (a)). This is due to size differences between the files.

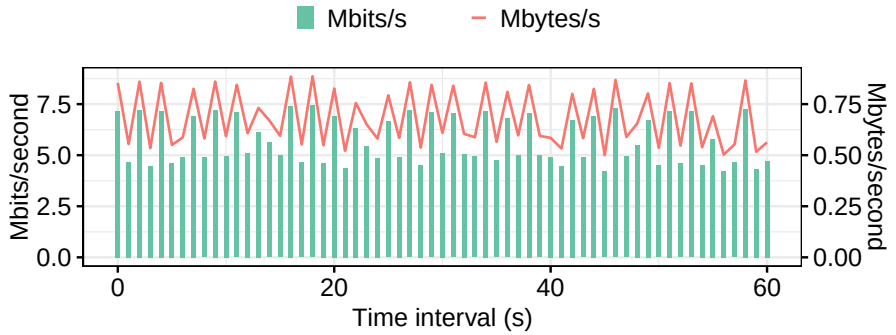
We compare the **sequential execution** time between the Local and Cloud environments and the **parallel execution** time between the Local and Cloud environments. The parallel version splits tasks between processes. For both executions, we set 5 processes.

Parallel executions on the Local and the Cloud show a gain of $\approx 71.12\%$ and $\approx 73.01\%$, respectively, compared to the sequential version. In comparison, the average

Figure 5.1 – Bandwidth used and total transferred.



(a) Without pre-processing.



(b) With pre-processing.

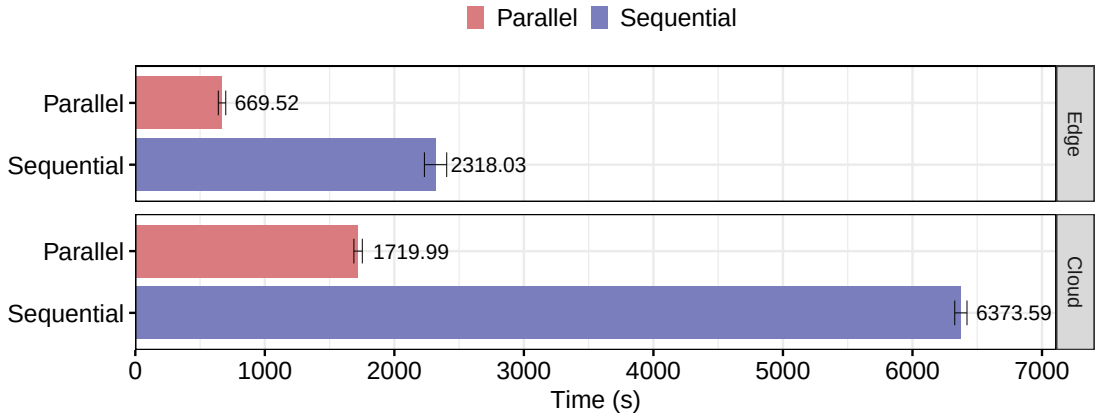
Source: The Author.

execution time of the sequential application on the Local was $\approx 2,318.03$s, a gain of $\approx 63.63\%$ compared to the same version running on the Cloud. In the parallel version, the average execution time on the Local was ≈ 669.52s, which represents a gain of $\approx 61.07\%$ compared to the same version running on the Cloud (Figure 5.2). As we can see, executions on the Local perform better than on the Cloud because they do not overload the network. By pre-processing the data on the Local, in addition to decreasing the total size of the data by $\approx 44\times$ (decreasing from $\approx 8,204.5$ MB to ≈ 185.7 MB), we also save bandwidth for transferring this data.

We present the application's throughput in Table 5.2. This throughput rate represents the number of images pre-processed and transferred every second. It can be seen that in the cloud sequential execution model, where the first step is sending the original data to the Cloud, the throughput is too low. This is because the dataset has a large variation in the sizes of the original images. In Figure 5.3 (a), we demonstrate this variation, with sizes ranging from ≈ 200 KB to $\approx 7,500$ KB. This disparity in values significantly impacts execution time and bandwidth consumption.

The best result in terms of throughput is obtained in the parallel version executed

Figure 5.2 – Sequential and parallel version execution times: comparing execution on the Local and in the Cloud.



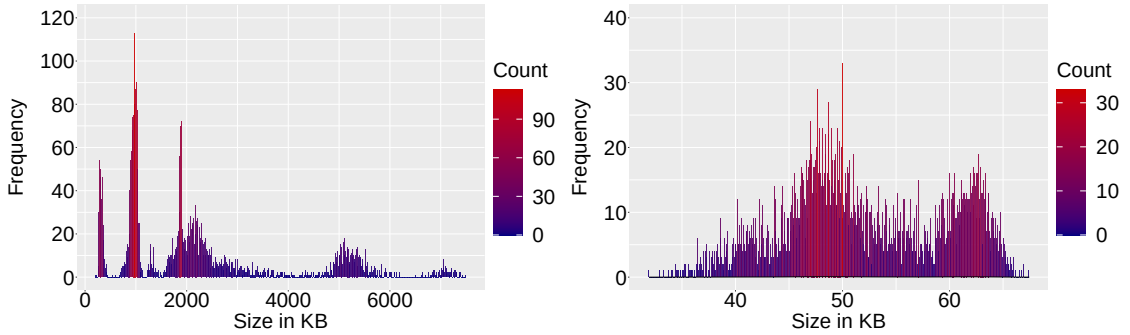
Source: The Author.

Table 5.2 – Throughput in images/second, in sequential and parallel runs, in Local and Cloud environments.

	Local	**Cloud**
Parallel	5.47 images/second	2.13 images/second
Sequential	1.58 images/second	0.57 images/second

Source: The Author.

Figure 5.3 – Frequency histogram of images by size in KB without and with pre-processing.



(a) Without pre-processing.

(b) With pre-processing.

Source: The Author.

on the Local, reaching ≈ 5 images/second. This is because the pre-processing step allows us to normalize the data, reducing the size of each image and the size variation observed in the original data (30 - 70 KB), as shown in Figure 5.3 (b).

6 EXPERIMENTAL EVALUATION

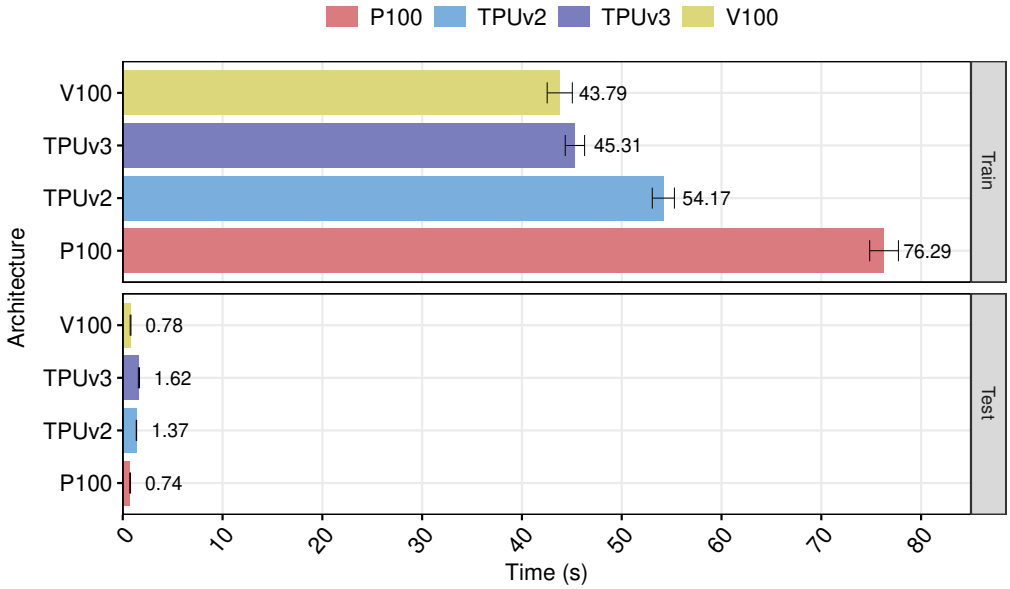
In this section, we showcase the performance evaluation results obtained from the experimental platform mentioned in Section 4. We present metrics for execution time for different architectures. We also present the accuracy achieved by our models and perform a cost-efficiency analysis when using TPUs compared to a local cluster.

6.1 Performance Evaluation

6.1.1 NN-TWINS

For **NN-TWINS** application, the performance metrics are illustrated in Figure 6.1. The t-test (KIM, 2015) was used to compare all sets of time metrics collected and indicates that the results of training times are significantly different from each other. The average training time on the **V100** was $\approx 1.74\times$ faster than on the **P100**, and the mean training time of **TPUv2** was $\approx 1.41\times$ faster than on the **P100**. However, using the **V100**, we observed a speedup of $\approx 1.24\times$ compared to the **TPUv2**.

Figure 6.1 – Training and testing times of NN-TWINS application in each hardware.



Source: The Author.

On the other hand, the average training time on the **TPUv3** was $\approx 1.68\times$ faster than on the **P100** and $\approx 1.20\times$ faster than on the **TPUv2**. But, the speedup observed on the V100 over the TPUv2 is not observed compared to the TPUv3. The **V100** was only

$\approx 1.03\times$ faster than on the **TPUv3**.

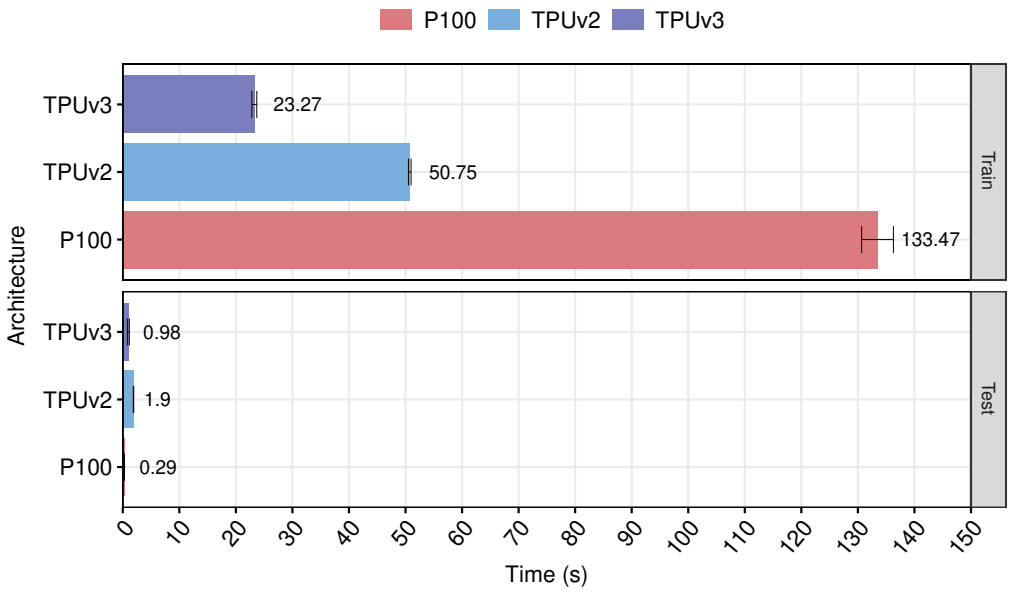
The first hypothesis to explain this result is that the application context may negatively influence performance since it is a relatively small model with few parameters. The second point is that TPUs can perform better when the batch size is larger, making better use of TPU memory. The documentation recommends that to optimize memory use, you should use the largest batch size that can fit in memory.

6.1.2 RNN-SA

Previous studies have shown that the batch size influences the training time, which tested batch sizes from 32 up to 2048, with little or no degradation in the model's accuracy (KÜNAS et al., 2022). For this reason, in this dissertation, we used a larger batch size that allowed shorter training times. Also, as explained earlier, it is essential to increase the batch size to better use the architecture's capacity.

The performance metrics of the **RNN-SA** application are illustrated in Figure 6.2. The t-test was used to compare all sets of time metrics collected and indicates that the results of training times are significantly different from each other. The average training time on the **TPUv2** was $\approx 2.63\times$ faster than on the **P100**, and the mean training time of **TPUv3** was $\approx 2.18\times$ faster than on the **TPUv2**. We observed a speedup of $\approx 5.74\times$ on the **TPUv3** compared to the **P100**.

Figure 6.2 – Training and testing times of the Neural Network in each computational resource.



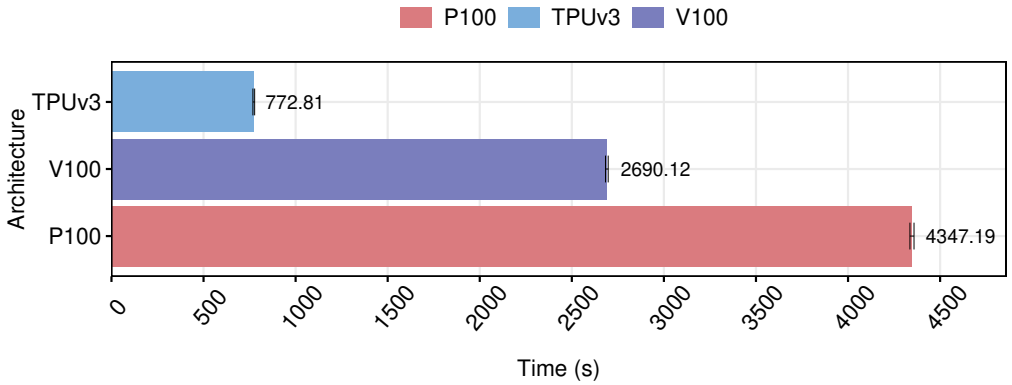
Source: The Author.

Based on the experimental results, using Cloud TPU appears to be a favorable option for training a sentiment analysis model. Despite the potential cost associated with transferring the dataset to the cloud (which we determined to be approximately 7 seconds for this particular dataset), it is outweighed by the benefits of faster training time and avoiding additional strain on local resources.

6.1.3 IC-DR

The performance results of **IC-DR** application are presented in Figure 6.3, showcasing that the **V100** outperformed the **P100** in terms of average training time, with a $1.63\times$ improvement. On the other hand, the **TPUv3** showed a remarkable performance with an average training time that was $3.48\times$ faster than the **V100**. This result is even more significant when compared to the **P100**, where the **TPUv3** demonstrated an improvement of $5.63\times$ in terms of average training time.

Figure 6.3 – Neural Network Training times in different hardware.

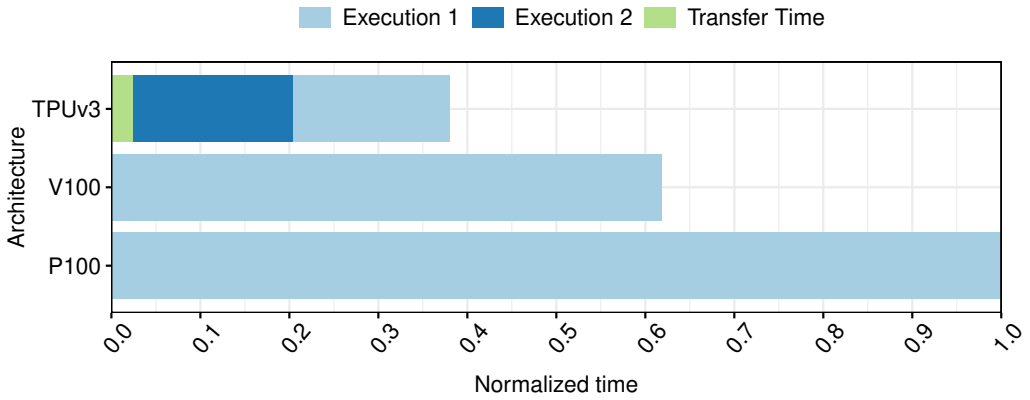


Source: The Author.

This gain is achieved without any code optimization. Furthermore, although there is a time spent to transfer the dataset to the cloud to run on the TPU device, as shown in Figure 6.4, the performance of **TPUv3** is still quite considerable, about $4.95\times$ and $3.06\times$ more effective than **P100** and **V100**, respectively. Even though it needs to transfer the data to the cloud, it is possible to run the model multiple times on the TPU (Execution 1 and 2) in less time compared to local runs where the data is already available. The information about the transfer is presented in Table 6.1. The measurements demonstrate that the average throughput was $\approx$ 12.2 MBytes/second, with an average size of 20.73 MBytes.

Although we did not explore optimization techniques for scaling training to large

Figure 6.4 – Comparison of execution times adding time to transfer data to the cloud (all values have been normalized).



Source: The Author.

batch sizes, such as selecting large batch optimizers and learning rate schedules, as well as utilizing distributed evaluation techniques and batch normalization, we still achieved good performance on the **TPUv3**, which indicates that Cloud TPU can be a good choice for deep learning models training, especially for the Diabetic Retinopathy use case.

Table 6.1 – Dataset transfer measurements to the cloud.

Parameter	**Local to Cloud**
Time	106.510 seconds
Average throughput	12.2 MBytes/second
Total Tranferred	932.2 MBytes

Source: The Author.

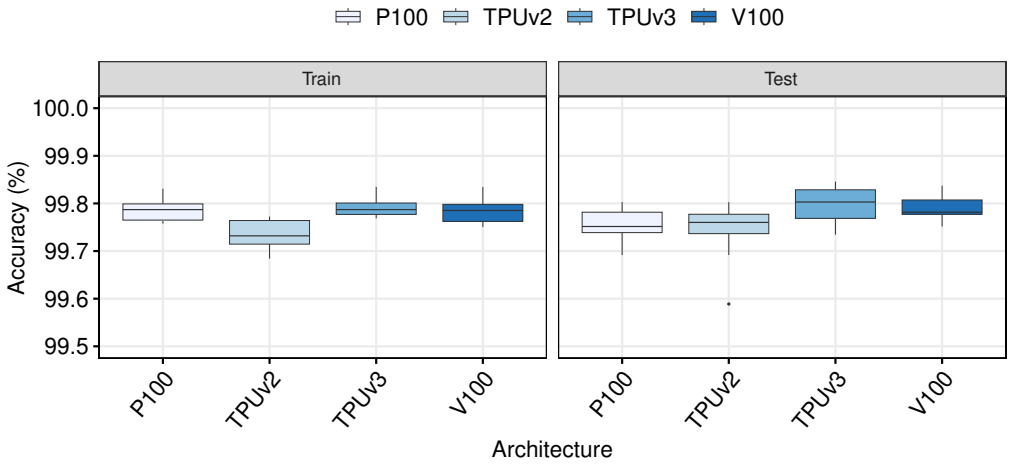
6.2 Accuracy Evaluation

6.2.1 NN-TWINS

We trained the **NN-TWINS** model on $27{,}238$ samples and tested it on $11{,}674$ samples, with a batch size of 32 and limiting to 50 epochs. It is important to notice that all three classes are correctly identified with reasonable probability. The model's accuracy exceeds 99% for training and testing, with a standard deviation of less than 1%. The statistical test has indicated that the training accuracy is not significantly different, but the testing accuracy is significantly different. Figure 6.5 depicts the accuracy when running in each platform. In this application, the global batch size was 256. Even using a TPU for

training, the accuracy remained similar to the baseline.

Figure 6.5 – Training and testing accuracy of NN-TWINS application.

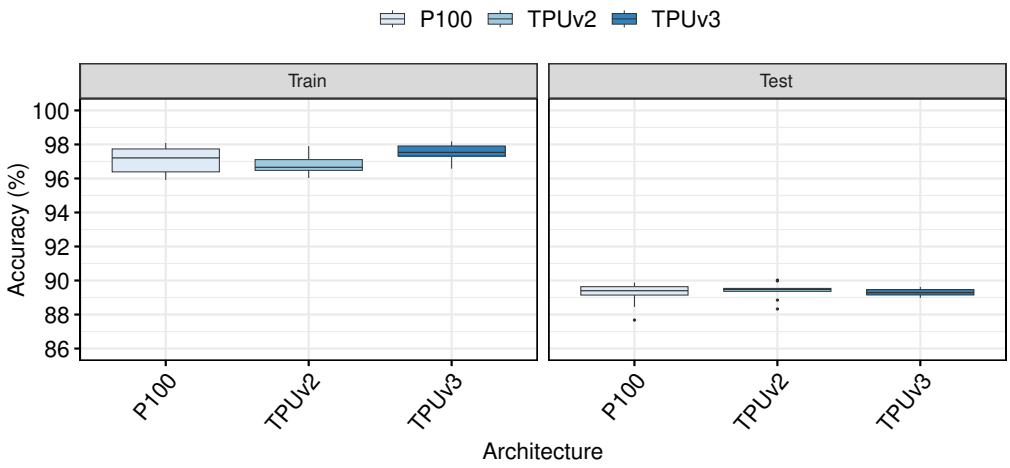


Source: The Author.

6.2.2 RNN-SA

We trained the **RNN-SA** model on $36{,}000$ samples, validate on $10{,}000$ samples, and tested it on $4{,}000$ samples, and limiting to 5 epochs. We chose the batch size of 1024 because, according to a previous study, it was the one that presented the best performance in the average training time. The model's average accuracy achieves 97.51% for training and 89.30% for testing with a standard deviation of less than 1% on both platforms. Figure 6.6 depicts the accuracy when running in each platform.

Figure 6.6 – Training and testing accuracy of the Neural Network model.

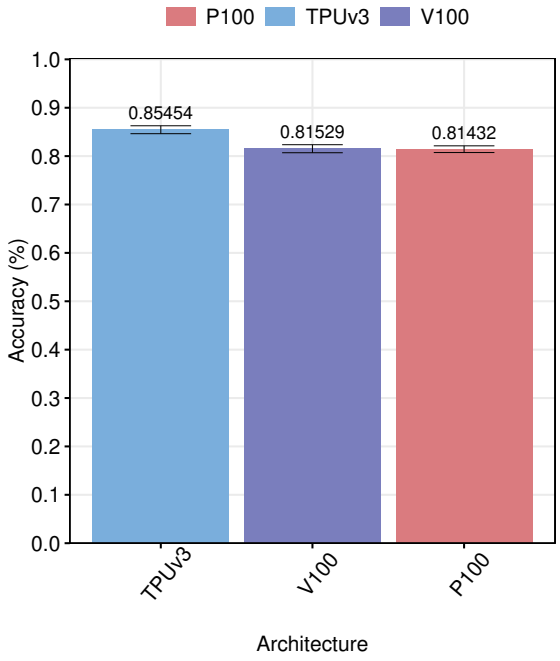


Source: The Author.

6.2.3 IC-DR

As mentioned previously, we use the Inception V3 architecture for the **IC-DR** model. After loading and initializing the network with the imagenet weights and adding all layers described in the previous section, we trained the model on $28{,}000$ samples and validated it on $7{,}126$ samples, with a batch size of 32 and limiting to 25 epochs. For the TPU, we use the strategy presented in Section 2.2.3, where each core processes 32 examples, resulting in a global batch size of 256. All values shown are averages, and the t-test is used to compare them. The model's average accuracy achieves 85.45%, 81.59% and 81.43% for **TPUv3**, **V100**, and **P100**, respectively, with a standard deviation of less than 1% on both architectures. Figure 6.7 depicts the accuracy when running in each architecture.

Figure 6.7 – Neural Network Accuracy in different hardware.



Source: The Author.

The 85% accuracy of the diabetic retinopathy detection model is deemed appropriate. This is reinforced by the fact that other studies have shown similar accuracy rates (LIN et al., 2018; GHOSH; GHOSH; MAITRA, 2017), including using the Inception V3 architecture (MOHAMMADIAN; KARSAZ; ROSHAN, 2017). It is important to highlight that the early detection of diabetic retinopathy is crucial to avoid serious vision complications. A model with an accuracy of 85% can provide reliable results and significantly contribute to the accurate diagnosis of the disease.

6.3 Cost Evaluation

This section performs a cost-efficiency analysis when using TPUs compared to a local cluster. We considered a local cluster for the cost estimation, the one with the worst training time.

NN-TWINS: This application presents the worst cost-efficiency. The local cluster cost estimated is nearly USD 0.060, while in a TPUv3 is USD 0.10. This means the cost is higher to run this application in the cloud than in a local cluster, about 66% more. In this case, the achieved performance, around $1.68\times$, is not worth the execution cost. The TPUv3 is about 41% less efficient compared to a local cluster.

RNN-SA: The TPUv3 achieves performance that is approximately $5.73\times$ better than the local cluster. If we were to train the model on the local cluster, we estimate the cost to be USD 0.106. On the other hand, training the model on the TPUv3 results in an average cost per training of USD 0.052. Compared to the local cluster, the cloud is about 51% more efficient in performing the same amount of work.

IC-DR: For the IC-DR application, the TPUv3 cost per hour is about $\approx 2.8\times$ higher than the local cluster. However, the performance achieved was $\approx 5.63\times$ better. On the local cluster, our estimated cost would be USD 3.44 to train the model. On the other hand, TPUv3, costing USD 8 per hour, gives us an average cost per training of USD 1.72. TPUv3 is about 50% more efficient, *i.e.*, it costs 50% less to perform the same amount of work in the cloud than in the local cluster. This indicates that Cloud TPU can be a good choice for training deep learning models, especially for this case, the Diabetic Retinopathy model.

Additionally, the cost per training can be further reduced by using preemptible TPUs. Preemptible TPUs cost much less than non-preemptible ones, about 70% less. However, it can be interrupted at any time. In this case, the application must be restart-resilient to save model checkpoints regularly and restores the most recent one upon restart. The estimated cost of using preemptible TPUv3 is presented in Table 6.2.

Table 6.2 – Estimated cost (in Dollar) for each application when using preemptible TPUv3.

Application	Cost	Cost-Efficiency (%)
NN-TWINS	$0.030	49.50%
RNN-SA	$0.016	84.91%
IC-DR	$0.52	84.88%

Source: The Author.

This cost is about $3.3\times$ better than the on-demand TPU in both cases and represents a cost efficiency of around 84.88%, 84.91%, and $49,50\%$ for **IC-DR**, **RNN-SA**, and **NN-TWINS** applications, respectively, compared to the local cluster. Compared to on-demand TPU, it is possible to observe that applications achieve better cost efficiency when using the preemptive TPU, including the **NN-TWINS** application.

6.4 Discussion

We sought to evaluate the performance and accuracy of DL training models by asynchronously offloading the training to the Cloud using TPU devices. Such an approach aids in alleviating the contention for high-demanded local HPC resources, allowing them to be focused on running applications.

The I/O access pattern classifier model presented up to $1.68\times$ gain compared to the other devices. Cloud TPUs are designed to excel in performing matrix operations in parallel. However, small models with small datasets may not have enough matrix operations to take advantage of the full parallelism offered by the TPU, leading to less efficient use of resources and resulting in lower performance. In addition, there is overhead associated with transferring data to and from the TPU and initializing the TPU for each task. With small datasets, this overhead may represent a more significant proportion of the total time required to complete the task, reducing overall performance. Training the model on cloud TPU devices is feasible in this case because the (re)training phase can take longer to complete and can be done asynchronously in the background. It is important to note that, in a production environment, the number of observations collected to (re)train the model will be more expressive, which may cause a more significant burden on the HPC system as the dataset gets larger. Furthermore, the model presented accuracy above 99% for the training and test datasets in all the computing environments we explored.

Differently, the performance achieved is better for the Sentiment Analysis, where the dataset is a little bigger, reaching $5.74\times$. This occurs because the dataset, although more extensive than the one used with NN-TWINS, still fits into the TPU memory and also because we increased the global batch size to 1024. The TPU cores operate on the XLA memory layout (GOOGLE, 2023), which requires that the batch dimension of each tensor be a multiple of 8 (JOUPPI et al., 2017) to more optimally utilize the memory of each TPU core and increase throughput. However, it's important to note that training with large batch sizes can lead to a degradation in model quality due to the "generalization

gap" (KESKAR et al., 2016). This has been observed compared to models trained with smaller batch sizes. In this application, we achieved reasonable accuracy rates in both architectures we explored. This accuracy rate is similar to that found in other studies that used the same dataset and similar models (QAISAR, 2020; HAQUE; LIMA; MISHU, 2019; WIDAYAT; ADJI et al., 2018; YENTER; VERMA, 2017).

For higher datasets is recommended to convert to another format that is easier to read from storage. TFRecord, TensorFlow's custom data format, is a powerful tool. It's natively supported by the high-performance tf.data API, can handle distributed datasets, and takes advantage of parallel I/O. Working with large datasets can greatly benefit from using a binary file format for storage. Binary data consumes less space on disk, is faster to transfer, and can be read more efficiently. Using a binary file format can lead to a faster import pipeline and reduce the model training time. In addition to performance benefits, the TFRecord file format is also optimized for use with TensorFlow. It simplifies combining multiple datasets and seamlessly integrates with the library's data import and preprocessing features. It is particularly useful for datasets too large to fit in memory, *e.g.*, the EyePACS dataset used in the IC-DR application with about 88.29 GB, as only the necessary data (*e.g.*, a batch) is loaded from the disk and processed at a time. Overall, the TFRecord file format provides a convenient and efficient way to work with large datasets in TensorFlow. As we mentioned earlier, this dataset was preprocessed by locating the center and radius of the eye fundus and redimensioning every picture to 256×256 pixels and then converted to TFRecord format and sent to storage on Google Cloud. This application achieved up to $5.63\times$ speedup in the best case with competitive accuracy, and similar to others studies (LIN et al., 2018; GHOSH; GHOSH; MAITRA, 2017; MOHAMMADIAN; KARSAZ; ROSHAN, 2017).

When evaluating cost efficiency, we found that TPUs offer better cost efficiency for larger applications, which indicates that Cloud TPU can be a good choice for training large DL models. However, small applications that end up not using all the hardware may have little or no cost efficiency, *i.e.*, the execution cost is higher and is not compensated by the performance obtained. In these cases, the choice to run in the Cloud, especially on TPU devices, must be done cautiously, noting if there are other benefits besides performance. For the NN-TWINS application, *e.g.*, such an approach is promising, considering that in a production environment, the number of observations collected to (re)train the model will be much larger than what we used in our experiments. In addition, the (re)training phase can take longer to complete and can be done asynchronously

in the background.

7 CONCLUSION AND FUTURE WORK

TPUs are specialized hardware devices designed to accelerate machine learning workloads, especially matrix operations used in deep learning algorithms. TPUs are considered better for deep learning tasks because they provide a high computational performance, efficient and cost-effective solution for accelerating these workloads.

In this thesis, we sought to evaluate the performance and accuracy of three applications by asynchronously offloading the training to the cloud using TPU devices. Such an approach aids in alleviating the contention for high-demanded local HPC resources, allowing them to be focused on running applications. We have observed promising results by modifying the neural network models to train on TPU devices. We also evaluate the cost-efficiency of running these applications on such devices.

The I/O access pattern classifier (NN-TWINS) presented accuracy above 99% for the training and test datasets in the three computing environments we explored. The average training time of this model on TPUv3 was $\approx 1.68\times$ faster than on P100. The execution of NN-TWINS on cloud TPU devices, although it is 41% more expensive than the local execution, it is still a feasible solution because the (re)training phase can take longer to complete and can be done asynchronously in the background. Also, it is important to remember that the amount of data collected to train this model will be larger in a production environment. The Sentiment Analysis model (RNN-SA) presented average accuracy of $\approx 97\%$ for the training and $\approx 89\%$ for the test datasets in the three computing environments we explored. The performance of TPUv3 was $\approx 5.74\times$ faster than on P100. Compared to the local cluster, the cloud is about 51% more efficiently performing the same amount of work for this application.

For the Diabetic Retinopathy detection model (IC-DR), the TPUv3 demonstrated an improvement of $5.63\times$ in terms of average training time compared to the P100This gain is achieved without any code optimization. Furthermore, although there is a time spent to transfer the dataset to the cloud to run this model on the TPU device, the performance of TPUv3 is still quite considerable, about $4.95\times$ more effective. The data size can negatively influence the application's execution time and congests the network interconnection. Thus, pre-processing data at the Edge can contribute to bandwidth savings. We studied this challenge in Section 5, where we considerably reduced the total data size. The accuracy of the diabetic retinopathy detection model, which stands at 85%, is considered suitable. This is further supported by other studies that have reported comparable

accuracy rates. It should be emphasized that timely detection of diabetic retinopathy is essential to prevent severe vision-related complications.

Our results provide a good starting point for those interested in improving the performance of their deep-learning models.

7.1 Future work

Future work will deepen the cost analysis and extend the performance evaluation will extend the performance evaluation to the Cloud TPUv4 and the TPU Pods, exploring optimization techniques, such as selecting large batch optimizers and learning rate schedules, to scale training to large batch sizes while keeping the accuracy rate. In addition, future work will focus on proposing an automatic mechanism to select efficient Cloud resources for DL workloads that optimize cost and performance.

7.2 Publications

The following papers were produced during this dissertation. We first list the ones strongly related to this work, including those under review:

- **KÜNAS, C. A.;** PADOIN, E. L.; NAVAUX, P. O. A. Accelerating Deep Learning Model Training on Cloud Tensor Processing Unit. In: 13th International Conference on Cloud Computing and Services Science, CLOSER 2023, (*Accepted*).
- **KÜNAS, C. A.;** PADOIN, E. L.; BEZ, J. L.; CARISSIMI, A.; NAVAUX, P. O. A. Executando na Nuvem um Detector de Padrões de Acesso de E/S. In: In: ESCOLA REGIONAL DE ALTO DESEMPENHO DA REGIÃO SUL (ERAD-RS), 2023. Porto Alegre: Sociedade Brasileira de Computação, 2023, (*Under Review*).
- **KÜNAS, C. A.;** SERPA, M. S.; BEZ, J. L.; PADOIN, E. L.; NAVAUX, P. O. A. Offloading the Training of an I/O Access Pattern Detector to the Cloud. In: 2021 International Symposium on Computer Architecture and High Performance Computing Workshops (SBAC-PADW) (pp. 15-19). IEEE, 2021.
- **KÜNAS, C. A.;** PINTO, D. R.; NAVAUX, P. O. A.; GRANVILLE, L. Z. Computaçao de Borda versus Computaçao em Nuvem: Impacto do Pré-processamento de Imagens de Retinas. In: Anais do XXIII Simpósio em Sistemas Computacionais de Alto Desempenho (WSCAD) (pp. 85-96). SBC, 2022.

- **KÜNAS, C. A.;** PINTO, D. R.; GRANVILLE, L. Z.; SERPA, M. S.; PADOIN, E. L.; NAVAUX, P. O. A. Edge Computing versus Cloud Computing: Impact on Retinal Image Pre-processing. In: 2022 International Symposium on Computer Architecture and High Performance Computing Workshops (SBAC-PADW) (pp. 51-56). IEEE, 2022.

The following papers were also published during this dissertation:

- **KÜNAS, C. A.;** SERPA, M. S.; PADOIN, E. L.; NAVAUX, P. O. A. Improving Performance of Long Short-Term Memory Networks for Sentiment Analysis Using Multicore and GPU Architectures. In: CARLA 2021 - Latin American High Performance Computing Conference (pp. 34-47). Springer, Cham, 2022.
- SERPA, M.; SILVA, P.; **KÜNAS, C.;** CARISSIMI, A.; PANETTA, J.; NAVAUX, P. Energy Efficiency of Reverse Time Migration on HPC Architectures. In: Third EAGE Workshop on HPC in Americas (Vol. 2022, No. 1, pp. 1-5). European Association of Geoscientists & Engineers, 2022.
- **KÜNAS, C. A.;** PADOIN, E. L.; SERPA, M. S.; DIENER, M.; NAVAUX, P. O. A. Managing the Residual Load Imbalance and Clock Frequency to Save Energy on Iterative Applications. In: CARLA 2022 - Latin America High-Performance Computing Conference. Workshop 6: Energy and HPC, Porto Alegre, Brazil, 2022.
- HECK, L. P.; **KÜNAS, C. A.;** SERPA, M. S.; PADOIN, E. L.; NAVAUX, P. O. A. Convolutional Neural Networks Applied to Emotions Recognition in Heterogeneous Architectures. In: CARLA 2022 - Latin America High-Performance Computing Conference. Workshop 1: Advanced Computing Trends in Latin America, Porto Alegre, Brazil, 2022.
- **KÜNAS, C. A.;** DIENER, M.; PADOIN, E. L.; NAVAUX, P. O. A. EnergyLB: combinando balanceamento de carga dinâmico com técnicas DVFS para reduzir o consumo de energia. In: In: ESCOLA REGIONAL DE ALTO DESEMPENHO DA REGIÃO SUL (ERAD-RS), 2023. Porto Alegre: Sociedade Brasileira de Computação, 2023, (*Under Review*).
- SCHUSSLER, B.S.; RIGON, P. H. C.; SERPA, M. S.; **KÜNAS, C. A.;** CARISSIMI, A.; PANETTA, J.; NAVAUX, P. O. A. Comparando o Desempenho entre Computação em Nuvem e Servidor Local na Execução do Método Fletcher. In: In: ESCOLA REGIONAL DE ALTO DESEMPENHO DA REGIÃO SUL (ERAD-RS), 2023. Porto Alegre: Sociedade Brasileira de Computação, (*Under Review*).

REFERENCES

ABADI, M. et al. Tensorflow: Large-scale machine learning on heterogeneous distributed systems. **arXiv preprint arXiv:1603.04467**, 2016.

ABADI, M. et al. Tensorflow: Large-scale machine learning on heterogeneous distributed systems. **arXiv preprint arXiv:1603.04467**, 2016.

ABIODUN, O. I. et al. State-of-the-art in artificial neural network applications: A survey. **Heliyon**, Elsevier, v. 4, n. 11, p. e00938, 2018.

BEN-NUN, T.; HOEFLER, T. Demystifying parallel and distributed deep learning: An in-depth concurrency analysis. **ACM Computing Surveys (CSUR)**, ACM New York, NY, USA, v. 52, n. 4, p. 1–43, 2019.

BEZ, J. L. et al. Detecting i/o access patterns of hpc workloads at runtime. In: **2019 31st International Symposium on Computer Architecture and High Performance Computing (SBAC-PAD)**. [S.l.: s.n.], 2019. p. 80–87.

BEZ, J. L. et al. Twins: Server access coordination in the i/o forwarding layer. In: IEEE. **2017 25th Euromicro International Conference on Parallel, Distributed and Network-based Processing (PDP)**. [S.l.], 2017. p. 116–123.

BOX, G. E.; COX, D. R. An analysis of transformations. **Journal of the Royal Statistical Society: Series B (Methodological)**, Wiley Online Library, v. 26, n. 2, p. 211–243, 1964.

BUYYA, R. et al. Cloud computing and emerging it platforms: Vision, hype, and reality for delivering computing as the 5th utility. **Future Generation computer systems**, Elsevier, v. 25, n. 6, p. 599–616, 2009.

CARNEIRO, T. et al. Performance analysis of google colaboratory as a tool for accelerating deep learning applications. **IEEE Access**, IEEE, v. 6, p. 61677–61685, 2018.

CHO, M. et al. PowerAI DDL. **arXiv preprint arXiv:1708.02188**, 2017.

GANDHI, U. D. et al. Sentiment analysis on twitter data by using convolutional neural network (cnn) and long short term memory (lstm). **Wireless Personal Communications**, Springer, p. 1–10, 2021.

GHOSH, R.; GHOSH, K.; MAITRA, S. Automatic detection and classification of diabetic retinopathy stages using cnn. In: IEEE. **2017 4th International Conference on Signal Processing and Integrated Networks (SPIN)**. [S.l.], 2017. p. 550–554.

GOODFELLOW, I.; BENGIO, Y.; COURVILLE, A. **Deep learning**. [S.l.]: MIT press, 2016. <http://www.deeplearningbook.org>.

GOOGLE. **Cloud TPU**. 2022. Available from Internet: <https://cloud.google.com/tpu>. Accessed in: Nov. 09, 2022.

GOOGLE. **Cloud TPU System Architecture**. 2022. Available from Internet: <https://cloud.google.com/tpu/docs/system-architecture-tpu-vm>. Accessed in: May 28, 2022.

GOOGLE. **XLA: Google's Accelerated Linear Algebra library**. 2023. [Accessed Jan. 26, 2023]. Available from Internet: <https://www.tensorflow.org/xla>.

GUPTA, S.; ZHANG, W.; WANG, F. Model accuracy and runtime tradeoff in distributed deep learning: A systematic study. In: IEEE. **2016 IEEE 16th International Conference on Data Mining (ICDM)**. [S.l.], 2016. p. 171–180.

HAHNLOSER, R. H. et al. Digital selection and analogue amplification coexist in a cortex-inspired silicon circuit. **Nature**, Nature Publishing Group, v. 405, n. 6789, p. 947–951, 2000.

HAQUE, M. R.; LIMA, S. A.; MISHU, S. Z. Performance analysis of different neural networks for sentiment analysis on imdb movie reviews. In: IEEE. **2019 3rd International Conference on Electrical, Computer & Telecommunication Engineering (ICECTE)**. [S.l.], 2019. p. 161–164.

HATCHER, W. G.; YU, W. A survey of deep learning: Platforms, applications and emerging research trends. **IEEE Access**, IEEE, v. 6, p. 24411–24432, 2018.

JOUPPI, N. P. et al. In-datacenter performance analysis of a tensor processing unit. In: **Proceedings of the 44th annual international symposium on computer architecture**. [S.l.: s.n.], 2017. p. 1–12.

KESKAR, N. S. et al. On large-batch training for deep learning: Generalization gap and sharp minima. **arXiv preprint arXiv:1609.04836**, 2016.

KIM, S.; CHUN, J.; DEY, A. K. Sensors know when to interrupt you in the car: Detecting driver interruptibility through monitoring of peripheral interactions. In: **Proceedings of the 33rd Annual ACM Conference on Human Factors in Computing Systems**. [S.l.: s.n.], 2015. p. 487–496.

KIM, T. K. T test as a parametric statistic. **Korean journal of anesthesiology**, Korean Society of Anesthesiologists, v. 68, n. 6, p. 540, 2015.

KIMM, H.; PAIK, I.; KIMM, H. Performance comparision of tpu, gpu, cpu on google colaboratory over distributed deep learning. In: IEEE. **2021 IEEE 14th International Symposium on Embedded Multicore/Many-core Systems-on-Chip (MCSoC)**. [S.l.], 2021. p. 312–319.

KÜNAS, C. A. et al. Computaçao de borda versus computaçao em nuvem: Impacto do pré-processamento de imagens de retinas. In: SBC. **Anais do XXIII Simpósio em Sistemas Computacionais de Alto Desempenho**. [S.l.], 2022. p. 85–96.

KÜNAS, C. A. et al. Offloading the training of an i/o access pattern detector to the cloud. In: IEEE. **2021 International Symposium on Computer Architecture and High Performance Computing Workshops (SBAC-PADW)**. [S.l.], 2021. p. 15–19.

KÜNAS, C. A. et al. Improving performance of long short-term memory networks for sentiment analysis using multicore and gpu architectures. In: SPRINGER. **High Performance Computing: 8th Latin American Conference, CARLA 2021, Guadalajara, Mexico, October 6–8, 2021, Revised Selected Papers**. [S.l.], 2022. p. 34–47.

LIM, R. **Accelerating Machine Learning via Multi-Objective Optimization**. Thesis (PhD) — University of Oregon, 2021.

LIN, G.-M. et al. Transforming retinal photographs to entropy images in deep learning to improve automated detection for diabetic retinopathy. **Journal of ophthalmology**, Hindawi, v. 2018, 2018.

MAAS, A. L. et al. Learning word vectors for sentiment analysis. In: ASSOCIATION FOR COMPUTATIONAL LINGUISTICS. **Proceedings of the 49th annual meeting of the association for computational linguistics: Human language technologies-volume 1**. [S.l.], 2011. p. 142–150.

MALISZEWSKI, A. M. et al. Performance and cost-aware hpc in clouds: A network interconnection assessment. In: **2020 IEEE Symposium on Computers and Communications (ISCC)**. [S.l.: s.n.], 2020. p. 1–6.

MALTA, E. M. **Selecting efficient virtual machines for training deep learning models on the cloud: Seleção de máquinas virtuais eficientes para o treinamento de modelos de aprendizado profundo na nuvem**. Dissertation (Master) — Universidade Estadual de Campinas, Instituto de Computação, Campinas, SP, 2021.

MALTA, E. M.; AVILA, S.; BORIN, E. Exploring the cost-benefit of aws ec2 gpu instances for deep learning applications. In: **Proceedings of the 12th IEEE/ACM International Conference on Utility and Cloud Computing**. [S.l.: s.n.], 2019. p. 21–29.

MAYER, R.; JACOBSEN, H.-A. Scalable deep learning on distributed infrastructures: Challenges, techniques, and tools. **ACM Computing Surveys (CSUR)**, ACM New York, NY, USA, v. 53, n. 1, p. 1–37, 2020.

MCCULLOCH, W. S.; PITTS, W. A logical calculus of the ideas immanent in nervous activity. **The bulletin of mathematical biophysics**, Springer, v. 5, p. 115–133, 1943.

MELL, P.; GRANCE, T. **The NIST definition of cloud computing**. 2011. (NIST Special Publication 800-145).

MOHAMMADIAN, S.; KARSAZ, A.; ROSHAN, Y. M. Comparative study of fine-tuning of pre-trained convolutional neural networks for diabetic retinopathy screening. In: IEEE. **2017 24th National and 2nd International Iranian Conference on Biomedical Engineering (ICBME)**. [S.l.], 2017. p. 1–6.

MOOLAYIL, J. An introduction to deep learning and keras. In: **Learn Keras for Deep Neural Networks**. [S.l.]: Springer, 2019. p. 1–16.

MUTINDA, J.; MWANGI, W.; OKEYO, G. Sentiment analysis of text reviews using lexicon-enhanced bert embedding (lebert) model with convolutional neural network. **Applied Sciences**, MDPI, v. 13, n. 3, p. 1445, 2023.

OVTCHAROV, K. et al. Accelerating deep convolutional neural networks using specialized hardware. **Microsoft Research Whitepaper**, Citeseer, v. 2, n. 11, p. 1–4, 2015.

PAKDEL, R.; HERBERT, J. Adaptive cost efficient framework for cloud-based machine learning. In: IEEE. **2017 IEEE 41st Annual Computer Software and Applications Conference (COMPSAC)**. [S.l.], 2017. v. 2, p. 155–160.

PEDREGOSA, F. et al. Scikit-learn: Machine learning in python. **the Journal of machine Learning research**, JMLR. org, v. 12, p. 2825–2830, 2011.

PINTO, C. et al. Hoard: A distributed data caching system to accelerate deep learning training on the cloud. **arXiv preprint arXiv:1812.00669**, 2018.

QAISAR, S. M. Sentiment analysis of imdb movie reviews using long short-term memory. In: IEEE. **2020 2nd International Conference on Computer and Information Sciences (ICCIS)**. [S.l.], 2020. p. 1–4.

ROLOFF, E. **Viability and performance of high-performance computing in the cloud**. Dissertation (Master) — Universidade Federal do Rio Grande do Sul, Instituto de Informática, Porto Alegre, RS, 2013.

ROSSUM, G. V.; DRAKE, F. L. **The python language reference manual**. [S.l.]: Network Theory Ltd., 2011.

SERVICES, A. W. **Amazon SageMaker**. 2022. Available from Internet: <https://aws.amazon.com/sagemaker/resources/>. Accessed in: Nov. 09, 2022.

SHARMA, V.; GUPTA, G. K.; GUPTA, M. Performance benchmarking of gpu and tpu on google colaboratory for convolutional neural network. In: SPRINGER. **Applications of Artificial Intelligence in Engineering: Proceedings of First Global Conference on Artificial Intelligence and Applications (GCAIA 2020)**. [S.l.], 2021. p. 639–646.

SPEARMAN, C. The proof and measurement of association between two things. **The American Journal of Psychology**, v. 15, p. 72–101, 1904.

SUBRAMANIAN, R. R. et al. A survey on sentiment analysis. In: IEEE. **2021 11th International Conference on Cloud Computing, Data Science & Engineering (Confluence)**. [S.l.], 2021. p. 70–75.

TIRUMALA, A. et al. Iperf: the tcp/udp bandwidth measurement tool (2005). **URL: http://iperf. sourceforge. net**, 2005.

VIELMA, C.; VERMA, A.; BEIN, D. Single and multibranch cnn-bidirectional lstm for imdb sentiment analysis. In: SPRINGER. **17th International Conference on Information Technology–New Generations (ITNG 2020)**. [S.l.], 2020. p. 401–406.

VOETS, M.; MØLLERSEN, K.; BONGO, L. A. Reproduction study using public data of: Development and validation of a deep learning algorithm for detection of diabetic retinopathy in retinal fundus photographs. **PloS one**, Public Library of Science San Francisco, CA USA, v. 14, n. 6, p. e0217541, 2019.

WANG, W. et al. Singa: Putting deep learning in the hands of multimedia users. In: **Proceedings of the 23rd ACM international conference on Multimedia**. [S.l.: s.n.], 2015. p. 25–34.

WANG, Y. E.; WEI, G.-Y.; BROOKS, D. Benchmarking tpu, gpu, and cpu platforms for deep learning. **arXiv preprint arXiv:1907.10701**, 2019.

WANG, Z. et al. Various frameworks and libraries of machine learning and deep learning: a survey. **Archives of computational methods in engineering**, Springer, p. 1–24, 2019.

WIDAYAT, W.; ADJI, T. B. et al. The effect of embedding dimension reduction on increasing lstm performance for sentiment analysis. In: IEEE. **2018 International Seminar on Research of Information Technology and Intelligent Systems (ISRITI)**. [S.l.], 2018. p. 287–292.

WONGPANICH, A. et al. Training efficientnets at supercomputer scale: 83% imagenet top-1 accuracy in one hour. In: IEEE. **2021 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)**. [S.l.], 2021. p. 947–950.

YENTER, A.; VERMA, A. Deep cnn-lstm with combined kernels from multiple branches for imdb review sentiment analysis. In: IEEE. **2017 IEEE 8th Annual Ubiquitous Computing, Electronics and Mobile Communication Conference (UEMCON)**. [S.l.], 2017. p. 540–546.

YEO, I.-K.; JOHNSON, R. A. A new family of power transformations to improve normality or symmetry. **Biometrika**, Oxford University Press, v. 87, n. 4, p. 954–959, 2000.

YING, C. et al. Image classification at supercomputer scale. **arXiv preprint arXiv:1811.06992**, 2018.

YOU, Y. et al. Large batch optimization for deep learning: Training bert in 76 minutes. **arXiv preprint arXiv:1904.00962**, 2019.

YOU, Y. et al. Imagenet training in minutes. In: **Proceedings of the 47th International Conference on Parallel Processing**. [S.l.: s.n.], 2018. p. 1–10.

YOU, Y. et al. Fast deep neural network training on distributed systems and cloud tpus. **IEEE Transactions on Parallel and Distributed Systems**, IEEE, v. 30, n. 11, p. 2449–2462, 2019.

APÊNDICE A — RESUMO EM PORTUGUÊS

In this chapter, we present a summary of this master thesis in the portuguese language, as required by the PPGC Graduate Program in Computing.

Neste capítulo, é apresentado um resumo desta dissertação de mestrado na língua portuguesa, como requerido pelo Programa de Pós-Graduação em Computação.

A.1 Introdução

As aplicações científicas de vários domínios de pesquisa (por exemplo, ciências da saúde, química, física, petróleo, clima) impõem requisitos de desempenho cada vez maiores para o campo da computação de alto desempenho (HPC) (MALISZEWSKI et al., 2020). Além disso, novas cargas de trabalho estão entrando nas instalações de HPC, de Big Data a aplicativos de Aprendizado de Máquina (ML), tornando os sistemas cada vez mais complexos. Esses requisitos justificam atualizações contínuas e implantação de novas plataformas de grande escala. À medida que a complexidade desses sistemas tende a crescer, também aumenta o número de parâmetros e fatores que podem afetar direta ou indiretamente o desempenho. Os modelos de Deep Learning (DL) têm sido cada vez mais usados para resolver problemas complexos. Porém, a complexidade imposta às aplicações acaba gerando overhead para a máquina local, que por sua vez tem seus recursos computacionais disputados por aplicações científicas e pelo treinamento ou atualização destes modelos. Com o aumento da complexidade e da quantidade de dados, o treinamento destes modelos tem exigido sistemas de computação cada vez mais poderosos com altos custos de aquisição e manutenção.

Uma alternativa natural para reduzir estes custos é o uso de serviços e/ou recursos computacionais na nuvem, um modelo de negócio que permite o acesso a diversos sistemas de informática, inclusive sistemas de alto desempenho, mediante pagamento por uso, sem que o usuário tenha que arcar com os custos de aquisição do equipamento (ROLOFF, 2013). No entanto, escolher o sistema de computador mais adequado para treinar um modelo de DL na nuvem é um processo trabalhoso. A escolha deve considerar fatores como tempo e custo de execução (PAKDEL; HERBERT, 2017). Diferentemente dos sistemas tradicionais, a Computação em Nuvem não requer investimentos iniciais em infraestrutura e licenças de software. Devido à elasticidade e ao modelo de faturamento de pagamento por uso, o custo total de manutenção pode ser próximo a zero quando os

recursos não estão em uso. Ademais, os custos de instalações, depreciação de hardware, consumo de energia e resfriamento são eliminados ou, pelo menos, bastante reduzidos. Ainda, a quantidade de recursos disponíveis é virtualmente ilimitada. Portanto, a motivação de mover aplicações para a nuvem é reduzir estes custos e, ao mesmo tempo, aumentar a escalabilidade e a disponibilidade.

Além disso, provedores de serviço de Nuvem Computacional atualizam frequentemente seus parques computacionais com novos aceleradores, como GPUs e TPUs, o que viabiliza o treinamento de modelos cada vez maiores e com custos menores à medida que a tecnologia avança. Este tipo de benefício é difícil de se obter com parques computacionais locais já que estes equipamentos são muito custosos e a renovação do parque computacional em pequenas empresas e universidades não ocorre com a frequência que novas versões destes equipamentos são lançadas. Assim, nosso trabalho visa a execução do treinamento de algoritmos de ML/DL na Nuvem, uma das tarefas que mais consome recursos computacionais e que pode demorar dias, evitando sobrecarregar o sistema local, que muitas vezes está altamente sobrecarregado por muitas aplicações científicas em execução[1].

A.1.1 Contribuições

O principal objetivo de nossa pesquisa é otimizar a execução do treinamento de algoritmos de Machine Learning (ML) e Deep Learning (DL) usando recursos de computação de alto desempenho na nuvem. Para isso, nossas principais contribuições são:

- Adaptamos e migramos aplicações DL para explorar recursos de Cloud Computing;
- Otimizamos a execução destas aplicações na Cloud utilizando Tensor Processing Units (TPUs);
- Estudamos o desempenho, a precisão e o custo do uso de TPUs na nuvem, identificando desafios.

A.2 Metodologia

Após o grande sucesso na resolução de problemas de classificação de imagens, a aplicação de técnicas de aprendizado de máquina ganhou velocidade e se espalhou para

[1] Projetos em andamento no Supercomputador SDumont: https://sdumont.lncc.br/projects_statistics.php

outras áreas. O treinamento de modelos maaiores e mais complexos, hardwares aceleradores tornam-se ferramentas essenciais. Com altos custos de aquisição de plataformas de alto desempenho, uma alternativa é a utilização de recursos de computação em nuvem. Neste paradigma, o usuário paga apenas pelo uso, eliminando custos de aquisição de equipamentos. Nesse contexto, nosso trabalho visa descarregar o treinamento de algoritmos ML/DL para a nuvem para evitar a sobrecarga do sistema local.

Para isso, adaptamos e migramos três aplicativos reais para a nuvem. Estudamos cada aplicativo cuidadosamente e identificamos as seções de código que precisavam ser modificadas para serem executadas na TPU. A intenção é adaptar o modelo o máximo possível, mas manter sua estrutura original, ou seja, sem adicionar ou remover camadas ou modificar hiperparâmetros. Portanto, as alterações devem ser oportunas e específicas para que o algoritmo possa se conectar ao dispositivo e permitir o treinamento distribuído em todos os núcleos de TPU disponíveis.

Para cada aplicação, coletamos métricas de desempenho e as comparamos com a linha de base. O desempenho é medido pelo tempo total de treinamento em segundos. Métricas de acurácia também foram analisadas para validar a migração da aplicação, ou seja, se o modelo apresentar índices de acurácia semelhantes aos encontrados na linha de base ou mesmo na literatura, a migração do modelo foi bem-sucedida.

Para analisar a relação custo-benefício do uso do Cloud TPU para treinamento de modelo dimensionamos o valor do desempenho com o preço por hora. O custo por hora de uma TPUv3 é de USD 8.00. O valor para o cluster foi calculado da seguinte forma. Consideramos o custo de hardware de uma máquina de $\$25.000$ e que a máquina será usada por um ano, então chegamos a um custo de hardware por hora de $\$2,85$.

A.3 Resultados e Conclusão

Nesta tese, procuramos avaliar o desempenho e a precisão de três aplicações descarregando de forma assíncrona o treinamento para a nuvem usando dispositivos TPU. Essa abordagem ajuda a aliviar a contenção de recursos locais de alta demanda de HPC, permitindo que eles se concentrem na execução de aplicativos. Observamos resultados promissores modificando os modelos de rede neural para treinamento em dispositivos TPU. Também avaliamos o custo-benefício da execução de tais aplicações nesses dispositivos.

O classificador de padrão de acesso de E/S (NN-TWINS) apresentou precisão

acima de 99% para os conjuntos de dados de treinamento e teste nos três ambientes computacionais que exploramos. O tempo médio de treinamento desse modelo no TPUv3 foi $\approx 1,68\times$ mais rápido do que no P100. A execução de NN-TWINS em dispositivos Cloud TPU, embora seja 41% mais cara que a execução local, ainda é uma solução viável porque a fase de (re)treinamento pode demorar mais para ser concluída e pode ser feita de forma assíncrona em segundo plano. Além disso, é importante lembrar que a quantidade de dados coletados para treinar este modelo será maior em um ambiente de produção. O modelo de Análise de Sentimentos (RNN-SA) apresentou precisão média de $\approx 97\%$ para o treinamento e $\approx 89\%$ para os conjuntos de dados de teste nos três ambientes computacionais que exploramos. O desempenho do TPUv3 foi $\approx 5,74\times$ mais rápido do que no P100. Em comparação com o cluster local, a nuvem é cerca de 51% mais eficiente executando a mesma quantidade de trabalho para este aplicação.

Para o modelo de detecção de retinopatia diabética (IC-DR), o TPUv3 demonstrou uma melhoria de $5,63\times$ em termos de tempo médio de treinamento em comparação com o P100. Esse ganho é alcançado sem nenhuma otimização de código. Além disso, embora haja um tempo gasto para transferir o conjunto de dados para a nuvem para executar esse modelo no dispositivo TPU, o desempenho do TPUv3 ainda é bastante considerável, cerca de $4,95\times$ mais efetivo. O tamanho dos dados pode influenciar negativamente no tempo de execução da aplicação e congestionar a interligação da rede. Assim, o pré-processamento de dados no Edge pode contribuir para a economia de largura de banda. Estudamos esse desafio na Seção 5, onde reduzimos consideravelmente o tamanho total dos dados. A acurácia do modelo de detecção da retinopatia diabética, que é de 85%, é considerada adequada. Isso é apoiado por outros estudos que relataram taxas de acurácia comparáveis. Deve-se enfatizar que a detecção oportuna da retinopatia diabética é essencial para prevenir complicações graves relacionadas à visão.

Nossos resultados fornecem um bom ponto de partida para aqueles interessados em melhorar o desempenho de seus modelos de aprendizado profundo.