# Scaling Deep Learning Workloads. Applications in Computer Vision and Seismology


**Leonel Cruz De La Cruz**

Supervisor: Dr. Ruben Tous
Dra. Beatriz Otero

Department of Computer Architecture
Universitat Politècnica de Catalunya

This dissertation is submitted for the degree of
*Doctor of Philosophy*

Universitat Politècnica de Catalunya April 2023

# Abstract

Deep learning techniques have an enormous impact on the state-of-the-art in many fields, such as computer vision, natural language processing, audio analysis and synthesis, and many others. The increasing computing power and the increasing amount of available data, and the algorithms' evolution are fostering this impact.

This thesis addresses the general goal of designing deep learning methods that are able to leverage the increasing computational resources and data. On the one hand, the thesis studies how deep learning workloads can be distributed over a High Performance Computing (HPC) infrastructure. A technology stack based on Apache Spark is deployed and evaluated on the MareNostrum supercomputer. In order to evaluate the performance and scalability of the proposed stack, different workloads and different deployment setups are tested. The goal is to provide insights into how the job configuration on a traditional HPC setup can be optimized to run this kind of workload efficiently. In addition to the performance evaluation, a use case related to the training deep CNNs for annotating and filtering images from social media is also explored. On the other hand, this thesis applies deep learning techniques to the problems of earthquake detection and location. Single-station 3-channel seismic waveforms are processed with convolutional neural networks trained over a medium-sized computer cluster. New data preprocessing techniques and new network architectures are proposed. Also, a new dataset, the first for the study region, is made public for reproducibility and benchmarking purposes. The proposed methods outperform the State of the Art methods for the target seismicity and show good generalization properties when applied to data from other areas.

# Table of contents

# Chapter 1

# Introduction

## 1.1 Challenge

These days, the use of Deep Learning (DL) techniques has generated a transformation in the state of the art in different areas of knowledge, such as computer vision, natural language processing, audio analysis and synthesis, and many others. It has become a boiling hot area of research due to the increasing computing power, the increasing amount of available data, and the evolution of algorithms.

On the one hand, this thesis applies DL techniques to large parallel systems to train and validate Neural Networks (NN) models for different applications. First, this work employs technology stacks to enable the distribution of deep learning workloads on a traditional High Performance Computing (HPC) setup, such as the MareNostrum supercomputer. Enabling the distributed training of deep neural networks on a large parallel system is a complex process involving integrating and configuring several layers of general-purpose and custom software. This thesis contributes to the design of such a technology stack based on Apache Spark and develops DL models to process image datasets retrieved from social media with the purpose of image classification and object recognition.

On the other hand, this thesis applies DL techniques to the problems of earthquake detection and location. In past decades, the number of seismic networks and monitoring sensors has experienced a significant increase, and the continuously growing seismic records call for efficient processing algorithms to ensure that seismic catalogs contain complete and accurate information. These catalogs are used in hazard analyses where the probability of earthquake occurrence under a given magnitude is quantified. The thesis applies convolutional neural networks to the processing of single-station 3-channel seismic waveforms. New data preprocessing techniques and new network architectures are proposed. A framework is built to rank different NN architectures and hyperparameterization according to their detection

and location performance on a given seismic dataset. Also, a new dataset, the first for the study region, is made public for reproducibility and benchmarking purposes.

## 1.2 Contribution

This thesis addresses the general goal of designing deep learning methods that are able to leverage the increasing computational resources and data. More specifically, the thesis studies how deep learning workloads can be distributed over a HPC infrastructure such as the MareNostrum supercomputer, and also how deep neural networks (DNNs) can be applied over the increasing volume of seismic data to address some problems in seismology such as earthquake detection. The thesis is divided into two main contributions:

- C1: Design, implement and evaluate a strategy to execute distributed DL workloads over an HPC infrastructure.
    1. Deployment and evaluation of a technology stack based on Apache Spark to distribute DL workloads on a real-world, petascale, HPC setup, the MareNostrum supercomputer.
    2. Study of a use case, the automatic annotation of social media images. The use case involves training DL models for image classification and object recognition.
- C2: Study new DL models to approach the earthquake detection and epicentral region estimation problems.
    1. With the data provided by The Venezuelan Foundation for Seismological Research (FUNVISIS), build a new dataset of seismic data to train and evaluate DL models for earthquake detection and source region estimation in North Central Venezuela.
    2. Design, train an evaluate a new DNN to approach the earthquake's P-wave detection and source region estimation problem by processing three-channel seismic signals. The resulting network will be applied over the new dataset of seismic data from North Central Venezuela and will be applied to facilitate the revision and generation of the FUNVISIS seismic catalogs.

## 1.3 List of Publications

- Leonel Cruz, Ruben Tous and Beatriz Otero. Distributed Training of Deep Neural Networks with Spark: The MareNostrum Experience. Pattern Recognition Letters, vol.

125, 2019, pp. 174-178. ISSN 0167-8655. DOI:10.1016/j.patrec.2019.01.020. Impact factor JCR = 1.952, Quartile = Q2 (JCR 2017).

- Ruben Tous, Mauro Gomez, Jonatan Poveda, Leonel Cruz, Otto Wust, Mouna Makni, Eduard Ayguadé. Automated curation of brand-related social media images with deep learning. Multimedia Tools and Applications, vol. 77, issue 20, pp. 27123–27142, 2018. DOI: 10.1007/s11042-018-5910-z. Impact factor JCR = 1.541, Quartile = Q2 (JCR 2017).

- Oscar Orti, Ruben Tous, Mauro Gomez, Jonatan Poveda, Leonel Cruz and Otto Wust. Real-time Logo Detection in Brand-related Social Media Images. Proceedings of the 15th International Work-Conference on Artificial Neural Networks (IWANN 2019). Lecture Notes in Computer Science, vol. 11507. DOI: 10.1007/978-3-030-20518-8_11. Conference CORE B.

- Ruben Tous, Leonardo Alvarado, Beatriz Otero, Leonel Cruz, Otilio Rojas. Deep Neural Networks for Earthquake Detection and Source Region Estimation in North Central Venezuela. Bulletin of the Seismological Society of America, vol. 110, issue 5, pp. 2519-2529, 2020. DOI: 10.1785/0120190172. Impact factor JCR = 2.910, Quartile = Q2 (JCR 2020).

- Sergi Mus, Norma Gutiérrez, Ruben Tous, Beatriz Otero, Leonel Cruz, David Llácer, Leonardo Alvarado, Otilio Rojas. Long Short-Term Memory Networks for Earthquake Detection in Venezuelan Regions. A: International Conference on Machine Learning, Optimization, and Data Science. "Machine Learning, Optimization, and Data Science, 5th International Conference, LOD 2019: Siena, Italy, September 10-13, 2019". Lecture Notes in Computer Science, vol. 11943, pp. 751-754. DOI:10.1007/978-3-030-37599-7_62. Best Computer Science Conferences for Machine Learning Artificial intelligence by Research.com.

- Leonel Cruz, Ruben Tous, Beatriz Otero, Leonardo Alvarado and Otilio Rojas. Epicentral Region Estimation Using Convolutional Neural Networks. International Conference on Machine Learning, Optimization, and Data Science. "Machine Learning, Optimization, and Data Science, 7th International Conference, LOD 2021: Grasmere, UK, October 4–8, 2021". Revised Selected Papers, Part I, Lecture Notes in Computer Science, vol. 13163, pp. 541-552, 2021. DOI:10.1007/978-3-030-95467-3_39. Best Computer Science Conferences for Machine Learning Artificial intelligence by Research.com.

## 1.4 Dissertation Outline

Table 1.1 shows the structure of the work of this thesis and summarises in an abbreviated form the contributions made in each chapter. We have classified our contributions according to the type of problem addressed in two parts, emphasising the Deep Learning methodology used.

Table 1.1 Thesis structure.

| | | |
|---|---|---|
| **Part I** Exploring the distributed Paradigm for Deep Learning Algorithms. | **Chapter 3.1** Distributed training of CNN with spark [C1.1] | **Chapter 3.2** Digital marketing a case study[C1.2] |
| **Part II** Exploring Seismic Data with Deep Neural Networks | **Chapter 4.1** Deep neural network for earthquake detection[C2.1] | **Chapter 4.2** Epicentral region estimation using deep neural networks[C2.2] |

Part I consists of Chapter 3 and its contributions divided into two subchapters, Chapter 3.1 shows how to exploit the distributed paradigm for deep learning algorithms on a petascale supercomputer, by deploying a stack of technologies to enable deep learning workloads on Marenostrum. These components of a layered architecture, based on the usage of Apache Spark, are described and the performance and scalability of the resulting system is evaluated. Chapter 3.2 presents a work consisting in using deep CNNs to facilitate the curation of brand-related social media images. The images are captured in real time and automatically annotated with multiple CNNs. In order to speed-up the training of custom CNNs we applied a transfer learning strategy, having as a final goal to facilitate searching and discovering user-generated content(UGC) with potential value for digital marketing tasks.

Part II, chapter 4. Moves the focus to use the deep learning techniques used in the first part of the research towards a practical application in the growing area of seismology studies. Taking account the number of seismic networks and monitoring sensors have steadily increased in recent years, and the continuous growth of seismic records have opened up the opportunity to explore new processing algorithms that assist in solving problems in seismology. Chapter 4.1 presents the results of applying a deep convolutional neural network, called UPC-UCV, over single-station three-channel signal windows for P-wave earthquake detection and source region estimation in north central Venezuela. Chapter 4.2 extends the

previous work using a relaxed UPC-UCV model; the epicentral region estimation problem is explored in more detail here. This part evaluates the hypothesis that the source region estimation accuracy is significantly increased if the geographical partitioning is performed considering the regional geological characteristics such as the tectonic plate boundaries. Also, it raises the transformation of the training data to increase the accuracy of the predictive model based on a Projected Coordinate Reference (PCR) System.

# Chapter 2

# Background

## 2.1 Artificial Neural Networks

Artificial neural networks (ANNs) [27][127] are non-linear mapping structures that draw inspiration from the human brain; ANNs are also called Feed forward neural networks (FFNNs) or Multilayer Perceptrons (MLP) [80]. Basically, ANNs can be defined as a direct graph where information travels forward (see figure 2.1).

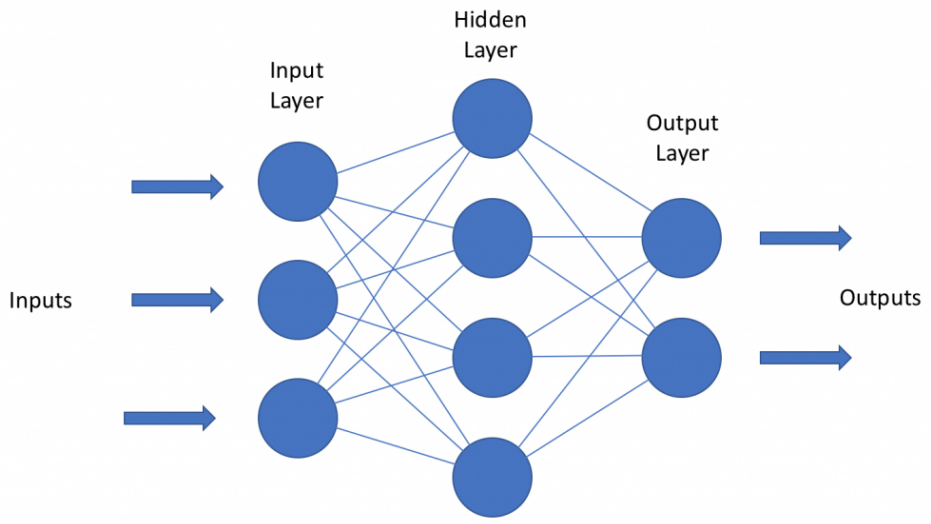


Fig. 2.1 Composition of an ANN represented as a direct graph. *Adapted from [6].*

The ANNs are composed of at least three layers interconnected, defined as an input layer, a hidden layer, and an output layer. Within each layer, we will find artificial neurons (AN) or perceptrons that make up each.

The work presented in [81] modelled the AN as a binary threshold unit, with two possible states: active or inactive; these receive inputs from other neurons and essentially contain a nonlinear function that processes them and produces outputs (see figure 2.2).

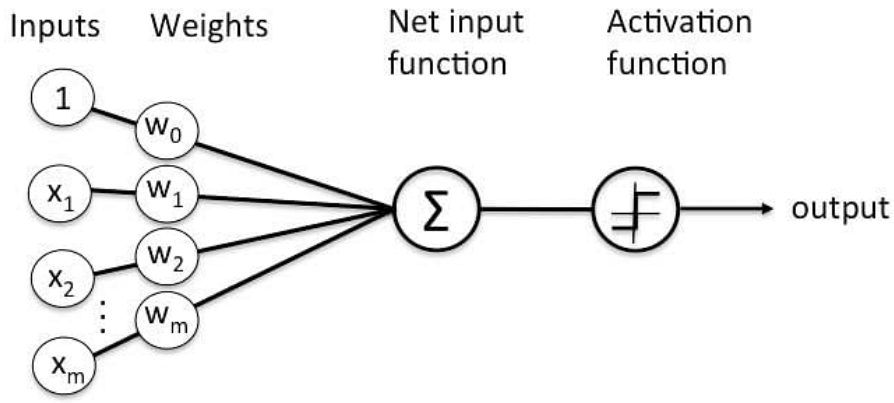


Fig. 2.2 The structure of an AN. *Adapted from [4].*

Each input is associated with a weight that will determine its importance; the output is the summation of the product of Xi and its weight Wi and b is the threshold value o bias. This scalar function is mathematically represented by the equation 2.1

$$y = b + \sum_{i=1}^{n} (WiXi) \tag{2.1}$$

Once the input value is calculated, the processing element then uses a transfer function or also known as activation function to produce its output. The transfer function transforms the AN's input value to generate the output signal. Typically this transformation involves the use of a sigmoid, hyperbolic-tangent, rectified linear units or other nonlinear function 2.2.

$$sgn(y) = \begin{cases} -1, & y < 0, \\ +1, & y \geqslant 0. \end{cases} \tag{2.2}$$

The main purpose of this nonlinear function is to convert an input signal of a node in an ANN to an output signal. This output signal is used as input to the next layer in the stack. The activation function is important for an ANN to learn and make sense of complicated problems to deal with such as images, videos, audio, speech, etc [33].

An additional set of learning rules makes use of backpropagation [49], a process through which the ANN can adjust its output results by taking errors into account also called cost

function. Through backpropagation, each time the output is labeled as an error during the supervised training phase, the information is sent backward as can be seen in the figure 2.3. Each weight is updated proportionally to how much they were responsible for the error.

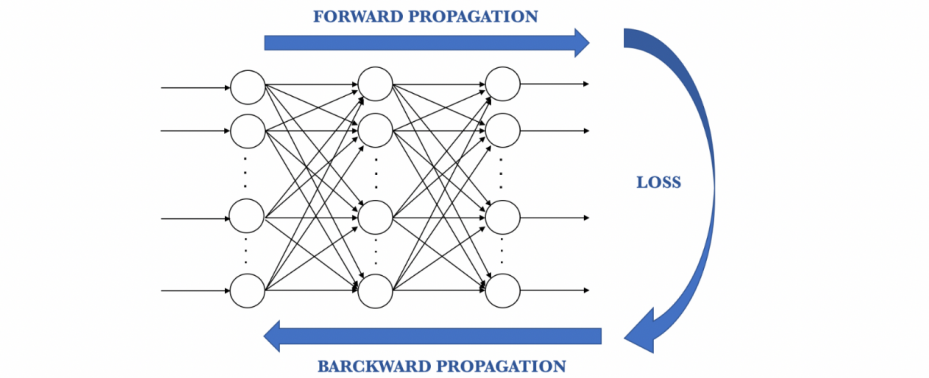


Fig. 2.3 Learning process of an ANN. *Adapted from [103].*

Therefore, the error is employed to re-calibrate the weight of the ANN's unit connections to consider the difference between the desired outcome and the actual one. In due course, the ANN will "learn" how to minimize the chance for errors (cost function) and undesirable results. This is achieved by running the ANN over and over again.

## 2.2 Convolutional Neural Networks

A Convolutional Neural Network (CNN) [66][78] is a class of ANN commonly used in image-related tasks but also in other problems such as, e.g. natural language processing [46] or speech recognition [28]. Due to their excellent performance, CNNs have become dominant in multiple tasks during the last decade. Besides fully-connected layers, CNNs incorporates convolutional layers and pooling layers. Because of their architecture, CNNs are able to automatically and adaptively learn spatial hierarchies of features.

In the figure 2.4, we can see an example of a CNN [66]. In the first steps, we can see a 1-channel image that is passed through a set of convolutional layers. Each convolutional layer consists on a set of filters (convolutional kernels) that are swiped over the layer input generating a set of feature maps. These feature maps detect or enhance features in the incoming data [70]. The kernels' parameters are automatically learned.

A problem with the output feature maps is that they depend on the location of the features in the input data. In order to make the results of convolutional layers more robust to changes in the position of the features (local translation invariance), typically a pooling layer is added after each convolutional layer. Pooling layers perform a dimensional reduction (subsampling)

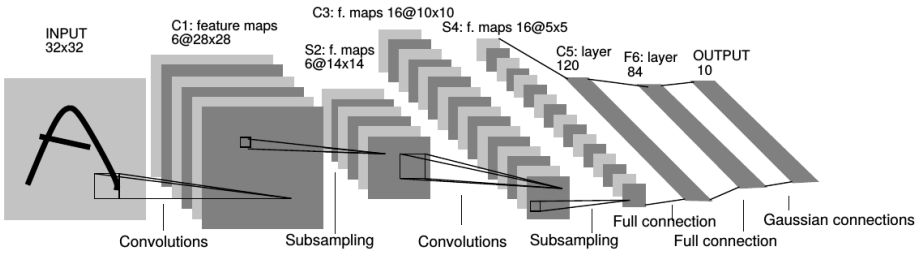


Fig. 2.4 Example of the composition of a CNN model. *Adapted from [66].*

summarizing the presence of features in patches of the feature map. Two common pooling methods are average pooling and max pooling. This process (convolution and pooling) is repeated multiple times. Whereas more layers are placed, the features obtained will be more representative. After the convolutional and pooling layers, some fully connected layers are usually added. Depending on the goal of the network (classification, regression, etc.) the last layers will end with, e.g., a softmax layer (for classification).

Although CNNs are widely known for their use with images, they have also shown benefits on other types of data, such as seismic signals. In the figure 2.5, we can see an example of a CNN architecture for processing seismic waves. Here, the input is a 3-channel seismic signal and the output are the probabilities that the signal is not an earthquake or is an earthquake whose epicenter is in one of the 6 given geographic regions (clusters).

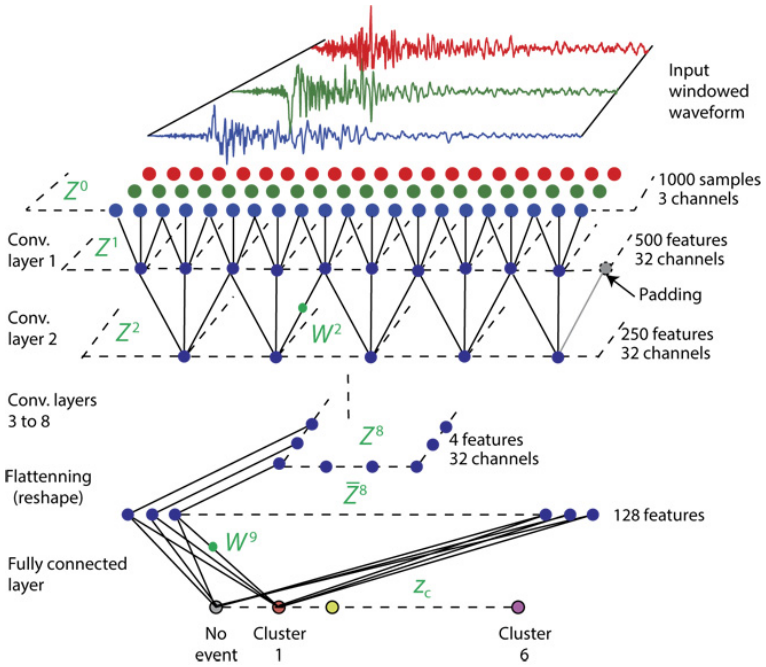


Fig. 2.5 CNN model used for processing seismic signals.

## 2.3 Metrics

All the NNs are evaluated using different metrics that ensure and test their efficiency. These metrics are the Accuracy (ACC), Loss, AUC, Precision and Recall. Firstly, the accuracy is represented in Equation 2.3 and defined as the True Positives (TP) plus the True Negatives (TN) divided by the sum of TP, TN, False Positives (FP) and False Negatives (FN):

$$Accuracy = \frac{TP+TN}{TP+TN+FP+FN} \tag{2.3}$$

Secondly, the most common loss is the binary cross-entropy loss which formula is represented in Equation 2.4. Note that the $y$ value represents the real output, whereas the $\hat{y}$ represents the output estimation:

$$Loss = -[y \cdot log(\hat{y}) + (1-y) \cdot log(1-\hat{y})] \tag{2.4}$$

Finally, the precision and the recall are metrics defined in Equations 2.5 and 2.6, respectively.

$$Precision = \frac{TP}{TP+FP} \tag{2.5}$$

$$Recall = \frac{TP}{TP+FN} \tag{2.6}$$

The metrics used, for these several target systems the false positives are as crucial as false negatives (it is not a medical diagnosing system) and accuracy was a convenient metric. Regarding the second metric, it is used to obtain the final model in the training process of the convolutional networks in each of the published works. The remaining ones are part of the parameters obtained to refine the selection of the models during each iteration.

## 2.4 Distributed CNNs

Nowadays, there are large amounts of data available thanks to social networks. These have greatly helped the rebirth of CNN, but have also represented a bottleneck for model processing and training. Even using GPUs, calculations can take days or weeks if only a single machine or node is used. Despite the increase in computational power in recent years, the use of parallelism techniques is increasingly necessary [56][105][118][119].

There are two paradigms for parallelizing the training of deep learning models. On the one hand, data parallelism, or across data dimension, where each machine or node contains

a complete replica of the model but only processes a part of the data. On the other hand, model parallelism, also called across model dimension, where different parts of the model are executed in distinct nodes in parallel.

In both approaches, some synchronization between workers is required. In model parallelism, neuron activities need to be communicated, while in data parallelism model parameters (weights and biases) are transmitted to ensure all models are trained evenly [62].

### 2.4.1 Data Parallelism

Data parallelism requires keeping a global model and some way of updating its parameters by gathering results from workers. The main algorithm to carry out the learning process is Stochastic Gradient Descent (SGD) [126], and to execute it in parallel, a global model is maintained. Workers send their gradients, which are aggregated, to the global model. There are two ways to update these gradients: synchronously (awaiting all the workers to finish before updating the model), and asynchronously (allowing model updates with just a part of nodes results).

Ultimately, the updated global model must be promoted to all workers. Therefore, techniques common in collective communication and HPC are highly relevant for model gradient update and propagation. One implementation of data parallelism is Parameter Server [95], which is efficient and scalable. This framework distributes data and model parameters across multiple nodes to disperse the workload.

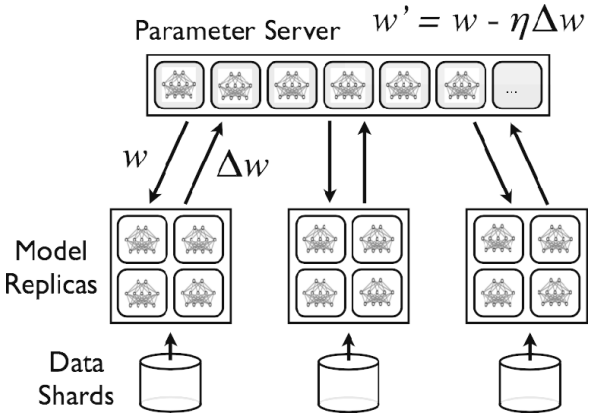


Fig. 2.6 Parameter server framework. *Adapted from [42].*

The implementation of the parameter server model is illustrated in figure 2.6, where the data is divided across nodes each containing a model replica. The parameter server collects model gradients ΔW from each machine and returns an aggregated model parameter W that is used to update every model replica.

### 2.4.2 Model Parallelism

In this paradigm the model is distributed or partitioned among the different nodes, synchronization between nodes is done when one node needs neuron activities output by another node as input. Communication overhead in NN model graph is mitigated by partitioning in a way that edges running between separated components (as shown in the figure 2.7) of the model are minimal or the amount of data that flows through the edges is low.

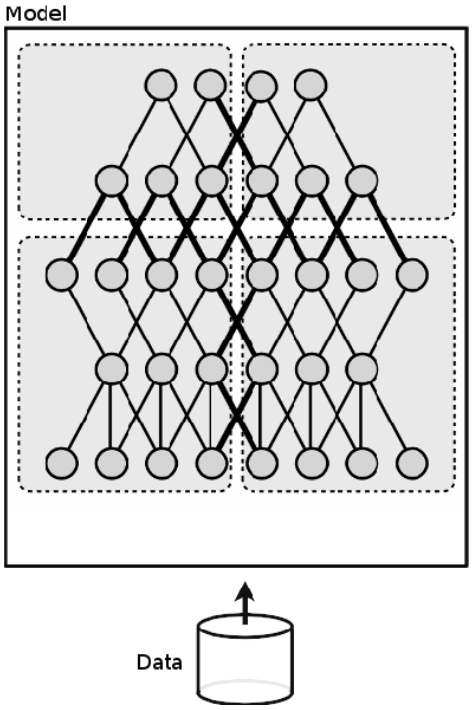


Fig. 2.7 Model parallelism. *Adapted from [42].*

A key advantage of the model-parallel approach is that it explicitly partitions the model parameters into subsets, allowing ML problems with massive model spaces to be tackled on nodes with limited memory, as for example would be applicable when using GPUs.

## 2.5 Earthquake Detection Techniques

A seismic wave is a wave of energy that travels through the Earth as shown in the figure 2.8. It can result from an earthquake but also from volcanic eruptions, avalanches, landslides, man-made explosions or other causes. Seismic waves that travel through the interior of the Earth are known as body weights, the ones that travel across the surface are known as surface waves. Body waves can be divided into primary (P-waves) [12] and secondary (S-waves)

[83]. P waves (see figure 2.9) can travel through any type of material, including fluids, and travel faster than the S waves (they arrive first to seismic stations).

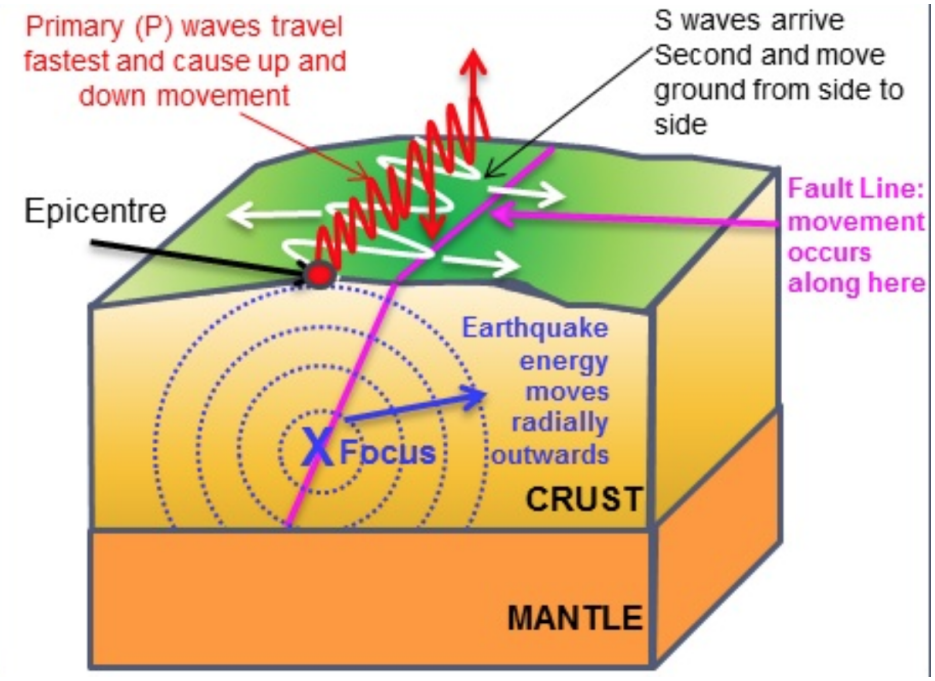


Fig. 2.8 Earthquake composition. *Adapted from [5].*

Seismometers continuously detect ground motions excited by seismic waves and record them into seismograms. A seismogram shows ground displacement on the y axis and time on the x axis. Modern seismograms include 3 channels (see figure 2.10), one for each movement direction: up-down, north-south, and east-west.

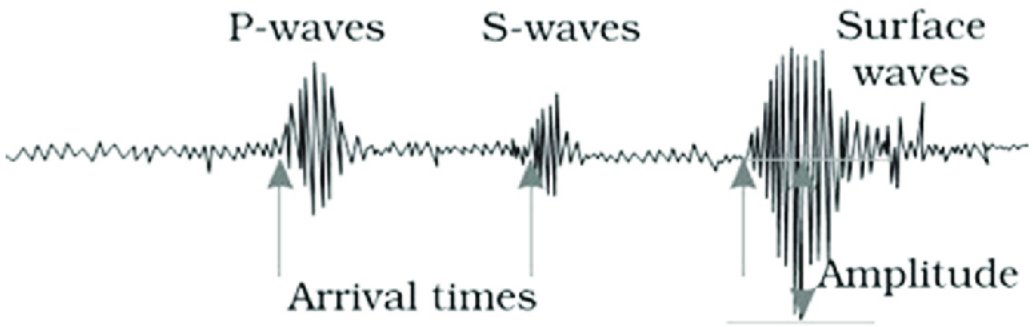


Fig. 2.9 Types of waves in a seismic signal. *Adapted from [100].*

Seismometers are located within seismic station. A number of interconnected seismic stations form a seismic network. Today, the data from all of the stations of a seismic network are transmitted (by radio, internet or satellite) in real-time to a data center, where the data are

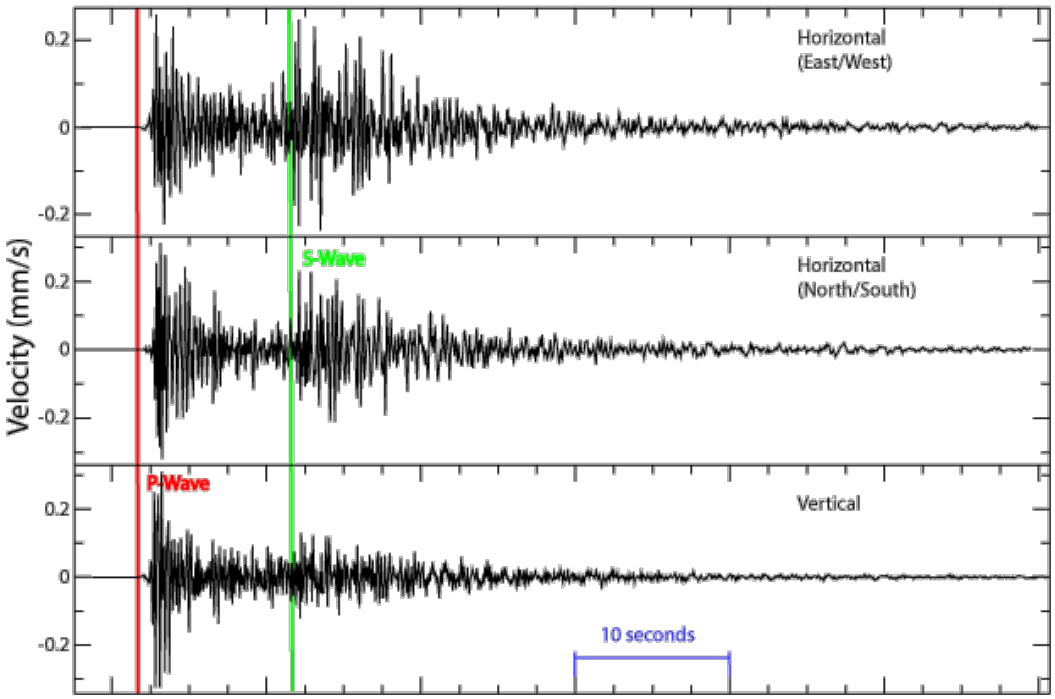


Fig. 2.10 Example seismograms

processed and an earthquake catalog is generated. An earthquake catalog is a description of seismic events, with each entry describing one earthquake. The description usually includes the location, origin time and magnitude of the event, and may include other information such as the focal mechanism. Some of the information of the catalogs are produced by human experts, which implies some limitations. On the one hand, as seismic monitoring equipment continue to spread and become more efficient, the amount of data surpasses the capabilities of human analysts. On the other hand, small-magnitude earthquakes (more frequent) sometimes pass undetected by trained analysts [91].

For these reasons, during the last four decades multiple research works have addressed the automation of interpretation tasks such as event detection, event identification, hypocenter location, and source mechanism analysis. Lately, recent advances are related to the application of ANNs to automate some of these interpretation tasks. Published results [87][101][124] show that these networks are able to learn the interpretation abilities of human analysts and improve the performance of traditional algorithms.

# Chapter 3

# Exploring The Distributed Paradigm for Deep Learning Algorithms.

## 3.1 Distributed Training of Deep Neural Networks with Spark: The MareNostrum Experience

### 3.1.1 Introduction

Over the past several years, DNNs have proven to be an incredibly effective tool for a variety of problems, from computer vision, speech recognition or natural language processing. Their number of parameters and complexity, and the size of the training datasets, have quickly grown, leading to be a first-class workload for HPC infrastructures. However, enabling deep learning workloads on a large parallel system is a very complex process, involving the integration and configuration of several layers of both, general-purpose and custom software. The details of such kind of deployments are rarely described in the literature.

The goal of the deployment is to be able to take profit of the computation resources provided by MareNostrum (almost 50K cores and more than 100TB of aggregated RAM) for training DNNs. Nowadays, the usage of GPUs has proven to be the more efficient alternative to train neural networks, speeding up common operations such as large matrix computations [67, 36]. As their price, performance and energy efficiency improves, GPUs are gaining ground in HPC (both in special-purpose systems and in hybrid general-purpose supercomputers). However, there are still many systems, such as MareNostrum, that are not equipped with GPUs.

The key element of the deployed layered architecture is Apache Spark [118]. In order to isolate machine-learning applications from the particularities of MareNostrum, Spark is

usually used as an intermediate layer (not only in MareNostrum, [111] does the same on a Cray X-series supercomputer). The deployment of Spark-enabled clusters over MareNostrum is not trivial, and it has been done with the help of Spark4MN [105], a custom interoperability layer. On top of this stack (Marenostrum, Spark4MN and Spark) we place a deep learning specific layer, DL4J. DL4J, that is written in Java and has a direct integration with Spark, enables distributed training of deep neural networks through a synchronous data parallelism method.

These four elements (DL4J, Spark, Spark4MN and MareNostrum) have been integrated enabling to efficiently train deep neural networks. Apart from the deployment details, the challenge is scalability and proper configuration. Simply running on many cores may yield poor benefits or even degraded performance due to overheads. We deal with this issue and we aim to make the first step towards systematic analysis of the several parameters and optimized configuration.

In order to evaluate the performance and scalability of the proposed software stack on MareNostrum, we have experimented with different workloads and different deployment setups (number of nodes, parallelism configuration, etc.). Through the following sections we explain the different components of the deployment in more detail. Then, we discuss the performed experiments and the obtained results, aiming to shed light onto the parameters that have the biggest impact and their effective configuration. We provide insights into how the job configuration on a traditional HPC setup can be optimized to efficiently run this kind of workloads. The derived conclusions should be useful to guide similarly complex deployments in the future.

### 3.1.2 Related Work

Several works have addressed the execution of deep learning workloads on large specific purpose clusters usually involving nodes equipped with GPUs. In [65], authors present a Caffe-based approach to execute deep learning workloads on a contemporary HPC system equipped with Xeon-Phi nodes. They use the Intel distribution of Caffe, that improves Caffe performance when running on CPUs. Authors report to be able, due to a hybrid approach, to overcome the limitations of synchronous systems scaling the training of a model up to thousands of nodes. In [116], authors describe another method (tested over Intel Knights Landing (KNL) clusters and multi-GPU clusters) with very good weak scaling efficiency (e.g. 92% for GoogleNet on 2176 cores with respect to a Intel Caffe baseline). Alternatively, distributed DNNs training can be deployed through an integrated software stack. Despite of the potential performance limitations, the possibility to take profit of thousands of underutilized cores to alleviate the pressing demand of computational resources

to train deep learning models with a solution with minimum cost and setup time is an option for many general-purpose HPC infrastructures, specially if they already provide the lower components of the stack. A common case are infrastructures with an Apache Spark abstraction layer. Enabling distributed DNNs training in these situations is straightforward through the integration of a DL4J layer. While the performance of Spark on HPC setups have been already studied by many works (e.g. [48][72][111]), as far as we know, there are no previous works evaluating the feasibility and scalability of a Spark-DL4J integrated solution when applied to an HPC setup. One potential limitation of DL4J is that its currently constrained to synchronous SGD-based training. The work described in [57] analyzes the main bottlenecks of the synchronous approach. The authors conclude that the issue is quickly turning into a vastly communication bound problem which is severely limiting the scalability in most practical scenarios.

### 3.1.3 Components for distributed training

#### Deep Neural Networks

DNNs are layered compositional models that enable learning representations of data with multiple levels of abstraction. State-of-the-art DNNs include many variants, specialized in different domains (CNNs, recurrent neural networks (RNNs), etc.). DNNs are usually trained by using iterative, gradient-based optimizers (typically mini-batch SGD) that drive a non-convex cost function to a local minimum. In every iteration step, we use information about the gradient $\nabla E$ at the current point. In iteration step $[t+1]$ the weight update $\Delta w[t]$ is determined by taking a step ($\gamma$ is the learning rate) into the direction of the negative gradient at position $w[t]$ such that (in the case of stochastic training):

$$\Delta w[t] = -\gamma \frac{\partial E_n}{\partial w[t]} \tag{3.1}$$

State-of-the-art networks have a huge number of weights $W$ and the core computation in their training is dominated by dense linear algebra. Usually, in order to improve the efficiency, the training dataset is split into mini-batches of size $B$ (typically chosen between 1 and a few hundreds) and the model is only updated (one iteration) after accumulating the gradients of all the training samples within a mini-batch.

DNNs training on a single node involves several software and hardware layers. At the top of the stack there is normally a deep learning framework such as DL4J, TensorFlow, Torch, etc. (there may be even an upper layer such as Keras). Below, the framework relies on an

underlying numerical library such as NVIDIA's cuDNN or Intel's MKL. Finally, the models are usually trained on NVIDIA GPUs or Intel's Xeon Phi processors.

When trained on multiple nodes, one can apply data parallelism (distributing training samples among nodes) and/or model parallelism (distributing model parameters among nodes). In our deployment, we only apply data parallelism. The $B$ training samples within a min-batch are split into $n$ equal sized sets of size $b$ (with $b = B/n$). The resulting mini-batch-splits are then fed to $n$ nodes holding a complete copy of the model. The results (gradients) off all nodes are then accumulated and used to update the model.

While DL4J limits us to perform this process synchronously (awaiting all the workers to finish before updating the model), it could be also performed asynchronously (allowing model updates with just a part of nodes results). Asynchronous data parallelism can potentially gain higher throughput, but depending on the infrastructure status we can have the *stale gradient problem*. By the time a slow worker has finished its calculations based on a given state of the model, the model may have been updated a number of times and the outdated update may have a negative impact. Some solutions to this problem (e.g. [76]) have been recently proposed.

#### DL4J

Deeplearning4j (DL4J) is a computing framework written for Java with wide support for deep learning algorithms. DL4J is powered by its own numerical computing library, ND4J, and provides distributed parallel versions (both for GPUs and CPUs) of the algorithms that integrate with Apache Hadoop and Spark. Through a C++ native library, Libnd4j (with a BLAS backend), ND4J provides intra-node parallelism for matrix operations (implemented with OpenMP vectorizable loops with SIMD support). With the help of JavaCPP and Java Native Interface (JNI), pointers to off-heap memory (allocated outside of the Java Virtual Machine (JVM) and not managed by the Garbage Collector (GC)) are passed to the underlying C++ code. In order to achieve distributed network training over Spark, DL4J performs a version of the synchronous data parallelism mechanism called parameter averaging. Instead of transferring gradients to the master, the nodes perform first a local model update and then they transfer the resulting weights to the master, where they are averaged. With respect to generic parameter averaging, in DL4J the Spark driver and reduction operations take the place of the parameter server (see figure 3.1).

There are several parameters that must be adjusted to optimize training time. These include, but are not limited to, mini-batch-split size, averaging frequency (too low averaging periods may imply too networking overhead), prefetching (how many mini-batch-splits a

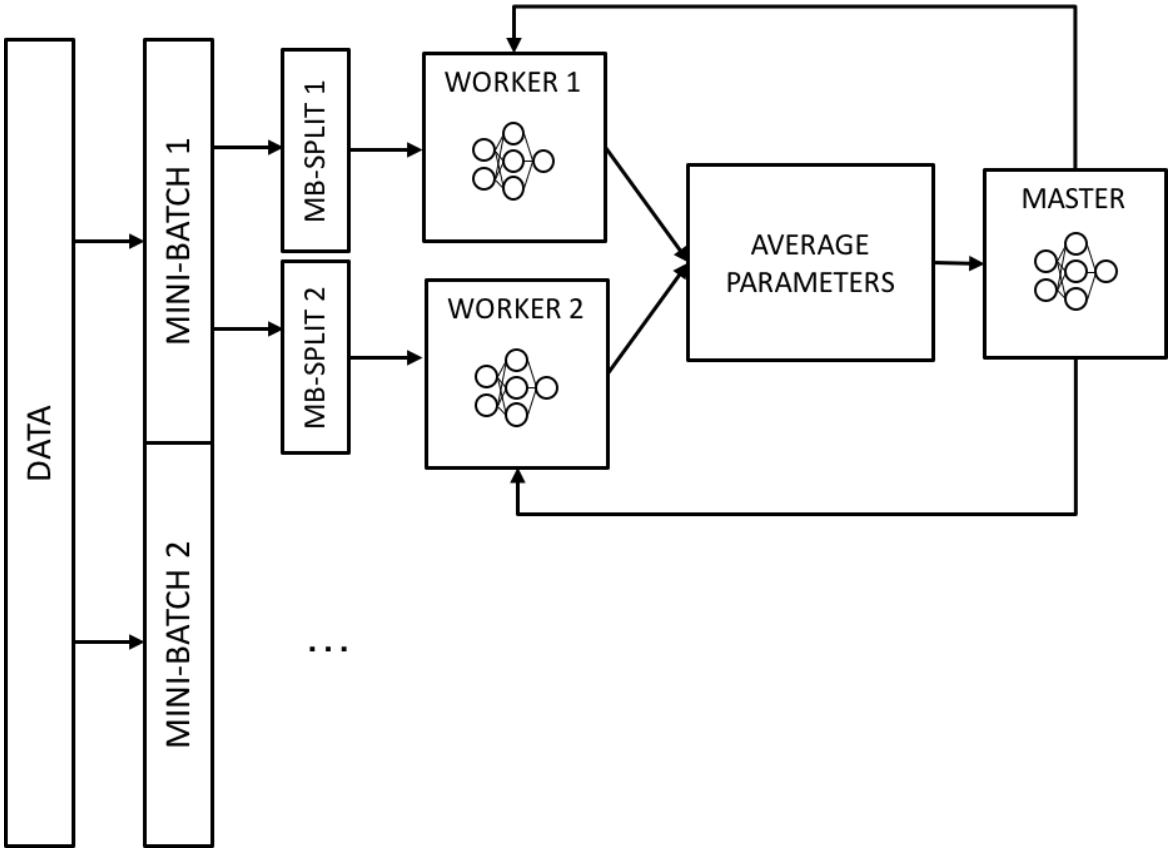


Fig. 3.1 Parameter averaging in DL4J over Spark (example using two mini-batches).

worker must prefetch to avoid waiting for the data to be loaded), and repartitioning strategy (when and how to repartition data to keep the partitions balanced).

### Apache Spark

As mentioned before, Apache Spark is the key component of the proposed framework. Spark is a distributed system for processing data-intensive workloads. It excels in an efficient memory usage, outperforming Hadoop for many applications [117]. Spark is being used to execute big data workloads on the MareNostrum supercomputer, isolating the applications from the particularities of this HPC infrastructure. Spark is designed to avoid the file system as much as possible, retaining most data resident in distributed memory across phases in the same job. Such memory-resident feature stands to benefit many applications, such as machine learning or clustering, that require extensive reuse of results across multiple iterations. Essentially, Spark is an implementation of the so-called Resilient Distributed Dataset (RDD) abstraction [117], which hides the details of distribution and fault-tolerance for large collections of items. The usage of Spark over alternatives with potentially better

performance, e.g. MPI, was a prerequisite of the described deployment. While Spark has some advantages such as fault tolerance, its main advantage over other alternatives in this case was that it minimized the deployment cost.

### The Spark4MN Framework

The MareNostrum supercomputer is accessed through an IBM LSF Platform workload manager. In order to be able to deploy Spark clusters over MareNostrum, we employ an intermediate layer called Spark4MN [105]. Spark4MN is also in charge of managing the deployment of any additional resource Spark needs, such as a service-based distributed file system (DFS) like Hadoop distributed file system (HDFS). Essentially, Spark4MN is a collection of *bash* scripts that deploy the Spark cluster's services, and executes the user applications. Spark4MN scripts read a configuration file, describing the application and the Spark cluster configuration, and submit one or more jobs to the MareNostrum workload manager. Once the cluster's job scheduler chooses a Spark4MN job to be executed, an exclusive number of cluster's nodes are reserved for the Spark cluster and (if requested) for the DFS (e.g. HDFS) cluster (may be the same nodes, depending on the configuration). After the resource allocation procedure, Spark4MN starts the different services. In Spark4MN, the Spark master corresponds to the *standalone* Spark manager, and workers are Spark worker services, where the Spark executors are received and launched. The cluster startup requires about 12 seconds. This is independent of the size of the cluster (the number of nodes). Each application is executed via *spark-submit* calls. During each Spark job execution, intermediate data is produced, e.g., due to shuffling. Such data are stored on the local disks and not on DFS by default (as in [72], this yields the best performance). Finally, Spark timeouts are automatically configured to the maximum duration of the job, as set by the user.

### Marenostrum supercomputer

MareNostrum is the Spanish Tier-0 supercomputer provided by BSC. It is an IBM System X iDataplex based on Intel Sandy Bridge EP processors at 2.6 GHz (two 8-core Intel Xeon processors E5-2670 per machine), 2 GB/core (32 GB/node) and around 500 GB of local disk (IBM 500 GB 7.2K 6Gbps NL SATA 3.5). Currently the supercomputer consists of 48896 Intel Sandy Bridge cores in 3056 JS21 nodes, with more than 104.6 TB of main memory and 2 PB of General Parallel File System (GPFS) disk storage. More specifically, GPFS provides 1.9 PB for user data storage, 33.5 TB for metadata storage (inodes and internal filesystem data) and total aggregated performance of 15GB/s. The GPFS filesystems are configured and optimized to be mounted on 3000 nodes. All compute nodes are interconnected through

an Infiniband FDR10 network, with a non-blocking fat tree network topology. In addition to the 40 Gb/s Infiniband, 1 Gb/s full duplex Ethernet is in place. With the last upgrade, MareNostrum has a peak performance of 1.1 Petaflops.

### 3.1.4 Experiment

The main goal of the experiments is to evaluate the scalability properties of the proposed deployment. To this end, we have experimented with different workloads and different deployment setups. Regarding the benchmarking workloads, we have chosen two widely used convolutional networks, AlexNet [64] and GoogLeNet [97]. Both networks have been used in other state-of-the-art works (e.g. [57]) and let us compare our results with others. While AlexNet implements a rather shallow network with many parameters, GoogLeNet is a very deep network with many convolutional layers. We apply both networks to dataset of the ImageNet [90] visual recognition challenge. For reproducibility, we stick to the ILSVRC2012 classification task training and test datasets and their standard evaluation procedure.

Table 3.1 Properties of the DNNs used in the experiments.

| **Properties** | **AlexNet** | **GoogLeNet** |
|---|---|---|
| Default batch size | 256 | 32 |
| Default step-size | 0.1 | 0.1 |
| # Iterations till convergence | 450k | 1000k |
| # Layers | 25 | 159 |
| # Convolutional layers | 5 | 59 |
| # Fully-connected (FC) layers | 3 | 1 |
| # Weights in FC layers | 55M | 1M |

Regarding the deployment setup, we have tested different values for the number of nodes, the number of Spark workers per node, the Spark data partition size, the DL4J averaging frequency and the persistence level. Figure 3.2 shows the speedup results obtained with $B = 256$ and $B = 1024$ (two Spark workers per node, averaging each 3 mini-batch-splits, Spark's persistence level set to *MEMORY_AND_DISK_SER* and automatic partitioning). The step sizes were increased according to the batch size as suggested by [52], while the number of iterations has been decreased by the same factor. For each different number of nodes $n$, each node processes mini-batch splits of size $b = B/n$.

Under a basic setup (averaging for each computed minibatch and uniform node workload) synchronous data parallelism trough parameter averaging is mathematically equivalent to a non-parallel computation and yields the same accuracy results. However, accuracy degrades

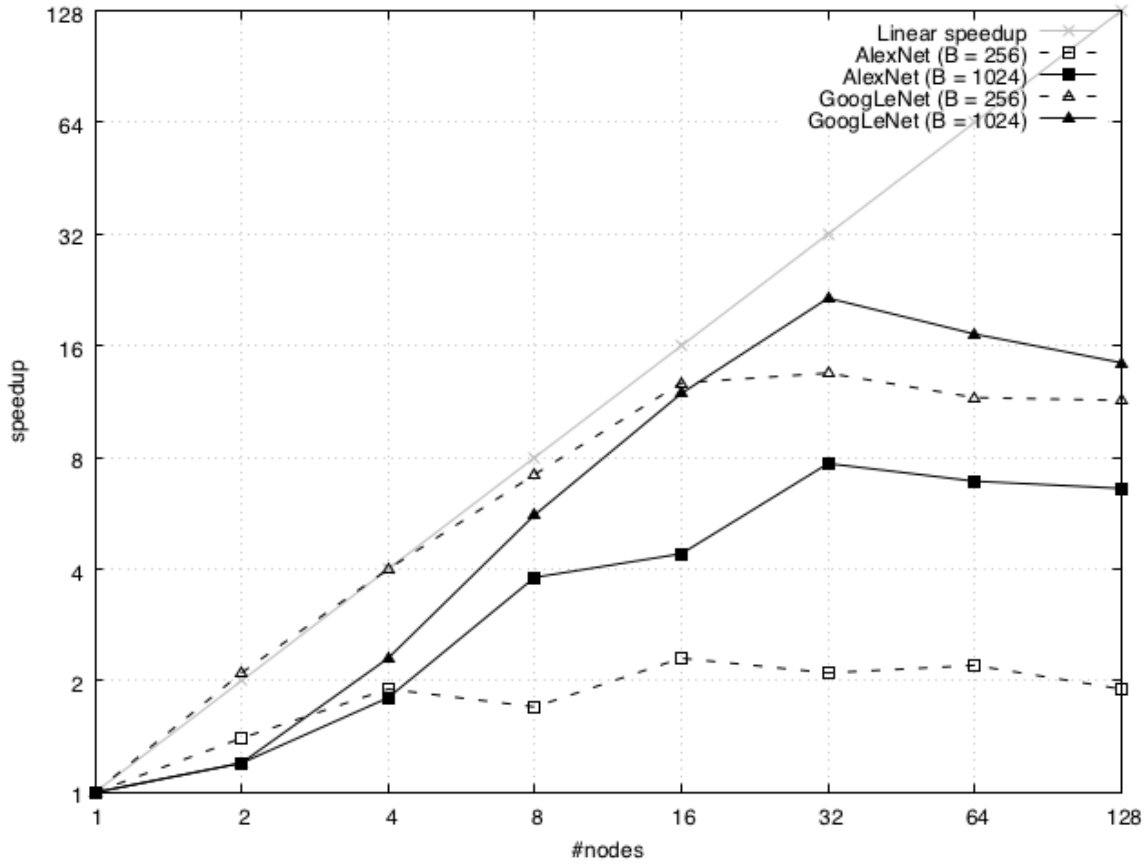


Fig. 3.2 Speedup results for AlexNet and GoogLeNet with different mini-batch sizes $B$.

(regardless of parallelization) when mini-batch sizes become too large [55], which imposes a constraint on scalability. Table 3.2 shows the accuracy results for the different configurations.

Table 3.2 Impact on accuracy of the different configurations.

| **mini-batch size** | **accuracy** |
|---|---|
| AlexNet (B = 256) | 56.9% |
| AlexNet (B = 1024) | 53.6% |
| GoogLeNet (B = 256) | 67.1% |
| GoogLeNet (B = 1024) | 65.4% |

The following table 3.3 shows the different elements used to set up the experiment.

Table 3.3 Experiment setup summary.

| **Dataset** | **Pre-Processing Data** | **Model** | **Evaluation Architecture** | **Software Version** |
|---|---|---|---|---|
| ImageNet | none | CNN:<br>- GoogLeNet[96]<br>- AlexNet[63] | Marenostrum 3:<br>*115.5TB RAM<br>*3,056 nodes<br>*3PB storage | Spark 1.5.2<br>Dl4j 0.4-rc3<br>Spark4MN |

The results of our evaluation show that DL4J and Spark are able to scale deep learning workloads over MareNostrum. However, the effective scaling stops above 32 nodes with the best configurations. This limitation agrees with the results reported in [57], that studies the theoretic constraints of synchronous data parallelism for DNNs training. The main bottleneck of the synchronous approach is the computation to communication ratio. The synchronous parallelization of DNN training requires the communication of the model $w_t$ and the computed gradients $\Delta w_t$ between all nodes in every iteration $t$. Since $w$ has to be synchronous in all nodes and $\Delta w_{t+1}$ can not be computed before $w_t$ is available, the entire communication has to be completed before the next iteration $t+1$. The problem is that $w$ and $\Delta w$ have the size of all weights in the neural network, which can be hundreds of megabytes. The compute times per iteration are rather low and decrease when scaling to more nodes. Depending on the model size and layout, the training problem becomes communication bound after scaling to only few nodes. Shallow networks with many neurons per layer (like AlexNet) scale worse than deep networks with less neurons (like GoogLeNet) where longer compute times meet smaller model sizes.

A second problem of the synchronous approach is that nodes process mini-batch-splits instead of mini-batches, and the size $b$ of these splits depends on the number of nodes $n$.

If *b* is too small (less than 32 samples in our experiments), there will be a negative impact on the inner parallel computation (within the node), especially in the case of the FC layers. One solution would be to increase the mini-batch size in parallel to the number of nodes, but too large batch sizes have been shown to cause slowdown in convergence and degrade the generalization properties of the trained model [65]. As larger mini-batch sizes enable lower absolute training times when many nodes are used, they are always preferred, with the limitation of accuracy degradation. While intra-node parallelism of fully connected layers improves with a larger mini-batch size, the poor scalability results observed at low core counts can be attributed to a poor intra-node parallelism of the other layers (dropout, pooling and LRN). The observed behavior is consistent with the results from Keuper and Pfreundt (2016). Another aspect negatively impacting scalability at low core counts (few nodes) can be related to data loading. Our implementation uses asynchronous data prefetching (as it is the default DL4J behavior). The next mini-batch-splits are loaded in another thread of the worker while training is proceeding in the main thread. Under ideal circumstances (fast disk access and small mini-batch-split size), asynchronous prefetching implies negligible data loading delays (except on the first iteration). However, MareNostrum nodes are equipped with relatively slow local disks (IBM 500 GB 7.2K 6Gbps NL SATA 3.5), a circumstance that can turn data loading into a bottleneck when mini-batch-splits are too big (i.e. when large mini-batches are distributed among few nodes) and the network is shallow.

A third problem is stragglers. The duration of the iteration depends on the slowest node. This effect gets worse with scale.

Asynchronous parallelization, not possible with the current version of DL4J, would solve these problems but, as mentioned before, has the *stale gradient problem* (though our nodes are homogeneous and the impact would be low). Some recent works like [65] propose a hybrid approach in which synchronous parallelism just takes place within groups of nodes. Partial solutions to the *stale gradient problem*, e.g. [76]), have also been proposed.

### 3.1.5 Conclusions

This work explores the feasibility and efficiency of using Apache Spark and DL4J for deploying deep learning workloads over a real-world, petascale, HPC setup, such as MareNostrum. To this end, we have designed a layered architecture consisting in both, general-purpose (Spark and DL4J) and custom components (Spark4MN). We have evaluated the deployment by training AlexNet and GoogLeNet over the ImageNet dataset. We have tested different deployment setups (number of nodes, number of Spark workers per node, data partition size, mini-batch size, mini-batch-split size, averaging frequency, prefetching and repartitioning strategy).

We conclude that it is feasible to rely on Apache Spark to deploy DL workloads over a traditional HPC setup. This approach minimizes deployment costs and enables a systematic tuning of the different configuration parameters, both at application level and at infrastructure level. However, the effective scaling is strongly limited by the synchronous parallelism approach applied by the latest DL4J version. Problems such as the communication overhead, mini-batch-split size and stragglers degrade the scalability beyond 32 nodes. In order to overcome this limitation, it would be necessary to replace the synchronous mechanism by a hybrid approach in which synchronization just takes place within fixed-size node sets. Assessing the impact of certain aspects, such as a quantitative evaluation of the effects on performance of asynchronous data prefetching, deserves further investigation and will be carried out in future work.

## 3.2 Digital Marketing: A Case Study

### 3.2.1 Introduction

Nowadays, there is a growing interest in exploiting the photos that users share on social networks such as Instagram or Twitter [22][107], a part of the so-called user-generated content (UGC). A significant part of these images has potential value for digital marketing tasks. On the one hand, users' photos can be analyzed to obtain knowledge about users' behavior and opinions in general, or with respect to a certain products or brands. On the other hand, some users' photos can be of value themselves, as original and authentic content that can be used, upon users' permission, in the different brands' communication channels. This work is related to this second use case, searching, discovering and exploiting UGC for digital marketing tasks, that has been traditionally addressed by the so-called *content curation* technologies.

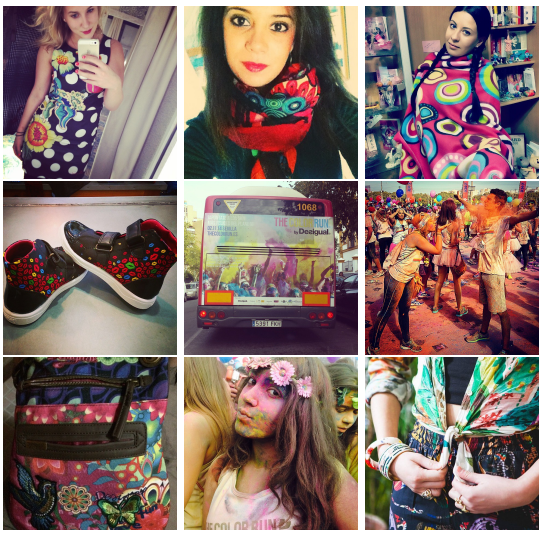

Fig. 3.3 Example images posted by Instagram users and tagged with Desigual's promotional hashtags (e.g. #lavidaeschula), used to feed the proposed solution

Platforms for photo-centric UGC are proliferating rapidly nowadays (e.g. Olapic [ola], Chute [chu] and Curalate[cur]), but discovering valuable images on social media streams is challenging. The potential bandwidth to analyze is huge and, while they help, user defined tags are scarce and noisy. A large part of current solutions relies on costly manual curation tasks over random samples. This way many contents are not even processed, and many valuable photos go unnoticed. Adoption of image recognition techniques in commercial UGC systems is currently very limited. In the best case, they provide generic classifiers whose categories and original training data were not specific to UGC. Often these classifiers

limit to the categories of the ImageNet Large Scale Visual Recognition Challenge (ILSVRC), but the vast majority of Instagram/Twitter photos are people-centric (selfies, food, clothes, etc.) while ILSVRC is more generic (fauna, flora, etc.). An important particularity of UGC is the huge amount of *spam images*, i.e. images that, in the most usage scenarios, have no value neither as knowledge carriers nor as a exploitable content. The incapacity of detecting the multiple types of spam images limits the usability and efficiency of existing solutions. Another difficulty of adopting image recognition techniques into UGC systems is the high computational cost of CNN-based image classifiers and object detectors. These systems need to process incoming streams of hundreds of images per second and a very volatile traffic. Any additional processing component need to be extremely efficient and scalable.

In this work, it has been proposed an approach based on deep CNNs and transfer learning to minimize manual curation as much as possible and to make it more efficient. As a result, we increase the number of photos processed several orders of magnitude, we increase the quality of the resulting photos (as more photos are analyzed and only the best ones go through manual curation), we enable near real-time discovery and, last but not least, we drastically reduce the cost.

### 3.2.2 Related Work

#### Classification and search of brand-related images in social networks

The work presented in this paper is related to recent works attempting to facilitate the classification and search of images in social networks such as Instagram and Twitter. Some works, such as [29][40][75][79][106], also apply scene-based and object-based image recognition techniques to enrich the metadata originally present in the images in order to facilitate their processing. All latest works rely on CNNs as an underlying technique. In our case, the applied image recognition techniques, while also relying in CNNs, are tuned for content curation for digital marketing tasks. This implies new problems, such as the need to recognize more abstract categories (e.g. "mediterranean") and the need to deal with smaller datasets (e.g. brand-based image datasets). Previous works such as [39][41][122] also classify social media data paying special attention to brands and products. Regarding the annotation of images with generic object categories, we reuse Google's Inception-v3 model [98], trained for the ImageNet Large Scale Visual Recognition Challenge and 1000 object categories with a top-5 error rate of 15.3%. Regarding the training of classifiers for new categories, we solved the overfitting problem related to the usage of small training sets by applying a transfer learning approach the same way Berkeley researches do in [32].

### Visual brand identity recognition

As far as we know, this is the first work that addresses the automatic recognition of visual brand identity in images. In [14] researchers from Georgia Tech and Yahoo labs identified a relationship between certain visual aspects (warmth, exposure, and contrast) and a photo's engagement on Flicker and Instagram. Some years before, researchers from the University of Portsmouth, UK, analyzed how wavelength hues influenced users' perception and reaction [23]. In [121], authors perform personalized (for each individual viewer) image emotion classification in social networks. In our approach, emotions are just one of the aspects to consider, as brands pay also attention to other aspects (lifestyle, values, etc.). Regarding the training of classifiers for the recognition of visual brand identity, we also applied a transfer learning approach to avoid overfitting as we need to deal with small training sets.

### Logo detection

Regarding the detection of logos, in [108] authors propose a dense histogram type feature to classify logo and non-logo image patches from the Sina Weibo platform, a Chinese microblogging site. In [82], authors propose CNN-based approach able to predict bounding boxes and class probabilities in just one evaluation, without the need to apply a sliding window. This approach is extremely fast and authors claim being able to process images at 45 frames per second. In our case, due to need to generate the detectors in an on-demand basis, we opted for a solution with worse real-time performance but that doesn't need too many resources for training (data and computation power). We have developed a system that automatically processes, in real-time, an incoming stream of social media images and detects and localizes all the occurrences of any of a set of supported logotypes. The system makes use of two state-of-the-art deep CNNs designed for object detection, SSD InceptionV2 and Faster Atrous InceptionV4 (that provides better performance on small objects). The resulting system is currently being integrated within a real commercial service, the Adsmurai's Visual Commerce Platform

In this part of the work, we describe the technical design of the system and the results of the performance evaluation experiments in which real images related with two commercial brands, Estrella Damm and Futbol Club Barcelona, have been used. We examine the impact of different configurations and derive conclusions aiming to pave the way towards systematic and optimized methodologies for automatic logo detection in UGC.

## 3.2.3 Experiments

### 3.2.3.1 User Generation Content

**Outline of the system**

The system developed processes images for one or more marketing campaigns. Each user (usually a brand's account manager) can operate multiple campaigns simultaneously. The functionality of the system can be divided into two different stages, data acquisition and data consumption.

During data acquisition the system captures and annotates new images from social media with potential value for a given campaign, and indexes them into a database. During this stage, new images are captured in real-time, as they are published on the underlying sources (Instagram and Twitter). Descriptors of the images (including the URL pointing to the image content) are acquired using the APIs provided by these underlying sources. These APIs impose limits over the number of images that can be obtained during a certain period of time. So, processing the entire stream of images produced by a given API is not possible. APIs provide the possibility to subscribe to certain filters, such as tags or geolocation bounding boxes. These filters produce partial streams that may be overlapped. In order to capture images with potential value for a campaign, our system first needs information about geographical areas and/or hashtags that are related to the campaign (e.g. promotional hashtags such as Desigual's "#lavidaeschula" or Estrella Damm's "#mediterraneamente"). These data are used by the system to program a set of subscriptions to the underlying sources. Each subscription will produce a continuous stream of images that we call "channel". The throughput of the channels may be extremely volatile, requiring a proper scalability strategy. Once a new image is captured, it is processed by multiple deep convolutional neural networks that automatically enrich the image's metadata with tags that describe their visual content (e.g. "selfie", "pizza", etc.) plus a score that measures how the image fits the visual identity of the brand.

During the data consumption stage users can navigate, search and select images from the database. Depending on the communication channel where an image is going to be used (paid ads, organic posts, images feeds, etc.) the user who post the image will be asked authorization. Both stages (acquisition and consumption) interact through the common images database, and they can occur concurrently (once images start feeding the database users can start using them). Figure 3.4 shows visually the overall data flow of the system.

**Image semantics recognition**

During the acquisition stage, captured images are processed by a set of multi-class and binary

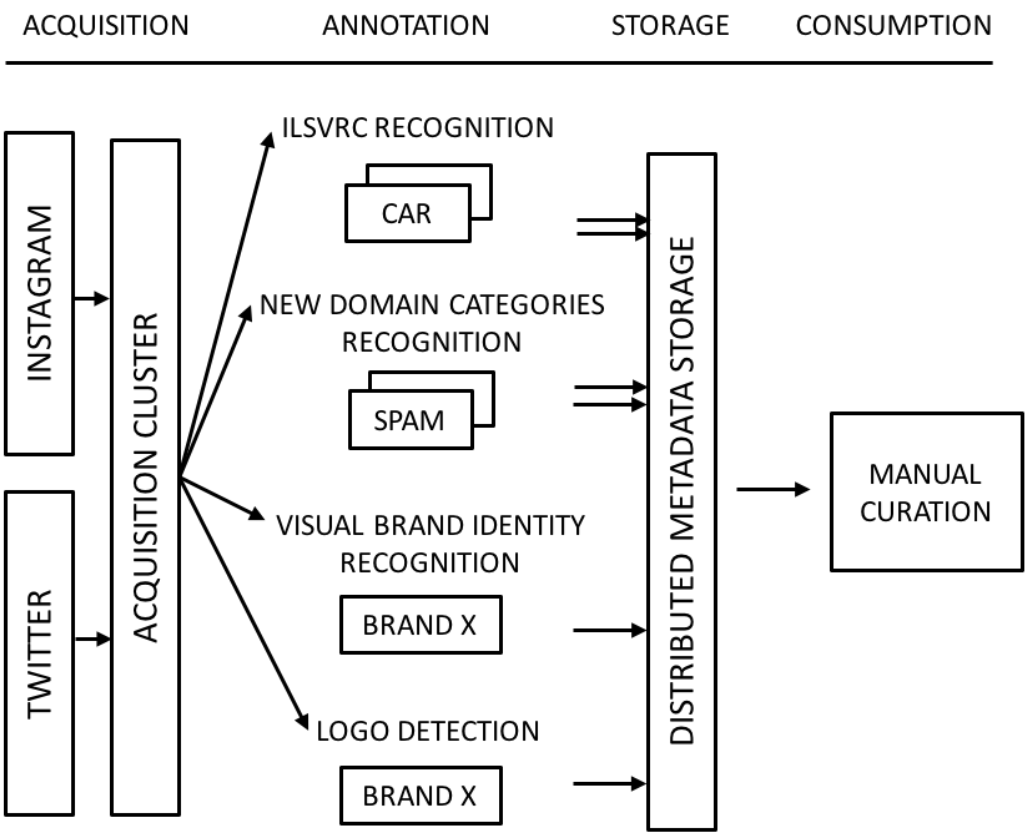


Fig. 3.4 Overall data flow of the system. Images are acquired and annotated in real-time. The resulting metadata are stored and queried by a manual curation graphical user interface.

image classifiers that try describing their visual content (for logo detection, explained below, we have applied a different approach). All the involved classifiers share the same architecture, Google's Inception-v3 [98], a deep convolutional neural network trained for the ILSVRC. One of the classifiers that we apply is Inception-v3 itself, which let us classify the images into 1000 different categories. As these categories are very specific terms from WordNet, we expanded ImageNet categories with their corresponding WordNet hypernyms reaching a total of more than 7,000 different tags. Table 3.4 shows some of them.

Table 3.4 Some of the 1000 ILSVRC tags and their expanded WordNet hypernyms.

| **Original ILSVRC tag** | **4-depth WordNet hypernyms** |
|---|---|
| Siberian husky | sled dog, working dog, dog, canine |
| beer bottle | bottle, vessel, container, instrumentality |
| red wine | wine, alcohol, beverage, food |
| consomme | soup, dish, nutriment, food |
| cowboy hat | ten-gallon hat, hat, headdress, clothing |
| burrito | dish, nutriment, food, substance |
| ... | ... |

However, we have observed that many objects and scenes that typically appear in Instagram/Twitter images do not appear in ILSVRC. The vast majority of Instagram/Twitter photos are people-centric (selfies, food, clothes, etc.) while ILSVRC is more generic (fauna, flora, etc.). Also, even if an object or scene appears in ILSVRC often it is not part of the ILSVRC categories dictionary (i.e. WordNet). In order to provide a more comprehensive and practical set of tags, we have trained our own classifiers (more than 100, Table 3.7 shows some of them), retraining the last layer of Inception-v3. The most part of them are binary classifiers, that enable us to determine if an image should be annotated with a given tag (one for classifier). Notable examples are *spam* and *selfie*, tags that have proven to be very useful when searching this kind of images. The criteria of inclusion of new tags is currently heuristic, driven by the feedback of users interacting with the final curation interface. Figure 3.5 shows some example images from the spam dataset.

**Visual brand identity (VBI) recognition**

Besides annotating the incoming images with tags that describe their semantics, we have also trained a set of CNNs that perform what we call VBI recognition. Nowadays, the main course in almost all branding initiatives is to develop a unique and consistent visual brand identity that expresses and reflects the brand's culture and character. A VBI may involve

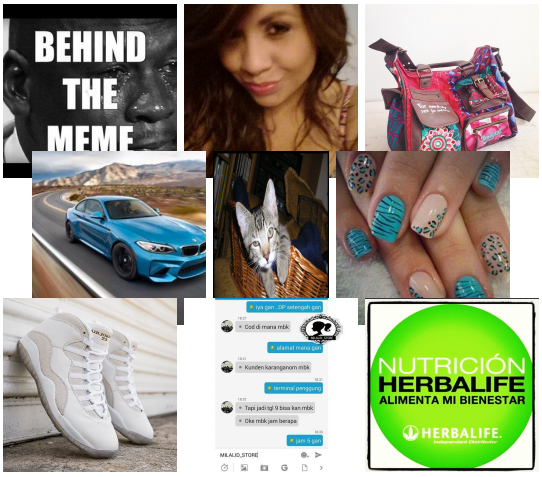


Fig. 3.5 Example images from different sub-categories of the spam dataset.

the preference for some colors, lighting, themes, etc. It's easy to find examples of visual identities for iconic brands such as Coca-Cola, Levis or McDonald's.

Each VBI classifier is a binary image classifier based on Inception-v3 and fine-tuned with a dataset provided by a brand. The classifier learns to distinguish which images satisfy some visual patterns found in the brand's imagery. Initially, the classifier is trained with a dataset provided by the brand (e.g. a set of images used in a previous marketing campaigns and that are representative of the brand's visual identity). Later, each time a certain amount of usage actions (selection of new images by the user) have been recorded (a training batch), the model weights are updated. Figures 3.6 and 3.7 show some example images used to train VBI classifiers for the Estrella Damm beer brand (related to the Mediterranean lifestyle) and Pepsi, respectively.

**Transfer learning approach**

In order to reduce overfitting, improve accuracy and reduce training times we have chosen a Transfer Learning approach for training the classifiers (for both the image semantics recognition and the VBI recognition). The method consists on fine-tuning a deep architecture already trained with millions of images on a set of traditional object recognition tasks. We start by processing each one of our training images through all the layers of the Google's Inception-v3 model [98] except the last one. For each image, we save the values of the penultimate layer of Inception (called the image *bottleneck*). Once we have computed all the bottlenecks, we replace the final layer of Inception with a new one, defined over the categories of the model that we want to train (e.g. a binary spam-detector model with just two classes). Then we run some (around 4K) training steps over the network (feeding the bottlenecks directly into the final layer).

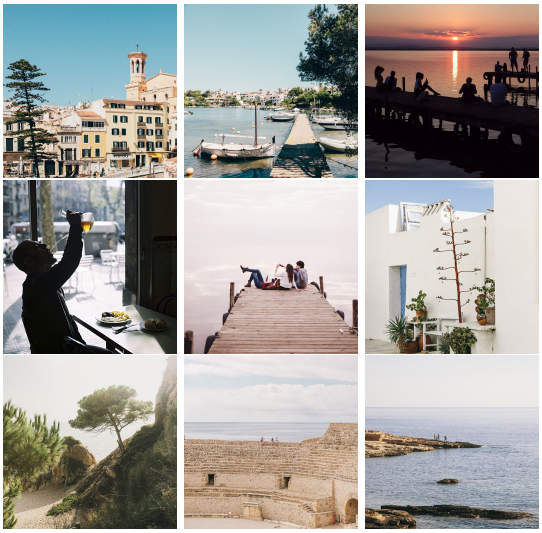

Fig. 3.6 Example images showing the visual brand identity of the Estrella Damm beer brand, related to the Mediterranean lifestyle.


Fig. 3.7 Example images showing the visual brand identity of the Pepsi.

**Alternative algorithms**

We have compared the obtained results with two other methods, a Bag of Words (BoW) approach and training our own lightweight CNNs. Regarding BoW, with the help of OpenCV we computed Opponent-SIFT descriptors from the images and clustered them to obtain a dictionary of k-dimensional visual words. With the help of the dictionary we transformed each image into a k-dimensional vector. With the obtained vectors, we trained a Support vector machine (SVM) classifier with an Radial basis function (RBF) kernel. We performed experiments with different configurations (different descriptors, different downscaling sizes, different kernels, etc.). Regarding the training of our own CNNs, we defined and trained, with the help of TensorFlow, a lightweight, 6-layer deep convolutional neural network (3 convolutional+relu layers, two fully connected layers and a softmax layer). We applied data augmentation and disabled Local Response Normalization.

**Datasets**

Table 3.5 shows the details of the datasets used in this work for which results are provided in the next section. We worked with three different groups of datasets, the ones for training new generic image recognition classifiers or NGR (e.g. "selfie" and "spam"), the ones for visual brand identity recognition classifiers or VBI (e.g. "Pepsi" and "FC Barcelona") and the ones for logo detection or LOGO (e.g. "Estrella Damm logo"). We only provide details about a representative subset of the classifiers/detectors actually trained. Regarding NGR, we finally trained more than 100 new classifiers (here we show details about 6 of them). Regarding VBI, we trained more than 20 classifiers (here we show details about 6 of them). Regarding LOGO, it is a new feature in which we are currently working and here we only provide results for the Estrella Damm logo.

Regarding datasets acquisition for NGR, the most part of the new models required to acquire training images that are not part of any public images dataset. In order to solve this problem, we combined images both from Instagram and the WWW. Instagram photos were obtained from the Instagram API, filtered with user defined tags and manually purged. As user defined tags are very noisy this method proved to be inefficient and very time-consuming. In order to facilitate the generation of more ground truth annotations and a larger training dataset we also obtained images from Google Images through the Custom Google Search API. This method, which allowed to automatically annotate a bigger set of images, turned out to be very useful as almost all the retrieved images showed the desired category (e.g. "handbag") and minimum manual purge was required. The resulting dataset contains more than 50K images distributed in 100 different categories.

Table 3.5 Summary of training datasets (NGR = New generic recognition, VBI = Visual brand identification, LOGO = Logo detection).

| tag | type | #positives | #negatives |
|---|---|---|---|
| selfie | NGR | 295 | 8,959 |
| group_selfie | NGR | 98 | 8,884 |
| spam | NGR | 319 | 8,979 |
| burguer | NGR | 474 | 8,845 |
| nails | NGR | 434 | 8,866 |
| sushi | NGR | 571 | 8,920 |
| Pepsi | VBI | 680 | 8,422 |
| FC Barcelona | VBI | 1,023 | 8,366 |
| Estrella Damm | VBI | 663 | 8,363 |
| Desigual | VBI | 1,381 | 8,862 |
| Catalunya Experience | VBI | 89 | 8,809 |
| Estrella Damm logo | LOGO | 322 | 1,681 |

Regarding datasets acquisition for VBI, the brands are responsible of providing a curated dataset of images that satisfy their VBI criteria. In our prototype, our main source of (positive) training images are the brand's Instagram profiles. With the help of the Instagram API we have been able to collect training sets with sizes varying from several hundred (e.g Ecooltra) to several thousand (e.g. Desigual). As negatives we use images from multiple NGR classifiers' datasets with low probability of semantic overlapping. While the performance of a VBI classifier depends initially on the quality of the starting dataset, it improves as the system is used.

The data acquisition for LOGO is the only one which required a totally manual process. First, we captured Instagram images annotated with the Estrella Damm promotional hashtag. Then we manually annotated the bounding boxes of all the logo occurrences with a tool that we developed for this task.

It's worth mentioning that we didn't use a public dataset such as Brand-Social-Net because the goal of the work was to evaluate the viability of the whole approach on a real scenario. This included to experience with the generation of training datasets for custom (brand-specific) classifiers on demand.

Regarding compliance with laws and ethical standards, the system analyzed publicly available photos as they were published in Instagram, without keeping them. All statistical data retained had no identifying information about individuals and no confidential data that allow statistical units to be identified, either directly or indirectly, thereby disclosing individual information. For photos identified as relevant, explicit consent from users was requested (through Instagram direct messaging tools) before publishing them in brands' communication channels.

**Software and hardware configuration**

The data acquisition system was implemented with Java and served as a set of RESTful APIs. The scalable image metadata database was implemented with Elasticsearch. The image recognition service was implemented with Python and TensorFlow, and also implemented as a set of RESTful APIs. Once in production, the system is running over a cluster of 6 Amazon EC2 t2.large instances (dual core 3.3 GHz Intel Xeon processor and 8 GB of memory). The CNNs were trained over a high-end server with a quadcore Intel i7-3820 at 3.6 GHz with 64 GB of DDR3 RAM memory, and 4 NVIDIA Tesla K40 GPU cards with 12 GB of GDDR5 each. The following table 3.6 shows the different elements used to set up the experiment.

Table 3.6 Experiment setup summary.

| **Dataset** | **Pre-Processing Data** | **Model** | **Evaluation Architecture** | **Software Version** |
|---|---|---|---|---|
| ImageNet | none | Inception-V3[99] | 6 Amazon EC2 (t2.large) | TensorFlow 0.12.1 |
| VBI-dataset | Re-Size data | VBI models | Server i7-3820 | Python 2.7 |
| ISR-dataset | Re-Size data | ISR models | 64GB Ram | ElasticSearch 5.3.0 |
| | | | 4 NVIDIA Tesla K40 | Docker Engine 1.13 |

**Results**

The combined method that we propose integrates multiple state-of-the-art image recognition and object detection algorithms. As mentioned previously, the goal of the work is not to re-evaluate the performance of these algorithms but to analyze their suitability among other alternatives and to asses the viability of the whole approach on a real scenario. Therefore, following we provide representative results for the three different groups of models that we trained (NGR, VBI and LOGO). The difference between NGR and VBI classifiers is the way the training data is obtained, but the underlying method is the same. In order to evaluate the suitability of the chosen method (Transfer Learning) we compared its classification accuracy, classification time and training time with two alternative methods (BoW+RBF-SVM and lightweight CNN). We also provide some results of the chosen method when applied to VBI classifiers.

**Classification accuracy**

Table 3.7 shows some representative results obtained for the image semantics recognition part. The results show that the classic BoW approach provides the worst accuracy but it is the fastest to train and predict and the one with a smallest memory footprint. The approach consisting

in defining and training our own deep CNN provides accuracy improvements of more than 10% with respect to BoW. However, with our small training sets this method implies a strong overfitting, as pointed in [32]. Training times in a high-end server with 4 NVIDIA Tesla K40 GPU cards are between 3 and 5 hours. One advantage of this method (with respect of the Transfer Learning approach that we finally chose) is that models have a small memory footprint. Another advantage is that predictions are faster (as the network is significantly simpler). Disadvantages of this method (with respect to Transfer Learning) are overfitting, significantly higher training times and (about 5%) lower accuracies. Finally, the transfer learning approach improves accuracies (with respect to training our own lightweight CNN) about 5%, reduces overfitting and reduces training times to less than 2 hours (significantly less if some images are reused as the bottlenecks need to be obtained just once). One significant disadvantage (specially when many models have to be served simultaneously) is that models have a big memory footprint. Another disadvantage is that prediction times are higher.

Table 3.7 Training results of some of the 100 new models that we have trained for image semantics recognition.

| **tag** | **BoW** | **CNN** | **CNN-TL** |
|---|---|---|---|
| selfie | 72% | 87% | 93% |
| group_selfie | 76% | 88% | 95% |
| spam | 69% | 78% | 91% |
| burguer | 81% | 89% | 95% |
| nails | 83% | 92% | 97% |
| sushi | 86% | 93% | 96% |

Because of its advantages in terms of accuracy, reduced overfitting and training times, we finally chose the Transfer Learning approach for training the classifiers. Its ability to work with small datasets is especially suited the visual brand identity recognition task. Table 3.8 shows the VBI classification results that we obtained for 5 real brands with the Transfer Learning approach.

Table 3.8 Training setup and results for 5 visual brand identity classifiers.

| **Brand name** | **training** | **accy.** |
|---|---|---|
| Pepsi | 5,922s | 87% |
| FC Barcelona | 6,141s | 96% |
| Estrella Damm | 5,789s | 93% |
| Desigual | 6,310s | 95% |
| Catalunya Experience | 5,624s | 76% |

**Classification time**

Classification time is critical as the system needs to process in real-time a huge and volatile amount of incoming images. Each image needs to be classified by multiple models. Figure 3.8 shows a decomposition of the average classification time of one low-resolution (320x320) Instagram image by one model. The values are just indicative as download times are context-dependent. The main component of the classification time is the bottleneck computation (0.4 seconds). This computation, along with the downloading of the image (0.25 seconds), need to be done just once, as the bottlenecks are the same for all the models and they can be fed directly into the final layer. As the time to process the final layer is very small (0.05 seconds), including more models does not imply a significant cost, neither in terms of time, being the memory the only limitation in practice. On average, we need 0.7 seconds to classify one image.

Regarding logo detection, on average this stage adds around 0.8 seconds to the annotation of a single image (increasing the total time to around 1.5 seconds/image). Our current implementation does not support reusing intermediate layers among multiple logo detectors, so this time gets multiplied if more than one logo detector has to be applied to the same image. Even if just one logo detector is applied, a 0.8 seconds increase is a big penalty. For this reason, we have opted to only apply the logo detector when certain conditions are met. These conditions are expressed in terms of the other classifiers (e.g. not applying the logo detector if the image has been classified as spam). These conditions are configurable in a per-campaign basis.

The overall annotation process is an embarrassingly parallel problem and the throughput of the system can scale linearly adding more computational resources. We run as many annotations in parallel as possible, depending on the available cpu and memory.

Figure 3.9 shows a slice of a time series for the amount of images acquired and indexed during September 2016. In that period, the system was running an average of 5 campaigns simultaneously, including, but not limited to, Estrella Damm, Desigual, Catalunya Experience, Ecooltra and Shakn. Acquisition was done mainly based on hashtags (around 5 per brand), but some geolocation-based filters were also used (e.g. some specific beaches from the Balearic Islands for the Estrella Damm campaign). Each captured image was classified with the corresponding VBI model, inception, and 5-10 of our own semantics recognition models. More than 1 million images were captured, classified and indexed during one month, providing, on average, more than 200K images for each campaign.

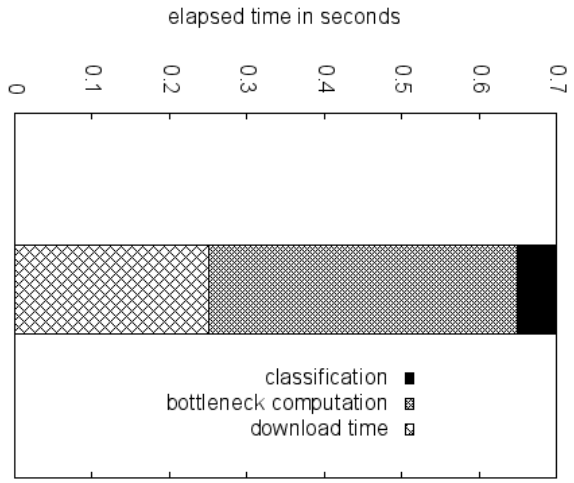


Fig. 3.8 Classification time for a single 320x320 image and one model.

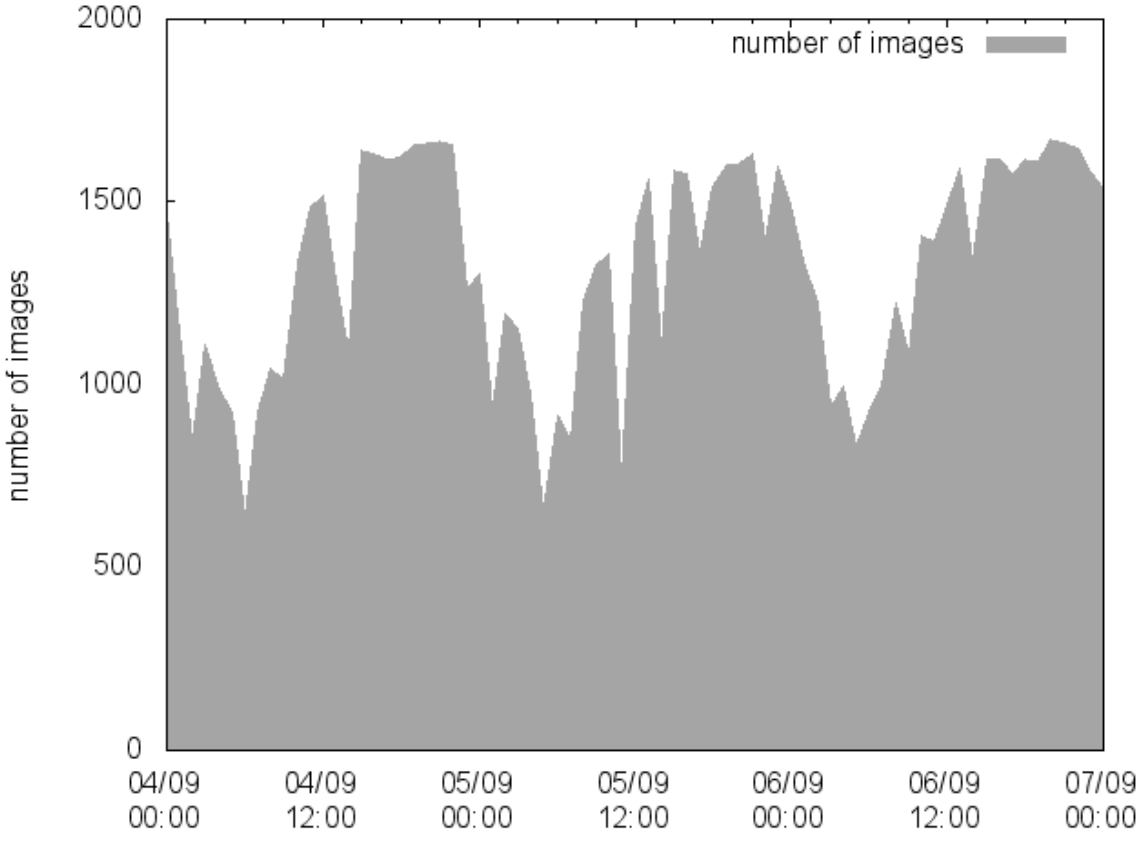


Fig. 3.9 Slice of the time series showing the amount of images acquired per hour during September 2016.

### 3.2.4 Conclusions

The research work presented analyzes the usage of deep CNNs for curating and filtering UGC for digital marketing tasks. We have built a system that captures images from Instagram and Twitter in real-time, and processes them by multiple CNNs that automatically enrich their metadata with tags that describe their visual content and also how they fit the visual identity of a brand. As far as we know, this is the first work that addresses the automatic recognition of visual brand identity in UGC. We have compared the results of three different methods (BoW+RBF-SVM, lightweight CNN and Transfer Learning) and we conclude that the Transfer Learning approach is the one that better suits this domain (best accuracy, less overfitting with small datasets, and low training times). With this method, we have trained VBI classifiers for more than 10 real brands and more than 100 classifiers for generic description of social media images. We have employed a ground truth of more than 50K images. Each model can be trained in less than 2 hours and the most part of resulting accuracies are always above 90%. We also process the images with Google's Inception-v3 and expand its 1000 WordNet categories with their corresponding hypernyms to obtain a dictionary of more than 7,000 tags. On average, we need 0.7 seconds to classify each image. As the bottleneck computation and image download consume the most part of this time, applying more models sequentially has just a sub-linear impact on the elapsed time. In practice, we run as many classifications in parallel as possible, depending on the available cpu and memory. During a experiment conducted on September 2016, the system captured, classified and indexed more than 1 million images related to 5 different brands. Discovering valuable images among them is finally done by using the provided search interface and applying the different filters (over the VBI tags, expanded inception tags, our own semantics tags, and any metadata provided by Instagram/Twitter). With respect of traditional curation methods, our approach minimizes human visual inspection, increases the number of photos processed several orders of magnitude, increases the quality of the resulting photos, enables near real-time discovery and reduces the cost drastically. There are some issues that need to be addressed, however. Our logo detection implementation has a high detection time and is currently single-label. Directions for future research include the evaluation of alternative logo detection algorithms. On the other hand, regarding image semantics recognition, the criteria of inclusion of new tags is currently heuristic, and should be addressed. Besides, as the number of possible tags becomes very large, it becomes more difficult to design a human curation interface able to take profit from them.

# Chapter 4

# Exploring Seismic Data with Deep Neural Networks.

## 4.1 Deep Neural Networks for Earthquake Detection

### 4.1.1 Introduction

The number of seismic networks and monitoring sensors have steadily increased in recent years, and the continuous growth of seismic records call for new processing algorithms that assist in solving problems in seismology. A fundamental endeavor is earthquake recognition, which pertains to the identification of seismic events in continuous data, with real-time application to early warning systems, or offline data postprocessing in the search for undetected past earthquakes. Based on time-dependent or spectral analyses of seismic traces, earthquakes of large or moderate magnitudes are easy to detect by conventional algorithms, some of which are cited below. Such earthquakes may impact the economy and human lives, but they are more rare and therefore scarce in seismic catalogs. Alternatively, small-magnitude earthquakes are more frequent, but they sometimes pass undetected by trained analysts or automatic recognition if, for instance, traces present poor signal-to-noise SNR ratios or recordings of overlapping events. Thus, seismic catalogs may be incomplete in the low magnitude range, as mentioned in [73][89][91][112]. Such catalogs are used in seismic hazard analysis for ground motion estimation under a given earthquake magnitude, which serves as a basis for building codes. In addition, accurate detection of the foreshocks and aftershocks associated with a main shock are used to constrain the actual fault configuration, total rupture area, and evolution of the tectonic stresses. All of these topics are fundamental in seismology, and they allow for better understanding of the hosting fault system and assessment of its destructive potential under surrounding geological conditions. Here we address

the earthquake detection and epicentral estimation problems for seismic data collected by broadband stations in north-central Venezuela during the time period from April 2018 to April 2019. Two main contributions of this study are as follows:

- A new analyst-labeled dataset, called CARABOBO according to the recording region, was built and made public for result reproducibility and benchmarking purposes. This dataset contains seismic data related to 949 earthquakes with magnitudes ranging between 1.1 Mw and 5.2 Mw.
- A deep CNN is proposed, called UPC-UCV and inspired by ConvNetQuake, to approach the P-wave detection problem by processing three-channel seismic signals in the mentioned geographic area. The method also performs source region estimation, mapping positive events to geographical partitions (automatically obtained with the k-means clustering algorithm) of the study region. Although our epicentral estimation lacks depth approximation for complete hypocentral location, it represents information that is available in real time and is complementary to standard earthquake location procedures.

### 4.1.2 Related Work

Conventional algorithms have been developed for automatic detection of seismic phases and picking the arrival times of P- and S- waves. Most of these techniques rely on general properties of seismic waves, waveform attributes, and statistical correlation or wave polarization analyses. A popular P-wave detector developed in [7] and [8] calculates the ratio of the average of the absolute amplitude of a signal in a short time window (STA) to the average of the signal in a long time window (LTA). The phase identification of S-waves is usually difficult due to superposition with P coda and converted phases, and some elaborate picking methods were developed in [13][21][31][84]. The P and S detection algorithm presented by [86] combines some of the aforementioned waveform analyses with STA/LTA measures. By using recordings from local stations, this method is also able to pick fault-zone head waves that may be triggered during earthquakes.

A different class of detection algorithms are based on pattern recognition or intensive autocorrelation. Most methods build appropriate pattern sets of earthquake and noise signals, and the association of new trace windows with one set based on sufficient similarity serves as the discrimination basis for detection. Among the early pattern recognition methods, one can find [9][20][58][69]. Earthquake signals that share the same source mechanism, along with similar path and site conditions, should present strong waveform similarities. Earthquake detection by intensive autocorrelation is undertaken by the methods in [44] and

[19]. This category of detection algorithms naturally adapts to general regional and network conditions and behaves reliably, but obtaining results is computationally intensive when processing large datasets. To reduce the computational cost, this class has evolved into the template matching algorithms that maintain a reduced database of representative waveforms to be fully compared against new traces or that extract and store key discriminative features from waveforms to be used in the similarity search. Some works focused on earthquake swarms and sequences of foreshocks and aftershocks are those of [16][47][92][94], and the Fingerprint and Similarity Thresholding (FAST) algorithm in [115]. In FAST, discriminative features are binary fingerprints, stored in a special dictionary according to their similarity, that allow for efficient searches owing to a locality-sensible hashing. Special FAST adaptations to sparse seismic networks have been recently proposed in [17].

Many of the latest methods use deep learning related techniques such as CNNs and RNN. Among the state-of-the-art CNNs for earthquake detection, we find ConvNetQuake developed by [101], the generalized phase detection (GPD) introduced in [88], and PhaseNet proposed by [125]. ConvNetQuake operates on three-channel seismograms from local stations for P-wave detection and has been used for the recognition of natural or human-induced (related to waste water injection) low-magnitude events in central Oklahoma, USA (hereafter referred to as the OKLAHOMA dataset). For training, the authors employ a seismic catalog of over two thousand events that occurred during 2014-2016 and then validate the network by using 209 additional events. This network is extensively used in this work as a reference, so additional ConvNetQuake characteristics and new performance assessments will be discussed in the following sections. The training and validation of GDP and PhaseNet make use of the extremely large seismic datasets, properly labeled by human analysts, that are available from the Californian seismic networks. In particular, GDP detection has also been tested on the 2016 Bombay Beach, California swarm of small and moderate ($\leq$ 4.8 Mw) events and with the data of the 7.0 Mw Kumamoto earthquake that occurred in 2016.

The aforementioned detection methods, as well as multiple alternative applications, could also be compared in terms of data preprocessing (trace normalization, filtering, windowing, etc.) and achieved accuracy, among other possible aspects. An interested reader, in addition to directly inspecting our listed references, may review any of the following survey papers: [30], [35], [59] and [85].

### 4.1.3 Methodology

#### The CARABOBO Dataset

The CARABOBO dataset of analyst-labeled P-wave arrivals is based on the data provided by FUNVISIS, the governmental agency for monitoring and reporting seismic activity in Venezuela. The FUNVISIS network includes 35 broadband stations recording three-channel continuous data at 100 Hz. The stations are mainly deployed in the regions with higher seismic activity which are close to the active fault systems in northern Venezuela. The dataset is the first for this region and contains seismic data (in miniSEED format) collected by broadband stations in north-central Venezuela, in the region of 9.5 to 11.5N and 67.0 to 69.0W, during the time period from April 2018 to April 2019. During this period, 949 earthquakes were recorded in that area by 5 seismological stations (BAUV, BENV, MAPV, TACV and TURV). The dataset also includes a catalog with the metadata related to the events (hypocenter, P-wave arrival times, magnitude, etc.) in Nordic format.

The method is also designed to perform source region estimation, mapping positive events to geographical partitions. Windows can be classified into k+1 classes: one class for windows not containing a P-wave (negatives) and k classes for windows containing a P-wave (positives) but classified into the k geographic regions that we want to discriminate. This section focuses on detection experiments and results. Source region estimation experiments and results are described in Section 4.2.

#### Earthquake Detection and Source Region Estimation

Earthquake detection and source region estimation are approached as a single multiclass classification problem, in which fixed-size windows of waveform data must be classified into $K+1$ classes. The first $K$ classes are for windows containing a P-wave from an event whose epicenter is located in one of the given $K$ source regions. The last class is for windows that are free of any P-wave arrival. The proposed method (UPC-UCV) is inspired by ConvNetQuake [101], which addresses these problems with a supervised learning approach. ConvNetQuake consists of a series of data preprocessing steps to convert the input waveforms into a set of fixed-size windows and the application of a deep convolutional network to infer the best class for a given window. ConvNetQuake was the first choice to perform detection over the CARABOBO dataset.

### Data Preprocessing

Several data preprocessing steps are applied in order to obtain the training and evaluation datasets. First, input streams with no data in any of the three channels are purged. Then, each input stream is normalized (independently for each of the three channels), and if the stream contains seismograms from multiple stations, it is divided into multiple 3-channel streams. Because different events can be detected by a different set of seismic stations (in different locations, a station is not always operative, etc.), processing single-station data enables obtaining a more homogeneous and larger training dataset. Each 3-channel stream is split into multiple 3-channel temporal windows of fixed size (window sizes in the range of 10-60 sec have been tested). With the event information obtained from the metadata files, the windows are divided into $K+1$ classes ($K$ classes for positive detection originating in one of the given $K$ source regions and one class for negative detection). These preprocessing steps are identical to those performed by ConvNetQuake, with the exception that ConvNetQuake only reports results with 10 sec windows. However, to improve the detection performance, the proposed method incorporates some preprocessing improvements. Unlike ConvNetQuake, which splits the input streams into non-overlapping windows, the proposed method applies a sliding window approach so that the resulting windows partially overlap. Another difference with respect to ConvNetQuake involves dataset balancing. ConvNetQuake works with an extremely imbalanced dataset, but the proposed method balances the training data by undersampling the negative data. Finally, unlike ConvNetQuake, bandpass filtering ([0.5, 10] Hz) is applied to attenuate the noise. This preprocessing setup applied over the CARABOBO dataset results in sixty thousand windows, 50% positive and 50% negative.

### UPC-UCV network architecture

The proposed model is a deep convolutional network inspired by the one described in [101] but enhanced to be able to work properly with the CARABOBO dataset. First, the convolutional part of the network is flattened by enlarging the size of the convolutional kernels (from 3 to 20) and reducing the number of convolutional layers to only 4 (from 8 in ConvNetQuake). Greater robustness to noise and distortions is obtained by applying max pooling after each convolutional layer. Second, two additional fully connected layers are incorporated. All of these changes result in a significant increase in the number of trainable parameters (to 70K, from 23K in ConvNetQuake) but not in the training time, as the depth of the network becomes slightly reduced. Figure 4.1 shows the proposed network architecture.

The network takes a multiple-channel seismogram window as input and outputs $K+1$ detection probabilities. The first $K$ values estimate whether the window contains a P-wave

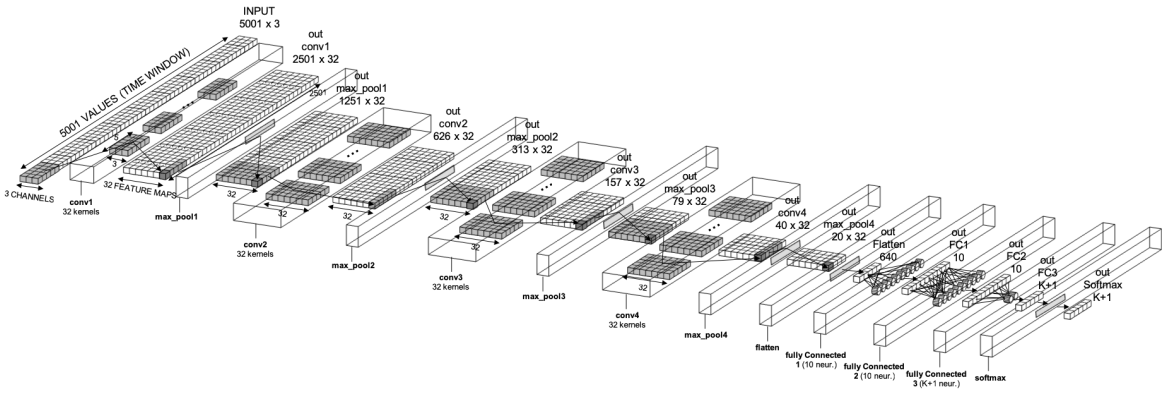


Fig. 4.1 Network architecture with a 3-channel input, 4 convolutional+max_pooling layers, 3 fully connected layers and a softmax layer.

triggered within one of the given source regions, and the last output is the probability that the window does not contain any P-wave. The input and output of the network are the same as those in ConvNetQuake, with the exceptions of the input window size (10 sec in ConvNetQuake; 50 sec in the best configuration of UPC-UCV) and the value of $K$ (6 for the OKLAHOMA dataset targeted by ConvNetQuake; 5 and 3 for the CARABOBO dataset targeted by UPC-UCV).

The input of the model is a window of, in the best configuration of UPC-UCV, 50 sec. As the signal is sampled at 100 Hz, the window contains 5000 samples. Each sample presents three values, one for each component (N-S, E-W, and up-down). This would result in a tensor of shape *[5000, 3]* (*[window size x sampling rate, number of channels]*). However, in the same way as in ConvNetQuake, UPC-UCV's network works with an input tensor of *[5000, 1, 3]* (*[window size x sampling rate, 1, number of channels]*), a more convenient shape to feed a ConvNet network, which is typically used to work with color images received as tensors with shape *[width, height, color channels]*. Thus, from a conventional ConvNet point-of-view, UPC-UCV's input is like a just-one-row image with three color channels. Figure 4.1 attempts to clarify this by showing the input tensor as a flat cuboid (while a color image would be a normal 3D cuboid).

The network has four convolutional layers, each with an associated max pooling layer. All of the layers have 32 convolutional kernels. Conventionally, the deeper layers of a ConvNet, which work with higher-level features, contain fewer kernels than the first layers. However, after testing different configurations, it was decided to retain the same configuration as in ConvNetQuake for this aspect.

The convolutions are applied in 1D fashion (only through the temporal axis), but the kernels are 2D to process the multiple input channels. Given a 2D kernel $k$ of size $s \times c$

(width $s$ over $c$ channels) of a given layer $l$, the weight matrix $W^{k,l} \in \mathbb{R}^{2s+1 \times c}$ is:

$$W^{(k,l)} = \begin{pmatrix} W^{(k,l)}_{-s,1} & \cdots & W^{(k,l)}_{-s,c} \\ \vdots & \vdots & \vdots \\ W^{(k,l)}_{0,1} & \cdots W^{(k,l)}_{0,0} \cdots & W^{(k,l)}_{0,c} \\ \vdots & \vdots & \vdots \\ W^{(k,l)}_{s,1} & \cdots & W^{(k,l)}_{s,c} \end{pmatrix} \tag{4.1}$$

Empirically, the value of the $s$ hyperparameter (size of the kernel) has been set to 20 (3 in ConvNetQuake). Thus, each convolutional kernel is a $20 \times c$ matrix, with $c$ denoting the number of output channels of the previous layer (3 for the first convolutional layer and 32 for the following ones). The $20 \times 3$ cuboids in figure 4.1 correspond to the convolutional kernels of the first layer, i.e., to Equation 4.1 when $l = 1$, $s = 20$ and $c = 3$. Each kernel is applied to all of the 20-size windows of the input tensor with a stride of 2 (the same as in ConvNetQuake). For each of these windows, the discrete convolution of the input 2D tensor Y with kernel $W^{k,l}$ at position $t$ of the input tensor (the kernel only moves in 1D) is given by:

$$(Y * W^{k,l})_t = \sum_{u=-s}^{s} \sum_{v=1}^{c} W^{k,l}_{u,v} * Y_{t+u,v} \tag{4.2}$$

The result of equation 4.2 is known as a *convolved feature map*. As each kernel is applied to different windows of the input tensor (different values of $t$), many convolved feature maps are obtained for each kernel (e.g., 2500 in the first convolutional layer). A bias and an activation function are applied to each convolved feature map to provide nonlinearity and obtain the final output of the layer (also known as a *rectified feature map*) at each position. The output at position $t$ of a convolutional layer $l$ and kernel $k$ is computed as:

$$Y^{(l)}_{k,t} = \sigma(b^{(l)}_k + (Y^{(l-1)} * W^{k,l})_{t'}) \tag{4.3}$$

in which $t'$ is the index of the input tensor. $t'$ ($t$ in Equation 4.2) depends not only on $t$ but also on the stride (2 in this case). $\sigma(x) = max(0,x)$ is the nonlinear ReLU (rectified linear unit) activation function, and $b^{(l)}_k$ is the bias term for kernel $k$ in layer $l$. After each convolutional layer, we apply a max pooling layer with a pooling window of size 5 and stride 3.

After the last convolutional layer, the resulting 2D tensor is flattened into a 1D tensor. This feature vector is then processed by three fully connected layers with ReLU activation functions (10 neurons for the first ones and $K+1$ neurons for the last one). The size of the last fully connected layer depends on the number of classes (two if only P-waves are being

detected and $K+1$ if the source region where the P-wave originated is also being inferred from $K$ predefined regions). The outputs of the last fully connected layer, the so-called *logits*, are raw prediction values with a range of $[-infinity, +infinity]$. To transform these values into probabilities (range of $[0,1]$, sum to one) a *softmax* function is applied. Each of the $K$ outputs of the network is the probability (a float value within the range of $[0,1]$) that the input window contains a P-wave that originated in one of the $K$ regions. One output of the network contains the probability that the input window does not contain a P-wave. This is the same output that is defined in ConvNetQuake.

The network is trained with 80% of the balanced dataset (48K windows), and the remaining 20% (12K windows) is used for the evaluation.

### 4.1.4 Experiment

The main goal of the experiments is to evaluate the detection performance of the different aspects (including the data preprocessing strategy and the network architecture) of the UPC-UCV method. The source region estimation performance is also addressed, but a deeper discussion about this aspect will take place in Section 4.2, focused this problem. All of the reported results are obtained when evaluating each configuration over a test dataset of previously unseen windows (20% of the dataset). The experiments were conducted on one of the computing clusters of the Computer Architecture Department of the UPC. The cluster is accessed through a Sun Grid Engine (SGE) batch-queuing system and comprises 200 heterogeneous nodes, the newest ones being equipped with 2x Intel Xeon E5-2630L v4 processors running at 2.20 GHz with 128 GB of RAM. The following table 4.1 gives an overview of the configurations used.

Table 4.1 Experiment setup summary.

| **Dataset** | **Pre-Processing Data** | **Model** | **Evaluation Architecture** | **Software Version** |
|---|---|---|---|---|
| CARABOBO | Normalize 3 channel<br>Under sampling balancing<br>Bandpass Filtering<br>Dataset split:<br>- 80% training<br>- 20% evaluation | ConvNet-Quake[101]<br>UPC-UCV[104] | Sun Grid Engine<br>200 nodes<br>Intel Xeon E5-2630L | obspy 1.0.2<br>Tensorflow 0.12.0<br>Python 2.7<br>Seisbos |

Before deciding the final configuration, different data preprocessing configurations (window size, frequency filtering, number of channels, etc.) and different network geometries and

parametrizations were tested. Table 4.3 summarizes the impacts of the different preprocessing choices on the performance. The obtained results confirm that all of the proposed modifications with respect to [101] exert a positive impact on the performance. The preprocessing modification with the highest impact is the slicing of the input streams into highly overlapping windows, which can be seen as a phase shifting-based data augmentation technique. The usage of the three channels, which is mainly aimed at improving source region estimation, exerts a small impact on the P-wave detection accuracy, as expected.

One important hyperparameter is the window size. Window sizes in the range of 10-60 sec have been tested. Figure 4.2 shows the impact of the window size on the detection accuracy for both UPC-UCV and ConvNetQuake. It is interesting to observe how the optimal window size differs between methods, with larger windows (approximately 50 sec) being better for the proposed method.

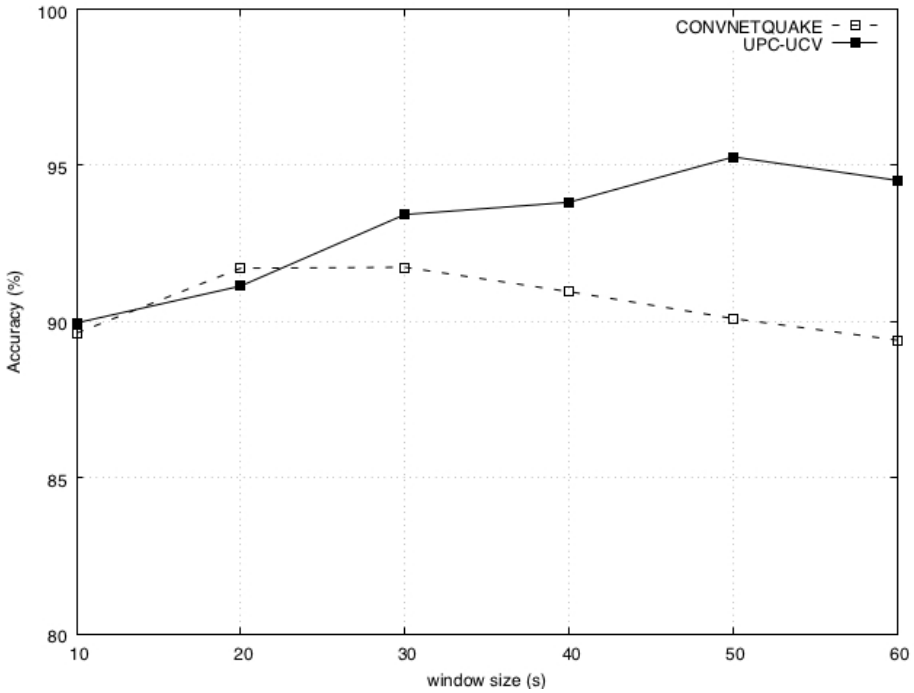


Fig. 4.2 Impact of window size on accuracy for both the reference model (ConvNetQuake) and the proposed model (UPC-UCV). Models are trained with the CARABOBO dataset.

Table 4.2 summarizes the results of UPC-UCV (with all the described preprocessing and network architecture improvements) in comparison with the reference method, ConvNetQuake. While the goal of this work is to provide reliable detection and source region estimation for the target region, its performance when trained and evaluated on the dataset employed in [101] (the OKLAHOMA dataset) has also been assessed to study the applicabil-

ity of the proposed network setup to different seismic data. Because the experiment is based on an already preprocessed dataset, the test is limited to a configuration of 10 sec windows without overlap. Although the UPC-UCV hyperparameters were not fine-tuned over this dataset, it achieves a 98.21% detection accuracy, while ConvNetQuake reaches a similar 97.32% (balanced) accuracy.

Table 4.2 Final P-wave detection results.

| **Preprocessing** | **Model** | **Win. size** | **Accuracy** |
|---|---|---|---|
| ConvNetQuake | ConvNetQuake | 10 | 81.95%* |
| ConvNetQuake | ConvNetQuake | 50 | 83.25%* |
| UPC-UCV | ConvNetQuake | 10 | 89.63% |
| UPC-UCV | ConvNetQuake | 50 | 90.10% |
| UPC-UCV | UPC-UCV | 10 | 89.96% |
| UPC-UCV | UPC-UCV | 50 | **95.27%** |

Detection accuracy results for both the reference model (ConvNetQuake) and the proposed model (UPC-UCV) on the CARABOBO dataset with two different window sizes (10 sec and 50 sec). *Accuracy values for the ConvNetQuake preprocessing are *balanced accuracies*, as the dataset is imbalanced.

Figures 4.3 and 4.4 compare test cases with satisfactory and wrong P-wave detections. Most of the false positives are windows shortly after the P-wave (out-of-phase detection). When applied in a real setup, these false positives could be avoided by introducing an exclusion window after the detection of the P-wave (if there is a true positive detection). In the worst cases, out-of-phase detection implies a false negative followed by a false positive. Irregular waveforms related to the heterogeneous characteristics and states of the sensors are the other main source of detection errors.

Regarding source region estimation, the method obtains a 95.68% accuracy with 3 geographical clusters and a 93.36% accuracy with 5 geographical clusters. These results are satisfactory considering that the source region is being estimated with the information of just one seismic station. A more detailed description of the source region estimation experiments will take place in Section 4.2, in which an alternative method will be proposed. Regarding the computational efficiency of UPC-UCV during inference time, processing a window (which happens every 10 sec of the input stream) takes an average of 4.4 milliseconds. Such performance would enable our network to process a one hour stream in less than 2 seconds, allowing an efficient real-time processing. This performance is very similar to that observed for ConvNetQuake, which takes an average of 3.8 milliseconds to process each window. Regarding the memory footprint, UPC-UCV (which has 70K parameters) occupies about 3.5 MB of memory during inference time. ConvNetQuake, which has only 23K parameters,

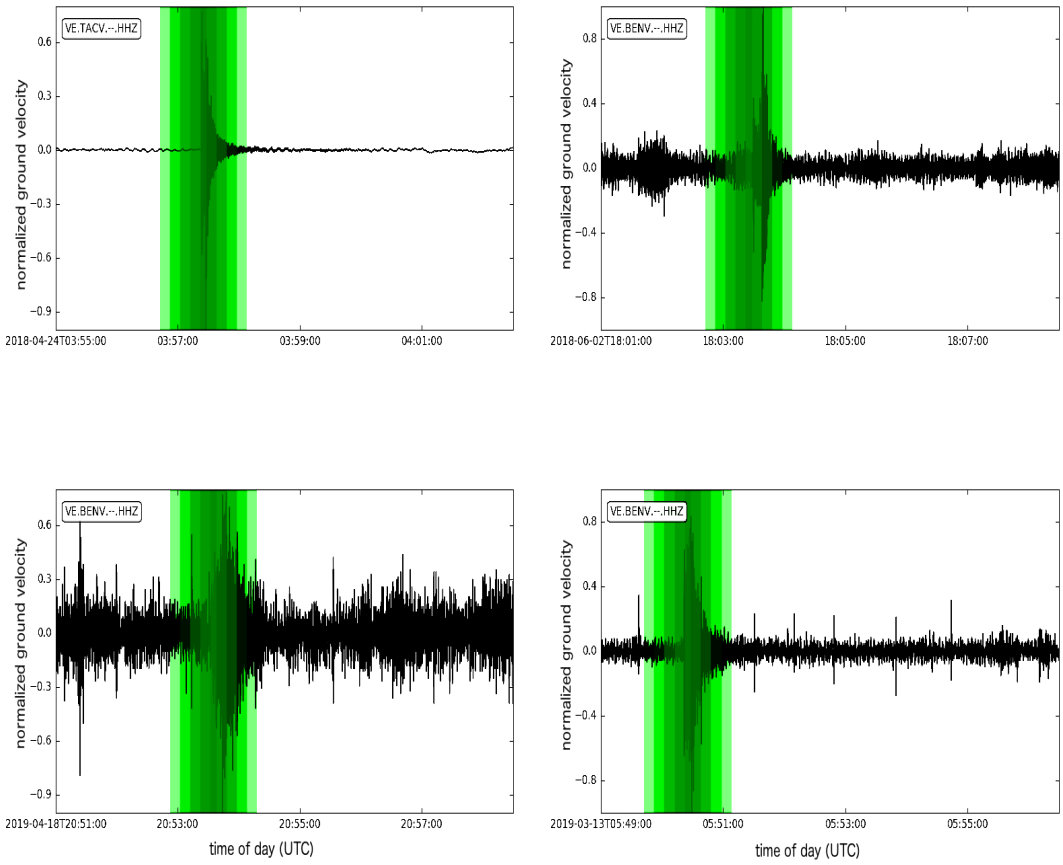


Fig. 4.3 Example seismograms (only Z component for readability) with P-waves successfully detected (true-positive windows are overlaid in green).

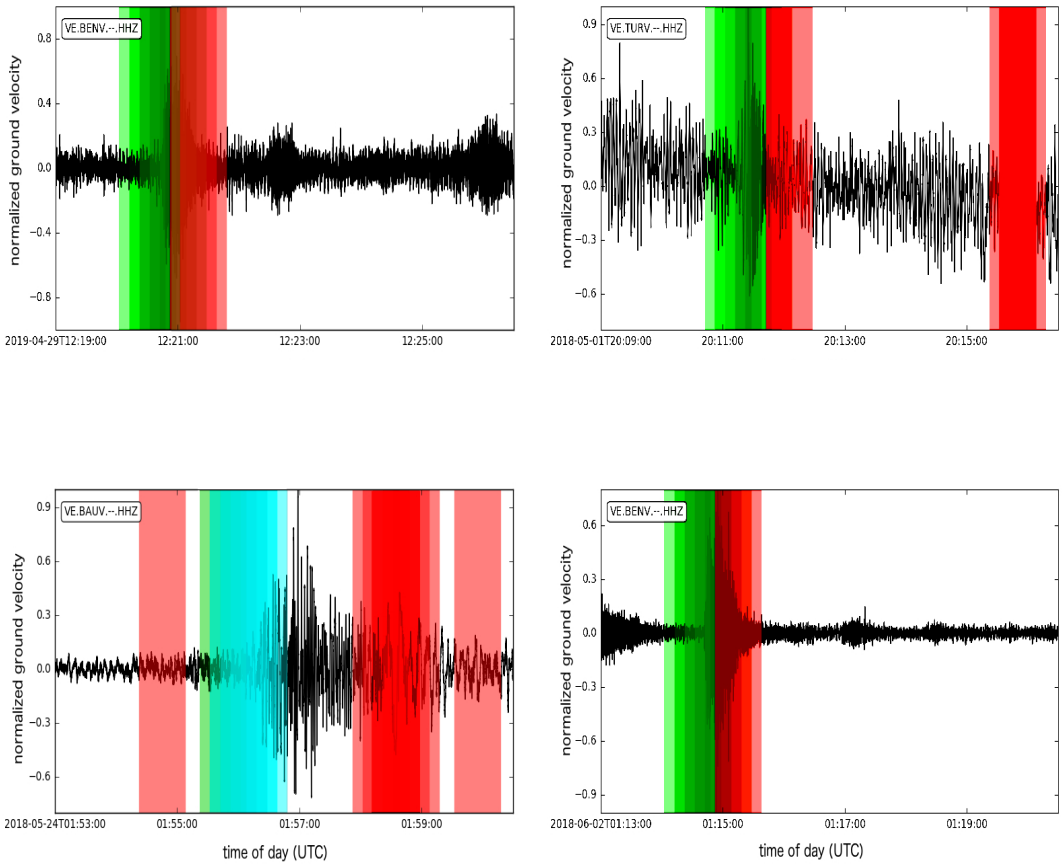


Fig. 4.4 Example seismograms (only Z component for readability) with detection errors (false positive in red and false negative in cyan).

occupies about 3 MB, that represents a similar cost. Some of the overheads introduced by TensorFlow are not actually proportional to the number of network parameters.

Table 4.3 Impact of different hyperparameters.

| Overlapping | Filtering [0.5, 10] Hz | Components | Accuracy |
|---|---|---|---|
| yes | yes | 3 | **95.27%** |
| no | yes | 3 | 89.9% |
| yes | no | 3 | 92.48% |
| yes | yes | 1 | 94.11% |

Impact of different hyperparameters on the detection accuracy of the proposed method on the CARABOBO dataset.

### 4.1.5 Conclusions

Results of the proposed UPC-UCV method, consisting of applying a convolutional neural network to single-station 3-channel waveforms for P-wave earthquake detection and source region estimation in north-central Venezuela, have been reported. In addition, the CARABOBO dataset of analyst-labeled P-wave arrivals, named after the studied seismic region, has been built and made public for reproducibility and benchmarking purposes. Both the UPC-UCV network and the CARABOBO dataset are the first developed for this geographic region. This convolutional network is inspired by ConvNetQuake developed by [101] but incorporates many improvements, both in the data preprocessing strategy and the network architecture, to yield higher detection accuracy (13.3 percentage point increase) for the new target seismicity. Regarding the preprocessing strategy, the main modifications are the usage of larger and overlapping windows, the balancing of the dataset and the application of a bandpass filter to input seismic signals. Regarding the network architecture, the main enhancements are the flattening of the convolutional part (fewer layers with larger kernels), max pooling and more fully connected layers. UPC-UCV achieves a 95.27% detection accuracy over a subset of windows of the CARABOBO dataset that were never seen during training, while ConvNetQuake obtains a 81.95% detection accuracy over the same subset. Regarding source region estimation, UPC-UCV achieves a 95.68% accuracy with $K = 3$ geographic partitions, while ConvNetQuake obtains an 84.58% accuracy. In the case of $K = 5$ geographic partitions, UPC-UCV reaches a 93.36% accuracy, compared to ConvNetQuake that yields an 82.08% accuracy. The dataset partitioning into $K$ geographical clusters is automatically performed by the k-means algorithm, and the optimality of these $K$ values has been assessed using the Elbow and the Silhouette methods. In addition, without any modification, this network yields excellent detection results when trained and evaluated

on the OKLAHOMA dataset, which corresponds to a totally different geographical region (98.21% detection accuracy; ConvNetQuake, fine-tuned for this dataset, achieves a 97.32% detection accuracy).

## 4.2 Epicentral Region Estimation using Deep Neural Networks

### 4.2.1 Introduction

Reliable earthquake detection and location algorithms are needed to properly catalog and analyze steadily growing seismic records. ANN have been employed in [74][101][120] and [61] to study earthquake detection and location of seismics events.

One of the most important tasks of seismic source characterisation is the estimation of epicentre location of event over a geographical area. Here, we study the application of the method described in Section 4.1 to the earthquake source region estimation problem, i.e. mapping positive events to geographical partitions of the study region. The proposed method is essentially a signal window classifier. With only two classes/labels (P-wave or not P-wave), the method behaves similar to a P-wave detector. Increasing the number of classes and properly labeling the windows that contain a P-wave enable inference of properties about the event that triggered the detected P-wave. One possibility, the one addressed in this work, is attempting to determine the epicentral source region of the event. First, the study area is partitioned into $K$ geographic subdivisions. Second, the positive windows are labeled with the identifier of the subdivision to which the event source location belongs. Third, a classifier with $K+1$ classes is trained. The first step, geographic partitioning, can be performed in many different ways, including manually.

The study region (see figure 4.5), characterized by shallow seismicity, includes the central part of the Caribbean mountain system with important cities such as Caracas, Valencia and Maracay. The magnitudes of the earthquakes range between 1.1 and 5.2 Mw (figure 4.6 shows the magnitude distribution of the events); P- and S- wave arrival times are reported in the FUNVISIS catalogs [37] [38]. Most seismic events belong to a seismic swarm with epicenters located within northwestern Valencia (Carabobo state). The seismicity of the region is associated with the San Sebastián and La Victoria fault systems, along with some minor faults such as the Las Trincheras and Morón faults, all of which are right lateral strike-slip faults [11] [93]. The San Sebastián and La Victoria faults belong to the continental scale Boconó - San Sebastián - El Pilar fault system along the interface between the Caribbean and South American plates [10].

It is worthy of mentioning that the study region in figure 4.5 concentrates a large population and a variety of big industries, and the human and industrial activities induce a high level of seismic noise. Unfortunately, some of the seismic stations of the local FUNVISIS network failed during the recording period of this study, and the seismic processing of the

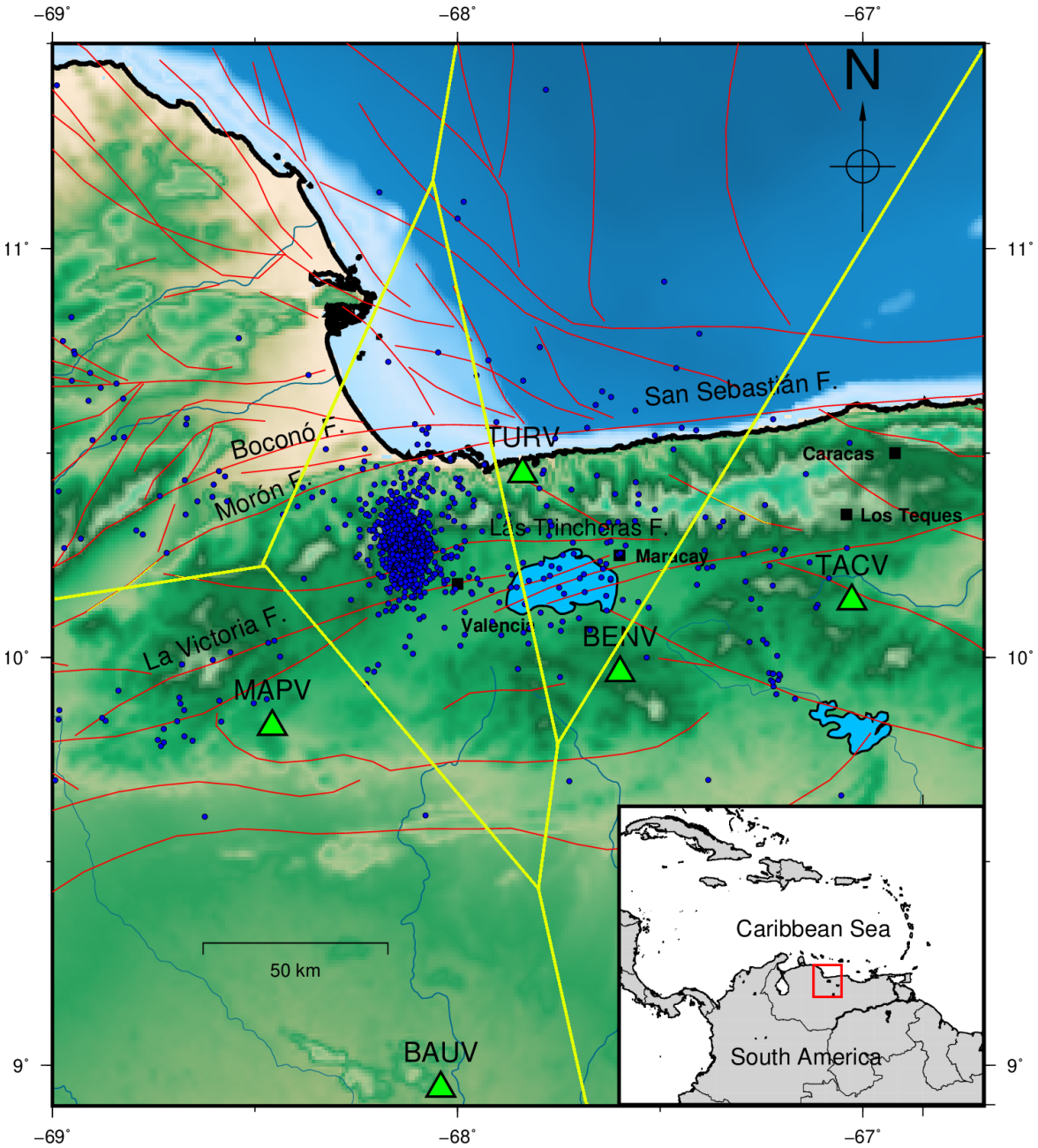


Fig. 4.5 Epicenters of seismic activity in the center-north region (67W - 69 W, 9.5N - 11.5N) of Venezuela (circles), seismological stations (triangles), and active faults (red lines) compiled by [15]; inset: relative location of the study area at the Caribbean-South American plate interaction. The figure also shows the geographical partitioning (obtained with k-means and K=5) of the events (yellow lines).

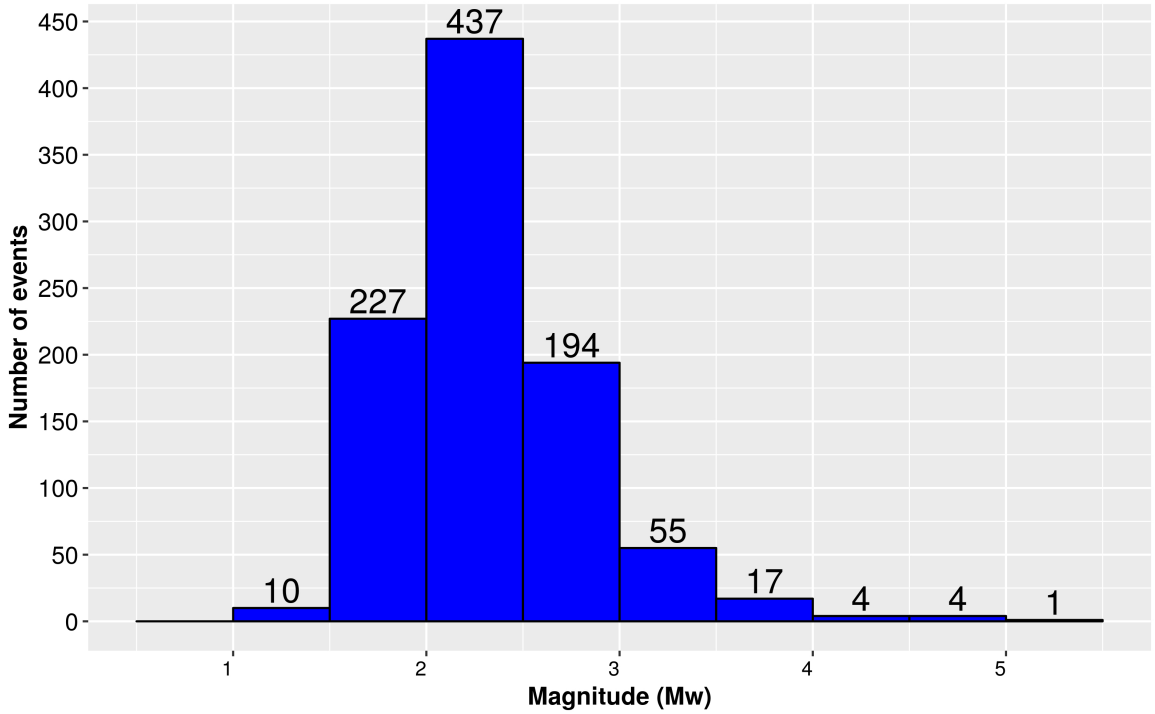


Fig. 4.6 Magnitude distribution of the events.

recorded data was carried out by trained analysts assisted by STA/LTA detection software. The combination of these facts leads to a detection deficiency of small events, and reflect on the apparent scarcity of the seismic activity below 2 Mw, as depicted in figure 4.6. This cumulative earthquake distribution does not follow a standard frequency magnitude relation, such as the Gutenberg–Richter law. However, more elaborate earthquake detection procedures could reveal several small events, allowing a future enrichment of the CARABOBO dataset. This work is based on dividing the geographical distribution process as follows:

- Carry out an automatic distribution using automatic clustering algorithms.
- Use an approach associated with manual distribution provided by an expert seismologist with Spatial Projection techniques.

### 4.2.2 Related Work

ANNs represent a suitable framework for earthquake detection and location due to their proficiency in complex pattern recognition. Since the mid-1990s, there has been a tremendous amount of contributions that vary in terms of the underlying learning technique, network architecture and functionality, and spatial extent of the documented application, among

other distinct aspects. Early works based on supervised learning involve simple FFNNs, which were followed by some scattered cases of RNNs, and we then find some recent applications of very deep CNNs. FFNNs for seismic phase detection have been proposed in [25][26][34][109][110][123] and applied to local and regional earthquake data. In particular, the testing dataset used in [25] consists of slightly more than 850 local events, while [123] employed over 1200 seismograms of the IRIS network for training and validation. Alternatively, the FFNNs introduced in [102] make use of a dataset of P-wave signals from 193 teleseismic events recorded at 3 short period stations in central Finland. In [43], the singular network IUANT2 presents a problem adaptive structure that is inferred during the training process. This technique is used for phase picking by employing records of 342 local earthquakes recorded by 23 different stations (approximately 5K traces). [113] implement an RNN for real-time detection of small magnitude (below 2.5 Mw) earthquakes in populated and noisy areas owing to an elaborate design of filter banks processing the STA/LTA ratios of seismic waveforms. This process allows for the extraction of relevant frequencies from input data, making this network less prone to false detection occurrences. The initial training relies on 170 events, including regional and teleseismic earthquakes, and the full operating network was tested during different time periods in 2009. Recent applications of RNNs to early warning systems and to earthquake prediction are given in [18] and [53], respectively.

For locating the earthquake hypocenter, as part of the core information in seismic catalogs, only a few scattered estimation approaches incorporating ANNs can be found. For instance, ConvNetQuake maps hypocenters to a cluster of different geographical areas based on a Voronoi partition. Alternatively, the detection CNN proposed by [60] is trained on an earthquake swarm of 2000 events recorded by several local three-component stations. During testing, this network locates approximately 900 earthquakes with standard deviations of nearly 56 m, 124 m and 136 m along the east-west, north-south and vertical directions, respectively. Lastly, the detection CNN introduced in [114] constructs a 3D probability volume of location likelihood inside the Earth. This work employs the OKLAHOMA dataset for training and evaluating the model, and reports an average error of 4.9 km in the epicenter location estimation (1.0 km in the depth estimation).

### 4.2.3 Methodology

#### Data

As in Section 4.1, the experiments have been done over the CARABOBO dataset. It contains data (signals in miniSEED format and metadata in Nordic format) collected among April 2018 and April 2019 through 5 seismological stations (BAUV, BENV, MAPV, TACV, and

TURV) in northcentral Venezuela, specifically on the region between the coordinates 9.5° to 11.5° N and 67.0° to 69.0° W. The earthquakes have a magnitude ranging from 1.7 to 5.2 Mw distributed over the region shown (see figure 4.7), whose epicenters are located on the northcentral states of Carabobo, Aragua and Miranda. These zones contain a set of interconnected faults such as: San Sebastián and La Victoria that make up an important seismic area, belong to a continental scale, converging to a larger fault system called Boconó. As well as El Pilar fault system that lies the Caribbean and South American plates.

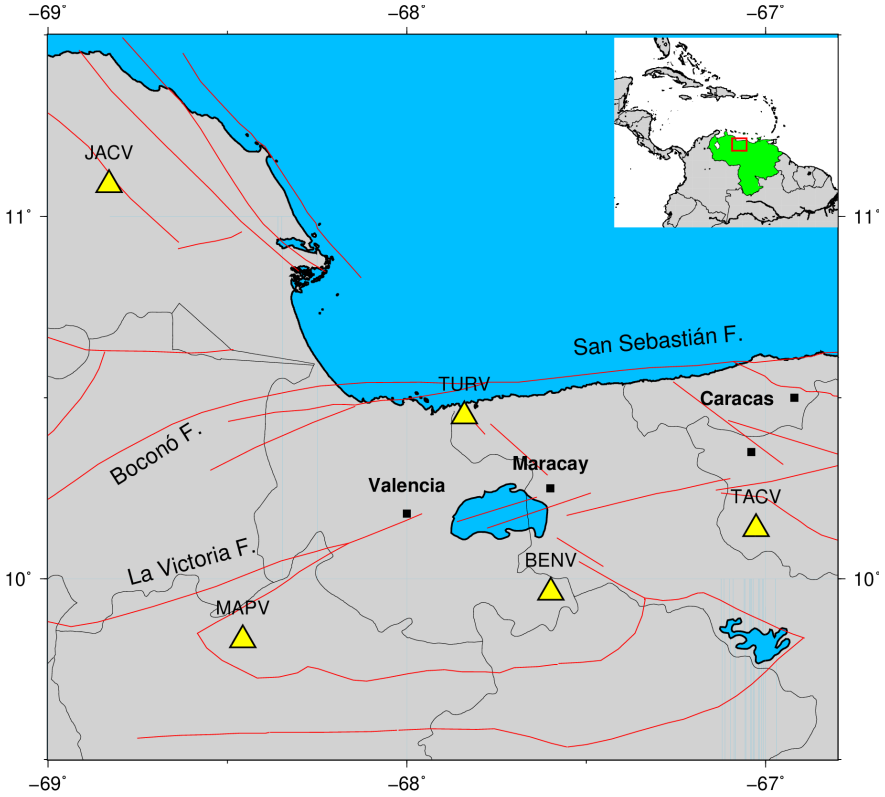


Fig. 4.7 Geographical distribution of the Boconó, La Victoria and San Sebastián faults, and locations of FUNVISIS stations.

### Earthquake Source Region Estimation

Source region estimation is a relaxed version of the earthquake location problem that consists on, first, partitioning a study area into $k$ geographic subdivisions and, second, attempting to determine to which one the earthquake epicenter belongs. Several works have demonstrated the possibility to estimate the source region of an earthquake from a single-station 3-channel waveform [101][102][104]. In this work, source region estimation is approached as a multiclass classification problem. First, the wave is divided into three-channel temporal

windows of a fixed-size, and these windows will be classified into k+1 classes: one class for windows not containing a P-wave (negatives) and *k* classes for windows containing a P-wave (positives) but classified into the *k* geographic regions that we want to discriminate.

### Partitioning the Study Area into Geographic subdivisions

In order to perform source region estimation, it is first necessary to partition the study area into multiple geographic subdivisions and to define the membership criteria.

Our work had two approaches, in the first instance of this work, it is performed by clustering all of the event locations in the dataset wit k-means [68] for different values of K (see figure 4.5). The tested values of *K* are the optimal ones resulting from the application of both the Elbow method and the Silhouette method. Figure 4.8 shows the results of both methods for finding the appropriate number of source regions.

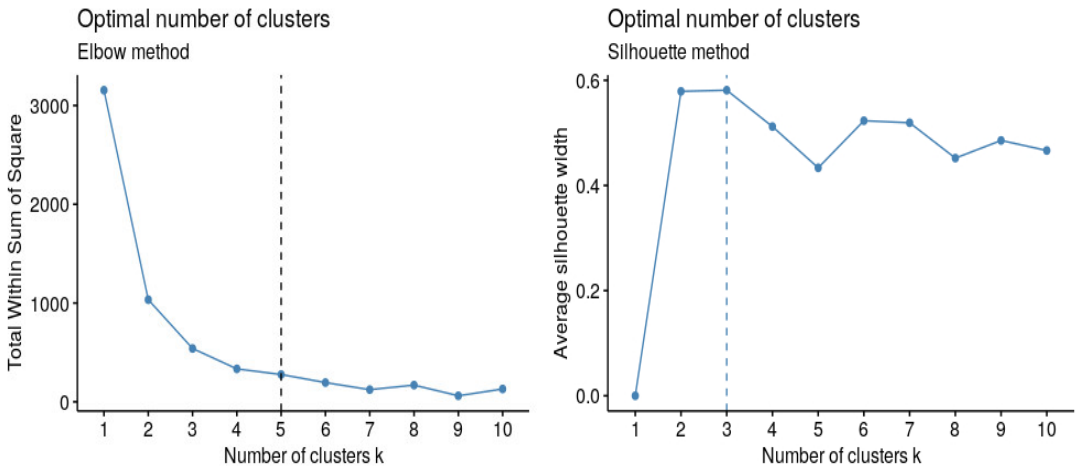


Fig. 4.8 Results of the methods for finding the appropriate number of source regions (clusters). Elbow method (left), and Silhouette method (right).

On the other hand, we have developed an approach based on the geographical subdivision that the FUNVISIS expert gave us (see figures 4.9 and 4.10), four different partitions were obtained (k = 3, 4, 5, 10). All these delimited by irregular polygons covering the main seismic faults of Venezuela, according to the following study [71].

Seismic hazard studies carried out north and central Venezuela by FUNVISIS analyze the seismic activity in shallow seismogenic regions [50] [51]. These regions were defined as polygons around each fault segment, which were chosen according to their tectonics and degree of seismicity (see figure 4.10). At first, 10 regions were delimited, which were later regrouped in new bigger regions with similar seismic-tectonic characteristics, leaving the whole study area split into the 4 regions shown in figure 4.9.

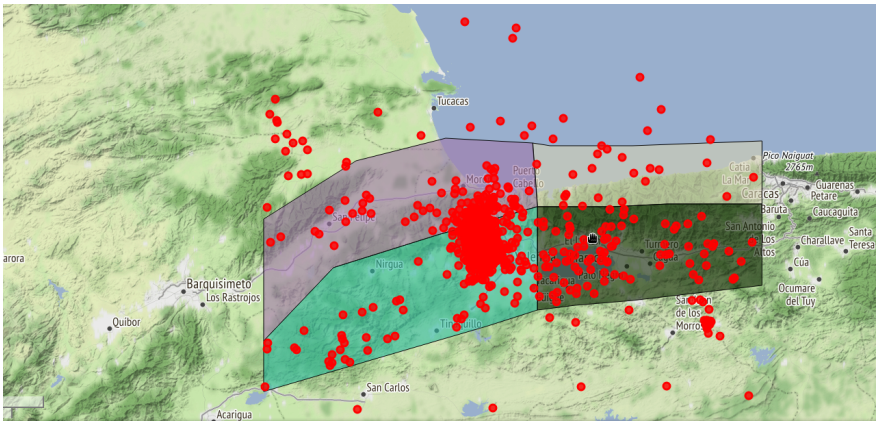


Fig. 4.9 Clustering the locations of all the earthquakes with fault-based geographic partitioning on 4 zones.

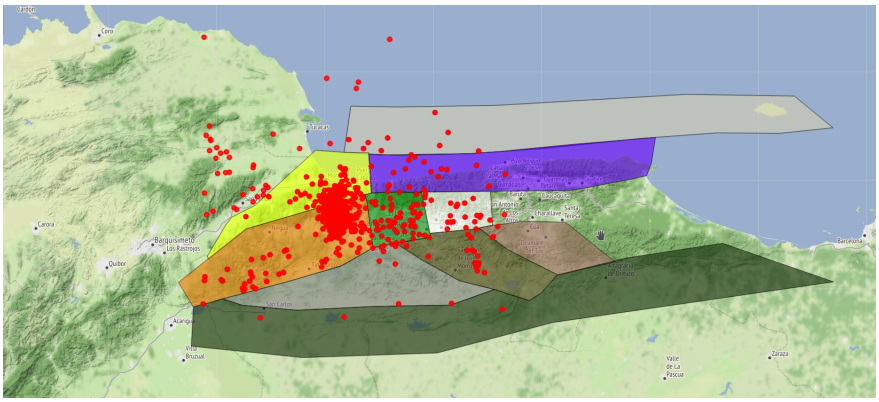

Fig. 4.10 Clustering the locations of all the earthquakes with fault-based geographic partitioning on 10 zones.

### Data Preprocessing

For the training of the model, a filtering is carried out first, purging the most dispersed events over the study region in order to reduce overfitting.

Input waveforms are first normalized and divided into single-station streams. Each 3-channel single-station stream is split into multiple 3-channel temporal windows of a fixed size. With the events information obtained from the metadata files, we divide the windows into $k+1$ classes: one class for windows not containing a P-wave(negatives) and $k$ classes for windows containing a P-wave(positives) but classified into the $K$ geographic regions that we want to discriminate. With a 50 sec./window our preprocessing stage yields $12,685$ positives and $84,911$ negatives. The classification of windows among the different K regions

is done using clustering the locations of all the earthquakes with a fault-based geographic partitioning provided by a seismologist.

Figures 4.9 and 4.10, we can observe the several segmentations granted by FUNVISIS and the distribution of the events over the study region that will be used for the comparison of the k-means clustering method.

### Spatial Data Preprocessing

In the Carabobo dataset, each of its records contains coordinates that indicate the epicenter of seismic events. To deal with this information is necessary to have a reference frame capable of making sense of these data to view and manipulate them and thus feed the CNN model. This reference frame is known as Coordinate Reference System (CRS) [54], classified in Geographical Coordinate System (GCS) and Projected Coordinate System (PCS). Figure 4.11 shows the representation of a specific place on earth, on the left, we have its three-dimensional visualization on the globe with a GCS, and on the right, the two-dimensional projection of this same point projected in PCS.

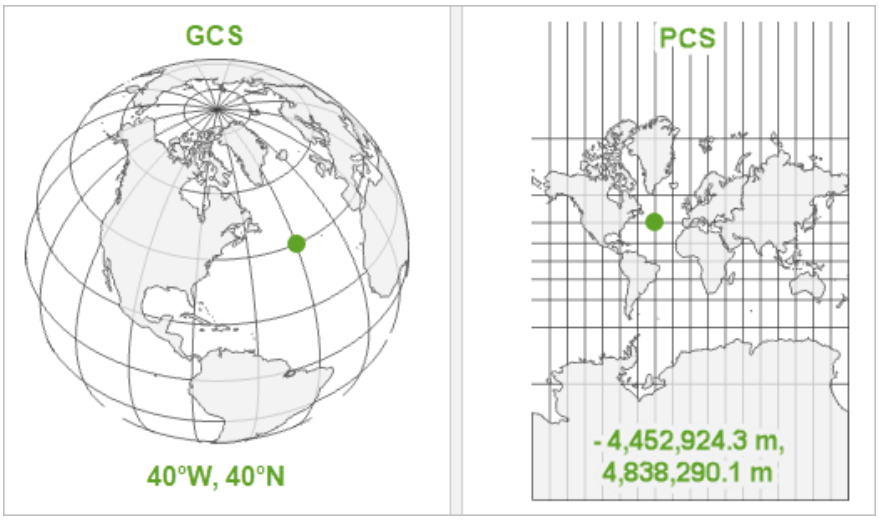


Fig. 4.11 Types of coordination reference system.

The first is a reference framework that defines the locations of features on a model of the earth. It's shaped like a globe-spherical. Its units are angular, commonly degrees. It's the best for the location and visualization of elements, but distance measurements have a distortion when using latitude and longitude due to the earth's shape. The second is flat. It contains a GCS, but it converts that GCS into a flat surface using a projection algorithm [54] and it is excellent for performing calculations involving distance measurements over geographic areas.

For our dataset, we have selected the GCS called WGS84 to work with our initial data and its corresponding PCR named Pseudo-Mercator, which is by de facto the standard for web services such as Google Maps, Bing Maps, OpenStreet, among others. The coordinates provided generate perimeters with undefined shapes, which in Spatial Data are known as irregular polygons. In order to handle this information, it is necessary to create a JSON (see listing 1) structure that could store an indeterminate number of points that formed these polygons.

```
{
 "cluster_type": "RCPoligons",
 "clusters":[

  {
  "id": 1,
  "label": "Zone_01",
  "points": [
     {"x": 10.41413, "y": -67.90393},
     {"x": 10.42613, "y": -66.10873},
     {"x": 10.39413, "y": -66.90323}]
  },

  {
  "id": 2,
  "label": "Zone_02",
  "points": [
     {"x": 10.71413, "y": -66.20393},
     {"x": 10.62613, "y": -66.40873},
     {"x": 10.79413, "y": -67.10323},
     {"x": 10.22763, "y": -66.30822},
     {"x": 10.11113, "y": -67.09223}]
  },

    ...
            ]
}
```

Listing 1 JSON Structure

The JSON file contains a key called clusters, whose value is composed of a list of elements that describe the n-regions of which the coordinates define each region's perimeter.

An essential section of the file is the points key, because here the coordinates that delimit the boundary and shape of the area are stored; unlike other approaches here, there is no fixed number of elements when forming the list.

For instance, the list 1 shows an irregular polygon is made up of 3 points while the second zone is made up of 5. After defining the type of CRS and the JSON structure that stores the shape of the polygons, we proceeded to use Computational Geometry techniques to assign each of the events to a specific cluster and label it for use model training process. To this, we face the Point in polygon (PIP) problem in which it is decided whether or not a point is in an irregular polygon, described below:

Given a point $R$ and a polygon $P$ represented by $n$ points: $P_0, P_1, ..., P_{n-1}, P_n = P_0$, determine whether $R$ is inside or outside the polygon $P$. When a line is drawn from $R$ to other point $S$ that is wagered to extend outside the polygon. If this line $\overline{RS}$ crosses the edges $e_i = \overline{P_i P_{i+1}}$ of the polygon an odd number of times, the points is inside P, otherwise it is outside.

To carry out the PIP queries, we used Shapely's binary predicates [45] that implement these algorithms to assess the topological relationship between geographic objects. It is based on the widely deployed Geometry Engine Open Source (GEOS), allowing work with three main Point, Line String, and Polygons objects. These algorithms are used to determine whether a seismic event falls within the perimeter of a given zone.

### 4.2.4 Experiment

This work's main objective is to assess the effectiveness of source region estimation using a geographic partitioning provided by an expert and determine the impact on the prediction's improvement using a PCR transformation, compared with the approach automatically generated with k-means as in [104]. The UPC-UVC network's best configuration was taken as a basis, considering the network's parametrization and geometry to carry out this new approach.

The experiments were performed on equipment provided by the Computer Architecture Department of the UPC. The device had Intel(R) Core(TM) i7-3770 CPU running at 3.40GHz and 8GB in RAM. With this configuration, the training process was carried out between 3 to 4 hours per model. An overview of the job configuration is shown in the table 4.4 below.

In the first instance, a set of experiments were performed to determine if Spatial-Data techniques within the pre-processing of the training data increased the accuracy of the model prediction results. The data in the Table 4.5 describe the experiment; in the first column, we partition the study area into k regions; in the second column, we have the experiment's result without using a pre-processing, taking by default a GCS as in the works [77][104]. In the last column, we have the effect after transforming the original data and converting them into

Table 4.4 Experiment setup summary.

| Dataset | Pre-Processing Data | Model | Evaluation Architecture | Software Version |
|---|---|---|---|---|
| CARABOBO | Normalize 3 channel<br>Under sampling balancing<br>Bandpass Filtering<br>Dataset split:<br>- 80% training<br>- 20% evaluation | ConvNet-Quake[101]<br>UPC-UCV[104]<br>UPC-UCV-GEO[24] | Intel(R) Core(TM) i7-3770 CPU 3.40GHz 8GB RAM | obspy 1.0.2<br>Tensorflow 0.12.0<br>Python 2.7<br>Seisbos |

a PCS. As shown in the table, the post-transformation result obtains an accuracy higher than 90% in each of the cases.

Table 4.5 Forecasting impact of source region estimation based on Coordinate Reference System

| K Zones | Accuracy GCS | Accuracy PCS |
|---|---|---|
| 4 | 86.52% | 95.43% |
| 10 | 87.72% | 91.78% |
| 16 | 88.93% | 91.43% |

Table 4.6 summarizes the results of UPC-UCV-GEO obtained using spatial data pre-procesing. The results of ConvNetQuake and UPC-UVC are provided for comparison. For a small number of geographic subdivisions (3-5), the obtained results don't enable to confirm the target hypothesis. The partitioning into 4 regions recommended by the expert (UPC-UCV-GEO with K=4) provided an accuracy of 95.43%, just slightly above than the results for a k-means based partitioning (UPC-UCV) with K=5 (93.36%) and slightly below than the results for a k-means based partitioning (UPC-UCV) with K=3 (95.68%). However, the hypothesis seems to be confirmed for a more fine-grained partitioning (K=10), as UPC-UCV-GEO obtains an accuracy of 91.43% while the accuracy of UPC-UCV degradates to 66.10%.

### 4.2.5 Conclusions

In this work, we have evaluated the hypothesis that the accuracy of methods (such as [101] and [104]) for the automated estimation of the epicentral source region of a seismic event is increased if the geographical partitioning is performed considering the regional geophysical

Table 4.6 Source region estimation results

| K Zones | Model | Accuracy |
|---|---|---|
| 3 | ConvNetQuake | 84.58% |
| | UPC-UCV | 95.68% |
| 4 | UPC-UCV-GEO | 95.43% |
| 5 | ConvNetQuake | 82.08% |
| | UPC-UCV | 93.36% |
| 10 | UPC-UCV-GEO | 91.78% |
| | UPC-UCV | 66.10% |

characteristics. The UPC-UCV-GEO deep convolutional neural network is applied over the CARABOBO dataset, consisting of three-channel seismic waveforms recorded in north-central Venezuela from April 2018 to April 2019. Instead of partitioning the data with K-means, we have applied several geographical tessellations provided by seismologists from the study area.

While the obtained results for a small number of geographic subdivisions are not better than the ones obtained with k-means clustering, the good results obtained with a large number of subdivisions (91.78% with K=10) outperform the k-means approach (66.10%). It should be noted that to obtain these results, the use of spatial-based techniques significantly improved the final model. This confirms the target hypothesis that the source region estimation accuracy is significantly increased if the geographical partitioning is performed considering the regional geophysical characteristics such as the tectonic plate boundaries.

# General Conclusions

In the last decades, we have seen an explosion of data in the world due to the digital revolution and the massive use of technological tools by persons and companies such as social networks, mobile devices, geo-location, and all kinds of sensors connected to the network. In the same way, the growing increase in computing power and storage has led to deep learning algorithms to become very relevant. In this thesis, novel deep learning methods were developed to leverage these growing computational and data resources. We focused on two main research directions, a strategy for executing distributed deep learning workloads over an HPC infrastructure and the application of deep neural networks to approach the earthquake detection and source region estimation problem.

Regarding the first part of this work, a technology based on Apache Spark was used to distribute deep learning workloads on a real-world, petascale, HPC setup, the MareNostrum supercomputer. Different workloads and deployment setups (number of nodes, parallelism configuration, etc.) were tested to evaluate the performance and scalability of the proposed software stack. Insights into how the job configuration on a traditional HPC setup can be optimized to efficiently run this kind of workloads were provided. The deployment was also tested in a use case study that analyzed the usage CNNs to curate and filter user-generated content for digital marketing tasks. We conclude that relying on Apache Spark to deploy deep learning workloads over a traditional HPC setup is feasible, minimizing deployment costs and enabling a systematic tuning of the different configuration parameters at the application level and infrastructure level. Nonetheless, the effective scaling is strongly limited by the synchronous parallelism approach applied by the 1.5 DL4J version. The derived conclusions should be useful to guide similarly complex deployments in the future. Future research could explore ways to overcome the identified limitations, such as replacing the synchronous mechanism by a hybrid approach in which synchronization just takes place within fixed-size node sets.

Regarding the second part of this work, a novel method, based on the usage of deep neural networks, for earthquake P-wave detection and source region was developed and evaluated. Models with hundreds of different parametrizations were trained in parallel on a medium-size

cluster of commodity servers. The method, called UPC-UCV, consists of applying a convolutional neural network to single-station 3-channel waveforms. It was applied to data recorded by the broadband stations of the Venezuelan Foundation for Seismological Research in the region of 9.5°–11.5°N and 67.0°–69.0°W during the time period from April 2018 to April 2019. A dataset, named CARABOBO, was also built and made public for reproducibility and benchmarking purposes. The method improves the detection accuracy of the state-of-the-art method ConvNetQuake by almost 10%. Regarding source region estimation, a novel approach consisting on using geographical tessellations provided by seismologists from the study area was compared to the state-of-the-art approach of geographically partitioning the data with k-means clustering. While the obtained results for a small number of geographic subdivisions are not better than the ones obtained with k-means, the results obtained with a large number of subdivisions (91.78% region estimation accuracy with K=10) outperform the k-means approach (66.10%). This confirms the target hypothesis that the source region estimation accuracy is significantly increased if the geographical partitioning is performed considering the regional geophysical characteristics such as the tectonic plate boundaries.

Several DL models have been developed to tackle the problem of earthquake recognition and the location of the epicentral region when processing seismic archives. These models are ranked according to their detection efficacy to propose optimal configurations that can be later applied to new datasets. This ranking and analysis serve to identify the best NN architecture and hyperparameterization for a given seismic dataset. The results are more general than Hyperband, which is only capable of parameter tuning for a given NN architecture. Future studies could investigate the possibility of applying the proposed method to the problem of S-wave detection on the CARABOBO dataset. Furthermore, future research should certainly study the application of the latest advances in deep learning (e.g. transformers, autoregressive models, diffusion models, etc.) to fundamental problems in seismology.

# Acronyms

**AN** artificial neurons. 6

**ANNs** Artificial neural networks. 6

**BoW** Bag of Words. 34

**CNN** Convolutional Neural Network. 8

**CRS** Coordinate Reference System. 62

**DFS** distributed file system. 20

**DL** Deep Learning. 1

**DL4J** Deeplearning4j. 16

**DNNs** deep neural networks. 2

**FAST** Fingerprint and Similarity Thresholding. 43

**FC** Fully-connected. 21

**FFNNs** Feed forward neural networks. 6

**FUNVISIS** Venezuelan Foundation for Seismological Research. 2

**GC** Garbage Collector. 18

**GCS** Geographical Coordinate System. 62

**GEOS** Geometry Engine Open Source. 64

**GPD** Generalized phase detection. 43

**GPFS** General Parallel File System. 20

**HDFS** Hadoop distributed file system. 20

**ILSVRC** ImageNet Large Scale Visual Recognition Challenge. 27

**JNI** Java Native Interface. 18

**JVM** Java Virtual Machine. 18

**KNL** Knights Landing. 16

**LTA** Long-time average. 42

**MLP** Multilayer Perceptrons. 6

**NGR** New generic recognition. 34

**NN** Neural Networks. 1

**PCS** Projected Coordinate System. 62

**PIP** Point in polygon. 64

**RBF** Radial basis function. 34

**RDD** Resilient Distributed Dataset. 19

**RNNs** recurrent neural networks. 17

**SGD** Stochastic Gradient Descent. 11

**SGE** Sun Grid Engine. 48

**SNR** Signal-to-noise ratios. 41

**STA** Short-time average. 42

**SVM** Support vector machine. 34

**UGC** user-generated content. 26

**VBI** Visual brand identity. 31

# References


[ola] Olapic. earned content platform.

[chu] Chute. enterprise ugc.

[cur] Curalate.

[4] Ajitesh Kumar, Data Analytics (2020). Java implementation for rosenblatt perceptron. https://vitalflux.com/java-implementation-for-rosenblatt-perceptron/.

[5] Ariel Ellison (2015). Earthquakes and subduction zone seismicity. http://ffden-2.phys.uaf.edu/webproj/212_spring_2015/Ariel_Ellison/Ariel_Ellison/page2.html.

[6] Patrick D. Smith, Packt Publishing (2018). Hands-on artificial intelligence for beginners. https://www.packtpub.com/product/hands-on-artificial-intelligence-for-beginners/9781788991063.

[7] Allen, R. (1978). Automatic earthquake recognition and timing from single traces. *Bulletin of the Seismological Society of America*, 68(5):1521–1532.

[8] Allen, R. (1982). Automatic phase pickers: Their present use and future prospects. *Bulletin of the Seismological Society of America*, 72(6B):S225–S242.

[9] Anderson, K. R. (1982). Syntactic analysis of seismic waveforms using augmented transition network grammars. *Geoexploration*, 20(1):161–182.

[10] Audemard, F. (2007). Revised seismic history of the el pilar fault, northeastern venezuela, from the cariaco 1997 earthquake and recent preliminary paleoseismic results. *Journal of Seismology*, 11(3):311–326.

[11] Audemard, F. A., De-Santis, F., Singer, A., and Ramos, C. (1995). Sistema de falla de la Victoria, Venezuela norcentral: trazas activas, complejidades estructurales, cinemática y sismicidad asociada. In de Energía y Minas, M., editor, *Trabajos de autores venezolanos presentados en el IX Congreso Latinoamericano de Geología*, pages 1–15, Caracas, Venezuela.

[12] Baer, M. and Kradolfer, U. (1987). An automatic phase picker for local and teleseismic events. *Bulletin of the Seismological Society of America*, 77(4):1437–1445.

[13] Baillard, C., Crawford, W. C., Ballu, V., Hibert, C., and Mangeney, A. (2014). An automatic kurtosis-based P- and S-phase picker designed for local seismic networks. *Bull. Seism. Soc. Am.*, 104(1):394–409.

[14] Bakhshi, S., Shamma, D. A., Kennedy, L., and Gilbert, E. (2015). Why we filter our photos and how it impacts engagement. In *Proceedings of the Ninth International Conference on Web and Social Media, ICWSM 2015, University of , Oxford, UK, May, 2015, pp. 12–21*, pages 26–29.

[15] Beltrán, C. (1993). Mapa Neotectónico de Venezuela. 1:2,000,000 scale, FUNVISIS.

[16] Benz, H. M., McMahon, N. D., Aster, R. C., McNamara, D., and Harris, D. B. (2015). Hundreds of earthquakes per day: The 2014 Guthrie, Oklahoma, earthquake sequence. *Seismol. Res. Lett.*, 86(5):1318–1325.

[17] Bergen, K. J. and Beroza, G. C. (2018). Detecting earthquakes over a seismic network using single-station similarity measures. *Geophys. J. Int.*, 213(3):1984–1998.

[18] Bhandarkar, T., Vardaan, K., Satish, N., Sridhar, S., Sivakumar, R., and Ghosh, S. (2019). Earthquake trend prediction using long short-term memory rnn. *International Journal of Electrical and Computer Engineering (IJECE)*, 9:1304–1312.

[19] Brown, J., Beroza, G., and Shelly, D. (2008). An autocorrelation method to detect low frequency earthquakes within tremor. *Geophysical Journal International*, 35(16):1–5.

[20] Chen, C. H. (1984). On a segmentation algorithm for seismic signal analysis. *Geoexploration*, 23(1):35–40.

[21] Cichowicz, A. (1993). An automatic s-phase picker. *Bulletin of the Seismological Society of America*, 83(1):180–189.

[22] Clark, M., Black, H. G., and Judson, K. (2017). Brand community integration and satisfaction with social media sites: a comparative study. *Journal of Research in Interactive Marketing*, 11:39–55.

[23] Clarke, T. and Costall, A. (2008). The emotional connotations of color: A qualitative investigation. *Color Research & Application*, 33:406–410.

[24] Cruz, L., Tous, R., Otero, B., Alvarado, L., Mus, S., and Rojas, O. (2022). Epicentral region estimation using convolutional neural networks. In Nicosia, G., Ojha, V., La Malfa, E., La Malfa, G., Jansen, G., Pardalos, P. M., Giuffrida, G., and Umeton, R., editors, *Machine Learning, Optimization, and Data Science*, pages 541–552, Cham. Springer International Publishing.

[25] Dai, H. and MacBeth, C. (1995). Automatic picking of seismic arrivals in local earthquake data using an artificial neural network. *Geophysical Journal International*, 128(3):758–774.

[26] Dai, H. and MacBeth, C. (1997). The application of back-propagation neural network to automatic picking seismic arrivals from single-component recordings. *Journal of Geophysical Research: Solid Earth*, 102(B7):15105–15113.

[27] Dastres, R. and Soori, M. (2021). Artificial Neural Network Systems. *International Journal of Imaging and Robotics (IJIR)*, 21(2):13–25.

[28] Dede, G. and Sazlı, M. H. (2010). Speech recognition with artificial neural networks. *Digital Signal Processing*, 20(3):763–768.

[29] Denton, E., Weston, J., Paluri, M., Bourdev, L., and Fergus, R. (2015). User conditional hashtag prediction for images. In *15, . ACM*, pages 1731–1740, KDD. Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining.

[30] der Baan, M. V. and Jutten, C. (2000). Neural networks in geophysical applications. *Geophysics*, 65(4):1032–1047.

[31] Diehl, T., Deichmann, N., Kissling, E., and Husen, S. (2009). Automatic S-wave picker for local earthquake tomography. *Bull. Seism. Soc. Am.*, 99(3):1906–1920.

[32] Donahue, J., Jia, Y., Vinyals, O., Hoffman, J., Zhang, N., Tzeng, E., and Darrell, T. (2014). Decaf: A deep convolutional activation feature for generic visual recognition. In *Proceedings of the 31th International Conference on Machine Learning, ICML 2014, Beijing, China, 21-26 June 2014*, pages 647–655.

[33] Dubey, S. R., Singh, S. K., and Chaudhuri, B. B. (2021). Activation functions in deep learning: A comprehensive survey and benchmark.

[34] Enescu, N. (1996). Seismic data processing using nonlinear prediction and neural networks. *IEEE NORSIG Symposium, Espoo, Finland*.

[35] Florido, E., Aznarte, J., Morales-Esteban, A., and Martínez-Álvarez, F. (2016). Earthquake magnitude prediction based on artificial neural networks: A survey. *Croatian Operational Research CRORS*, 7(2):159–169.

[36] Fujimoto, N. (2008). Faster matrix-vector multiplication on geforce 8800gtx. In *IPDPS*, pages 1–8. IEEE.

[37] FUNVISIS (2018). Boletín sismológico de Venezuela. 1–4(35).

[38] FUNVISIS (2019). Boletín sismológico de Venezuela. 1(36).

[39] Gao, Y., Wang, F., Luan, H., and Chua, T. (2014). Brand data gathering from live social media streams. In *Proceedings of the International Conference on Multimedia Retrieval*, pages 1–04, Glasgow, United Kingdom - April 0, 2014, p. 169. ICMR 2014.

[40] Gao, Y., Zhao, S., Yang, Y., and Chua, T. (2015). Multimedia social event detection in microblog. *In: MultiMedia Modeling - 21st International Conference, MMM 2015, Sydney, NSW, Australia, 5-, Proceedings, Part*, 7(2015):269–281.

[41] Gao, Y., Zhen, Y., Li, H., and Chua, T. (2016). Filtering of brand-related microblogs using social-smooth multiview embedding. IEEE trans. *Multimedia*, 18:2115–2126.

[42] Gebremeskel, E. (2018). Analysis and comparison of distributed training techniques for deep neural networks in a dynamic environment.

[43] Gentili, S. and Michelini, A. (2006). Automatic picking of p and s phases using a neural tree. *Journal of Seismology*, 10:39–63.

[44] Gibbons, S. J. and Ringdal, F. (2006). The detection of low magnitude seismic events using array-based waveform correlation. *Geophys. J. Int.*, 165(1):149–166.

[45] Gillies, S., Bierbaum, A., Lautaportti, K., and Tonnhofer, O. (2007). Shapely: manipulation and analysis of geometric objects. *Available online: github. com/Toblerity/Shapely (accessed on 15 June 2019)*.

[46] Goldberg, Y. (2017). Neural network methods for natural language processing. *Synthesis Lectures on Human Language Technologies*, 10(1):1–309.

[47] Harris, D. B. and Dodge, D. A. (2011). An autonomous system for grouping events in a developing aftershock sequence. *Bull. Seism. Soc. Am.*, 101(2):763–774.

[48] Haut, J. M., Paoletti, M., Plaza, J., and Plaza, A. (2017). Cloud implementation of the k-means algorithm for hyperspectral image analysis. *The Journal of Supercomputing*, 73(1):514–529.

[49] Hecht-Nielsen (1989). Theory of the backpropagation neural network. In *International 1989 Joint Conference on Neural Networks*, pages 593–605 vol.1.

[50] Hernández, J. (2009). Revisión de la sismicidad y modelo sismogénico para actualización de las evaluaciones de amenaza sísmica en la región norcentral de venezuela. In *IX Congreso Venezolano de Sismología e Ingeniería Sísmica*, Caracas.

[51] Hernández, J. and Schmitz, M. (2017). Modelo sismogénico de venezuela para evaluaciones de la amenaza sísmica. In *XI Congreso Venezolano de Sismología e Ingeniería Sísmica*, Caracas.

[52] Iandola, F. N., Moskewicz, M. W., Ashraf, K., and Keutzer, K. (2016). Firecaffe: Near-linear acceleration of deep neural network training on compute clusters. In *2016 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2016, Las Vegas, NV, USA, June 27-30, 2016*, pages 2592–2600.

[53] Ibrahim, M. A., Park, J., and Athens, N. (2018). Earthquake warning system: Detecting earthquake precursor signals using deep neural networks.

[54] Janssen, V. (2009). Understanding coordinate reference systems, datums and transformations. *International Journal of Geoinformatics*, 5.

[55] Keskar, N. S., Mudigere, D., Nocedal, J., Smelyanskiy, M., and Tang, P. T. P. (2016). On large-batch training for deep learning: Generalization gap and sharp minima. *CoRR*, abs/1609.04836.

[56] Keuper, J. and Pfreundt, F. (2016a). Distributed training of deep neural networks: Theoretical and practical limits of parallel scalability. *2016 2nd Workshop on Machine Learning in HPC Environments (MLHPC)*, pages 19–26.

[57] Keuper, J. and Pfreundt, F. (2016b). Distributed training of deep neural networks: Theoretical and practical limits of parallel scalability. In *2nd Workshop on Machine Learning in HPC Environments, MLHPC@SC, Salt Lake City, UT, USA, November 14, 2016*, pages 19–26.

[58] Klumpen, E. and Joswig, M. (1993). Automated reevaluation of local earthquake data by application of generic polarization patterns for P- and S-onsets. *Comput. Geosci.*, 19(2):223–231.

[59] Kong, Q., Trugman, D., Ross, Z., Bianco, M., Meade, B., and Gerstoft, P. (2019). Machine learning in seismology: Turning data into insights. *Seismological Research Letters*, 90(1):3–14.

[60] Kriegerowski, M., Petersen, G., Vasyura-Bathke, H., and Ohrnberger, M. (2018a). A deep convolutional neural network for localization of clustered earthquakes based on multistation full waveforms. *Seismological Research Letters*.

[61] Kriegerowski, M., Petersen, G. M., Vasyura-Bathke, H., and Ohrnberger, M. (2018b). A Deep Convolutional Neural Network for Localization of Clustered Earthquakes Based on Multistation Full Waveforms. *Seismological Research Letters*, 90(2A):510–516.

[62] Krizhevsky, A. (2014). One weird trick for parallelizing convolutional neural networks. *CoRR*, abs/1404.5997.

[63] Krizhevsky, A., Sutskever, I., and Hinton, G. E. (2012). Imagenet classification with deep convolutional neural networks. In Pereira, F., Burges, C., Bottou, L., and Weinberger, K., editors, *Advances in Neural Information Processing Systems*, volume 25. Curran Associates, Inc.

[64] Krizhevsky, A., Sutskever, I., and Hinton, G. E. (2017). Imagenet classification with deep convolutional neural networks. *Commun. ACM*, 60(6):84–90.

[65] Kurth, T., Zhang, J., Satish, N., Mitliagkas, I., Racah, E., Patwary, M. M. A., Malas, T. M., Sundaram, N., Bhimji, W., Smorkalov, M., Deslippe, J., Shiryaev, M., Sridharan, S., Prabhat, and Dubey, P. (2017). Deep learning at 15pf: Supervised and semi-supervised classification for scientific data. *CoRR*, abs/1708.05256.

[66] Lecun, Y., Bottou, L., Bengio, Y., and Haffner, P. (1998). Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324.

[67] Lee, V. W., Kim, C., Chhugani, J., Deisher, M., Kim, D., Nguyen, A. D., Satish, N., Smelyanskiy, M., Chennupaty, S., Hammarlund, P., Singhal, R., and Dubey, P. (2010). Debunking the 100x GPU vs. CPU myth: an evaluation of throughput computing on CPU and GPU. In *37th International Symposium on Computer Architecture (ISCA 2010), June 19-23, 2010, Saint-Malo, France*, pages 451–460.

[68] Macqueen, J. (1967). Some methods for classification and analysis of multivariate observations. In *In 5-th Berkeley Symposium on Mathematical Statistics and Probability*, pages 281–297.

[69] Manfred, J. (1990). Pattern recognition for earthquake detection. *Bulletin of the Seismological Society of America*, 80(1):170.

[70] Marr, D. (1976). Early processing of visual information. *Philosophical Transactions of the Royal Society of London B: Biological Sciences*, 275(942):483–519.

[71] Michael, S., Julio, H., Cecilio, M., Jean, D., Victor, R., Vallée, M., Tagliaferro, M., Delavaud, E., Singer, A., Amaris, E., Danna, M., González, M., Leal, V., el., C., and Franck, A. (2011). Principales resultados y recomendaciones del proyecto de microzonificación sísmica de caracas. *Revista Facultad de Ingeniería*, 26:113–127.

[72] Michael, S., Thota, A., and Henschel, R. (2014). Hpchadoop: A framework to run hadoop on cray x-series supercomputers. In *Cray USer Group (CUG)*.

[73] Mignan, A. and Woessner, J. (2012). Estimating the magnitude of completeness for earthquake catalogs. *Community Online Resource for Statistical Seismicity Analysis*, pages 1–45.

[74] Mus, S., Gutiérrez, N., Tous, R., Otero, B., Cruz, L., Llácer, D., Alvarado, L., and Rojas, O. (2019). Long short-term memory networks for earthquake detection in venezuelan regions. In Nicosia, G., Pardalos, P., Umeton, R., Giuffrida, G., and Sciacca, V., editors, *Machine Learning, Optimization, and Data Science*, pages 751–754, Cham. Springer International Publishing.

[75] Nguyen, D. T., Alam, F., Ofli, F., and Imran, M. (2017). Automatic image filtering on social networks using deep learning and perceptual hashing during crises. *CoRR*, abs/1704.02602.

[76] Nguyen, L. M., Nguyen, P. H., van Dijk, M., Richtárik, P., Scheinberg, K., and Takác, M. (2018). SGD and hogwild! convergence without the bounded gradients assumption. In *Proceedings of the 35th International Conference on Machine Learning, ICML 2018, Stockholmsmässan, Stockholm, Sweden, July 10-15, 2018*, pages 3747–3755.

[77] Novianti, P., Setyorini, D., and Rafflesia, U. (2017). K-means cluster analysis in earthquake epicenter clustering. *International Journal of Advances in Intelligent Informatics*, 3(2):81–89.

[78] O'Shea, K. and Nash, R. (2015). An introduction to convolutional neural networks. *CoRR*, abs/1511.08458.

[79] Park, M., Li, H., and Kim, J. (2016). HARRISON: A benchmark on hashtag recommendation for real-world images in social networks. *CoRR*, abs/1605.05054.

[80] Pinkus, A. (1999). Approximation theory of the mlp model in neural networks. *Acta Numerica*, 8:143 – 195.

[81] Pitts, W. H. and Mcculloch, W. S. (1947). How we know universals: The perception of auditory and visual forms. *Bulletin of Mathematical Biophysics*, 9:127–147.

[82] Redmon, J., Divvala, S. K., Girshick, R. B., and Farhadi, A. (2016). You only look once: Unified. In *real-time object detection*, pages 27–30, Las Vegas, NV, USA, June, 2016, pp. 779–788. Proceedings of the 2016 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2016.

[83] Roberts, R. G., Christoffersson, A., and Cassidy, F. (1989a). Real-time event detection, phase identification and source location estimation using single station three-component seismic data. *Geophysical Journal International*, 97(3):471–480.

[84] Roberts, R. G., Christoffersson, A., and Cassidy, F. (1989b). Real-time event detection, phase identification and source location estimation using single station three-component seismic data. *Geophysical Journal International*, 97(3):471–480.

[85] Rojas, O., Otero, B., Alvarado, L., Mus, S., and Tous, R. (2019). Artificial neural networks as emerging tools for earthquake detection. *Computación y Sistemas*, 23(2):335–350.

[86] Ross, Z. E. and Ben-Zion, Y. (2014). Automatic picking of direct p, s seismic phases and fault zone head waves. *Geophysical Journal International*, 199(1):368–381.

[87] Ross, Z. E., Meier, M., Hauksson, E., and Heaton, T. H. (2018a). Generalized Seismic Phase Detection with Deep Learning. *Bulletin of the Seismological Society of America*, 108(5A):2894–2901.

[88] Ross, Z. E., Meier, M.-A., Hauksson, E., and Heaton, T. H. (2018b). Generalized seismic phase detection with deep learning. *Bulletin of the Seismological Society of America*.

[89] Ross, Z. E., Trugman, D. T., Hauksson, E., and Shearer, P. M. (2019). Searching for hidden earthquakes in Southern California. *Science*, 364(6442):767–771.

[90] Russakovsky, O., Deng, J., Su, H., Krause, J., Satheesh, S., Ma, S., Huang, Z., Karpathy, A., Khosla, A., Bernstein, M. S., Berg, A. C., and Li, F. (2015). Imagenet large scale visual recognition challenge. *International Journal of Computer Vision*, 115(3):211–252.

[91] Rydelek, P. A. and Sacks, I. S. (1989). Testing the completeness of earthquake catalogues and the hypothesis of self-similarity. *Nature*, 337:251–253.

[92] Shelly, D. R., Beroza, G. C., and Ide, S. (2006). Low frequency earthquake swarms and non-volcanic tremor under Shikoku, Japan. In *AGU Fall Meeting Abstracts*, volume 1, pages T41A–1546. American Geophysical Union.

[93] Singer, A., Nevado, F., Gómez, A., and Rodríguez, L. M. (2014). Evidencias de actividad tectónica en la falla de Las Trincheras, VI Coloquio de Microzonificación Sísmica, Valencia, Venezuela.

[94] Skoumal, R. J., Brudzinski, M. R., Currie, B. S., and Levy, J. (2015). Optimizing multi-station earthquake template matching through re-examination of the Youngstown, Ohio, sequence. *Earth Planet. Sci. Lett.*, 405:274–280.

[95] Smola, A. and Narayanamurthy, S. (2010). An architecture for parallel topic models. *PVLDB*, 3:703–710.

[96] Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D., Erhan, D., Vanhoucke, V., and Rabinovich, A. (2014). Going deeper with convolutions.

[97] Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S. E., Anguelov, D., Erhan, D., Vanhoucke, V., and Rabinovich, A. (2015). Going deeper with convolutions. In *IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2015, Boston, MA, USA, June 7-12, 2015*, pages 1–9.

[98] Szegedy, C., Vanhoucke, V., Ioffe, S., Shlens, J., and Wojna, Z. (2016a). Rethinking the inception architecture for computer vision. In *Proceedings of the 2016 IEEE Conference on Computer Vision and Pattern Recognition*, pages 27–30, Las Vegas, NV, USA, June, 2016, pp. 2818–2826. CVPR 2016.

[99] Szegedy, C., Vanhoucke, V., Ioffe, S., Shlens, J., and Wojna, Z. (2016b). Rethinking the inception architecture for computer vision. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2818–2826.

[100] Tariq, H., Touati, F., Al-Hitmi, M., Crescini, D., and Ben Mnaouer, A. (2019). A real-time early warning seismic event detection algorithm using smart geo-spatial bi-axial inclinometer nodes for industry 4.0 applications. *Applied Sciences*, 9.

[101] Thibaut, P., Michaél, G., and Marine, D. (2018). Convolutional neural network for earthquake detection and location. *Science Advances*, 4(2).

[102] Tiira, T. (1999). Detecting teleseismic events using artificial neural networks. *Computers & Geosciences*, 25:929–938.

[103] Torres, J. (2018). *DEEP LEARNING Introducción práctica con Keras*. Independently published, 1st edition.

[104] Tous, R., Alvarado, L., Otero, B., Cruz, L., and Rojas, O. (2020). Deep neural networks for earthquake detection and source region estimation in north-central venezuela. *Bulletin of the Seismological Society of America*, 110(5):2519–2529.

[105] Tous, R., Gounaris, A., Tripiana, C., Torres, J., Girona, S., Ayguadé, E., Labarta, J., Becerra, Y., Carrera, D., and Valero, M. (2015a). Spark deployment and performance evaluation on the marenostrum supercomputer. *2015 IEEE International Conference on Big Data (Big Data)*, pages 299–306.

[106] Tous, R., Torres, J., and Ayguade, E. (2015b). Multimedia big data computing for in-depth event analysis. In *Proceedings of the 2015 IEEE International Conference on Multimedia Big Data (BigMM)*, pages 20–22, Beijing, China, pp. 144–147. IEEE. April.

[107] Tous, R., Wust, O., Gomez, M., Poveda, J., Elena, M., Torres, J., Makni, M., and E., A. (2016). User-generated content curation with deep convolutional neural networks. In *Proceedings of the 2016 IEEE International Conference on Big Data, BigData 2016, DC, USA, December, 2016, pp. 2535–2540*, pages 5–8.

[108] Wang, F., Qi, S., Gao, G., Zhao, S., and Wang, X. (2016). Logo information recognition in large-scale social media data. *Multimedia Syst.*, 22:63–73.

[109] Wang, J. and liang Teng, T. (1997). Identification and picking of s phase using an artificial neural network. *Bulletin of the Seismological Society of America*, 87(5):1140–1149.

[110] Wang, J. and Teng, T.-L. (1995). Artificial neural network-based seismic detector. *Bulletin of the Seismological Society of America*, 85(1):308–319.

[111] Wang, Y., Goldstone, R., Yu, W., and Wang, T. (2014). Characterization and optimization of memory-resident mapreduce on hpc systems. In *IPDPS*, pages 799 – 808.

[112] Wiemer, S. and Wyss, M. (2002). Mapping spatial variability of the frequency-magnitude distribution of earthquakes. *Adv. Geophys.*, 45:259–302.

[113] Wiszniowski, J., Plesiewicz, B. M., and Trojanowski, J. (2014). Application of real time recurrent neural network for detection of small natural earthquakes in poland. *Acta Geophys*, 62(3):469–485.

[114] Xiong, Z., Jie, Z., Congcong, Y., Sen, L., Zhibo, C., and Weiping, L. (2018). Locating earthquakes with a network of seismic stations via a deep learning method.

[115] Yoon, C., O'Reilly, O., Bergen, K., and Beroza, G. (2015). Earthquake detection through computationally efficient similarity search. *Science Advances*, 1:e1501057–e1501057.

[116] You, Y., Buluç, A., and Demmel, J. (2017). Scaling deep learning on GPU and knights landing clusters. *CoRR*, abs/1708.02983.

[117] Zaharia, M., Chowdhury, M., Das, T., Dave, A., Ma, J., McCauly, M., Franklin, M. J., Shenker, S., and Stoica, I. (2012). Resilient distributed datasets: A fault-tolerant abstraction for in-memory cluster computing. In *NSDI*, pages 15–28.

[118] Zaharia, M., Chowdhury, M., Franklin, M., Shenker, S., and Stoica, I. (2010). Spark: Cluster computing with working sets. In *HotCloud*.

[119] Zhang, J., Xiao, J., Wan, J., Yang, J., Ren, Y., Si, H., Zhou, L., and Tu, H. (2017). A parallel strategy for convolutional neural network based on heterogeneous cluster for mobile information system. *Mobile Information Systems*, 2017.

[120] Zhang, X., Zhang, J., Yuan, C., Liu, S., Chen, Z., and Li, W. (2020). Locating induced earthquakes with a network of seismic stations in oklahoma via a deep learning method. *Scientific Reports*, 10.

[121] Zhao, S., Yao, H., Gao, Y., Ji, R., Xie, W., Jiang, X., and Chua, T. (2016a). Predicting personalized emotion perceptions of social images. In *Proceedings of the 2016 ACM Conference on Multimedia Conference*, pages 15–19, Amsterdam, The Netherlands, October, 2016, pp. 1385–1394. Conference on Multimedia Conference 2016.

[122] Zhao, S., Yao, H., Zhao, S., Jiang, X., and Jiang, X. (2016b). Multi-modal microblog classification via multi-task learning. *Multimedia Tools Appl.*, 75:8921–8938.

[123] Zhao, Y. and Takano, K. (1999). An artificial neural network approach for broadband seismic phase picking. *Bulletin of the Seismological Society of America*, 89:670–680.

[124] Zhu, W. and Beroza, G. C. (2018a). PhaseNet: a deep-neural-network-based seismic arrival-time picking method. *Geophysical Journal International*, 216(1):261–273.

[125] Zhu, W. and Beroza, G. C. (2018b). Phasenet: A deep-neural-network-based seismic arrival time picking method. *http://arxiv.org/abs/1803.03211v1*.

[126] Zinkevich, M. A., Weimer, M., Smola, A., and Li, L. (2010). Parallelized stochastic gradient descent. In *NIPS*.

[127] Zou, J., Han, Y., and So, S.-S. (2009). Overview of artificial neural networks. *Methods in molecular biology (Clifton, N.J.)*, 458:14–22.