Universitat Politecnica De Catalunya

BARCELONA TECH

Master Thesis Project
Master in Innovation and Research in Informatics

Image Analysis using Deep-Learning over a Distributed Platform such as Spark in the Marenostrum

Author: Leonel Cruz De La Cruz

Advisors: Dr. Ruben Tous
Dr. Jordi Torres

June 28, 2016

Acknowledgements

First of all, I would like to deeply thank my Advisor Dr. Ruben Tous for the useful comments, remarks and engagement through the learning process of this master thesis, not forgetting my other Advisor Dr. Jordi Torres who always support me also even in the most difficult moments.

Also, I would like to convey my best regards to the Department of Architecture Computer and the Barcelona SuperComputer Center at Campus Nord for providing the access to the resources of these entities.

And finally I want to thank my mother for all their support from the distance.

Abstract

In recent years the digital universe has grown exponentially, the common use of the Internet has led to a new way to generate and consume information, only 10 years ago, it was necessary a deployment of technological infrastructure a lot expensive in order to collect data from several areas, they are ranging from a simple market analysis to the most specialized scientific research in complex areas such as medicine, physics, astronomy etc. But because of the maturity of the information age, costs are down, access to information is available or can be generated from low-cost devices giving rise to Bigdata.

This opens doors to a new world of opportunities such as image processing and possible both commercial and scientific applications. Clear examples of current commercial solutions such as image recognition for security systems of vehicles in circulation or current investigations as automatic driving or detection of diseases from images of organs of patients obtained by scanners.

This thesis deals the problem of the processing and classification of images in categories, obtained from any data source, using for this approach Deep-Learning. For this we have implemented a convolutional neural network architecture based on Java. Composed of an ETL module that handles the loading of images in raw format and transforms into tensioners for treatment within ConvNet, the configuration of the convolutional network is made up of 12 layers, 6 convolutional layers and 5 MaxPooling layers, together with a fully-connected layer. The convolutional layers were configured with Relu as activation function, while the last layer that performs the classification was treated with Softmax function.

The application works in two ways developed stand alone and distributed, in both cases the framework used for the development of this thesis was Deeplearning4j which is based on java, for the development of convolutional networks. In addition to handling n-dimensional array, linear algebra and signal processing functions NDj4 were used which is a scientific computing libraries for Java that complements the aforementioned framework. For the distributed environment Apache Spark was used as a distributed cluster, and RDD (Resilient Distributed Datasets) for the distribution model over the nodes that form the Spark cluster.

The results obtained from the experiments were different depending on the parameters supplied, these ranged from 60 to 80 of accuracy. The duration of the training process model in stand alone mode take considerable time unlike the distributed (minutes vs. hours). For future work this project could be coupled

to any other system performing object recognition, especially if they are developed in Java thus being a multiplatform solution.

Contents

Chapter 1

Introduction

1.1 Overview

Increased data digital world has greatly increased in recent years[3] , access to devices that allow to be connected all the time and platforms that allow consumption has led to another level data acquisition.

Deep Learning techniques are based on artificial neural networks, the result of research with more than 30 years of history. They have recently experienced a boom due mainly to advances in computing power and the systems distributed.

In the field of artificial vision and imaging has played a decisive role since 2012 where these techniques had a high yield when making classifications in the ImageNet Large Scale Visual Recognition Challenge. Since then large companies like Google, Facebook, Baidu or IBM, are betting strongly on these technologies.[9]

1.2 Motivations and goals

In this project the main motivations to its development were the following facts:

- The interest in the exploration of new techniques of data sciences, applied to massive data analysis provided by the different sources of information with which we interact every day, called Bigdata. Departing from the usual traditional approaches of Machine Learning by the new tendency named deep learing. With the purpose of obtain statistical information of features able to find patterns that represent real-world concepts. Especially in our project with the image processing.

- Application of these techniques for high-performance environments such as marenostrum which is the supercomputer that we have thanks to BSC, and learn from everyone involved with projects that take place with this technology, contributing to my research.
- Ecosystems Systems Distributed nowadays are the support to analysis of huge data, its ability to parallelize tasks make this technology very efficient at the time of image processing , it is for this reason that seek an integration between this platform and algorithms deep learning. At the time of this research there are native components in these distributed frameworks containing machine learning algorithms, but these do not implement the neural network approach. Therefore, frameworks that can work on distributed environments turn out to be an interesting field of research.
- The use of open source tools for processing and analyzing large amounts of data in this new era. Exploiting the fact that they can be reinvented and adapted to the needs of different problems sciences data. Such is the case of distributed systems like Hadoop, Spark and libraries for data analysis as Mlib, R, H2O, Cafe among other environments.

The goals of this project can be summarized below:

- Analysis of different Opensource alternatives available in the way of the realization of the solution over distributed systems.
- Develop a ETL system capable of processing raw data, in this case images and transform them into a format that is capable of being treated by different algorithms deep learning.
- Implement algorithms that perform automatic recognition of images using supervised classification scheme and integrate the use of deep learning on distributed environments.

1.3 Contributions

In this project we have implemented an algorithm based on Convolutional Neural Networks, which classifies images that can be acquired from different data sources. The language used for this was precisely java to extend the functionality offered by the framework used.The application is made using the deepLearning4j for neural network design along with Apache Spark to work its counterpart distributed.

The application is made using the deepLearning4j for neural network design along with Apache Spark to work its counterpart distributed. Making the performance improve considerably when running the algorithm. Also it was used the library Canova for treating the raw data, with respect to this was added a pre-preprocessing step before using load images.

Finally, this entire project was tested on Apache Spark on Marenostrum SuperComputer, whose results will be used for the preparation of papers and also the results could make them follow updated frameworks of this type on this Super-Computer.

1.4 About this document

The rest of the document contains five chapters more. Chapter 2 discusses the necessary background about the algorithms used in the project, taking account since artificial neural networks until distribution frameworks.

The Chapter 3 exposes the new tendencies with respect to the image classification following with the Chapter 4 explaining the methodology used in this project composed for network architecture, testing environment and optimization of solution.

The Chapter 5 shows the outlines achieved with our configuration measuring in terms of performances and scalability. And in the last Chapter exposes the conclusions obtained behind the experimentation and their possible applications in future works.

Chapter 2

Background Information

During the last two decades the recognized of images have been one of the most exciting field in the research, with the arrival of the massive acquired of data throughout several sensors, spectral cameras, drones, webcams, even more with the rapid development of Internet use and the advent of social networking platforms such as Facebook, Instagram, Flickr inter alia and with video transmission platforms such as Youtube[11], have opened a new field for Multimedia Big Data Computing Analysis.

With this in mind science he has tried to emulate how humans learn and perceive reality as the processing power of the human brain is infinite, the result of these questions has resulted in a research field called deep learning that born from neural networks.

Neural networks are not a new concept, born in the 40s, where the first neural network model was created based on mathematics and algorithm known as threshold logic by McCulloch and Pitts[5]. In the late of 50s It was created the perceptron[7] by Frank Rosenblatt, that is an algorithm for pattern recognition in base of two-layer computer learning, using simple operations of sum and multiplication in order to its use.

with the time algorithms as MLP were developed, basically it is an extended concept of perceptron until that in the 80s was developed the Backpropagation algorithm by Rumelhart, from this point forward we can say that the deep learning born.

The limitations of that time made that the models acquired to make predictions take too long on many occasions weeks or months to obtain, but now in our days since 2006 with the growing of current computing power can use more efficiently these algorithms adapting them also to that work in parallel to maximize the time of obtaining results.

2.1 Image Recognition

For humans to recognize objects is done easily thanks to our brain. Distinguish between a beach landscape vs a mountain we do immediately, but this is not so simple in the computing world. All this is done with algorithms that involve a huge amount of mathematical calculations that need to be computed for current processors.

Current research, make use of algorithms that features of the images. The extraction of features is very important when performing the classification process. It is necessary to take into consideration external agents that can modify in some way the object, as it is position, light intensity, or similar objects in the same image. And here it is where the ANN take advantage of other algorithms such as Support Vector Machines, Bayes among others.

2.2 Neural Network Definition

The way of implement deep learning is based in Artificial Neural Networks(ANN)[8], therefore in first place is necessary explain the basic concept of the them, it is based in the emulation of brain human and these are a set of algorithms designed to recognize patterns. This patterns recognized are numerical, storage in objects named vectors, these are a representation of the real-world data, such as images, text, sound, video.

The ANN can be used to cluster and classify, the main difference between the deep-learning and ANN is the number of hidden layer.

For instance, imagine that you want to determine a group of people with risk of cancer, so your training set could be an array of cancer patients or person without cancer, for each people with cancer it has a set of images representing anomalies in the scanners of any organ. With this input, It will training the neural network to determine if the new cases will be or not potential patients with this affliction.

2.3 Neural Network Elements

The Neural Network is composed for a set of layers, each layer is formed by nodes, the nodes is the representation of the a neuron in the human brain and in these nodes are where the action is generated if this achieve a enough stimulus.

It mix input from data with a set of coefficients or weights that amplify or decrease that input. This product weights and inputs are summed and this result is passed through a node that contain a “activation function”, the same that has a behavior of a classifier.

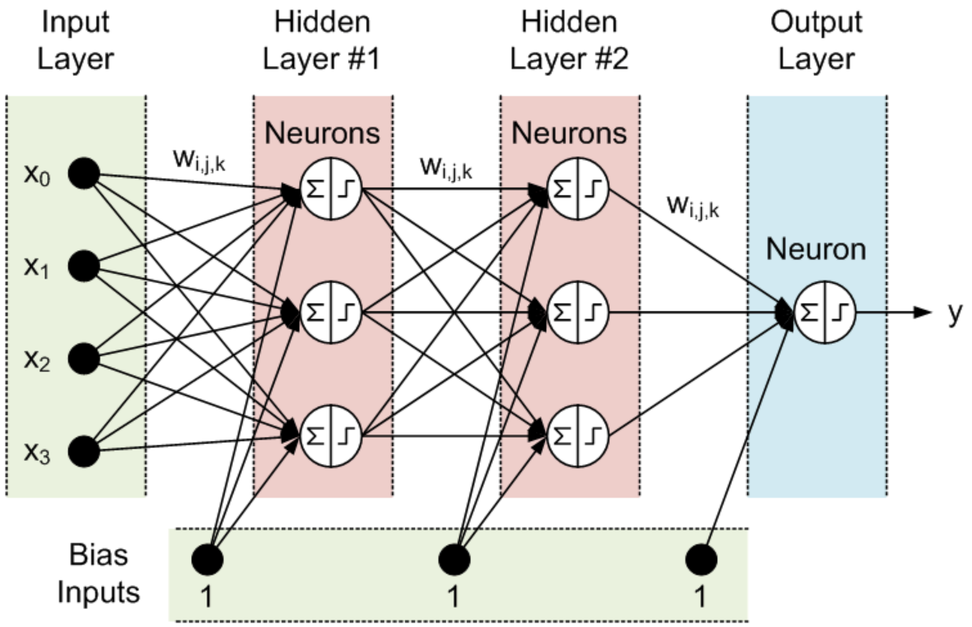


Figure 2.3.1: Neuronal Network

The Figure 2.3.1 represent the layers in the ANN, like can view each layer is represented like a column, each layer's output is at the same time the next layer's input.

Here we can named the following layers:

- Input Layer, that contain the input or raw data
- Hidden Layer, formed by the nodes named neurons, in which are realize the sums and activation functions.
- Output Layer, which consist of one or several neurons, whose outputs are the network outputs.

The graph we can see that each input node is connected to all the neurons in the next layer, this connection is accompanied by a weight.

The main operation performed by the neural network is to multiply the values of neurons by the weights of their outgoing connections. Then each neuron layer receives the following values of several incoming connections and then sum these values. This process is performed on all hidden layers until the output layer.

As explained above can be summarized in the following formula:

In this example the output layer is composed of a single neuron, which could give a value between 0 and

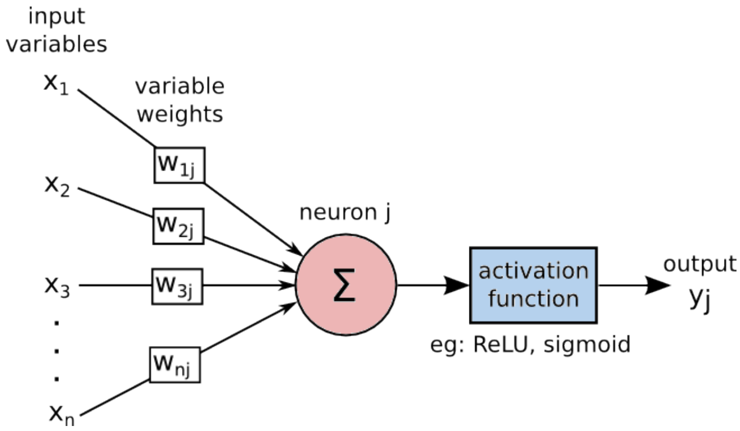


Figure 2.3.2: Activation Function

1 to determine whether or not the input represents a concept.

2.3.1 Activation Function

Until now we have only done multiplications and additions on the neural network, but the value of neurons will be the result of the assessment by the activation function of the sum realized in the last layer. One objective of the activation function is to maintain the results obtained by each neuron in a given range (0 and 1)

In the Figure 2.3.2 we can see how work the function network, depending of selected function is possible obtain better results.

There are several functions such as:

$$Sigmoid : \sigma(x) = \frac{1}{(1+e^{-x})} \tag{2.1}$$

$$Tanh : tanh(x) = 2\sigma(2x) - 1 \tag{2.2}$$

$$ReLu : f(x) = max(0, x) \tag{2.3}$$

2.3.2 Bias

It is a further neuron, which is in the hidden layers and the output, these neurons always have the value 1, but its associated weight can be varied. It is used to improve the convergence properties of the network and generates a threshold effect on the part of the network operating.

The principal difference between the deep learning and ANN is the named "depth" in other words the number of layers that are able contain, typically in the cn the configuration is composed by 3 layers give us a shallow learning net, technically the word "depth" is a term that means more than one hidden layer.

2.4 ANN Learning

2.4.1 Configuration

Once we know the number of neurons in the input layer neurons and resulting in the output layer, in the example described above they are 4 and 1 respectively. We must make the following analysis that will be key to the correct functioning of our model.

- Number of hidden layers required.
- Number of neurons in each hidden layer.
- The weight of the links of each layer.

With regard to the first two items, we can define them with different criteria, commonly trial and error. But it is noteworthy that the more layers the model we can learn more and if we increase the number of neurons in the hidden layers will have greater benefits in obtaining more representative of the concepts features. Of course the more complex the network more processing power needed.

The third point can be done automatically thanks to Backpropagation Algorithm, a set of training data is needed for the input layer and output labels to last layer. In the case of the images, for example if we wanted to determine whether an image is a person, we need a set of images containing people and others not, so the last layer determined by a percentage if it falls into one category or another. This is known as supervised learning

2.4.2 Backpropagation

At the moment we have a set of examples, we can see easily the obtained operating model and know the error that has been generated, in other words we can evaluate the value expected (real) vs the value obtained by the model.

To fix the errors the idea is to look for neuron provoked that response, in other words search guilty. Each neuron is connected to neurons in the previous layer by weights W, these weights are used to determine how much contribute the neurons of the preceding layer to the final error, this process extends to the first layer and adjusting the weights to improve the outcome of model.

When updating the weights must consider how these changes affect to error. In other words, we need to know how fast the error changes regarding weights and thus minimize the error as fast as possible. This is calculated with partial derivatives on the activation function is selected.

There are several algorithms like deep belief network(DBN), Deep Autoencoder, Recurrent Net able to deal a specific problem such as sentiment analysis, Document classification or Voice recognition.

This work make use of set of algorithms and tools in order to realize image classification with supervised learning. Specifically in this case It was used Convolutional Networks to deal the classification of the images.

2.5 Convolutional Networks

Convolutional neural nets are used to classify images grouped by similarity and realize object recognition inside of scene of picture, these can identify faces, persons, streets, transit signal and very much, these are a neural network specialized, therefore share the same concepts studied such as layer, neurons, weights, bias, activation function, backpropagation and so. Also named simply ConvNets. One difference with respect to the neural networks are the inputs, because the ConvNet makes the assumption that the inputs are images, using this such as advantage at the moment of encode certain properties into the Architecture in order to improve the efficiency, reducing the number of parameters of model.

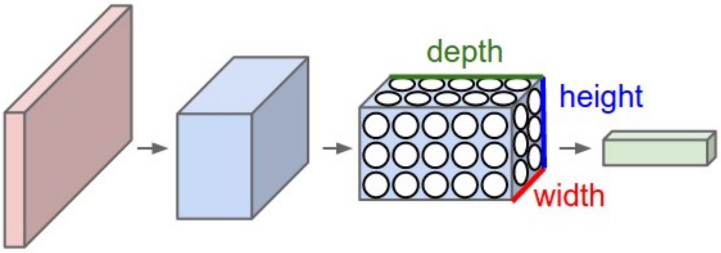


Figure 2.5.1: 3D volumes of neurons

2.5.1 Architecture

Recalling normal neural networks. It is known that these receive an entry in the form of simple vector and transformed through the different hidden layers. The layers have a number of neurons and each neuron is fully connected to all neurons in the previous layer.

This model is not efficient at the moment of look at inputs with high dimensionality, a simple example could be the Cifar Dataset which has images with a dimensionality of 32x32x3, so one single neuron would get 32*32*3= 3072 weights, This is still manageable but with real world where the size picture are typically 200x200x3 giving 120.000 weights thus the things changes, because the weights easily can generate Overfitting due the full connectivity.

Unlike an Artificial Neural Network, in CNN the neurons inside the layers are composed by 3 dimensions: width, height, depth. It must note that ”The depth” is the representation of the different color layers containing an image, also known as channels and these are represented by 3 channels in the case of images of colors thanks to RGB (Red, Blue, Black) encoding, and channel in the case of monochrome images, often is composed The key in this configuration is that each neuron is connected only to a small part of the neurons of the before layer.

Now instead of take each pixel in image the ConvNet will take a square patch of the pixels and these will be passed through a filter. This filter have the same dimension of the patch and it is more small than original image.

2.5.2 Layers in ConvNets

ConvNet is a set of layers with a specific order , wherein each layer transforms a volume of activations to another through activation functions defined in different layers. To build a classical ConvNet can distinguish 3 types of layers: ” Convolutional Layer, Pooling Layer and Fully Connected Layer”. The first two regularly work together and its number will depend of complexity that is desired in the features obtained, while the third layer type will only be one inside whole architecture of the ConvNet.

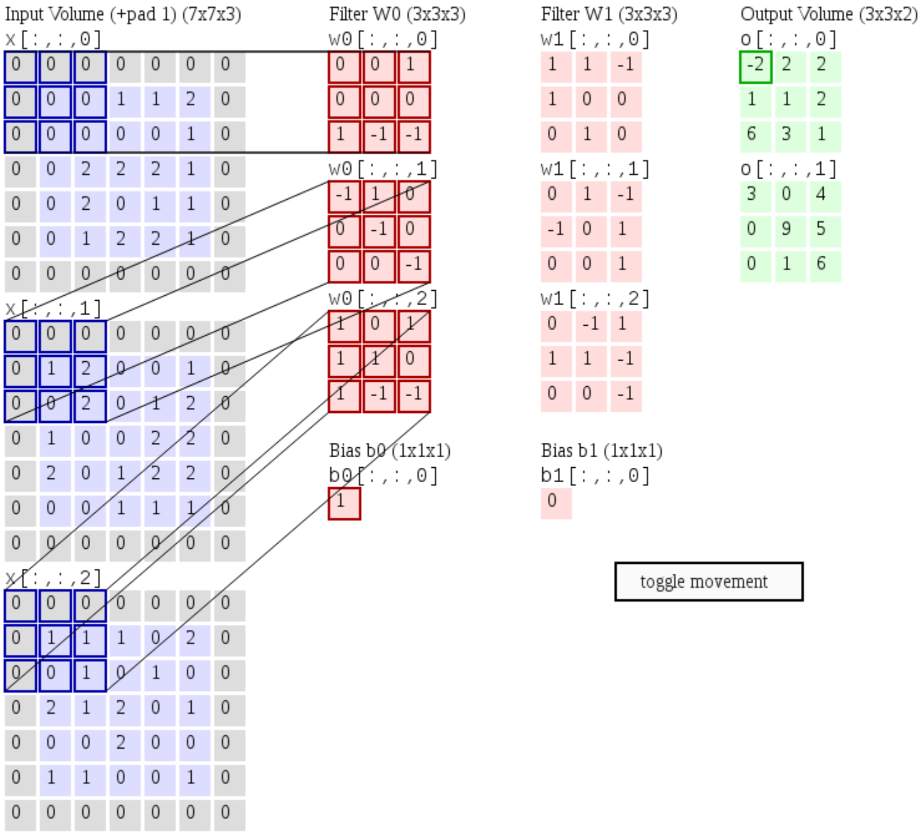
Convolution Layer

The CONV layer’s parameters consist of a set of learnable filters also called kernels. Every filter is small spatially (along width and height), but extends through the full depth of the input volume. For instance, a filter on a first layer of a ConvNet might have size 3x3x3 (i.e. 3 pixels width and height, and 3 because images have depth 3, the color channels). During the forward pass, we slide (the right word is convolve) each filter across the width and height of the input volume and compute dot products between the entries of the filter and the input at any position. As we slide the filter over the width and height of the input volume we will produce a 2-dimensional activation map that gives the responses of that filter at every spatial position using in this part any activation function. Intuitively, the network will learn filters that activate when they see some type of visual feature such as an edge of some orientation or a blotch of some color on the first layer, or patterns on higher layers of the network.

Now, we will have an entire set of filters in each CONV layer (e.g. 12 filters), and each of them will produce a separate 2-dimensional activation map. We will stack these activation maps along the depth dimension and produce the output volume.

Local Connectivity

When working with images, we encounter high dimensionality inputs, now it is inefficient to have each neuron entirely connected to all the neurons of the previous layer. Then the output found for this problem is to connect the neuron in question to a small portion of the neurons of the previous layer, the size of this portion is defined by the filter size or kernel that we have defined above, following the example above we will be 3x3. This small portion is formally known as ”receptive field” of the neuron.



Input Volume (+pad 1) (7x7x3)

x[:,:,0]

0	0	0	0	0	0	0
0	0	0	1	1	2	0
0	0	0	0	0	1	0
0	0	2	2	2	1	0
0	0	2	0	1	1	0
0	0	1	2	2	1	0
0	0	0	0	0	0	0

x[:,:,1]

0	0	0	0	0	0	0
0	1	2	0	0	1	0
0	0	2	0	1	2	0
0	1	0	0	2	2	0
0	2	0	1	2	2	0
0	0	0	1	1	1	0
0	0	0	0	0	0	0

x[:,:,2]

0	0	0	0	0	0	0
0	1	1	1	0	2	0
0	0	1	0	1	0	0
0	2	1	2	0	1	0
0	0	0	2	0	0	0
0	1	1	0	0	1	0
0	0	0	0	0	0	0

Filter W0 (3x3x3)

w0[:,:,0]

0	0	1
0	0	0
1	-1	-1

w0[:,:,1]

-1	1	0
0	-1	0
0	0	-1

w0[:,:,2]

1	0	1
1	1	0
1	-1	-1

Bias b0 (1x1x1)

b0[:,:,0]

1

Filter W1 (3x3x3)

w1[:,:,0]

1	1	-1
1	0	0
0	1	0

w1[:,:,1]

0	1	-1
-1	0	1
0	0	1

w1[:,:,2]

0	-1	1
1	1	-1
0	0	-1

Bias b1 (1x1x1)

b1[:,:,0]

0

Output Volume (3x3x2)

o[:,:,0]

-2	2	2
1	1	2
6	3	1

o[:,:,1]

3	0	4
0	9	5
0	1	6

Figure 2.5.2: Convolutional Layer

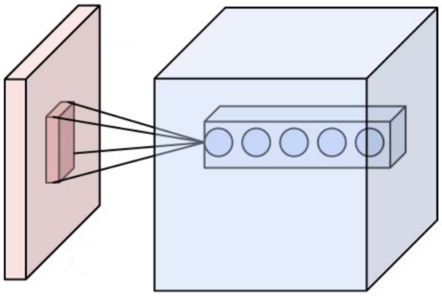


Figure 2.5.3: Receptive Field

In order to determine the number of neurons contained in the output volume perform convolution product of the filter and all the neurons of the preceding layer must consider the following hyperparameters:

- depth: It is defined by the number of filters that we we place in the convolutional layer. A clear example of this would be in the first part of our architecture. Wherein the first convolutional layer takes as input the raw image, in this different point stored in each filter neurons could be activated if they detect edged oriented, texture or blobs of color.
- stride:This parameter defines the size as we move into the input layer, its value is 1 regularly depending on how much want to share weights within the image. Returning to the example of the first convolutional layer using Fig 3.1.2 reference. As input we have an input volume 7x7x3 and 3x3x3 filter if we define a stride of 2 then the convolution patch filter or two units would shift forward over the entire image.

As we have already mentioned in the preceding paragraphs one of the main problems with images is that they are high-dimensional, which means they cost a lot of time and computing power to process. ConvNets are designed to reduce the dimensionality of images in a several ways. Filter stride is one manner to reduce dimensionality. Another manner is thanks to the downsampling method, contained in MaxPooling Layer.

MaxPooling Layer

Another layer in take account is the commonly named MaxPooling, downsampling and subsampling. The activation maps are the result of convolution operation and these are fed into a downsampling layer, and like convolutions, this method is applied one patch at a time.

The idea is to make a kind of zoom out of an area input volume. Basically taking and passing a patch THROUGH whole layer of previous entry. The following example will clarify the process mentioned.

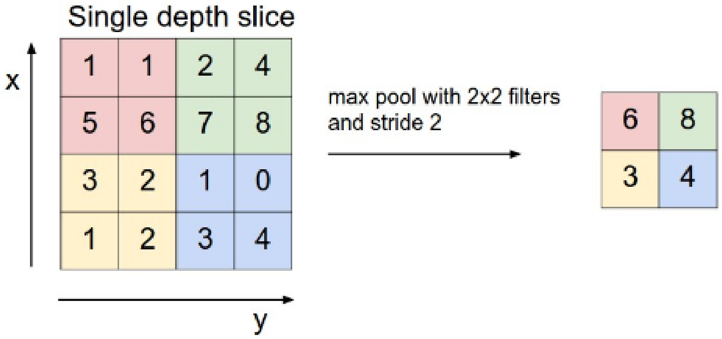


Figure 2.5.4: Work of MaxPooling

In Figure 3.1.4 we have a 4x4 input volume, so we use a patch of dimensions 2x2 with 2 stride for the operation of downsampling. What is done is that in each region composed of 2x2 values select the highest value in the first region for example we get the value 6 in the region because it contains values we [1,1,5,6]. And thus advancing 2 by 2 will make our dimensionality reduction. It should be noted that the value of stride not generate a superimposition of regions within the treated input unlike convolutional process.

Fully-connected layer

Neurons in a fully connected layer have full connections to all activations in the previous layer, as seen in regular Neural Networks. Their activations can hence be computed with a matrix multiplication followed by a bias offset.

To make the classification categories used in this case the algorithm softmax give us a percentage result of where our sample will be classified.

2.6 Deepleaning4j

Is the library used in order to realize these experiments, DeepLearning4j is a framework OpenSource written in java with a wide range of deeplearnig algorithms. Between the implemented algorithms we have:

- Restricted Boltzmann machine.
- Convolutional Nets.
- Recurrent Nets/LSTMs(time series and sensor data)
- Recursive autoencoders.
- Deep belief net.
- Deep autoencoder.

This set of algorithms support distributed parallel versions that can be integrated with Hadoop and Apache Spark, there are APIs to work with scala and Clojure directly. It has its own numerical computing library named ND4j and can be used with cpu or gpu highly efficient.

also deeplearning4j has a commercial support offered for SkypeMind Company.

2.6.1 Scientific computing for the JVM

Deeplearning4j has an n-dimensional class array using ND4J which permit scientific computing in Java and Scala. Similarly to numpy in Python. This solution is based on a library for linear algebra and matrix handling over production environment.

2.6.2 Vectorization

This is the process in which the data is passed or transformed from raw data to data representation able to be treated in the neural network. This is a one of the biggest problems in the moment to deal with several data source, because the nets read vectors.

Deeplearning4J provides Canova library named “Rosseta Stone” too in order to acquisition framework, this library can be used to deal source such as: CSV, images, sound, text and video. In our case It’ll use to images processing.

2.6.3 Parallelization

One of the most important points in this framework is that the training process is carried out in a cluster, that in this case over Apache Spark, the neural networks are trained in parallel using to do this RDD to distribute the work over the nodes of the cluster.

2.7 Spark

Apache Spark[10] is an open source cluster computer framework to general propose and quick. Initially was a project created for University of California, Berkeley AMPLab, nowadays is part of Apache Software Foundation. This framework being a general-purpose solution it can be used for both data and data science applications in order to parallelize their tasks in an efficient and quick way. Spark uses Mapreduce model for processing data in a parallel way across of its cluster composed of hundreds to thousands of machines. Spark (maintains) conserve MapReduce's linear scalability and fault tolerance, nevertheless expands it ability in three ways:

- In first place: instead of uses one strict map-reduce format , it can run a more general directed acyclic graph(DAG) of operators. This implies that while the map-reduce needs a storage the intermediate result in a distributed file system, the spark can send the intermediate result in the next pipeline.
- Second: With respect to the first point spark provides a set of transformations that help to the users to deal the programing of a way more natural with a few code lines, also spark provides APIs in Scala, Java, Python and now R.
- Third: Spark implements the processing over memory, this is possible with the use of Resilient Distributed Dataset(RDD) abstraction, this permit to the developers materialize any point in a processing pipeline into memory across the cluster, this means that is not necessary reload newly from disc to that other process take over of data.

The main advantage of spark is its manage of memory reducing the constant uses of read and write operations over the disc, therefore Spark's in-memory caching make it ideal para the use of machine

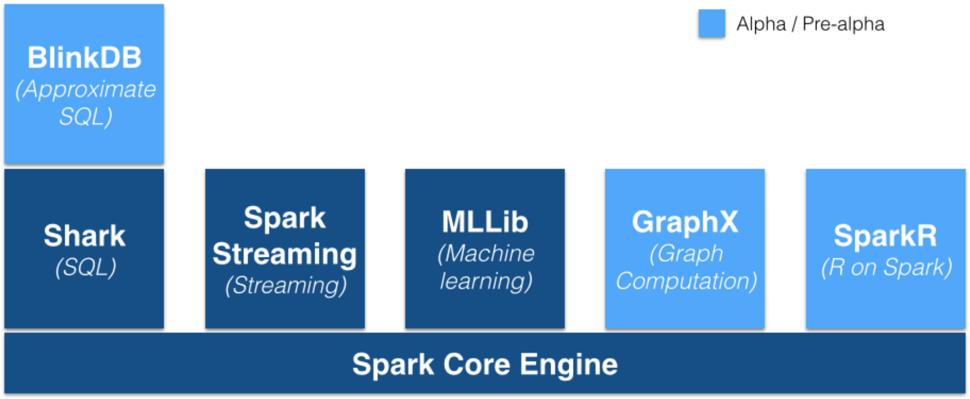


Figure 2.7.1: Spark stack

learning algorithms that makes several passes over their training set and these will be cached in memory for its respective dealing. data can be available in memory for the long time.

2.7.1 Components Apache Spark

Spark needs a cluster manager and distributed file system to work, in the case of cluster manager, Spark can be used with standalone(own spark cluster), Hadoop Yarn or Apache Mesos also in the cloud with service EC2 of Amazon or docker, In the case of distributed file system spark can work with an arrange of options such as: Hadoop Distributed File System (HDFS), Casandra, OpenStack Swift, Amazon S3, kudu.

In this project, we'll working with a framework based in java therefore we'll use the APIs java to integrate the algorithms. Now explain briefly the standard components inside Spark.

Spark Core

Spark Core contains the most important functions of project. This supply components for task scheduling, distributed dispatching, memory management, fault recovery and more gave to the users through an APIs (for Java, Python, Scala and R) focused on RDD abstraction.

It's precisely here in where we'll put to deeplearning4j framework in order to create the classification algorithms contains in the neural networks.

Spark SQL

Spark SQL is a component on top of Spark Core that introduces a new data abstraction called DataFrames, which provides support for structured and semi-structured data Spark SQL provides a domain specific language to manipulate DataFrames in Scala, Java, Python or R.

Spark Streaming

Spark Streaming is a component able to realize streaming analytics. It takes in data inside mini-batches and execute RDD transformations on those mini-batches of data.

Mlib

Spark contains a distributed machine learning framework on top of Spark Core, that provides the most commons machine learning algorithms such as:

Summary statistics: correlations, random data generation and hypothesis testing. Classification: support vector machines, decision trees, multinomial/binomial naive bayes classification. Regression: logistic regression, isotonic regression, linear regression with L1, L2 and elastic-net regularization. Reduction of dimensionality: Principal Component Analysis (PCA) and singular value decomposition(SVD). Clustering: k-means, bisecting k-means, Gaussian mixtures(GMM), power iteration clustering and more.

Although this library has the most used algorithms implemented in order to be used with RDD, this not implement actually Neuronal Network that is the base of deep-learning and object from this work. For this reason we'll use deeplearning4j framework.

GraphX

Graph X is a distributed graph-processing framework on top of Spark Core. It provides and set of operators for manipulating graphs and common graph algorithms.

2.7.2 Resilent Distributed DataSet

Resilient Distributed DataSet (RDD) is the component in which it base whole the Spark project. It is an immutable distributed collection of object. And in this project will be used with deeplearning4j.

The dataset in RDD are divided logically in small portions, which can be processed in several nodes of the cluster. These RDD can contain several kind of object for instance Java, Python, Scala objects, even object personally objects created by the users.

There are two forms to instances RDD: Loading the data from external store system such as: HDFS, HBASE or someone datasource ability of understand the Hadoop input format. Parallelizing a collection in our driver program.

Data Sharing using Spark RDD

As we mentioned in the introduction of this chapter, one of the disadvantage of hadoop is the behavior of MapReduce which is slow due to replication, serialization and the I/O Operations. Most applications designed spend over 90

With this problem in mind, It was created Apache Spark, basing in the idea of Resilient Distributed Datasets (RDD); It store the state of memory as an object across the jobs and the object is shareable between those jobs. Of course this is faster than access to disk or network, a 10 to 100 times faster.

Iterative operations

RDD Operations

RDD provides two kind of operations, transformations and actions. The main difference between these of operations is the type of object returned.

Transformations

Transformations are operations that return a new RDD, a representative example could be the log messages of a WebServer, in first place this should be load in a RDD if need only the messages related with spark we need to execute a filter(action) and obtain from this a new RDD, which could be base in order to treatment of data.

Actions

Actions are operations that return a final value, which could be used for driver program or sent to external storage system. The actions force the evaluation of the required for the RDD. Following the previous example, we want know how many errors has the log of the WebServer. So in this point we need use the function count() and take().

The behaviour standard, implies that each transformed RDD could be recomputed each time that run an action. But it could be changed, doing it persist in RDD memory. In this point Spark will keep the elements around on the cluster in order to improve the access speed, the next time that we need to do a query.

Chapter 3

State of the Art/Related Works

3.1 Classic approach and deep learning

3.1.1 Classical Image Processing

During the last decade technologies have been many improvements in image processing, the key to all the processing is based on the correct extraction of features. It is difficult to find common features that can be displayed within a category due to the effects of light, orientation, scale and others.

In 2004 appear techniques such as SIFT(Scale-Invariant Feature Transform), whose purpose was to extract distinctive invariant features from images, this was an important feature that was not affectation of rotation and lighting in the images sources. Shortly after based on this algorithm is developed PCA-SIFT, one of his strengths was that he could deal with distorted images and his weakness was present at the slow response. Until in 2006 partially inspired by SIFT, SURF(Speeded up robust features) is created and its main advantage over the previous was that it was noticeably faster.

Below is a brief description of these methods:

- SIFT is formed of 4 stages: scale-space extrema detection, keypoint localization, orientation assignment and keypoint descriptor. The first stage used difference-of-Gaussian function to identify potential interest points, DOG was used instead of Gaussian to improve the computation speed, in the second stage reject the low contrast points and delete the edge response. The stage 3 is centered in a orientation histogram was formed from the gradient orientations of sample points within a region around the keypoint in order to get an orientation assignment.

- PCA-SIFT makes use of dimensionality reduction algorithm PCA (Principal Component Analysis) instead of histogram to normalize gradient patch.
- SURF although it is based on sift, the how to extract features makes the difference between SIFT and SURF. The first builds an image pyramids, filtering each layer with Gaussians of increasing sigma values and taking the difference meanwhile the SURF creates a "stack" without 2:1 down sampling for higher levels in the pyramid resulting in images of the same resolution.

Other interesting related work in the world of the classification is the **Bag of the Words Models**.

The bag-of-keypoints model proposed by Csurka et. al. and was considered such as one of the state of the art algorithms. This model produces a single representative vector of an image from a set of local descriptors and is then coupled with a classifier to perform image recognition.

Most of the large scale image search and recognition systems were based by the bag of words model unfortunately it does not take into consideration the color in the images. Wengert et. al. proposed a bag of colours model like solution to this problem, mixing the descriptors used for bag of words model, with a colour signature that can be used as a global or local descriptor. Their findings quickly showed an interesting perspective to improve the image recognition methods until this days taking colour information in consideration.

These approaches can be considered like classics in the world of the images processing, until in 2012, new trends based on algorithms **deep learning** appear with their innovative applications.

3.1.2 New Age of deep learning

Currently the Convolutional Neural Networks or ConvNets are presented as the "Roseta stone" in the image classification, thus generating researches that have put these architectures in the state of the art in this field. One of the most important architectures CNN has been called "Inception module"(Colocar referencia a Going deeper with convolutions) the same that won the ImageNet Large-Scale Visual Recognition Challenge 2014(ILSVRC14). It all started with Letnet describing a configuration of convolutional network composed of convolutional layers and the last layer a Fully-connected layer, starting from here there are variants of this structure, for example the number of layers needed to obtain better and more refined features, or the amount of neurons in each layer, adjusting regularization parameters to mitigate overfitting.

The principal idea of the Inception architecture is based on finding out how an optimal local sparse structure in a convolutional vision network can be approximated and covered by readily available dense components. All we need is to find the optimal local construction and to repeat it spatially. Inception modules in convolutional networks were designed to allow for deeper and larger convolutional layers while at the same time allowing for more efficient computation.

This is done by using 11 convolutions with small feature map size, for example, 192 2828 sized feature maps can be reduced to 64 2828 feature maps through 64 11 convolutions. Because of the reduced size, these 11 convolutions can be followed up with larger convolutions of size 33 and 55. In addition to 11 convolution, max pooling may also be used to reduce dimensionality. In the output of an inception module, all the large convolutions are concatenated into a big feature map which is then fed into the next layer (or inception module).

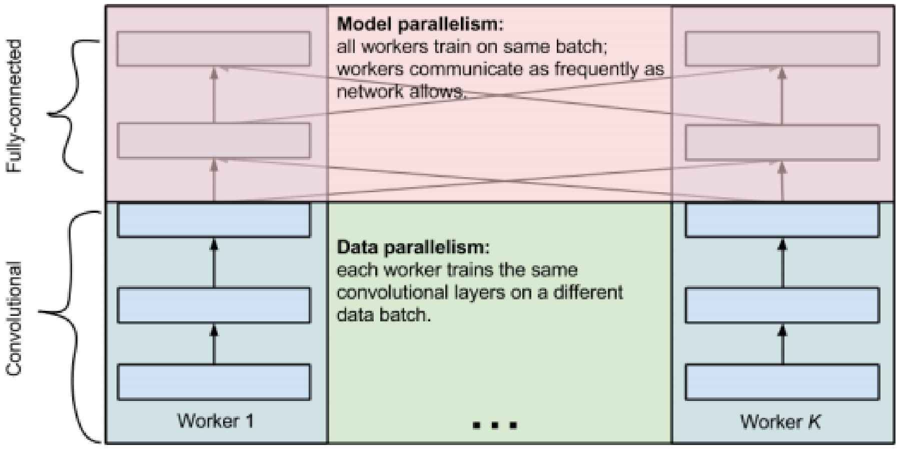

Model parallelism:
all workers train on same batch;
workers communicate as frequently as
network allows.
Data parallelism:
each worker trains the same
convolutional layers on a different
data batch.
Fully-connected
Convolutional
Worker 1
...
Worker K

Figure 3.2.1: ConvNets

3.2 Parallelism

Our model is developed using deeplearning4J as development framework of ConvNet, exploiting integration with Apache Spark for the algorithm parallelization and MN3 infrastructure for the test. The parallelization process is based on the solution provided in: (paper dl4j).

Currently there are other works focused in the creation of schemes for implementation over distributed environments. One of the most recognized and has contributed toward a new path, is the Krizhevsky work's Krizhevsky14. Here It's describe the two ways of how can be parallelized the training process inside ConvNet, basically It can be done considering the orientation to the data or to the model. This means that in the first approach the Dataset will be splitted equally and and subsequently distributed among several cores, the principal constraint in this approach is that the parameters of each core must be synchronized throughout the training process meanwhile in the second approach is designed so that the model is divided into a set of neurons and these in turn associated with a thread of execution called worker, the main idea is that this worked must be synchronized with other worker containing the rest of the model, thus generating a chain of processes.

The proposal of Krizhevsky take a bit of each approach(view figure 3.2.1), considering that the architecture of the current ConvNet are composed of two types of layers that are several Convolutional and Fully-Connected which is the last layer. He proposes has k workers inside of the convolutional layers, each one of these will handle a sample of the global dataset, will compute the data with parameters at the end of the convolutional layers these workers going to switch to mode parallelism in order to compute

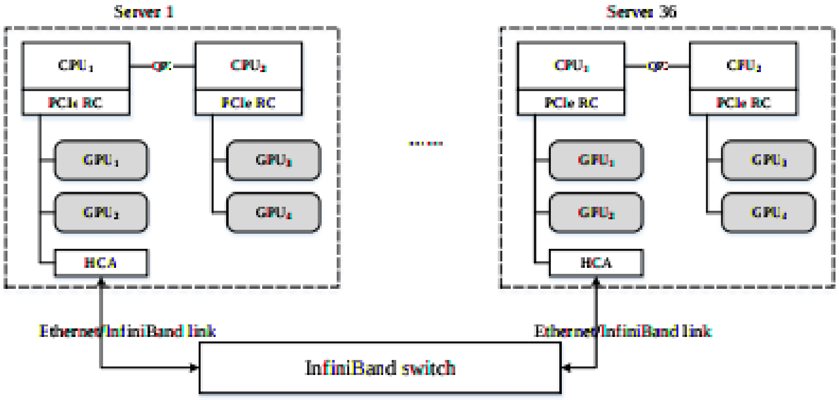


Figure 3.2.2: Architecture Minwa

the fully-connected layer activities. This can be done of following ways:

- Each worker sends its last-stage convolutional layer activities to each other worker, basically a broadcast. Then these assemble a big dataset produced by each of the workers in its convolutional layer computing the fully-connected activities.
- Only one worker sends its last-stage convolutional layer activities to each other worker. The workers then compute the fully-connected activities on this sample and then begin to propagate the gradients at the same time with this result, the next worker repeat the same process with the rest of the workers.

It is also necessary to consider how the hardware configuration on which our model is deployed can help improve the performance of our models, this is shown in the work of Baidu (Referncia a Deep Image: Scaling up Image Recognition) Here we can see the emphasis that is given to the use of GPUs(Graphics processors Units) considering that their behavior is very optimal with compute-intensive algorithms. The Google Brain Project (hacer referencia a 4) has three GPU servers with 4 GPUs each, rival the same performance of a 1000 nodes (16000 cores) CPU cluster, taking into account that the SGD (Stochastic Gradient Descent) require high computing power, this type of architecture is to be an attractive solution when deploying a deep learning algorithm.

The figure 3.2.2 shows the physical architecture of a supercomputer called Minwa(colocar referencia 2) It is comprised of 36 server nodes, each with 2 six-core Intel Xeon E5-2620 processors. Each server contains 4 Nvidia Tesla K40m GPUs and one FDR InfiniBand (56Gb/s) which is a high-performance low-latency interconnection and supports RDMA. The peak single precision floating point performance of each GPU is 4.29TFlops and each GPU has 12GB of memory.

Finally be given a look at the solution recently released by google.[1] The Tensorflow is a framework recently released by google is based in python and It has a powerful component to numerical computation another to creation of neural networks also as we mentioned in chapter 2 Apache Spark is a open source cluster computing framework of general purpose.

The cited article shows how you can exploit the benefits of TensorFlow with spark, do not forget that Apache has 3 programming interfaces java, scala and python. Therefore using the Python programming module can install the library TensorFlow and therefore distribute the dataset for model training. On the other hand Spark can be used to obtain the hyperparameters necessary for best performance of our model, an example of these hyperparameters can be learning rate, number of neurons in each of the convolutional layer, values on regularization techniques for overfitting.

3.3 Novel Applications with DeepLearning

Currently the applications to these new techniques have grown in different ways, here are some novel applications

New theories such as **transfer learning** are already implemented in various jobs, an example of this is presented by Luis Alexandre [2], using RGB-D images for 3d recognition with convolutional networks, briefly explain this case.

RGB-D is a format used by the current cameras to provide more information about the intensity of the color of the images, the letter D means ”deep”. therefore the raw input would be composed by 4 channels one to each color further of deep channel.

The classic procedure would be create a ConvNet with n convolutional layers and m neurons in each layer in order to process this data type with four channels,running the backprogation algorithm to obtain the parameters of the model and proceed to classify the new images. Instead of this it proposed is create a ConvNet for each channel, at the end the 4 ConvNet will be combined in order to create the classification model, exploiting the fact that the similarity between the three channels is described. The key to the use of transfer learning would be to use the parameters o weights obtained from model with the minimum error, and use these such as initial parameters in the other ConvNets.

According with this survey better results than the original ConvNet were obtained, both in error and time.

New fields in art have been opened With deep Learning, nowadays can experience the sensation of creating paintings with the style of artists such as Picasso, Bangot using ConvNets.

The main idea is to use one pre-trained network of images and use its parameters or weights for the combination of two pictures the first with an original state and the second with the artist’s style required.

With the figure 3.3.1 going to explain the process in question.

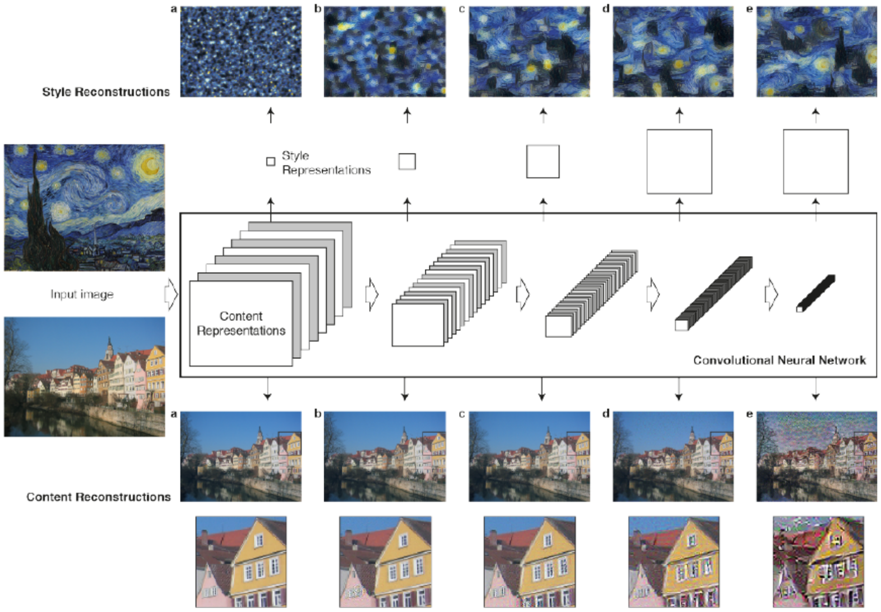


Figure 3.3.1: ConvNets

Chapter 4

Methodology.

4.1 Dataset Description

The dataset used contains photographs of landscapes, four particular environments are: the forest, beach, desert and mountains. All photographs are full color, its dimensions are 75x75 pixels. It was decided to use this kind of images to determine if the ConvNet could obtain key features for the classification of images, since a beach easily can have many elements in common with a desert.

Here's a sample in Figure4.1.1

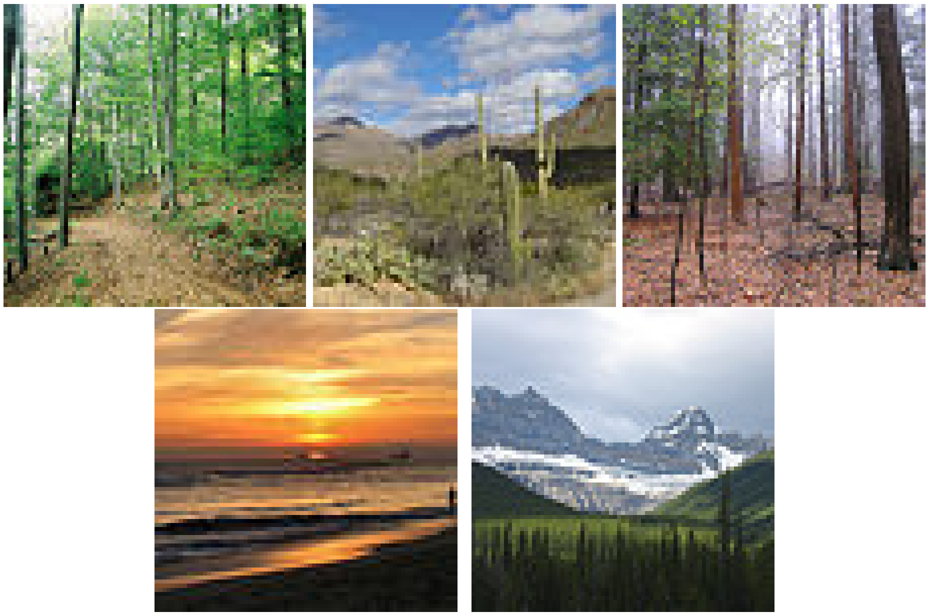


Figure 4.1.1: Image Examples.

Many of the graphics could get into the same category because of their similarity. For example the first two represent two different environments the forest and the desert, but there are many features that could make the classification is not good.

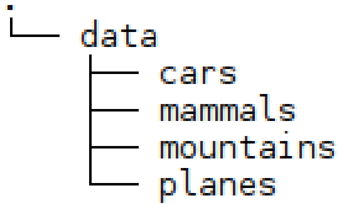
4.2 ConvNet Standalone version

4.2.1 Overview

For the development of the standalone version of supervised classifier, it used the DeepLearning4J(DL4J) framework, which is based on Java. This has a wide range of algorithms capable of handling various neural networks, as mentioned in previous chapters. At the time of this implementation we use the version 3.7. Being an Open Source product, some functions were not implemented or lacked documentation for use, so had to make knowledge transfer to the community that uses this library.

Furthermore the Canova library in its 13th version was used for the function of extraction, processing and load(ETL) of images from their raw format into its vector representation. It is noteworthy that it was necessary to create a small module that validates the physical state of the input data, since these should keep the same format for batch processing can be carried out without any problem. This I realized in Java to standardize the solution provided by Canova and DL4J.

This application requires three essential components: the ETL, the module defining the architecture and type of the neural network and the module that trains and evaluates the model.



```
.
└── data
    ├── cars
    ├── mammals
    ├── mountains
    └── planes
```

Figure 4.2.1: Directory Structure of DataSource

4.2.2 Creation of Input Layer

Load Data

In first place, it's necessary put the data on the local FileSystem in order to deal the raw data, to get this is necessary create a directory structure in a certain order. Within a directory must be created for each set of images that represent a real concept a subdirectory.

In the figure 4.1.1 it show an example of the structure mentioned, as we can see It has a main directory called data and several subdirectories inside it named with the concept they represent: cars, planes mammals, mountains.

Image Pre-processing

After preparing the data source, it must load the training data, one particular detail is that these data(image in native formats, jpg, png) must be scaled and standardized. For data recovery is used the classes ImageRecordReader and RecordReaderDataSetIteraror, the first takes as parameters the dimensions height, width and depth of the images contained in all subdirectories and the result is instanced in a java object. While the second uses the object obtained from the first and performs a load of certain number of data.

The third important point is the standardization of the training data for the this It used the StandardScaler class. It takes as its only parameter to the instantiated object of RecordReaderDataSetIterator class, this contain all data training in a vectorized form. The results are the global means and std deviations of whole images that will be stored in two files mean.bin and std.bin.The following code realize this job.

```
DataSet temporal;
DataSetIterator iter;
```

```
RecordReader trainReader = new
   ImageRecordReader(numRows,numColumns,nChannels,true);
      trainReader.initialize(new FileSplit(new File(labeledPath)));
      File meanFile = new File("mean.bin");
      File stdFile = new File("std.bin");

      if(!meanFile.exists() || !stdFile.exists()) {

          iter = new
             RecordReaderDataSetIterator(trainReader,4500,numColumns *
             numRows * nChannels,3);

          if (iter.hasNext()){
            temporal=iter.next();
         scaler.fit(temporal);
         scaler.save(meanFile,stdFile);
          }

      }
      else {
         scaler.load(meanFile,stdFile);
      }
```

Now again the data is load in memory and It must to be used the means and standard deviations obtained in the last step. The function transform belonging to the class StandarScaler is used. Also it's realized a random rearrangement of Training Data, with the following code:

```
   DataSet trainingSet;
      DataSetIterator iter2;
      File training = new File("train2.bin");
      if(!training.exists()) {
         iter2 = new
            RecordReaderDataSetIterator(trainReader2,4500,numColumns *
            numRows * nChannels,3);
         trainingSet = iter2.next();
         trainingSet.save(training);
      }
      else {
         trainingSet = new DataSet();
         trainingSet.load(training);
      }

      scaler.transform(trainingSet);
      trainingSet.shuffle();

      trainIter = new
          SamplingDataSetIterator(trainingSet,1000,trainingSet.numExamples());
```

4.2.3 Configuration Convolutional Layers

In this subsection we going to explore the configuration of ConvNet, focusing on the convolutional layers and a final fully-connected on order to get final categorization of the images. This represent the core of the algorithm.

Schematic Representation

The figure 4.1.3.1 represent the ConvNet implemented for this work.

In this case the input layer is composed of seven layers, the first six are of type convoulional while the last is pf type fully-connected. The following table describe each layer of this solution.

Convolution of the configuration is interesting. Always work in pairs. in other words convolutional and a pool. The first layer obtains the relevant features from the input data. This result is passed to pooling layer in order to reduce of dimensionality of this result.

Now vain to use our data to describe these two first layers. We have in this case 4 sets of images that represent places we have listed as mountains, beaches, forests and deserts.

In order to illustrate the process, we will use an image of one of the sets, this image is full color and its dimension in pixels is 75x75, being full color we deal with the RGB spectrum and therefore possess an

	Type	Patch Dimension	Number of neurons
Layer 1	Convolutional	4x4	20
Layer 2	SubSampling/Pool	2x2	–
Layer 3	Convolutional	5x5	40
Layer 4	SubSampling/Pool	2x2	–
Layer 5	Convolutional	5x5	70
Layer 6	SubSampling/Pool	2x2	–
Layer 7	Dense/Fully-conected	–	200
Layer 8	Output	–	number of classes

Table 4.2.1: List of Layers in the present ConvNet

entry 75x75x3, the last digit represents the number of channels using this image.

On this image we apply a patch of 4x4 pixels and slide this 2 pixels each time over the entire image of 75x75 pixels to complete the scanning. This will activate the convolution between the input image and the convolution layer. It is the same process as in a neural network so that an activation function is used. In each of the convolutional layers, this framework allows enable a global configuration of this function or this can do in each of the layers independently.

In our project the activation function selected was ReLu(Rectified Linear Unit) over the classical sigmoid, because nowadays this represent the state of the art of the activation function. This makes the training process is faster[4] than other algorithms without compromising accuracy.

The next step is realize a DownSampling over the 20 feature maps obtained from the last convolutional layer. This patch in this case of 2x2 will get the max value of this area. At the end we will reduce the resolution of this results.

Landing in the framework dl4j, architecture mentioned in the table 4.2.1 is displayed in the following code fragment:

```
   MultiLayerConfiguration.Builder builder = new
       NeuralNetConfiguration.Builder()

//hyperparameters configuration.
              .seed(seed)
```

```
            .iterations(iterations)
            .activation("relu")
            .weightInit(WeightInit.XAVIER)
            .gradientNormalization(GradientNormalization.RenormalizeL2PerLayer)
            .optimizationAlgo(OptimizationAlgorithm.STOCHASTIC_GRADIENT_DESCENT)
            .learningRate(0.01)
            .momentum(0.9)
            .regularization(true)
            .l2(0.00005)
            .updater(Updater.ADAGRAD)
```

HyperParameters

Before of set the layers, first is necessary instance the class NeuralNetConfinguration which contains several hyperparameters that going to be used to the right performance of the ConvNet.

- **.seed**: It is an initialization parameter associated with the random number generation. It does not require any knowledge to give the value regularly digits 123 can be used.
- **.iterations**:: This is a parameter of the ConvNet and determines the number of times (n) to be trained a set of data on the entire model. Thanks to this the ConvNet update its parameters n times, reaching a convergence in the classification model.
- **.activation**: This is the activation function of each of the layers of the model in our case we have selected ReLu, is a global setting but can be overridden in the creation of each of the layers that make up the model, an example of this is the softmax activation in the output Layer. It is used to classification of inputs, this converts vectors into class probabilities corresponding to some label(mountains, beach, deserts, forest)
- **.WeightInit**: It is a parameter that defines how will the initial weights of our network, as we know the first convolutional layer is fed by the value of a given pixel by an associated weight, this initial weight can be generated randomly or also subject under certain function. This function must ensure that the weights are neither too large nor too small. The algorithm xavier was chosen to do this. Referencia a Understanding the difficulty of training deep feedforward.
- **.optimizationAlgo**: Takes different values: LINE.GRADIENT.DESCENT, stochastic GRADI-

ENT.DESCENT, BFGS, in our case we use SGD because optimizes gradient descent and minimizes the loss function during network training.(Verify this)

- **.regularization**: Active several techniques of regularization such as L1/L2, with the purpose of avoid the overfitting in our model. Basically adds an extra term to the loss function. To set the value of the L1/L2, it must use .L1(value)
- **.updater**: This terms refers to training mechanisms such as momentum, RMSProp, adagrad. Using these methods can obtain quickly a network training, using it with a optimization algorithm. Once that we had selected the updater for instance momentum, is necessary setup this value with next sentence .momentum(value)

Once defined the Hyperparameters, we going to continue with the configuration of the convolutional layers. So quickly we'll give an overview of the necessary elements for configuration.

```
.layer(0, new ConvolutionLayer.Builder(4, 4)
                .name("cnn1")
                .nIn(nChannels)
                .stride(2, 2)
                .nOut(20)
                .build())
```

The first line defines the size of the patch or kernel and the type of layer, in this case is the first Convolutional layer, the next parameters inside of constructor layer define the following:

- .nIn(values): involves the number of channels of the input image in our case the value is 3 because the image is full color.
- .stride(shape): is the parameter defined as the kernel to slide within the image.
- .nOut(value): determines the number of feature maps or kernels that we will use. It should be noted that the activation function is the same as defined above (RELU).

```
.layer(1, new SubsamplingLayer.Builder(SubsamplingLayer.PoolingType.MAX,
   new int[]{2, 2})
                .name("pool1")
                .build())
```

The following code is easy to interpret. In the constructor we .layer define the type of layer to be created in this case is a subsampling, we decided the algorithm used to perform the reduction in this case MAX, although there are other like AVERAGE but for our purpose and according to the different papers the max pool is preferable. As the last parameter in the constructor have the size of the region to explore.

```
.layer(4, new DenseLayer.Builder()
                .name("ffn1")
                .nOut(200)
                .dropOut(0.5)
                .build())
```

In this code the Fully-Connected layer is defined, we noticed little difference in the constructor of the layer, this has changed ConvolutionalLayer to DenseLayer representing this new type of layer. Another interesting detail is that a new regularization technique called Dropout is added.

```
.layer(5, new
   OutputLayer.Builder(LossFunctions.LossFunction.NEGATIVELOGLIKELIHOOD)
                .nOut(outputNum)
                .activation("softmax")
                .build())
```

As a last fragment we have the creation of the Output layer, important points to mention are the parameter nOut inside of constructor, which should be equal to the number of groups we have in our dataset in this case the value is 3. The activation function changes Relu to softmax. Not forgetting that this layer is where it's defines the loss function in our case takes the value of NEGATIVELOGLIKEHOOD.

4.2.4 Training and Evaluation Module

Its function is to divide the training data in mini batches and start using these data the ConvNet iterating until you reach the data converge thanks to the backpropagation algorithm. Basically this module uses the configuration of the network model defined in the previous section. And this configuration is passed as a parameter to the MultilayerNetwork class.

The following code performs this function

```
      MultiLayerNetwork trainedNetwork = new MultiLayerNetwork(conf);
      trainedNetwork.init();
      trainedNetwork.setListeners(new ScoreIterationListener(0));

    StandardScaler scaler = new StandardScaler();
       DataSetIterator iter = new
          RecordReaderDataSetIterator(recordReader,4500,
          16875,labels.size());
         DataSet TrainingSet = iter.next();
         scaler.fit(TrainingSet);

         ////begin the training
        scaler.transform(TrainingSet);
        TrainingSet.shuffle();
        DataSetIterator trainIter = new
            SamplingDataSetIterator(TrainingSet,300,TrainingSet.numExamples());

        while(trainIter.hasNext()) {
         DataSet next = trainIter.next();
         next.labelCounts());
         trainedNetwork.fit(next);
      }
```

This part loads the data and their labels, makes a standardization and suffle With This. At the end takes a part of the data for training. and begins to train and to adjust the weights of the model.

The evaluation module makes it the same way as the training.

- Load the data from testing dataset.
- These are standardized
- A random sample for testing is obtained.
- This is passed to the trained model and this is classified.

As a result we obtain a matrix of confusion that will be able to deliver the necessary data to obtain quality metrics such as recall, f1-score, precision and accuracy.

In this code we can see how the class evaluation is used to create the confusion matrix.

```
StandardScaler scaler_testing = new StandardScaler();

DataSetIterator itertest = new
   RecordReaderDataSetIterator(recordReaderTest,150,
   16875,labels.size());
DataSet TestingSet = itertest.next();
scaler_testing.fit(TestingSet);
scaler_testing.transform(TestingSet);
TestingSet.shuffle();

Evaluation evaluation = new Evaluation(labels.size());
DataSet testIterNext = TestingSet.sample(60,true);

evaluation.eval(testIterNext.getLabels(),
   trainedNetwork.output(testIterNext.getFeatureMatrix(), false));
System.out.println(evaluation.stats());
```

4.3 Distributed Version

4.3.1 Overview

In this section we'll look at the version distributed ConvNet for this going to uses the Apache Spark platform, because it has an interface for programming in java so it becomes a native solution for our framework DL4j.

The whole process will use JavaRDD or just RDD to realize the parallelization, this is the heart of Apache Spark and provides us with a high level of abstraction for its implementation.

The outline to follow basically is to use what has already been seen at the time, with the only difference that the loaded data handling it done with RDD, in other words we will use a distribution scheme oriented data. Dividing this in a number of subsamplings and making the model train with your data. Then in each of the steps to perform an average of weights or parameters found during the parallel training.

4.3.2 Parallelization Process

As we described in section 2, Spark has an administrator named ”Driver” that is responsible for coordinating the execution of processing threads within the cluster. Within the DL4J framework possess the SparkDL4jMultilayer class that contains the definition of our model, number of layers, subsampling, patches among others.

We will use an instance of JavaSparkContext to initialize parallel environment, indicating that in each iteration parallel, the threads communicate parameters or weights obtained in each of the parallelized models and average value for the entire model.

```
final JavaSparkContext sc = new JavaSparkContext(new
  SparkConf().setMaster("local[*]").set("spark.driver.maxResultSize","3g")
     .setAppName("Paisajes").set(SparkDl4jMultiLayer.AVERAGE_EACH_ITERATION,
       String.valueOf(true)));
```

The load process for training and testing data have been encapsulated in a class called Datossetup for better handling. We load the dataset in a list of the DataSet class and send to parallelize this, in this case we have divided to 10 the total size of the dataset for processing.

```
DataSetSetup setSetup = new DataSetSetup();
setSetup.setup();
DataSet next = setSetup.getTrainIter().next();
next.shuffle();
List<DataSet> list = new ArrayList<>();
for(int i = 0; i < next.numExamples(); i++) {
   list.add(next.get(i).copy());
}
JavaRDD<DataSet> dataSetJavaRDD = sc.parallelize(list,list.size() /
   10);
```

In the training process we will use:

- The configuration of the ConvNet obtained in the same manner as in the standalone version.
- The SparkDL4JMultilayer class, which used a spark context object to take control of part of the model and send it to one of the cores of the cluster.

This can be summarized in the following code lines.

```
MultiLayerConfiguration conf = setSetup.getConf();
//train the network
SparkDl4jMultiLayer trainLayer = new
    SparkDl4jMultiLayer(sc.sc(),conf);
for(int i = 0; i < 5; i++) {
   //fit on the training set
   MultiLayerNetwork trainedNetwork =
      trainLayer.fitDataSet(dataSetJavaRDD);
}
```

Chapter 5

Evaluations and Results

5.1 Quality of Classification

It'll be used the following metrics: precision. recall, accuracy and F-Score in order to evaluate the results obtained from this ConvNet.

The precision in the field of classification is the number of true positives divided by the total number of elements labeled as belonging to the positive class.

$$Precision = \frac{true.positives}{true.positives + false.positive} \tag{5.1}$$

The true positive is the number of items correctly labeled as belonging to the positive class while false positive is represented by items incorrectly labeled as belonging to the class.

The same way in this context Recall is defined as the number of true positives divided by the total number of elements that actually belong to the positive class.

$$Recall = \frac{true.positive}{true.positive + false.negative} \tag{5.2}$$

Accuracy is a weighted arithmetic mean of Precision and Inverse Precision as well as a weighted arithmetic mean of Recall and Inverse Recall [6] and its formula is defined by:

$$Accuracy = \frac{true.positive + false.positive}{true.positive + false.positive + true.negative + false.negative} \tag{5.3}$$

Finally we have the F-score that takes account both the precision and the recall, using the concept of harmonic mean with the precision and recall. And the classical F score is defined by:

$$FScore = 2 \times \frac{precisionxrecall}{precision + recall} \tag{5.4}$$

Dl4j provides the matrix of confusion for our analysis when evaluating the model with test data, also it has functions to achieve the above metrics

5.2 Results Standalone version

For the StandAlone version, we have used the following hardware environment: Intel Core i5-2400 CPU (3.10 GHZ), 4GB in RAM and 250GB hard disk drive. The experiments were realized with images in full color.

For this part of the experiment it was used a dataset of 2100 color images for the training, divided in 3 categories: beach, desert, forest. The training process was done using mini-batches of 300 images with 10 iterations. Likewise 60 elements were used, randomly distributed in the categories for testing the model.

The result of the model executing during 02:31h gave us the following matrix of confusion:

	B	D	F
B	12	0	6
D	4	19	2
F	4	7	6

Table 5.2.1: Confusion Matrix

Using the functions for F1 score, Recall, precision and accuracy shown in the previous section, we obtain the following results:

Another interesting graph is seen as the model learns as their weights are adjusted backpropagation method. Here is how this model as the minutes pass, this is tied to the number of iterations performed by

Accuracy: 0.6167
Precision:0.5932
Recall: 0.5864
F1 Score: 0.5898

Table 5.2.2: Results of Model

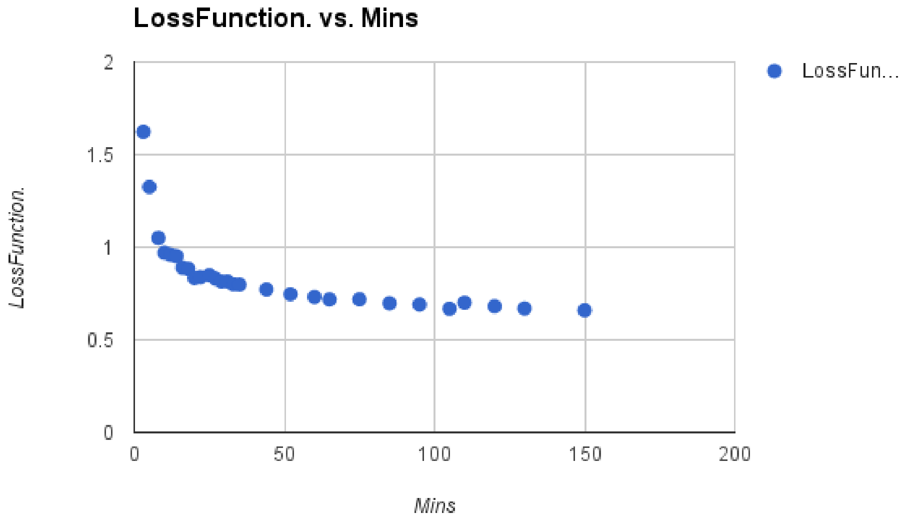


Figure 5.2.1: Learning Model

the model in a mini-batch

This type of model learns more when a lot of image in inputs layer, the hardware limitations Unfortunately not allowed to expand too much data entry. Since the more images, we have more input nodes, more convolution operations that make thus generating a high memory consumption and the classic examples such as JavaHeap Executions errors.

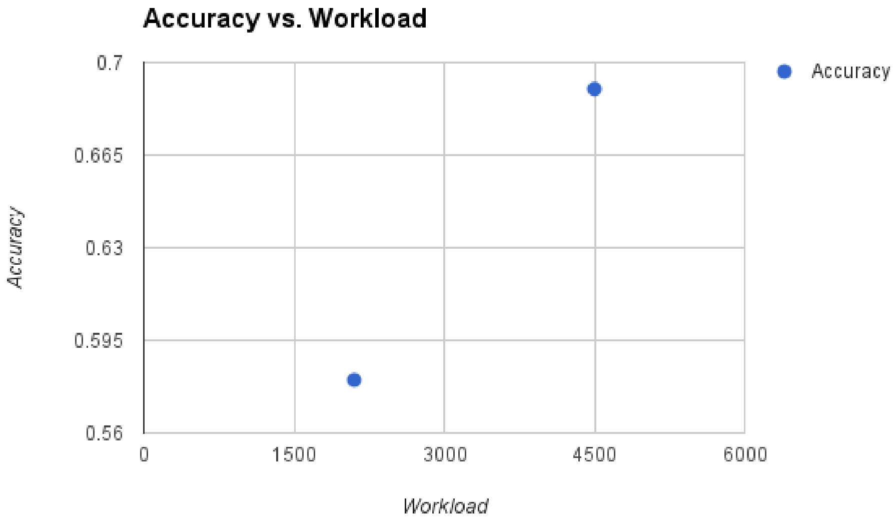


Figure 5.3.1: Learning Model

5.3 Results Distributed version

For these testing, we have used Spark on MareNostrum (Spark4mn), which is a implementation of Apache Spark over Marenostrum SuperComputer. In first place I'm going to give a overview of this supercomputer. Marenostrum[10] is the Spanish supercomputer provided by BSC. It is an IBM System X iDataplex based on Intel Sandy Bridge EP processors at 2.6GHZ,(two 8core Intel Xeon processors E5-2670 per machine), 2GB/core(32GB /node) and 500GB of disk(IBM 500 GB 7.2k 6Gbps NL SATA 3.5). Nowadays, the supercomputer has 48.896 Intel Sandy Bridge cores in 3056 JS21 nodes, and 85 Xeon Phi 5110P in 42 nodes, with more than 104.6TB of main memory and 2PB of GPFS disk storage. All nodes are interconnected with technology of Inifiniband FDR10 network. Marenostrum has a peak performance of 1.1 Petaflops.

To use the Spark on MareNostrum, we need the load module named spark4mn and submit a jar file that contains our code, also is necessary add a configuration file that define the use of resources assigned. Defining the number of nodes in spark and the workers per nodes, furthermore the path with the application inside of Server.

For this set of tests, the number of images will increase to 4500. In addition began to modify certain hyperparameters. The most significant was ratelearning and regularization L2.

In Figure 5.3.1 we can see how it affects the number of images treated in the training phase to the accuracy of our model. Without the restriction of hardware because the execution is the run on the supercomputer we have a better quality measures classified data.

Using the same configuration standalone environment, We can get the following forecasting:

	B	D	F
B	57	51	0
D	20	24	5
F	5	1	77

Table 5.3.1: Confusion Matrix Spark

Analyzing this result, we can look to the classified images that have an overlap are Beach and Desert, could be that many features of the images retain certain correlation between shapes and objects that compose it.

Accuracy: 0.6623
Precision:0.6376
Recall: 0.6423
F1 Score: 0.6348

Table 5.3.2: Scores of Model

In the Figure**??** we can see how It have changed the time of learning with respec to the standalone version, of course that we have a better hardware but is interesting view the difference between them, dealing initialy in minuts and now seconds.

Finally the same elements of the ConvNet were used to determine its scalability, when increasing the number of cores in the environment the results shows the behavior is expected, reducing training time as the cores are increased. However it should be noted that the training time is not reduced in the same proportion as resources are increased.We see in Figure 5.3.3

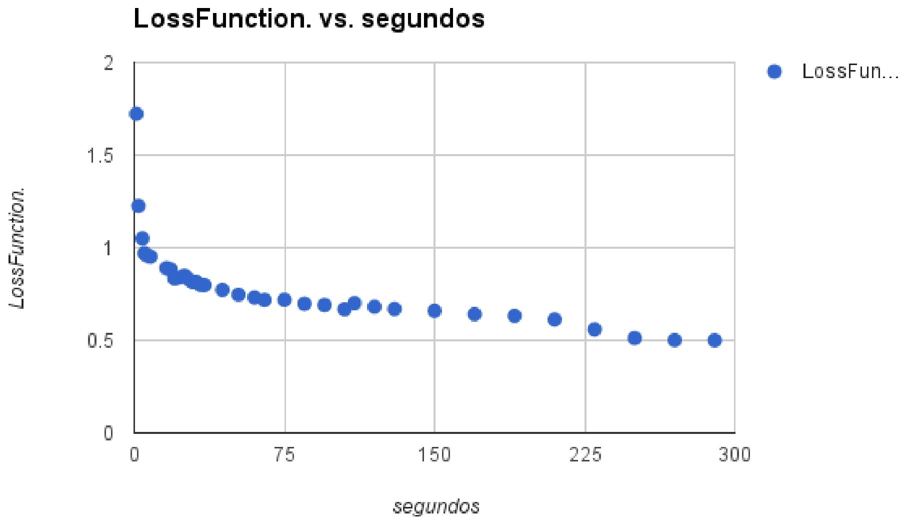


Figure 5.3.2: Learning Model

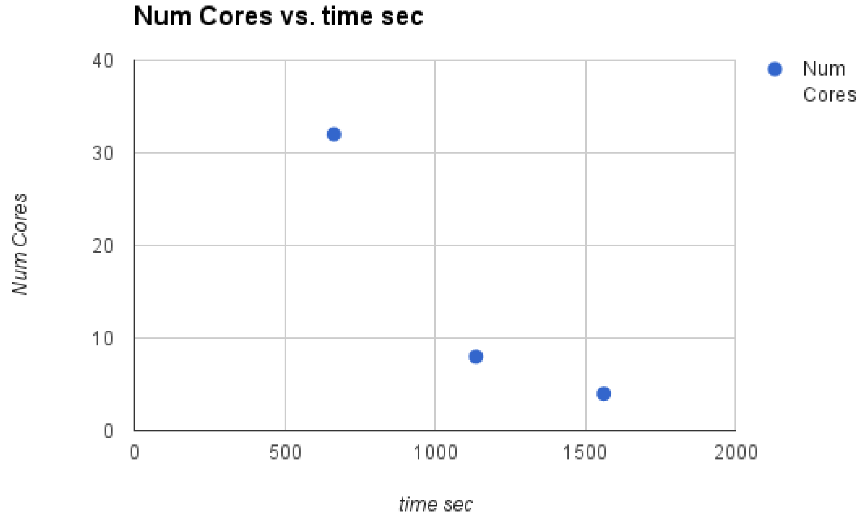


Figure 5.3.3: Learning Model

Chapter 6

Conclusion and Future Works

6.1 Conclusions

Image recognition has always been a challenge in the field of artificial intelligence and computer vision. With this work he managed to unite two approaches that were separate, they are: the analysis of data on images using techniques of deep learning and use of current platforms able to parallelize services as it is Spark, even more determine whether in high-performance as in the case of supercomputers specifically on Marenostrum, can coexist smoothly.

The tests were given correct answers in managing forecasts, for example an increase in the input data leads to a better predictive model, during the tests on the distributed environment it was found that at every step of calculating weights, node or core that possessed at that time a part of the dataset sent this and averaged with the other cores to continue training, and this it was made thanks to the coordination of the different threads of execution that provides Spark and are exploited by DL4J. I also came to the conclusion that by modifying certain parameters in the ConvNet such as regularization methods by assigning small values provide us with a better result when testing the model.

The DL4j use for the realization of this project was successful in view that their native language is java and perfectly adapted to the environment distributed spark, compared to standalone model but there is a limit on the hardware where it ran, a moderate set of images could reach 59 percent accuracy, this is an acceptable result since it must be recognized that the set of images to be analyzed in many instances they shared similar characteristics. In the tables of confusion we can easily see that beach and desert images overlapped easily. By moving to the distributed version we improved performance as the algorithm was shown to be portable between versions since no drastic changes for operation in Spark is needed. Besides their accuracy 10 points was raised with the increase in input data. In addition during the tests I just took you initially achievement deploy the application on a local spark and observe how the cores of the PC on which he settled spark were used by 90 percent of its capacity.

6.2 Future Works

Now that this ecosystem has been tested in a controlled environment, this project could be taken as a supplement to other projects requiring Image classifying them. It should be noted that the best models were obtained by modifications in certain hyperparameter, this leaves the door open to continue the creation of a library looking for the best hyperparameter for a better result.

In the current version uses a file system to store training data, this could be modified to accept data without having to be temporarily stored to disk, taking into account that spark performs all processing in memory, using RDD for this.

An important detail to mention is that although the model can be saved for later use, this is limited to not implement trasnfer-learning at the moment. This may be modified to add one more layer of adaptability to the framework.

One of my personal goals is to use this type of study to achieve solve problems in my country as anomaly detection or diseases in cocoa farms, where images can be captured by drones to process them and based on supervised learning algorithms prevent the disease spreading to other crops.

Bibliography

[1] ABADI, M., AGARWAL, A., BARHAM, P., BREVDO, E., CHEN, Z., CITRO, C., CORRADO, G. S., DAVIS, A., DEAN, J., DEVIN, M., ET AL. Tensorflow: Large-scale machine learning on heterogeneous distributed systems. *arXiv preprint arXiv:1603.04467* (2016). 30

[2] ALEXANDRE, L. A. 3d object recognition using convolutional neural networks with transfer learning between input channels. In *Intelligent Autonomous Systems 13*. Springer, 2016, pp. 889–898. 31

[3] DATA, B. for better or worse: 90% of world's data generated over last two years. *SCIENCE DAILY, May 22* (2013). 3

[4] KRIZHEVSKY, A., SUTSKEVER, I., AND HINTON, G. E. Imagenet classification with deep convolutional neural networks. In *Advances in Neural Information Processing Systems 25*, F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, Eds. Curran Associates, Inc., 2012, pp. 1097–1105. 38

[5] MCCULLOCH, W. S., AND PITTS, W. A logical calculus of the ideas immanent in nervous activity. *The bulletin of mathematical biophysics 5*, 4 (1943), 115–133. 6

[6] POWERS, D. M. Evaluation: from precision, recall and f-measure to roc, informedness, markedness and correlation. 46

[7] ROSENBLATT, F. The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological review 65*, 6 (1958), 386. 6

[8] SCHMIDHUBER, J. Deep learning in neural networks: An overview. *Neural Networks 61* (2015), 85–117. 7

[9] SUN, Y., WANG, X., AND TANG, X. Deep learning face representation from predicting 10,000 classes. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition* (2014), pp. 1891–1898. 3

[10] TOUS, R., GOUNARIS, A., TRIPIANA, C., TORRES, J., GIRONA, S., AYGUADÉ, E., LABARTA, J., BECERRA, Y., CARRERA, D., AND VALERO, M. Spark deployment and performance evaluation on the marenostrum supercomputer. In *2015 IEEE International Conference on Big Data, Big Data 2015, Santa Clara, CA, USA, October 29 - November 1, 2015* (2015), pp. 299–306. 49

[11] TOUS, R., TORRES, J., AND AYGUADÉ, E. Multimedia big data computing for in-depth event analysis. In *Multimedia Big Data (BigMM), 2015 IEEE International Conference on* (April 2015), pp. 144–147. 6