Uso y adaptación de un sistema de Deep Learning para análisis de imágenes en un Supercomputador

Borja Arias Navarro

Ponente: Jordi Torres

Director: David Vicente

Co-director: Jorge Rodríguez

19 de abril de 2016

Universitat Politècnica de Catalunya

Barcelona Supercomputing Center
Centro Nacional de Supercomputacion

Índice

Resumen **4**

Resum **5**

Abstract **6**

Agradecimientos **7**

1 Introducción **8**

1.1 Contexto 8

1.2 Objetivo 9

1.3 Partes interesadas 10

1.4 Áreas de estudio 11

1.5 Estado del arte 18

2 Gestión del proyecto **20**

2.1 Metodología 20

2.2 Planificación 26

2.3 Hardware 32

2.4 Software 33

2.5 Presupuesto 34

2.6 Sostenibilidad 40

3 Implementación de la solución **42**

3.1 Instalación local 42

3.2 Instalación remota 43

3.3 Análisis de resultados 45

4 Conclusión **52**

4.1 Alternativas 52

4.2 Trabajo futuro 53

Appendices **59**

Apéndice A Script extractor de datos del output **59**

Apéndice B Script contabilizador de las horas de cpu **61**

Apéndice C Código definitivo ejemplo MNIST **64**

Apéndice D Diagramas de Gannt **77**

D.1 Planificación inicial 77

D.2 Revisión de la planificación 78

Índice de figuras **79**

Índice de cuadros **80**

Resumen

El Big Data está a la orden del día en todo tipo de ámbitos. Por otro lado, el Deep Learning se deja ver menos fuera del ámbito informático, pero cada vez esta más presente en nuestras herramientas diarias. El Deep Learning es un conjunto de técnicas de aprendizaje automático que necesita de muchos datos para poder ofrecer buenas predicciones. Esto hace que el concepto del Big Data y el de Deep Learning estén relacionados.

En el BSC no se dispone de máquinas exclusivas para Big Data ni Deep Learning. Pero en un intento por modernizar parte de los servicios a estos paradigmas, se puso en marcha un sistema que permitía al usuario lanzar herramientas de Big Data en un cluster tradicional, como el MareNostrum. Encima de estas herramientas de Big Data puede pueden montarse otras de aprendizaje automático.

Este proyecto trata de la implantación y las pruebas de rendimiento con el análisis de imágenes de un sistema de Deep Learning, concretamente DL4J, encima de Spark.

Resum

El Big Data està a l'avantguarda en tot tipus d'àmbit. D'altra banda, el Deep Learning és menys visible fora de l'àmbit informàtic, però cada vegada està més present en les eines que utilitzem cada dia. El Deep Learning és un conjunt de tècniques d'aprenentatge automàtic, la qual necessita moltes dades per tal d'oferir unes bones prediccions. Això fa que el concepte de Big Data i el de Deep Learning estiguin relacionats.

Al BSC no disposem de màquines exclusives per al Big Data i tampoc pel Deep Learning. Però, en un intent de modernitzar part dels serveis cap a aquests paradigmes, es va posar en marxa un sistema que permit a l'usuari engegar eines de Big Data en un clúster tradicional com el MareNostrum. A sobre d'aquestes eines de Big Data, es poden muntar d'altres per l'aprenentatge automàtic.

Aquest projecte tracta sobre la implantació i les proves de rendiment amb l'anàlisi d'imatges d'un sistema de Deep Learning, concretament DL4J, sobre Spark.

Abstract

The Big Data is on the cutting edge in all kind of areas. On the other hand, Deep Learning is less visible outside of computing scopes, but it his presence is increasing in the tools we use every day. Deep Learning is a set of techniques of machine learning which require big amount of data in order to make good predictions. This fact is what creates a link between the concept of Big Data and Deep Learning.

In the BSC we don't have any machine designed exclusively for Big Data neither for Deep Learning. Anyway, in an attempt for modernize some of the services towards this paradigms, a system to allow the user start Big Data tools was deployed in a traditional cluster like MareNostrum. On top of this tools for Big Data, some other machine learning tools can be deployed.

This project is about the implantation and the image analysis' performance tests of a Deep Learning system, more concretely DL4J, on top of Spark.

Agradecimientos

Me gustaría dar las gracias a la gente, que directa o indirectamente ha participado en este proyecto haciendo que tenga un sentido y dándome ánimos.

Primeramente a mis directores (David Vicente y Jorge Rodriguez) y mi ponente (Jordi Torres), por guiarme en el proyecto y corregir todos los errores que han ido encontrando. También me gustaría agradecer a mis compañeros del BSC por el interés, ánimos e ideas que han podido aportar. Especialmente a Carlos Tripiana, que me ayudó con Spark4MN y que va arreglando todos los errores que le veo.

Por último, y no menos importante: a mis amigos que me han ayudado a despejarme cuando era necesario; a mi pareja, Anna, que me ha escuchado todos los problemas y me ha hecho ver soluciones con sus comentarios; y a mis padres y mi hermano, por prestarse a escucharme y revisar los documentos y me han perseguido para que no me despistara.

1 Introducción

1.1 Contexto

En el departamento de Operaciones del Barcelona Supercomputing Center – Centro Nacional de Supercomputación (BSC de ahora en adelante) nos ocupamos, entre otras cosas de mantener al día el supercomputador MareNostrum, así como de acomodar las nuevas tecnologías a la máquina y la máquina a las nuevas tecnologías.

Para equipos personales o pequeños clusters, adaptarse a nuevas tecnologías, arquitecturas o paradigmas puede ser sencillo (dependiendo, evidentemente, del presupuesto y los planes). Cuando pasamos a clusters más grandes como MareNostrum, en el que tenemos 3.056 nodos de cómputo (sin contar los nodos de almacenamiento de datos, etc) esto supone bastante más trabajo y un presupuesto mucho mayor. Por ello, no es una opción intentar adaptar este tipo de clusters a las tendencias del momento que, además, pueden ser específicas para cierto tipo de aplicaciones y no del todo generales.

Desde hace unos años se habla, cada vez más, del Big Data y no solo en el ámbito informático. Al tratar con gran número de datos, los procesos relacionados con Big Data necesitan sistemas específicos con los que trabajar. Los clusters dedicados a Big Data siguen un paradigma de computación centrado en datos, en cuyo caso, la variable a maximizar es el número de datos tratados (medido en archivos, o bytes). Es decir, se considera mejor aquel sistema que permita procesar más cantidad de datos por unidad de tiempo. Por contra, en los clusters tradicionales (como podría ser MareNostrum) están centrados en cálculo; con el objetivo de hacer el mayor número de cálculos en coma flotante por segundo como sea posible.[1] No tienen por qué ser contrario uno de otro, pues a mayor potencia de cálculo también puedes procesar mayor cantidad de datos. Sin embargo, cada herramienta está diseñada para una tarea y suele

dar mejor resultado usar cada una en su ámbito.

Hace algún tiempo que también se oye hablar de unas técnicas de aprendizaje automático que han resurgido para convertirse en el estado del arte para algunos tipos de problema de inteligencia artificial. Hablamos de Deep Learning y las redes neuronales, cuyos algoritmos han demostrado tener mejor rendimiento en GPUs debido a que se suelen usar enormes redes[2]. Las cuales pueden operarse como matrices, y en esto las GPUs son especialistas. Estas redes necesitan una gran cantidad de datos para entrenarse a fin de poder mejorar las predicciones. Aquí es donde entra el Big Data. Por contra, esto hace que no sean la mejor opción para problemas con pocas muestras, debido a que es costoso entrenar a una de estas redes y que otros algoritmos/estructuras clasificadores (por ejemplo los Arboles de decisión).

1.2 Objetivo

A finales de 2014 empezó un proceso por adaptar software para clusters de Big Data y encajarlo dentro del sistema de colas (LSF), los nodos del sistema, la configuración de la red, etc. Este proceso empezó con la adaptación de Spark[9] con una serie de scripts (de configuración y control) y variables para permitir a los usuarios del sistema levantar una Spark (Spark4MN) totalmente funcional, como si estuviese funcionando de forma nativa.

En este proyecto se pretende montar un framework[1] de Deep Learning encima de Spark4MN e investigar hasta que punto dicho framework escala[2]. Además de esto, trabajaremos con un caso real de deep learning en el que analizaremos diferentes data sets de imágenes con el fin de asegurar que la *performance* de la configuración el la mejor que puede ofrecer este *stack* de software en el hardware de MareNostrum.

[1] conjunto de herramientas de software: librerías, aplicaciones, etc.
[2] cuanto mejora el rendimiento al aumentar el número de workers

1.3 Partes interesadas

Se considera parte interesada todo agente que pueda llegar a hacer algún servicio de este proyecto. Con lo cual, se tienen en cuenta los siguientes.

1.3.1 Usuarios de MareNostrum

Los usuarios de MareNostrum tienen como objetivo conseguir reducir el tiempo tanto de desarrollo como de ejecución de sus programas de Deep Learning en un 20 %, durante lo que duren las pruebas de sus investigaciones en el Supercomputador. Reducción del tiempo de desarrollo ya que el framework evita que tengan que implementar por si mismos los algoritmos, dándoles una capa de abstracción agilizando el trabajo hacia los resultados.

1.3.2 Barcelona Supercomputing Center

El BSC como empresa, tiene el objetivo de habilitar nuevos servicios que usen las tecnologías más actuales para conseguir aparecer en nuevas publicaciones sobre, por ejemplo, Big Data y Deep Learning y conseguir un 40 % más de proyectos privados en un año.

1.3.3 Desarrolladores DL4J

Los desarrolladores de DL4J tienen por objetivo expandir su software a más usuarios para conseguir el doble de feedback de cara a estabilizar el software y poder lanzar la release[3] oficial en 9 meses.

[3] lanzamiento

1.4 Áreas de estudio

En este proyecto hay distintos áreas de estudio fácilmente identificables. El primero es Deep Learning y las Convolutional Neural Networks (CNN de ahora en adelante) ya que el software que se pretende instalar proporciona herramientas de este tipo de aprendizaje automático. El segundo es Spark y el paradigma centrado en procesamiento de datos. Finalmente, tenemos la arquitectura de un Supercomputador. Aunque sobre este último no se va a trabajar directamente, es necesario conocerlo y definirlo para contemplar las diferentes soluciones de software que podemos escoger.

1.4.1 Deep Learning y CNN

En este proyecto se pretende instalar un framework de un tipo de aprendizaje automático (Deep Learning) y probar que funciona de forma eficiente. Por lo tanto, es necesario conocer que tipos de algoritmos usa y de que manera funcionan.

Deep Learning: Es una colección de técnicas estadísticas de aprendizaje automático utilizados para aprender características jerárquicas. A menudo, se basan en redes neuronales. Estas técnicas han mejorado cosas como el reconocimiento de voz, la visión por computador o el procesamiento del lenguaje natural. [6, 7] En otras técnicas de aprendizaje automático se definen los patrones que se quieren detectar, pero esto no es posible en aplicaciones del mundo real. Por ejemplo, si queremos reconocer coches en una imagen, podemos contar el número de ruedas. Tendremos, pues, que definir cómo es una rueda, pero esto puede ser complicado en imágenes no óptimas, por ejemplo con sombras o reflejos. Las técnicas de Deep Learning nos permite expresar representaciones complejas (o abstractas) a partir de otras más simples.[31]

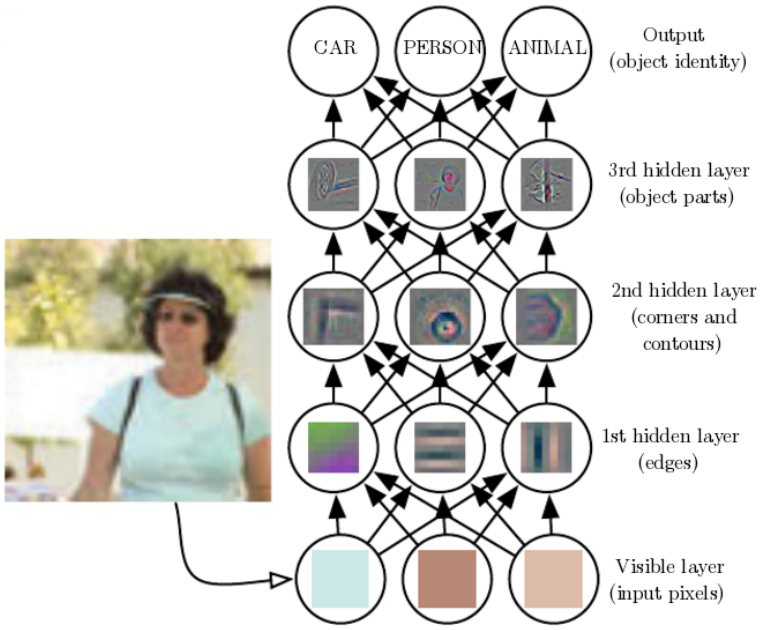


Figura 1: Ejemplo de red neuronal analizando una imagen [31]

Red Neuronal Multicapa: Una red neuronal artificial (ANN por sus siglas en ingles) se basa en las redes neuronales biológicas (BNN por sus siglas en ingles), pero a una escala mucho más pequeña. En una ANN tenemos dos elementos. Por un lado las neuronas y los enlaces, que conectan una neurona con otra. Cada enlace en una red neuronal tiene un peso y cada neurona un umbral (b).

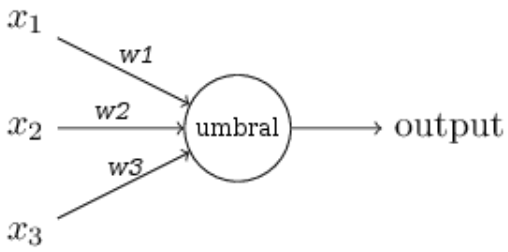


Figura 2: Representación de una neurona en una ANN

Las neuronas son consideradas una unidad computacional. En el caso más simple, la neurona de tipo *perceptrón* calcula el valor de salida con una función escalón unitario:

$$\text{output} \quad = \quad \begin{cases} 0 & \text{if } \sum_j w_j x_j \leq \text{ b} \\ 1 & \text{if } \sum_j w_j x_j > \text{ b} \end{cases} \tag{1}$$

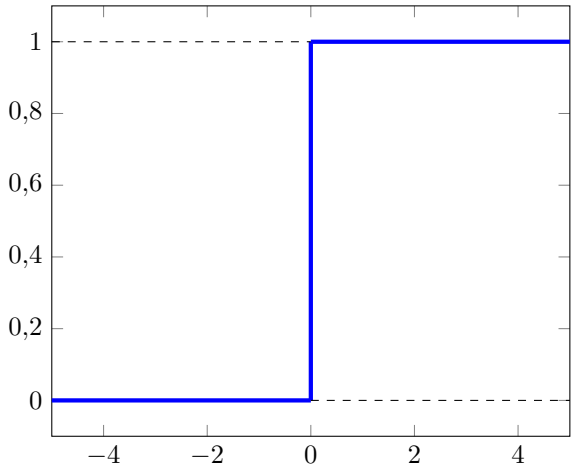


Figura 3: Función escalón unitario

Otro tipo de neurona, algo más interesante son las que siguen una función *sigmoidea* y se conocen como neurona sigmoidea (a veces llamadas también neuronas logísticas). Estas neuronas permiten que los valores de entrada y salida sean reales entre 0 y 1, ofreciendo una mejor precisión.

$$output = \sigma(w \cdot x + b) \equiv \frac{1}{1 + e^{(-\sum_j w_j x_j - b)}}. \tag{2}$$

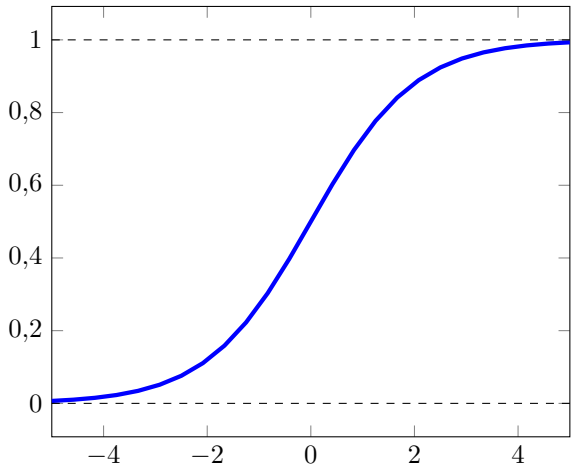


Figura 4: Función sigmoidea

Una ANN es, pues, un conjunto de neuronas enlazadas unas con otras. Dependiendo de la geometría resultante de enlazar las neuronas, se diferencian los tipos de redes.

Las ANN datan de 1943, cuando Warren McCulloch (neuropsicólogo) y Walter Pitts (matemático) modelaron una simple red neuronal con un circuito eléctrico[23][36]. No es hasta 1960 que aparecen, de la mano de Frank Rosenblatt, las neuronas llamadas perceptrón. En esos años las redes neuronales,

por motivos tecnológicos, se construían como una capa de entrada enlazada con una de salida. Con esto, las redes neuronales podían resolver funciones lineales. En 1969, Marvin Minsky y Seymour Papert demostraron que las neuronas perceptrón. no eran capaces de simular una puerta lógica XOR. Esto puso punto y final a los más de diez años de desarrollo de las redes neuronales, ya que se considero que era una vía muerta [27]. En la década de los 80 aparecen reglas de aprendizaje más potentes, se ven por primera vez el perceptrón multicapa (MLP[4]) invalidando la sentencia de Minsky y Papert [24].

MLP: Un perceptrón multicapa es una red neuronal de perceptrones (también puede estar formada por sigmoideas), pero además de la capa de entrada y la capa de salida puede tener una o más capas intermedias. A estas capas se las llaman ocultas.

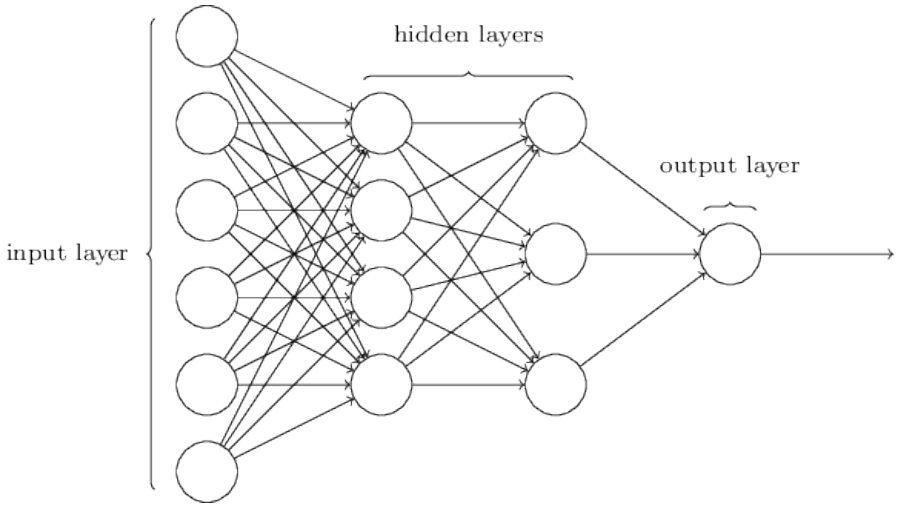


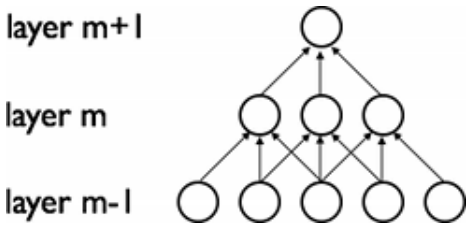
Figura 5: Representación de una MLP con 4 capas: una de entrada, dos ocultas y una de salida.[37]

En las MLP estándar, cada neurona de la capa de input tiene un enlace a la siguiente capa. Ninguna neurona esta enlazada con la capa anterior ni de su

[4]MultiLayer Perceptron

misma capa (este tipo de redes se conocen como Redes Neuronales Recurrentes[40]). En una MLP se diferencian varios tipos de capa según el numero de conexiones o la topología de éstas.

CNN: Es una MLP compuesta de una o más capas convolucionales seguidas de una o más capas totalmente conectadas como en una red neuronal multicapa estándar.



Cada capa neurona de la capa $m-1$ de la CNN está conectada con el conjunto de neuronas adyacentes de la capa m, es decir, las neuronas más cercanas. Las CNN están diseñadas para sacar partido de las estructuras en 2 dimensiones como las imágenes. Por ejemplo la primera capa siendo todos los píxeles. Cada neurona de esta primera capa tendría conexiones con la segunda de forma que una neurona de la segunda capa solo tendría conexiones de entrada con neuronas cuyos píxeles son adyacentes en la imagen. Además estas redes son más fáciles de entrenar y tienen muchos menos parámetros que las redes totalmente conectadas[8], al haber menos enlaces entre capas. Por ejemplo, suponemos 728 neuronas en la capa x y 728 en la $x+1$. En una capa totalmente conectada tendríamos un total de 529984 (728^2) enlaces, con su respectivo peso. Mientras que en una capa convolucional se reduce hasta 5492 ($104(laterales) \cdot 5 + 4(esquinas) \cdot 3 + 620(interior) \cdot 8$) sí suponemos que tiene conectividad 8[5].

[5]un píxel se conecta con los adyacentes en los lados y en diagonal

1.4.2 Spark, centrado en los datos

Spark es la capa subyacente al framework de Deep Learning. Por tanto, es necesario también saber de que manera funciona y como usarlo.

Spark: Framework de procesamiento de Big Data desarrollado en el AMPLab de UC Berkeley y convertido a Open Source en 2010 como proyecto Apache. Proporciona un framework unificado para procesamiento de datos de distinto tipo y de distintas fuentes.

Los desarrolladores aseguran que Spark es más 100 veces más rápido con procesamiento en memoria y 10 veces más rápido en el procesamiento de datos desde el disco. Eso sí, en un cluster de Hadoop que no es nuestro caso. [12][9]

1.4.3 Arquitectura de un Supercomputador

Entender la arquitectura de la máquina que tenemos en la última capa de todo el ecosistema es muy importante si se quiere tener una aplicación que realmente sea eficiente en esa máquina y aproveche todo lo mejor que nos proporciona.

MareNostrum Es el supercomputador más potente de España y el número 77 del mundo [10]. Cuenta con 3056 nodos con procesadores de 16 cores (48896 cores), más de 115 TB de memoria principal, red Infiniband y Gigabit Ethernet. 6.5 PB de espacio de disco sobre GPFS (General Parallel File System) [13]. Se diferencian 3 tipos de nodos en términos de memoria principal.

- High-mem: Nodos con 128 GB de memoria principal instalada. En MareNostrum hay un total de 128 nodos con esta cantidad de memoria.
- Mid-mem: Nodos con 64GB de memoria principal instalada. En MareNostrum hay un total de 128 nodos con esta cantidad de memoria.

- Low-mem: Nodos con 32 GB de memoria principal instalad. Son el resto de nodos disponibles en MareNostrum (unos 2752).

Teniendo en cuenta estos datos se pueden ejecutar pruebas con distintas configuraciones para ver cual se ajusta mejor a lo que buscamos y de que manera influye que no sea un cluster diseñado exclusivamente para Big Data.

1.5 Estado del arte

El estado del arte se refiere a las actuales y más avanzadas formas de solucionar un problema en un momento concreto. Aquí analizamos el estado del arte de los casos de estudio que hemos definido anteriormente.

1.5.1 Deep Learning

DistBelief [16] framework de Deep Learning desarrollado por Google con intención de poder escalar las redes neuronales en sistemas distribuidos, probado hasta los 10000 cores. Es el software responsable de las mejoras en el reconocimiento de voz en las aplicaciones de Google, reconocer gatos en Youtube, etc[17].

Caffe [25] framework de Deep Learning desarrollado por el Berkeley Vision and Learning Center (BVLC) y con contribuciones de la comunidad, publicado bajo licencia BSD-2. Esta programado en C++, pero también tiene interfaces para Python. Este framework funciona tanto en CPU como en GPU.

Thenao [34] Es una librería de Machine Learning (ML) para Python publicada bajo licencia Apache. Otras librerías de ML funcionan encima de ésta. Funciona tanto en GPU como en CPU.

Tensor Flow [32] Es una librería para Python y C++ desarrollada por Goo-

gle. TensorFlow es la continuación de DistBelief, que no era accesible para todo el mundo [17]. Se liberó bajo licencia Apache 2.0 en Noviembre del 2015. También funciona tanto en GPU como en CPU.

Excepto DistBelief, todos los frameworks nombrados se basan en GPU. Aunque tienen la posibilidad de correr en CPU, no hay una implementación distribuida, con lo que solo pueden ejecutarse en distintos cores bajo una misma máquina.

1.5.2 Arquitectura de un Supercomputador

CPU/GPU Con los años, cada vez son más los Supercomputadores en el top500 que incluyen una GPU. En 2012 eran 52 [18], 5 veces más que en 2010, solo hablando de GPU/Aceleradoras de Nvidia. En 2015 ya son 88 supercomputadores. Este tipo de arquitecturas favorecen a los algoritmos de Deep Learning,que son más eficientes en una GPU/Aceleradora que en CPU.

Baidu Minwa Es el supercomputador pensado para trabajar con visualización por computador, usando técnicas de Deep Learning, que este año a rebajado el porcentaje de fallos récord en reconocimiento de imágenes a 4.58 % frente a los 4.82 % que marco Google en 2014. Este computador se compone de 72 procesadores y 144 GPUs. [19][20]

2 Gestión del proyecto

2.1 Metodología

El software a probar es DeepLearning4j [14]. Es un framework de Deep Learning open source y escrito en Java, que está en continuo desarrollo con, aproximadamente, una RC[6] cada 2 meses y cambios grandes entre versiones. Este software funciona encima de Spark y se aprovecha del auto-paralelismo para aumentar su velocidad de cálculo (eso es lo que queremos comprobar).

En el proyecto se trabajará en Java dado que es el lenguaje de DeepLearning4J. Se usarán también lenguajes de scripting como Bash para gestionar algunos procesos intermedios, como podría ser la creación de job scripts o instalación de software. Se usa el software IntelliJ IDEA Ultimate Edition de JetBrains [33] para el desarrollo del código en el ordenador personal. Se usa GIT[5] para el control de versiones, con dos repositorios distintos, uno para el código y otro para la documentación del proyecto. Ambos repositorios están alojados en bitbucket.org.

La metodología que se ha escogido para seguir el proyecto es **Kanban** con la herramienta online Trello[3][4]. Esta metodología fue creada por Toyota en 1962 y consiste en crear y clasificar tarjetas donde estaba escrita la tarea a realizar. Se clasifican según su estado y su prioridad ya que se pueden ordenar las tareas, ponerles fecha de vencimiento y asignarlas a alguien en concreto. Esta metodología es simple, fácil de entender y usar, provee información de manera rápida y permite saber en que punto del proyecto nos encontramos de manera sencilla y que tareas ha hecho quien.

[6] Release Candidate

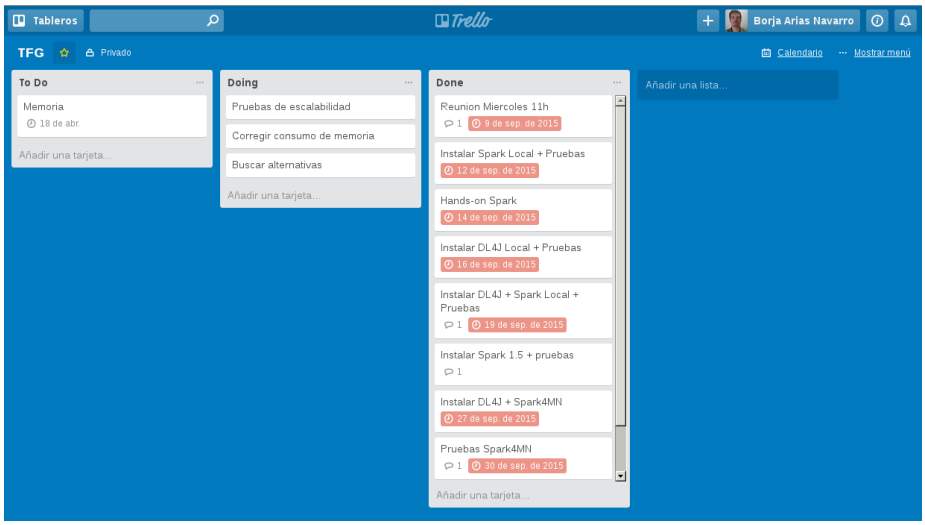


Figura 6: Impresión de pantalla del software online Trello

Aplicando esta metodología, se han creado 3 listas de tareas en Trello: *por hacer*, *haciendo* y *hecho*. Se han colgado de la lista de *por hacer* todas las tarjetas con las tareas iniciales y se le ha asignado una fecha de vencimiento, si es necesario. A medida que se iban escogiendo tareas de la lista de *por hacer*, se mueven a *haciendo* y una vez completadas a *hecho* y cuando aparecen tareas nuevas se ponen de la lista pertinente. La idea es no empezar una tarea hasta acabar la anterior, pero, como veremos en la planificación, hay tareas que se pueden o se deben hacer en paralelo. Además, Trello, nos ofrece un panel de histórico, donde podemos ver cuando una tarea ha sido movida y a donde.

Describiremos las tareas más en profundidad en el apartado de planificación. Pero para tener una idea general, de resumen aquí. En el proyecto tenemos dos ámbitos de trabajo. Uno, es en el que estaremos más tiempo trabajando explícitamente, es el equipo personal; donde se instalará el software primeramente para comprobar que todo funciona. El otro es MareNostrum, donde estaremos menos horas trabajando, pero gastaremos más tiempo de cálculo. En MareNostrum ejecutaremos la pruebas para comprobar la potencia del fra-

mework.

2.1.1 Caso de prueba

Para probar el software, se usa un problema sobre reconocimiento de números escritos a mano, MNIST [35]. Este es un problema básico en las redes neuronales convolucionales. Tenemos imágenes de números en blanco y negro y queremos saber qué dígito representa esa imagen. Para ello se usa una red neuronal, que se define de la siguiente manera:

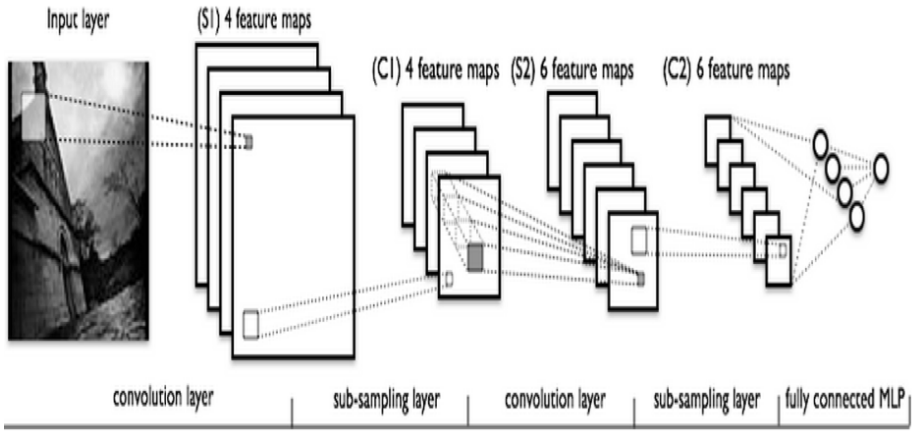


Figura 7: Convolutional Network Layer decomposition[15]

La capa convolucional ya se explica en el apartado1.4.1. La capa de subsampling es una capa optativa que se puede usar para reducir el tamaño de entrada en la siguiente capa convolucional. Para ello, se escoge un tamaño XxY para el filtro, un stride[7] Sx, Sy y un tipo de pooling:

- *Max pool*: escoge el máximo de los valores dentro del filtro y descarta el resto.

[7]stride: salto hasta el siguiente elemento de la estructura

- *Min pool*: escoge el mínimo de los valores dentro del filtro y descarta el resto.
- *Avg pool*: calcula la media de los valores dentro del ciclo.

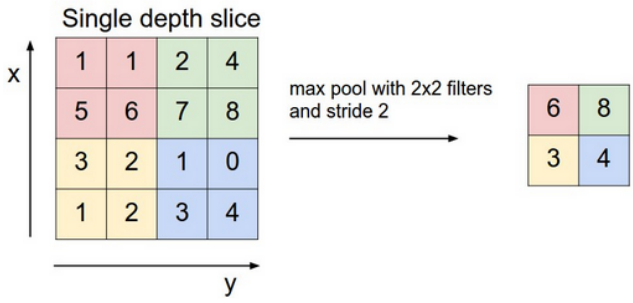


Figura 8: Ejemplo del resultado de aplicar una capa de submuestreo. Entrada y salida, izquierda y derecha respectivamente[15]

El dataset de entrenamiento consta de 60000 imágenes en blanco y negro, y 60000 etiquetas correspondientes al número que representa cada imagen.

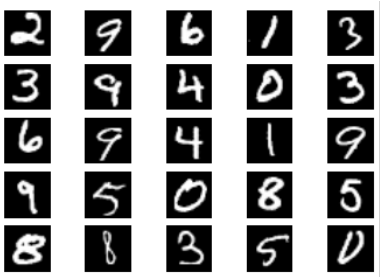


Figura 9: Muestra aleatoria de la base de datos MNIST[39]

2.1.2 Validación de resultados

Las ejecuciones que se correrán en MareNostrum habrán sido probadas con anterioridad a una menor escala. Para ver si los resultados son validos tendre-

mos que comprobar que las salidas sean similares y que no se han informado de errores. Cabe destacar que el framework de deep learning se encuentra en estado experimental, así que, es probable que programas ya verificados fallen.

Por otro lado para comprobar si una clasificación de imágenes es correcta o no, deberemos observar la matriz de confusión que se genera de la clasificación y, con el umbral definido, validaremos o no el resultado. En una matriz de confusión se contabilizan los falsos positivos (fp), los falsos negativos (fn), los positivos (tp) y los negativos (tn) sobre el total de muestras (n).

- *accuracy* (exactitud) $a = \frac{tp+tn}{n}$
- *precision* (precisión) $p = \frac{tp}{tp+fp}$
- *recall* (exhaustividad) $r = \frac{tp}{tp+fn}$
- *F1 score* $F_1 = 2 \cdot \frac{p \cdot r}{p+r}$

Etiqueta	Veces predicho									
	0	1	2	3	4	5	6	7	8	9
0	**1143**	1	0	2	1	7	5	4	13	3
1	0	**1293**	3	8	0	0	0	2	8	1
2	3	11	**1121**	18	9	2	3	25	29	8
3	2	5	16	**1137**	0	12	0	9	23	11
4	1	6	3	0	**1144**	1	10	1	5	38
5	2	4	0	10	0	**1025**	1	1	26	7
6	4	3	0	0	4	19	**1114**	0	21	0
7	0	4	10	7	4	0	0	**1186**	4	22
8	0	6	2	8	3	4	3	3	**1111**	20
9	3	8	1	10	5	4	2	11	10	**1131**

Cuadro 1: Matriz de confusión de una ejecución con 48000 imágenes de entrenamiento y 12000 de prueba.

La matriz de confusión del Cuadro 1 es la salida proporcionada por la ejecución de una prueba con el conjunto de imágenes al completo. En este caso tenemos un F_1Score de 0.9531 con 11405 positivos. En cuanto a los falsos positivos y falsos negativos, los tenemos que analizar etiqueta a etiqueta, como si la pregunta fuese binaria. Por ejemplo, en el caso del 0, consideraremos que las opciones son ó 0 ó cualquier otro número.

Etiqueta	Veces predicho									
	0	1	2	3	4	5	6	7	8	9
0	1143	1	0	2	1	7	5	4	13	3
1	0	1293	3	8	0	0	0	2	8	1
2	3	11	1121	18	9	2	3	25	29	8
3	2	5	16	1137	0	12	0	9	23	11
4	1	6	3	0	1144	1	10	1	5	38
5	2	4	0	10	0	1025	1	1	26	7
6	4	3	0	0	4	19	1114	0	21	0
7	0	4	10	7	4	0	0	1186	4	22
8	0	6	2	8	3	4	3	3	1111	20
9	3	8	1	10	5	4	2	11	10	1131

positivo
falsos negativos
falsos positivos
negativos

Cuadro 2: Matriz de confusión identificando falsos negativos, falsos positivos, positivos y negativos

Sumando cada uno de los tipos de resultado, podemos obtener, para cada etiqueta su *accuracy*, *precision* y *recall*. Aunque la *exactitud* es la misma para todas, ya que se calcula sobre el total de imágenes. Una vez tenemos la *precisión* y la *exhaustividad* calculada para cada elemento, solo tenemos que calcu-

lar la media, para tener el valor de todo el test.

Etiqueta	tp	tn	fp	fn	a	p	r
0	1143	10262	15	36	0.9528	0.9870	0.9695
1	1293	10112	48	22	0.9528	0.9642	0.9833
2	1121	10284	35	108	0.9528	0.9697	0.9121
3	1137	10268	63	78	0.9528	0.9475	0.9358
4	1144	10261	26	65	0.9528	0.9778	0.9462
5	1025	10380	49	51	0.9528	0.9544	0.9526
6	1114	10291	24	51	0.9528	0.9789	0.9562
7	1186	10219	56	51	0.9528	0.9549	0.9588
8	1111	10294	139	49	0.9528	0.8888	0.9578
9	1131	10274	110	54	0.9528	0.9114	0.9544
media					0.9528	0.9535	0.9527
$F_1 Score$			0.9531				

Cuadro 3: Calculo de los valores de a, p, r para cada una de la etiquetas

Estos valores, los podemos obtener directamente del framework, con lo que nos sirven para valorar los resultados, pero no tendremos que hacer estos cálculos manualmente. De todas maneras se ha escrito un script en Python (ver apendice A) que recorre el fichero de salida de la ejecución para calcular estos datos y los imprime en un formato filas de tabla de LaTeX para poder mostrarlos en este u otro documento.

2.2 Planificación

En esta sección se explicará la planificación inicial de proyecto. Para cada tarea se ha asignado un tiempo estimado. En esta previsión se incluía un margen para imprevistos aunque, como suele pasar, finalmente se alargó más de lo esperado.

2.2.1 Fases del proyecto

Hemos descompuesto el proyecto en 4 fases grandes, dentro de las cuales se encuentran tareas generales. Estas tareas se componen de otras más pequeñas y especificas.

Cuadro 4: Tareas

#	Tarea	Tiempo (días)
1	Instalar Spark Local	3
2	Instalar DL4J Local	2
3	Instalar DL4J + SparkML Local	1
4	Hands-on Spark4MN	3
5	Instalar DL4J + Spark4MN	1
6	Tests escalabilidad	3
7	Validación de Tests	3
8	Segunda fase	15
	Total	21

Cuadro 5: Hitos

Tarea	# Dependencia
Reunión definición de objetivos	3
Contactar Jordi Torres	5
Reunión definición de segunda fase	7

Dependencias de tareas

Algunas de las tareas se pueden hacer independiente de las demás, esto no quiere decir que se puedan hacer en paralelo (dado que solamente hay una persona dedicada a ello), si no que no depende del éxito o finalización de otra.

Por contra, hay otras tareas que dependen estrictamente de que una anterior acabe. [8]

Estas son:

[8] ver diagrama de Gantt en el apéndice D

Cuadro 6: Dependencias de tareas

# Tarea	# Dependencia (Tipo)
3	1, 2 (Fin-Inicio)
5	3 (Fin-Inicio)
6	5 (Fin-Inicio)
7	6 (Inicio-Inicio)
8	7 (Fin-Inicio)

Local

En la fase local, se trata de probar el sistema que queremos desplegar en MareNostrum en nuestro equipo portátil, para asegurar que tenemos todas las dependencias controladas y una instalación básica funciona. Esta fase implica a 3 subfases generales. Esta parte estaba limitada por la memoria del sistema 8GB, más adelante veremos hasta que punto nos limita el numero de imágenes a tratar.

Instalar Spark Local Se trata de descargar e instalar la última versión de Spark en nuestro portátil. Además, esta tarea se compone de 3 subtareas que pueden ser cíclicas y dar paso a retrasos.

- Instalación: descargar he instalar dependencias.
- Ejecución tests: ejecutar el paquete de pruebas del desarrollador del software. Si da errores volver al paso 1.
- Tutorial: aprender los aspectos básicos para que hacerlo funcionar.

Instalar DL4J Local Como con Spark, se trata de descargar e instalar la última versión de DL4J e instalarla en nuestro portátil. Esta tarea se descompone en 2 subtareas.

- Instalación: descargar he instalar dependencias.

- Ejecución tests: ejecutar el paquete de pruebas del desarrollador para el modo standalone.

Instalar DL4J + SparkML Se trata de comprobar que el framework DL4J funciona correctamente con las librerías ML (Machine Learning) de Spark. Este paso puede llevar a empezar de cero si hubiese algún error de incompatibilidad.

Remota

En la fase Remota, se trata de instalar el sistema en el supercomputador MareNostrum. Esta fase se puede descomponer en otras 2 más pequeñas.

- Instalación: como que en MareNostrum no hay conexión al exterior, necesitaremos descargar todo en el portátil, para subirlo al supercomputador y, una vez allí, compilarlo.
- Ejecución de tests: ejecutar el paquete de pruebas del desarrollador tanto para la versión de Spark que se instalará, como para el framework de Deep Learning.

Escalabilidad

En esta fase se pretende demostrar la eficiencia de la instalación que se ha hecho. Para ello procederemos a probar diferentes geometrías[9] para Spark. Esto se hará con la clasificación de dígitos MNIST que se ha explicado en la sección 2.1.1. Además, estas pruebas también nos permitirán probar con distintos de los algoritmos que nos proporciona el Framework y ver como se ajustan estos a las diferentes configuraciones (geometrías)

[9] manera en la que se distribuyen los recursos

Fase extra

Por último, nos reservamos una fase extra, que aún esté por determinar. Esto es así porque depende de como avancen las tareas anteriores y de alguna colaboración de algún proyecto de Máster sobre Deep Learning.

2.2.2 Desviaciones

Uno de los primeros errores de la planificación fue suponer que la asignatura de Gestión del Proyecto (GEP) no nos quitaría tiempo y, por lo tanto, no se contó en el desvío de las tareas del proyecto. A parte de esto, han habido más problemas de lo esperado en cuanto al desarrollo de las ejecuciones de pruebas. Estos problemas se han presentado durante todas las fases del proyecto (desde la fase local a las de los tests de Escalabilidad, pasando por las pruebas remotas). El peor de todos ellos fue darme cuenta que los datos que estaba obteniendo de las ejecuciones en MareNostrum eran falsos, ya que el programa aborta debido a un consumo de memoria excesivo pero el sistema reportaba que la ejecución había finalizado correctamente.

Con todo ello, a mitad del proyecto se hizo otra planificación, teniendo en cuenta el desvío provocado por GEP y los errores.

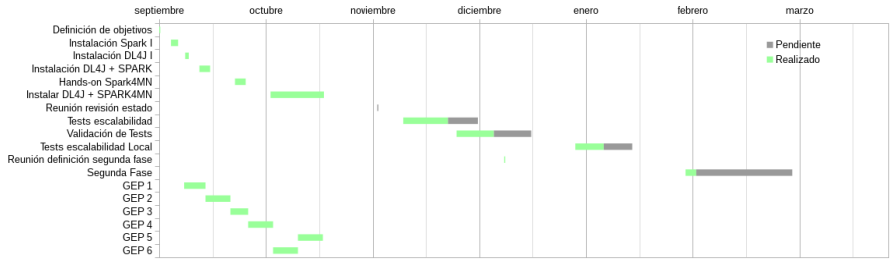


Figura 10: Diagrama de Gantt revisado. Ver Apendice D.2

2.2.3 Problemas

Durante los test de escalabilidad en MareNostrum han surgido varios problemas. Aquí explicaremos cuáles han sido y como se han resuelto, si se ha podido.

- Consumo de Memoria
  Las ejecuciones con más de 2000 imágenes de entrada, en MareNostrum, consumían toda la memoria del nodo y hacían que la ejecución no muriese. La solución se explicará más adelante en la sección 3.3.

- NULL Pointers
  Con las nuevas versiones de DL4J también aparecen nuevos bugs. En este caso la librería de álgebra lineal que usa DL4J (ND4J) presenta algún puntero nulo en sus operaciones internas. La solución se explicará más adelante en la sección 4.1.

2.2.4 Recursos

En esta sección se pretende dar una previsión de los recursos necesario, pero no en términos económicos, sino solamente de los requerimientos del proyecto.

Personal

Todo proyecto necesita de unos recursos humanos para llevarse a cabo. En este caso tenemos 4 personas que están implicadas, en mayor o menor grado.

Borja Arias (yo mismo) Como desarrollador principal del proyecto.

Jordi Torres Como ponente del proyecto y profesional con experiencia en Big Data.

Rubén Tous Como colaborador y profesional con experiencia en Deep Learning y, concretamente en DL4J.

Carlos Tripiana Como desarrollador de Spark4MN, con lo que aporta su experiencia en este software en MareNostrum y da soporte.

Jorge Rodriguez y David Vicente Ambos como directores del proyecto. Sus aportaciones han sido revisiones, tanto de la documentación como del estado del proyecto. Así como la revisión de los cambios de rumbo posibles.

2.3 Hardware

Además de los recursos humanos, necesitaremos equipo donde realizar las pruebas, los análisis y las instalaciones.

Equipo Portátil Donde se realizarán las instalaciones locales y desde donde se subirán los archivos a MareNostrum para poder instalarlos después. Finalmente se pasó a una workstation con más memoria para poder ejecutar los

ejemplos localmente.

MareNostrum El supercomputador donde se instalará el sistema y donde se realizarán las pruebas.

2.4 Software

Linux Mint 17.1 sistema operativo del equipo portátil que se usa.

LaTeX lenguaje para crear los documentos y presentaciones.

TeXstudio IDE [10] para documentos LaTeX.

IntelliJ Suite, Community Edition IDEA, PyCharm, ambos IDEs de IntelliJ, cada uno diseñado para un lenguaje de programación concreto: Java y Python respectivamente.

BitBucket & Git sistema de versiones que se usará desde el equipo portátil para mantener los documentos.

Trello aplicación web para llevar el seguimiento de las tareas.

LibreOffice Calc aplicación para elaborar las hojas de cálculo.

[10] Integrated Development Environment: Ambiento de desarrollo integrado. Software que integra un conjunto de herramientas para hacer que el desarrollo de algo concreto sea más sencillo

2.5 Presupuesto

Esta sección se compone de la estimación de costes del proyecto entre las cuales se contemplan Costes Directos, Costes Indirectos e Imprevistos.

2.5.1 Costes directos

Estos costes están directamente relacionados con el proyecto. Concretamente provienen de las tareas descritas en el diagrama de Gantt del Entregable 2. En esta categoría incluiremos los costes de personal, así como el equipo necesario para el desarrollo de la actividad y otros materiales.

Recursos Humanos

- **Project Manager**: Son responsables de establecer el plan del proyecto y coordinar los recursos para completarlo en el tiempo y el presupuesto acordado. Con el rol de Project Manager (Rol 1).
- **Senior HPC Assistant**: Soporte HPC y plataforma de Spark4MN.
- **Junior HPC Assistant**: encargado de la instalación del nuevo sistema y pruebas de eficiencia del sistema.

Teniendo en cuenta los roles que hemos definido y las tareas e hitos del diagrama de Gantt. Los recursos humanos se reparten de la siguiente manera las tareas y la participación en los hitos/reuniones.

Cuadro 7: Repartición de tareas

#	Tarea	Rol 1 (%)	Rol 2 (%)	Rol 3 (%)
1	Instalar Spark Local	10	0	90
2	Instalar DL4J Local	0	0	100
3	Instalar DL4J + SparkML Local	0	0	100
4	Hands-on Spark4MN	0	10	90
5	Instalar DL4J + Spark4MN	0	40	60
6	Tests escalabilidad	30	0	70
7	Validación de Tests	20	0	80
8	Segunda fase	0	0	100
H1	Reunión definición de objetivos	100	0	100
H2	Contactar Jordi Torres	100	0	100
H3	Reunión definición de objetivos	100	0	100

Contamos que las reuniones que se derivan de los hitos toman 1.5 horas cada. Se computan así en los costes directos derivados de los recursos humanos.

Cuadro 8: Presupuesto de recursos humanos

Rol	Precio por horas	Horas	Unidades	Coste
Project Manager	25	18.9	2	945
Senior HPC Assistant	12	4.8	1	57.6
Junior HPC Assistant	9	153.3	1	1379.7
Total		177		2382.3

Recursos Materiales

En esta sección del presupuesto contaremos todo el material que nos va a permitir llevar a cabo el proyecto. Esto incluye, tanto equipos informáticos como mobiliario. Para cada concepto contemplaremos los años de amortización.

Cuadro 9: Presupuesto de recursos materiales

Concepto	€/Unid.	Uso(días)	Unid.	VU(años)	Coste €
Ordenador Portátil	1900.00	21	1	5	30.57
MareNostrum	4.08	1	341	-	1391.28
Pantalla Externa	130.00	2	1	4	2.74
Escritorio	300.00	22	1	8	3.16
Silla	150.00	22	1	4	3.16
Teclado + Ratón	100.00	22	1	4	2.11
Total					1430.92

El caso del supercomputador MareNostrum hemos aproximado lo que costaría durante una serie de tests partiendo del coste de la máquina en 2013 con una amortización de 4 años, es decir hasta 2017. Lo contaremos por nodos, ya que la máquina se usa a la vez por cientos de usuarios, pero en general los nodos se usan en exclusiva.

Costes de software

En este caso no contamos con ningún caso de software, ya que todo el que se usará durante el desarrollo del proyecto, o bien será software libre o gratuito.

Este software es el siguiente: Linux Mint 17.1, LaTeX, Geany, TeXstudio, BitBucket,Git, Trello, LibreOffice Calc.

2.5.2 Costes Indirectos

Otros recursos que no están directamente relacionados con el proyecto también están lo están con su desarrollo, por lo que también tienen que tenerse en cuenta.

Como que todo el proyecto se realiza de forma digital el único coste indirecto que tenemos es la electricidad, para el ordenador portátil, para MareNostrum y el habitáculo donde se trabaja.

Cuadro 10: Costes indirectos derivados del consumo eléctrico

Sistema	€/hora.	Uso	Coste €
Ordenador Portátil	0,124107	21 días	0.47
MareNostrum	0.037333	81940(horas)	3059.00
Total			3059.47

El cálculo del consumo eléctrico de MareNostrum es por core/hora. El consumo anual de electricidad es de 1M€de media. Teniendo en cuenta que tenemos un total de 48896 cores, obtenemos un consumo medio de 0.037333 $e \cdot core/hora$.. Para el Ordenador Portátil se considera un consumo media de 20W.

2.5.3 Imprevistos

En este proyecto se contempla el caso de que se necesiten más pruebas por falta de claridad, o que se tengan que repetir algunas de las pruebas puntuales. Tenemos una probabilidad del 30 % de que se tengan que repetir los test. Otro Riesgo, con menor probabilidad, es el que nos demos cuenta de que la solución escogida no es suficientemente buena y tengamos que escoger otra.

Cuadro 11: Costes de Imprevistos

Imprevisto	Precio	Probabilidad	Coste €
Repetir test	3914.21	30 %	1174.263
Escoger otra solución		5 %	343.63

2.5.4 Resumen

El control de costes del proyecto se realizará mediante las herramientas de las que dispone MareNostrum para indicar el consumo de horas que las distintas ejecuciones tienen. En concreto se dispone de un Elasticasearch con todo el histórico de las ejecuciones y los datos relacionados con ellas. Dado que es el coste más importante es el relativo a MareNostrum el que más se debe controlar. En el peor de los casos tendremos que volver a empezar, planificando otra solución, pero esto se podría ver antes de finalizar el 100 % del proyecto.

Cuadro 12: Resumen de Presupuesto

Concepto	Coste €
Costes directos	3813.22
Costes Indirectos	3059.47
Imprevistos	1517.89
Contingencia(10 %)	839.05
Total	9229.63

2.5.5 Consumo real

Este presupuesto se hizo inicialmente suponiendo que la duración total del proyecto fuese 4 meses. Como finalmente se extendió otros 4 más, el presupuesto se ha superado. En cuanto a los recursos humanos, el rol de **Junior HPC Assistant** es el que más tiempo ha tenido que dedicar al desvío. Por parte de los costes indirectos, han habido, finalmente, más ejecuciones de las esperadas en MareNostrum aunque más pequeñas. Por último, hubo que hacer una ampliación de la Memoria del equipo local (dos dimms de 4GB) para poder hacer pruebas localmente.

Para calcular el total de horas consumidas en MareNostrum se ha creado un script en Python (ver apendice B) que utiliza una API Web de Elastic-

search[28] del BSC. Con esta API se extraen los datos de las ejecuciones de MareNostrum para ver los datos individuales de cada una.

Con los datos del Elasticsearch calculamos que se han consumido un total de 20Kh de CPU en MareNostrum aproximadamente (un total de 392 ejecuciones finalizadas y 259 canceladas en algún punto). Con la extensión del proyecto se calcula que el Junior HPC Assistant pasa de las 153.3 horas previstas inicialmente a un total de 451.6.

Cuadro 13: Margen de presupuesto inicial

Concepto	Precio por unidad	unidades	Coste €	Variación original
Junior HPC Assistant	9	451.6(horas)	4064.4	+2684.7
MareNostrum	0.037333	19429.72(horas)	725.37	-2333.63
DIMM 4GB	19.99	2	39.98	+39.98
Total				+391.05

Como teníamos presupuestado 1517.89€ de imprevistos pese al desvío no nos salimos del presupuesto inicial.

2.6 Sostenibilidad

Aspecto económico

Si bien para este proyecto se han analizado los costes antes de empezar, tanto de recursos humanos como de materiales, la solución a implementar no depende únicamente de si misma, además de que el software que se quiere instalar está en versión experimental y que realmente no sabremos, hasta acabar, si esta solución se ajusta a lo que queremos (en términos de eficiencia/coste de despliegue). Vemos bastante improbable que en aplicaciones reales esta solución no cumpla la expectativa y, pese al coste nos permitirá reducir el tiempo de test de otros proyectos futuros. La mayoría del coste de este proyecto está asociado a MareNostrum, cuyo coste no deberíamos pagar directamente ya que estamos desarrollándolo desde el departamento de Operaciones del BSC y el coste se amortiza con la modernización de parte del sistema.

Aspecto social

Actualmente en España, debido a la crisis, cada año disminuye el presupuesto dedicado a I+D[29][30]. Este hecho hace que las empresas públicas (como el BSC) que se dedican a ello tengan que ajustarse un poco más. Por suerte, el presupuesto del BSC no depende solamente del gobierno Español o Catalán, sino que también recibe subvenciones por parte de Europa con el proyecto PRACE[38], además de ser considerado centro de excelencia Severo Ochoa[26]. Con este proyecto se pretende modernizar servicios que se ofrecen con el supercomputador MareNostrum para seguir siendo competitivos y poder mantenerse en el proyecto PRACE y como centro Severo Ochoa.

Por parte de los usuarios a los que va dirigido el software final, al ofrecer un algo concreto, deberán adaptarse a él y aprender como funciona. Este proceso no tiene que ser muy tedioso. Se ha escogido este framework precisamente por

su facilidad de uso.

Aspecto Medioambiental

En este aspecto cabe comentar que el principal beneficio medioambiental de este proyecto es poder reducir el tiempo de ejecución de algunos programas sin tener que cambiar/ampliar el hardware. Con esto conseguimos dos puntos a favor, ya que no se generará contaminación derivada de la producción de nuevos componentes ni del desecho de viejos y, además, al reducir el tiempo de ejecución de un programa hacemos que, en definitiva, gaste menos energía. Esto último es cierto, siempre y cuando al mejorar la eficiencia en tiempo no se esté usado más recursos. Por ejemplo, que tarde la mitad pero trabajando con el doble de hardware.

Cuadro 14: Matriz de sostenibilidad

Etapa	Económico	Social	Medioambiental
Planificación	6	7	9
Resultado	6	7	7
Riesgos	0	0	0
Total	42		

3 Implementación de la solución

En esta sección se explica el proceso que se ha seguido a cada paso del proyecto y los resultados de estos.

3.1 Instalación local

La instalación local es bastante sencilla. Se explica, para cada paquete a instalar, el procedimiento seguido y como se integran entre ellos, si es necesario.

Spark En las pruebas en local se ha utilizado la versión 1.5.0 (que era la última disponible el 9 de septiembre de 2015), descargando los binarios de la página oficial[9]. Se usa un script para cargar el entorno necesario en el sistema y ya podemos arrancar.

./scripts/load–spark.sh

```
1 export PATH=~/bin/spark-1.5.0/bin:~/bin/spark-1.5.0/sbin:$PATH
  export SPARKHOME=~/bin/spark-1.5.0/bin
```

La configuración local de Spark arranca 4 workers o ejecutores, que serán los que reciban los datos y las instrucciones de lo que deben hacer con ellos. Estos workers tienen asignada 1 CPU cada uno y 3GB de memoria RAM, guardando el resto para el sistema y el Master de Spark.

ND4J: Es una dependencia de DL4J. Podría instalarse directamente con la instalación de DL4J, pero al descargar los archivos fuente podemos mirar en el código de una forma más cómoda en el caso de que se necesite, cambiar entre versiones y ramas del repositorio y actualizarlo fácilmente. En este caso, basta con clonar el repositorio desde `https://github.com/deeplearning4j/nd4j` y compilar e instalar usando Maven[11]. Con esto ya descarga todas las dependencias del software e instala los jars en el repositorio de maven. En

nuestro caso hemos usado la versión 0.4rc-3.8, con lo que tendremos que hacer checkout a del tag *nd4j-0.4-rc3.8* antes de compilar con Maven.

DL4J: La instalación es la misma que para ND4J, pero en este caso el repositorio es `https://github.com/deeplearning4j/deeplearning4j`. También se usa la versión 0.4rc-3.8 (tag *deeplearning4j-parent-0.4-rc3.8*). Es recomendable que las versiones de ND4J y DL4J vayan a la par, ya que la dependencia de DL4J con ND4J es bastante fuerte. Luego se compila con Maven.

ND4J y DL4J se prueban conjuntamente con los ejemplos del repositorio de los desarrolladores `https://github.com/deeplearning4j/dl4j-0.4-examples`. De estos ejemplos partiremos para hacer nuestro ejemplo de caso de estudio.

Estos tres paquetes, al instalarlos con Maven, acaban bajo la carpeta .m2/repositories de nuestra carpeta de Inicio. Antes de cada nueva instalación, hacemos una copia de esta carpeta (como copia de restauración) y luego la borramos para empezar limpiamente.

3.2 Instalación remota

Por contra, la instalación en MareNostrum es bastante más complicada. Desde esta máquina no tenemos acceso al exterior, es decir, que no podemos acceder a los repositorios, ni compilar con Maven y dejar que este nos descargue todas las dependencias automáticamente. La instalación de Spark en MareNostrum ya está solucionada con el despliegue de Spark4MN, el cual gestiona Carlos Tripiana. En cuanto a DL4J, se probaron varias soluciones.

Repositorio centralizado: Dentro de cada versión de Spark en MareNostrum (des de Spark 1.5.2). La idea es indicarle a Maven que el repositorio se encuentra en la carpeta que provee Spark en MareNostrum. El problema de esta solución es que, solo se puede indicar un repositorio local, y el usuario normal no puede escribir en el repositorio de Spark. Otra opción sería con-

figurar un Mirror dentro de Maven, que redirija las conexiones externas al repositorio centralizado. Para esto se necesitaría tener corriendo un servicio que respondiese a estas peticiones por HTTP y por motivos de seguridad se ha decidido no hacerlo. En este repositorio, por ahora, esta incluida la versión 0.4rc-3.7-SNAPSHOT, y se puede usar si se indica en el archivo de configuración de spark4mn.

conffile.job

```
...
SPARK_PACKAGES="DL4J"
...
```

Repositorio personal: Como solo se puede usar un repositorio local, la opción más sencilla es compilar todo en el ordenador propio y copiar el directorio /.m2 a MareNostrum. Una vez hecho esto, puedes mover el proyecto de Java a MareNostrum y compilarlo allí con todas las dependencias en un solo archivo *jar*. Las ventajas de esta opción es que si tus dependencias no cambian mucho, pero sí que lo hace tu código, solamente tienes que copiar una vez la carpeta pesada (en MB) y puedes trabajar en el código directamente en la máquina remota. Por contra, la carpeta .m2 con todas las dependencias de DL4J y el proyecto se puede ir a unos cientos de megas, aunque se puede acelerar usando *rsync* para sincronizar solo los archivos no existentes de las carpetas.

jar-with-dependecies: Como que lo que hacemos funcionar es un Java y es independiente de la máquina en la que se ejecuta también podemos compilar el proyecto en un jar con todas las dependencias incluidas, subirlo a MareNostrum y usarlo directamente. Para ello en el archivo de definición del proyecto (pom.xml) marcamos el *scope* de las dependencias que queramos incluir en el jar como *compile*. Dejamos las que no queramos incluir como *provided*, por ejemplo Spark (que ya está en MareNostrum).

```
1       ...
        <groupId>org.deeplearning4j</groupId>
3           <artifactId>dl4j-spark-ml</artifactId>
            <version>0.4rc-3.8</version>
5           <scope>compile</scope>
        </dependency>
7       <dependency>
        ...
```

La ventaja de este método es que no tienes que subir el cogido fuente a MareNostrum y solo habrá que compilar una vez. Como desventaja, cada vez que se haga una modificación en el código, por pequeña que sea, se tiene que subir el *jar-with-dependencies* completo, que en este caso son 150 MB.

Todos los métodos que se han comentado requieren mucha transferencia entre el ordenador personal y MareNostrum. Lo que disminuye el tiempo de trabajo efectivo. Dentro de la red del BSC esto no es un problema, pero en el mayor de los casos no disponemos de esas tasas de transferencia. Se ha hablado de habilitar de alguna manera que Maven acceda a internet de forma segura. Por ejemplo, Maven proporciona una configuración para usar proxis para conectarse al repositorio central (o otros repositorios). De todas formas esto es algo que se deja para el futuro.

3.3 Análisis de resultados

Como ya se ha comentado, el código que se usa para probar la escalabilidad de DL4J sobre spark4mn es una clasificación de dígitos escritos a mano. Para ello, se uso como ejemplo el código del repositorio https://github.com/deeplearning4j/dl4j-spark-cdh5-examples.

El código original accede a internet para descargar los archivos de imágenes y de etiquetas de MNIST. Como en MareNostrum no tenemos acceso a internet

se modificó para extraer estos datos de los ficheros ya descargados en disco. Una de las cosas que se quiere probar es como afecta el número de imágenes a tratar, así que se crea la función de lectura de las imágenes teniendo en cuenta que se lean solamente el número de imágenes que se pide, cosa que no hace el código original.

Con esta primera versión se intentaron correr 2 tipos de benchmarks. El primero, variando en número de imágenes a tratar manteniendo la cantidad de workers. El segundo, al contrario. Para ambos benchmarks se mide tanto el tiempo de ejecución, como las medidas de calidad de la matriz de confusión.

Problema Las ejecuciones con más de 2000 imágenes de entrada, en MareNostrum, consumían la memoria del nodo y hacían que loa ejecución no siguiese.

Fue el momento que se decidió incrementar la memoria disponible en el equipo local hasta 16GB (la mitad de un nodo normal de MareNostrum). Con esto se podía ejecutar programas de debug en Spark y poder ver un poco mejor que es lo que pasaba.

Una vez aumentada la memoria del equipo local, se pudo ejecutar el ejemplo hasta 20000 imágenes. Al no verse ningún motivo en concreto por lo que se consumía la memoria, se cambió algunos parámetros del código. En concreto, se probó a particionar los datos en la función *paralellize* del Spark Context (ver código definitivo en Anexo C). Esto no resolvió el problema por si solo, aunque para conseguir el paralelismo en Spark tenemos que particionar los datos para enviarlos a cada worker, con lo que el cambio se mantiene.

Otra de las pruebas que se hizo, fue usar Hadoop Distributed File System (HDFS) por debajo de Spark, para ver si el problema venia a la hora de leer el archivo desde disco. Por ejemplo, que el fichero de imágenes fuese duplicado a cada worker y eso pudiese generar más memoria de la cuenta. Se creó otro código que simplemente copiaba el fichero de GPFS a HDFS, desde donde los

workers lo leerían.

Finalmente, se optó por reformar el código (sin HDFS), y hacer que el entrenamiento de la red neuronal fuese incremental. En el nuevo código se tratan las imágenes en grupos de 2000. De cada grupo, se guarda un 80 % para entrenar a la red neuronal y un 20 % para hacer los tests con la red entrenada. Con esto, se consigue que como máximo, en el sistema se tengas estructuras de datos para un total de 1600+12000 imágenes cuando se trata con el conjunto de datos al completo. Aunque se usan más de 2000 imágenes a las vez en la ejecución llega a un consumo máximo de memoria de 90GB. Para este tamaño de memoria necesitamos los nodos High-mem, que pueden usarse desde la versión 3.2.0 de Spark4mn, que se habilitó el 29 de Marzo de 2016.

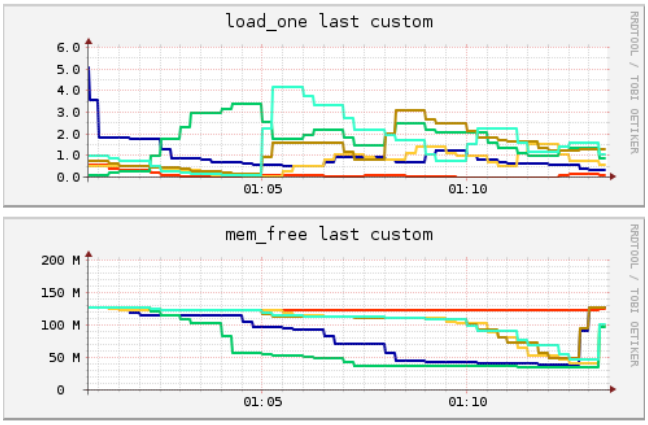


Figura 11: Gráficos de uso de cpu (arriba) y de memoria(abajo) de una ejecución correcta del ejemplo

En estas gráficas[11] vemos el uso de memoria de los nodos de una ejecución del ejemplo. Se puede observar el aumento escalonado del consumo de RAM hasta parar por debajo de los 50GB libres.

[11] las unidades del eje vertical de la gráfica de memoria son GB

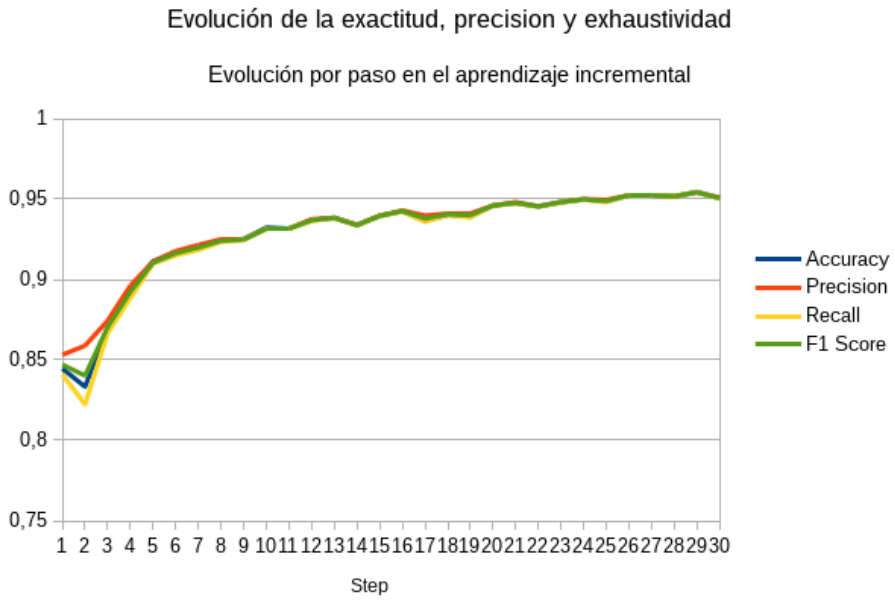


Figura 12: Cálculo de la exactitud, precisión y exhaustividad a cada paso del aprendizaje incremental

La Figura 12 muestra la evolución de los valores que se calculan de la matriz de confusión a medida que se va entrenando la red neuronal de forma incremental. Cada paso representa 2000 imágenes más digeridas por la red neuronal y podemos observar como se va acercando al máximo (en este caso 0.95[12] con una forma logarítmica. El pico descendiente que se aprecia en el paso 2 se debe más a la muestra de imágenes, pues de cada conjunto de 2000 se escogen aleatoriamente las que serán de entrenamiento y cuales para probar.

[12]valor máximo de F1 para una ejecución con 60000 imágenes

3.3.1 Resultados Benchmark

Una vez se tiene un código que funciona se realizan 2 pruebas diferentes.

Variación del número de Workers: Cada workers de spark que se lanza corre solo en un nodo, con lo que tiene disponible para si un total de 16 cores y 124.800 MB de memoria principal. El número de imágenes a procesar es siempre 60.000.

Viendo las gráficas de uso de CPU (ver figura 11), los workers no usan todas las CPUs del nodo, pero lo necesitamos así para reservar toda la memoria.

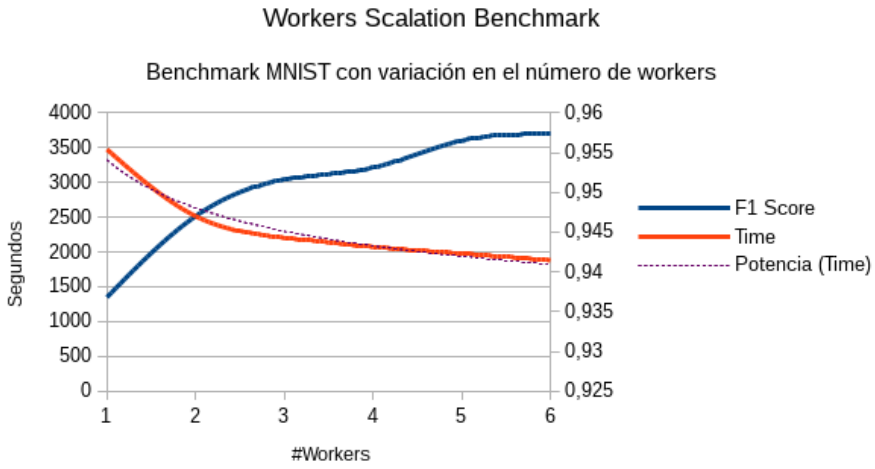


Figura 13: Escalabilidad por worker

Como vemos en la figura 13, es cierto que a mayor número de workers menor es el tiempo de ejecución. Pero a medida que aumentan cada vez es menor la mejora y, podemos ver que tiene tendencia a aplanarse con más workers. En cuanto a la marca F1, la diferencia máxima de los valores es de 2 centésimas, cosa que se explica por la aleatoriedad de la toma de muestras.

Variación en el número de imágenes: Para esta otra prueba se establece fijo el número de workers a 4. Cada worker usa la misma configuración que en caso anterior. Se empieza con 1875 imágenes y se dobla a cada paso.

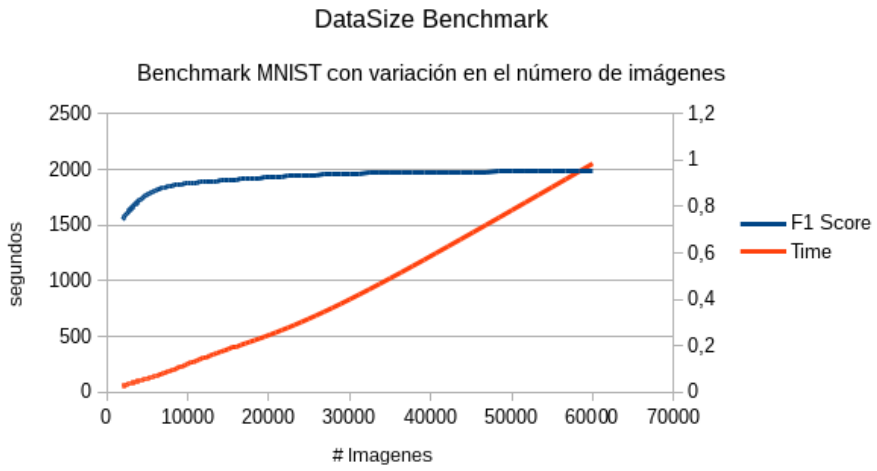


Figura 14: DataSize Benchmark

En la gráfica de la figura 14 podemos ver que con al incrementar el número de imágenes a tratar, el tiempo de procesarlas también aumenta, de forma casi lineal. Esta tendencia (salvo que se disparase a partir de cierto punto) nos indica que podríamos trabajar con cantidades de datos aún mayor sin que el tiempo de procesado se hiciese demasiado grande.

A partir de 4 workers la inestabilidad del programa se hace ver. De 4 a 6 workers, la ejecución se aborta por errores diversos, como fallos en la lectura de objetos serializados o punteros nulos al acceder al actualizador de la red neuronal. Más de 6 workers hacen que la ejecución falle siempre (al menos las veces que se han observado).

Esto parece tener alguna relación con el particionado de los datos, pues se ha ejecutado un código en el que la partición de los datos se hace de forma dis-

tinta. En la pruebas anteriores, se le decía a Spark que particionara los datos en tantas partes como workers hubiesen en el sistema. Se cambia este valor y decide de la forma siguiente

```
int nPartitions = Math.min(nWorkers, 4)
```

Se escoge el número 4 como mínimo ya que se ha observado que es el mayor número de workers que no suele fallar. Con este cambio, se prueban ejecuciones de hasta 64 workers terminando siempre con éxito. Pero los resultados no son tampoco satisfactorios.

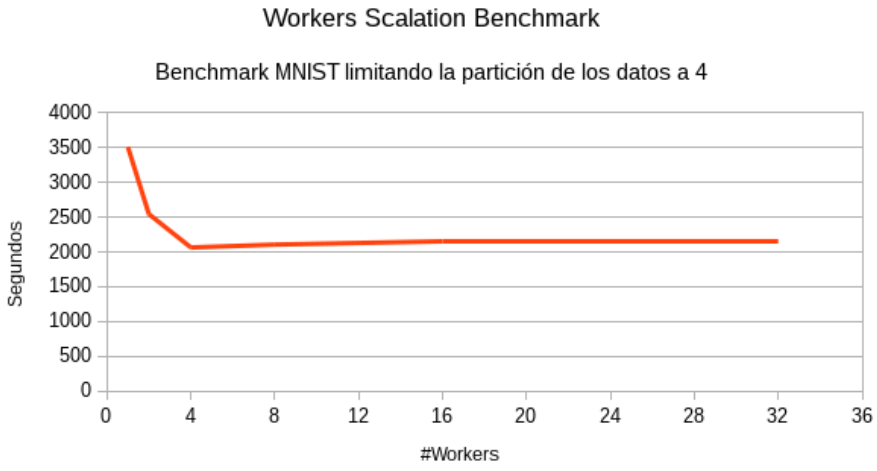


Figura 15: Escalabilidad con limitación de 4 particiones

La figura 15 muestra como, al limitar el número de particiones a 4 también se limita la mejora a 4 workers. De hecho, a partir de 4 workers el tiempo es ligeramente superior, cosa que se deba seguramente a la sincronización o la comunicación entre workers.

4 Conclusión

El framework de DeepLearning4J está teniendo mucho desarrollo, y va avanzando poco a poco. Dados los resultados que hemos obtenido en MareNostrum, no parece que se vaya a recomendar su uso por el momento, ya que esta pensada (MareNostrum) para ejecuciones de mucho mayores que las que hemos podido hacer (aunque aún hay muchas cosas que podríamos probar).

Por otro lado, si se arreglaran el problema del consumo de memoria, estoy seguro de que se podrían conseguir mejores números. Dado el caso, podríamos probar más configuraciones. Como se ve en la figura 11 el uso de las CPUs no es muy bueno, pues se espera que se usen los 16 cores del nodo, y el máximo en este caso es 4. Por tanto, se podría poner más de un worker por nodo como en el equipo local. De esta forma se usarían los nodos de manera más eficiente.

De todas maneras, hasta la última versión (0.4rc-3.8) el ritmo de publicaciones de nuevas versiones era de una cada mes o cada dos. Viendo la actividad del repositorio, puede que en un futuro no muy lejano puede que los problemas que hemos tenido sean resueltos, por nosotros mismos o algún otro colaborador. Yo por mi parte, le seguiré la pista y seguiré jugando con él para seguir aprendiendo y mejorándolo si puedo.

4.1 Alternativas

Dados los problemas que se estaban teniendo con DL4J se decidió probar con la versión más nueva posible, que está todavía en desarrollo 0.4rc-3.9 (tanto DL4J como ND4J). Esta versión resultó ser peor que la anterior, debido a que ni tan solo funcionaba en local.

Con la nueva versión también aparecen nuevos bugs. En este caso la librería

de álgebra lineal que usa DL4J (ND4J), que también presenta algún puntero nulo en sus operaciones internas. Dado el error, se intenta ver si es cosa del código de la aplicación o no y se descubre que DL4J estaba generando datos nulos al llamar a una función que devuelve NULL en las capas del tipo subsampling y no estaba siendo controlado. Fue reportado a los desarrolladores en github [13] y arreglado en el siguiente *commit*. Aunque detrás de esta solución habían más errores que están por resolver.

En el transcurso de las pruebas y errores, se planteó la idea de cambiar de framework. Se propuso TensorFlow, pero, ni era lo que queríamos ver (ya que se quería usar con Spark) ni se pudo instalar en MinoTauro, la máquina con GPUs del BSC, debido a que la versión de la librería de CUDA era incompatible.

En Noviembre de 2015 se publicó un proyecto de Apache Incubator, SystemML [22]. Este framework para Machine Learning, con un equipo de colaboradores de IBM, Databricks y Netflix[21], también funciona sobre Spark. Aunque no funciona fuera de la Spark shell ni tiene soporte para redes neuronales, con lo que fue descartado tras investigar un poco.

4.2 Trabajo futuro

Como ya he dicho, se pueden hacer más cosas aún con este framework. Por un lado, solo hemos probado un ejemplo y puede que con otro tipo de red neuronal u otro código totalmente distinto, quizá podamos obtener mejores resultados. Siguiendo este camino se pueden hacer pruebas diversas con versiones más recientes de DL4J según vaya siendo publicadas

Otra posibilidad es tener Spark en MinoTauro (la máquina con GPUs del BSC). Allí se podría también probar DL4J pero usando las GPUs de los nodos, con las que seguramente se obtendría mejor rendimiento.

[13] issue: https://github.com/deeplearning4j/deeplearning4j/issues/1358

Por último, siempre estar alerta de nuevo software que se publique o actualizaciones de otros. Por ejemplo, SystemML, si en algún momento incluyen redes neuronales entre sus algoritmos, o TensorFlow si se publica una versión de CPUs distribuida.

Referencias

[1] Jordi Torres. *Spark Deployment and Performance Evaluation on a Petascale HPC Setup*. Ed. por Spark-submit.org. 2015. URL: https://spark-summit.org/eu-2015/events/spark-deployment-and-performance-evaluation-on-a-petascale-hpc-setup (visitado 20-09-2015).

[2] Vincent Vanhoucke, Andrew Senior y Mark Z. Mao. *Improving the speed of neural networks on CPUs*. Ed. por Googleusercontent.com. URL: http://static.googleusercontent.com/media/research.google.com/en//pubs/archive/37631.pdf (visitado 22-09-2015).

[3] *Trello, kanban based project manager*. URL: https://trello.com/.

[4] *Trello, kanban based project manager Android App*. URL: https://play.google.com/store/apps/details?id=com.trello.

[5] *GIT version control system*. URL: https://git-scm.com/.

[6] Ahmed Banafa. *What is deep learning?* Ed. por Linkedin.com. 2014. URL: http://radar.oreilly.com/2014/07/what-is-deep-learning-and-why-should-you-care.html (visitado 23-09-2015).

[7] Dallin Akagi. *A Primer on Deep Learning*. Ed. por Datarobot.com. URL: http://www.datarobot.com/blog/a-primer-on-deep-learning/ (visitado 23-09-2015).

[8] Stanford.edu, ed. *Convolutional Neural Network*. URL: http://ufldl.stanford.edu/tutorial/supervised/ConvolutionalNeuralNetwork/ (visitado 23-09-2015).

[9] *Apache Spark Home page*. URL: https://spark.apache.org/ (visitado 23-09-2015).

[10] Top500, ed. *Top500 List - June 2015*. URL: http://www.top500.org/list/2015/06/ (visitado 23-09-2015).

[11] *Apache Maven Home page*. URL: https://maven.apache.org/ (visitado 18-04-2016).

[12] Srini Penchikala. *Big Data Processing with Apache Spark*. Ed. por Infoq.com. 2015. URL: http://www.infoq.com/articles/apache-spark-introduction (visitado 23-09-2015).

[13] BSC.es, ed. *MareNostrum III (2013) System Architecture*. URL: http://www.bsc.es/marenostrum-support-services/mn3 (visitado 23-09-2015).

[14] Deeplearning4j.org, ed. *Deep Learniing for Java Home page*. URL: http://deeplearning4j.org/ (visitado 23-09-2015).

[15] Deeplearning4j.org, ed. *Convolutional Networks in Java - Deeplearning4j: Open-source, distributed deep learning for the JVM*. URL: http://deeplearning4j.org/convolutionalnets.html (visitado 15-04-2016).

[16] Jeffrey Dean y col. *Large Scale Distributed Deep Networks*. Ed. por Googleusercontent.com. URL: http://static.googleusercontent.com/media/research.google.com/es//archive/large_deep_networks_nips2012.pdf (visitado 18-10-2015).

[17] Jeff Dean y Rajat Monga. *TensorFlow - Google's latest machine learning system, open sourced for everyone*. Ed. por GoogleResearch. URL: http://googleresearch.blogspot.com.es/2015/11/tensorflow-googles-latest-machine_9.html (visitado 15-04-2016).

[18] Sumit Gupta. *NEW TOP500 LIST: 4X MORE GPU SUPERCOMPUTERS*. Ed. por Nvidia.com. 2012. URL: http://blogs.nvidia.com/blog/2012/07/02/new-top500-list-4x-more-gpu-supercomputers/ (visitado 23-09-2015).

[19] Tom Simonite. *China derrota a Google en aprendizaje profundo para reconocer imágenes*. Ed. por Technologyreview.es. 2015. URL: http://www.technologyreview.es/informatica/47417/china-derrota-a-google-en-aprendizaje-profundo/ (visitado 23-09-2015).

[20] Aaron Mamiit. *Baidu AI Supercomputer Minwa Beats Google, Microsoft, And Humans In Image Recognition*. Ed. por Techtimes.com. 2015. URL: http://www.techtimes.com/articles/53270/20150516/baidu-ai-supercomputer-minwa-beats-google-microsoft-and-humans-in-image-recognition.htm (visitado 23-09-2015).

[21] *Apache SystemML - Community Members*.

[22] *Apache SystemML - Declarative Large-Scale Machine Learning*.

[23] *Artificial Neural Networks Technology*. 10 de abr. de 2007. URL: http://psych.utoronto.ca/users/reingold/courses/ai/cache/neural4.html (visitado 12-04-2016).

[24] Istvan S. N. Berkeley. *A Revisionist History of Connectionism*. 1997. URL: http://www.ucs.louisiana.edu/~isb9112/dept/phil341/histconn.html (visitado 12-04-2016).

[25] *Caffe - Deep Learning Framework*. URL: http://caffe.berkelyvision.org (visitado 25-03-2016).

[26] *Centros de Excelencia "Severo Ochoa"*. 2014. URL: http://www.idi.mineco.gob.es/portal/site/MICINN/menuitem.7eeac5cd345b4f34f09dfd1001432ea0/?vgnextoid=cba733a6368c2310VgnVCM1000001d04140aRCRD (visitado 10-10-2015).

[27] Harbey A. Cohen. *The Perceptron = Perceptual maths and Neural net History*. URL: http://harveycohen.net/image/perceptron.html (visitado 12-04-2016).

[28] *Elastic - Revealing insights from data*. URL: https://www.elastic.co/.

[29] Instituto Nacional de Estadística de España. *Estadística sobre Actividades en I+D. Resultados definitivos. Año 2013*. 2015. URL: http://www.ine.es/prensa/np889.pdf.

[30] Ignacio Fariza. "España sigue cediendo puestos en la tabla europea de inversión en I+D". En: *El País* (17 de nov. de 2014). URL: http://elpais.com/elpais/2014/11/17/ciencia/1416239255_999222.html.

[31] Ian Goodfellow, Yoshua Bengio y Aaron Courville. “Deep Learning”. Book in preparation for MIT Press. 2016. URL: http://www.deeplearningbook.org (visitado 10-04-2016).

[32] Google Inc., ed. *TensoFlow - an Open Source Software Library for Machine Intelligence*. 2015. URL: http://tensorflow.org.

[33] JetBrains. *Intellij IDEA the Java IDE*. https://www.jetbrains.com/idea. 2001–2016.

[34] LISA Lab., ed. *Theano at a glance*. 2016. URL: http://deeplearning.net/software/theano/introduction.html#introduction.

[35] Yann LeCun, Corinna Cortes y Christopher J.C. Burges. *The MNIST (Mixed National Institute of Standards and Technology) Database*. 2016. URL: http://yann.lecun.com/exdb/mnist/ (visitado 23-03-2016).

[36] *Neural Networks - History*. 16 de dic. de 2006. URL: https://cs.stanford.edu/people/eroberts/courses/soco/projects/neural-networks/History/history1.html (visitado 12-04-2016).

[37] Michael Nielsen. *Neural networks and deep learning*. URL: http://neuralnetworksanddeeplearning.com/chap1.html (visitado 12-01-2016).

[38] *PRACE Research Infrastructure*. 2014. URL: http://www.prace-ri.eu/ (visitado 13-04-2016).

[39] *Random Sampling of MNIST*. web—photo. URL: http://andrew.gibiansky.com/blog/machine-learning/k-nearest-neighbors-simplest-machine-learning/images/mnist-example-ipy.png.

[40] *Recurrent neural network*. 6 de abr. de 2016. URL: https://en.wikipedia.org/wiki/Recurrent_neural_network (visitado 13-04-2016).

Apéndice A Script extractor de datos del output

scripts/TextToConfMat.py

```
import re
import sys
import numpy as np
import time

p = re.compile(ur'([0-9]).*([0-9]): ([0-9]+)')
file = open(sys.argv[1])
test_str = file.read()
matches = re.findall(p, test_str)

M = np.empty((10, 10,))
for m in matches:
    key = int(m[0])
    predicted = int(m[1])
    times = int(m[2])
    M[key][predicted] = times

total = M.sum()
rows = M.sum(axis=1)
cols = M.sum(axis=0)
diag = np.diag(M)
tn = diag.sum() - diag
fn = rows - diag
fp = cols - diag
time.sleep(1)
A = (diag + tn)/total
P = diag / (diag+fp)
R = diag / (diag+fn)
for i in range(10):
    print "%d & %d & %d & %d & %d & %.4f & %.4f & %.4f \\\\" %(i,
        diag[i], tn[i], fp[i], fn[i], A[i], P[i], R[i])

A = A.sum()/10
P = P.sum()/10
```

```
R = R.sum()/10
F1 = (2*(P * R/(P+R)))

print "Accuracy: %f" %(A)
print "Precision: %f" %(P)
print "Recall: %f" %(R)
print "F1: %f" %F1

for i in range(10):
    sys.stdout.write("%d & " %i)
    for v in M[i]:
        sys.stdout.write("%d & " %v)
    sys.stdout.write('\b\b \\\\ \n')
```

Apéndice B Script contabilizador de las horas de cpu

scripts/ElasticQuery.py

```
import requests,json,time,datetime

esHost = 'http://mn3mon2.bsc.es/marenostrum/_search'
initial_date = '09/09/2015'
t = time.mktime(datetime.datetime.strptime(initial_date, "%d/%m/%Y"
    ).timetuple())
print t
data = '''{
  (...)
}'''
nfrom = 0
size = 10000
actual_data = data %(t, nfrom, size)
headers = {'content-type': 'application/json', 'Accept-Charset': '
    UTF-8'}
r = requests.post(esHost, data=actual_data, headers=headers)

datastring = json.dumps(r.json())

jsondata = json.loads(datastring)

class Job:

    def __init__(self, json):
        def ifExists(job, string, default=""):
            if string in job:
                return job[string]
            else:
                return default
        job = json['_source']
        self.nhosts = job[u'num_hosts']
        self.hosts = ifExists(job, u'hosts')
```

```
            self.host = ifExists(job, u'host')
            self.status = ifExists(job,u'status')
            self.elapsedTime = ifExists(job, u'elapsed')
            self.jobscript = ifExists(job, u'jobscript')
            self.jobid = ifExists(job, u'jobid', default=0)
            self.cpu_hours = ifExists(job,u'cpu_hours', default=0)
            self.ncpus = ifExists(job, u'cpus', default=0)

        def jobScriptContains(self, string):
            return string in self.jobscript

    if not r.ok or jsondata["_shards"]["successful"] == 0 or jsondata['
        timed_out']:
        print "Something went wrong"
    else:
        print "The query took: %d milliseconds" %jsondata["took"]
        hits = jsondata['hits']
        print "The query returns %d hits" %hits['total']
        sparkjobs = []
        for e in hits['hits']:
            j = Job(e)
            if j.jobScriptContains("spark4mn_job"):
                sparkjobs.append(j)
        print "There is %d spark4mn jobs in the query" %len(sparkjobs)
        print "Total consumed time executing %.2f hours" %reduce(
            lambda x,y: x+y, (map(lambda x: x.cpu_hours, sparkjobs)))
        def sumT (x, y):
            if y[0] == u'done':
                x[0] += y[1]
            elif x[0] == u'exited':
                x[0] += y[1]
            else:
                x[2] += y[1]
            return x

        status = reduce(sumT, [[0,0,0]] + map(lambda x: (x.status, 1),
            sparkjobs))
        print "job status summary %d done, %d exited, %d cancelled" %
```

```
tuple(status)
```

Apéndice C Código definitivo ejemplo MNIST

scripts/JavaMNistClass.java

```
package org.deeplearning4j.ml;

import com.sun.jndi.toolkit.url.Uri;
import org.apache.avro.generic.GenericData;
import org.apache.spark.SparkConf;
import org.apache.spark.SparkContext;
import org.apache.spark.api.java.JavaRDD;
import org.apache.spark.api.java.JavaSparkContext;
import org.apache.spark.broadcast.Broadcast;
import org.apache.spark.input.PortableDataStream;
import org.apache.spark.rdd.RDD;
import org.apache.spark.sql.SQLContext;
import org.apache.spark.storage.StorageLevel;
import org.codehaus.janino.Java;
import org.deeplearning4j.datasets.iterator.BaseDatasetIterator;
import org.deeplearning4j.eval.Evaluation;
import org.deeplearning4j.nn.api.OptimizationAlgorithm;
import org.deeplearning4j.nn.conf.MultiLayerConfiguration;
import org.deeplearning4j.nn.conf.NeuralNetConfiguration;
import org.deeplearning4j.nn.conf.Updater;
import org.deeplearning4j.nn.conf.layers.ConvolutionLayer;
import org.deeplearning4j.nn.conf.layers.DenseLayer;
import org.deeplearning4j.nn.conf.layers.OutputLayer;
import org.deeplearning4j.nn.conf.layers.SubsamplingLayer;
import org.deeplearning4j.nn.conf.layers.setup.
    ConvolutionLayerSetup;
import org.deeplearning4j.nn.conf.preprocessor.
    CnnToFeedForwardPreProcessor;
import org.deeplearning4j.nn.conf.preprocessor.
    FeedForwardToCnnPreProcessor;
import org.deeplearning4j.nn.multilayer.MultiLayerNetwork;
import org.deeplearning4j.nn.weights.WeightInit;
import org.deeplearning4j.spark.impl.multilayer.SparkDl4jMultiLayer
    ;
```

```
import org.dmg.pmml.True;
import org.json4s.FileInput;
import org.nd4j.linalg.api.complex.LinearViewComplexNDArray;
import org.nd4j.linalg.api.ndarray.BaseNDArray;
import org.nd4j.linalg.api.ndarray.INDArray;
import org.nd4j.linalg.dataset.DataSet;
import org.nd4j.linalg.dataset.api.iterator.DataSetIterator;
import org.nd4j.linalg.factory.Nd4j;
import org.nd4j.linalg.lossfunctions.LossFunctions;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import scala.Tuple2;

import javax.xml.crypto.Data;
import java.io.BufferedReader;
import java.io.File;
import java.io.FileInputStream;
import java.io.PrintWriter;
import java.lang.management.ManagementFactory;
import java.nio.file.Files;
import java.nio.file.Path;
import java.nio.file.Paths;
import java.util.*;

/**
 * Created by tfg on 27/02/16.
 */
public class JavaMNistClass {

    private static final Logger log = LoggerFactory.getLogger(
        JavaMNistClass.class);

    private static ArrayList<java.lang.Long> mem;
    private static int nChannels = 1;
    private static int outputNum = 10;
    private static int iterations = 1;
    private static int seed = 123;
```

```
68      /*
        FOR DEBUGGING
70      export SPARK_JAVA_OPTS=-agentlib:jdwp=transport=dt_socket,
            server=y,suspend=n,address=5005
         */
72
        /**
74       * Converts an array of bytes to a binary INDArray
         * @param b array of bytes
76       * @param threshold minimum value to set byte to 1
         * @return INDArray
78       */
        public static INDArray bytesToINDArray(byte[] b, int threshold)
             {
80          INDArray ndArray = Nd4j.create(1, b.length);
            int i = 0;
82          for (byte B : b) {
                long l = ((long) B & 0xffL) > threshold? 1 : 0;
84              ndArray.put(0, i, l);
                ++i;
86          }
            return ndArray;
88      }

90      /**
         * Read the Mnist file providing the path with an string
92       * @param img Path to the images file
         * @param lbl Path to the label file
94       * @param records Number of images to read
         * @param offset Number of images to skip before start reading
96       * @return Iterator<DataSet>
         * @throws Exception
98       */
        public static Iterator<DataSet> readMnistFile(String img,
            String lbl, int records, int offset) throws Exception
100     {
            return readMnistFile(new Uri(img), new Uri(lbl), records,
                offset);
```

```
102     }

104     /**
         * Read the Mnist file provifing the path as on Uri
106      * @param imgPath Uri to the images file
         * @param lblPath Uri to the label file
108      * @param records Number of images to read
         * @param offset Number of images to skip before start reading
110      * @return Iterator<DataSet>
         * @throws Exception
112      */
        public static Iterator<DataSet> readMnistFile(Uri imgPath, Uri
            lblPath, int records, int offset) throws Exception
114     {
            // offset is the number of images to skip
116         FileInputStream imgFile = new FileInputStream(imgPath.
                getPath());
            FileInputStream lblFile = new FileInputStream(lblPath.
                getPath());
118         List<DataSet> ds = new ArrayList<DataSet>();
            int imagepixels = 28*28;
120         byte[] pixel = new byte[imagepixels];
            byte[] label = new byte[1];
122         int i = 0;
            String cont = "show";
124
            long imgBytesSkipped = imgFile.skip(16+offset*imagepixels);
126         long lblBytesSkipped = lblFile.skip(8+offset);

128         while (i < records && imgFile.read(pixel) != -1 && lblFile.
                read(label) != -1) {
                byte[] labelarr = new byte[] {0,0,0,0,0,0,0,0,0,0};
130             int labelInt = label[0];
                labelarr[labelInt] = 1;
132 //              if (cont.equals("show") || cont.equals("") ) {
    //                  printNumber(pixel, label);
134 //                  cont = System.console().readLine();
    //              }
```

```
136             INDArray pisels = bytesToINDArray(pixel, 80);
                INDArray indlabel = bytesToINDArray(labelarr, 0);
138             DataSet record = new DataSet(pisels, indlabel);
//              ds.addRow(record, i);
140             ds.add(record);

142             ++i;
            }
144         return ds.iterator();
        }
146
        /**
148      * Print the numbers to the terminal for Debug porpouses
         * @param pixel Array of pixels when 0 means background.
150      * @param label Binary array, the position with 1 is the label
         */
152     private static void printNumber(byte[] pixel, byte[] label) {
            for (int i = 0; i < 28; ++i) {
154             for (int j = 0; j < 28; ++j) {
                    long l = ((long) pixel[i*28+j] & 0xffL);
156                 char x;
                    if (l == 0)
158                     x = '.';
                    else
160                     x = 'x';
                    System.out.print(x);
162             }
                System.out.print("\n");
164         }
            int k = 0;
166         while (k < label.length) {
                System.out.print(label[k++]);
168         }
            System.out.print("\n");
170     }

172     /**
         * Get the NN configuration as is in the example of MNist from
```

```
         the repository
174      * @return MultiLayerConfiguration.Builder
         */
176     public static MultiLayerConfiguration.Builder get6LayerConf ()
            {
            return new NeuralNetConfiguration.Builder()
178                 .seed(seed)
                    .iterations(iterations)
180                 .regularization(true).l2(0.0005)
                    .learningRate(0.1)
182                 .optimizationAlgo(OptimizationAlgorithm.
                        STOCHASTIC_GRADIENT_DESCENT)
                    .updater(Updater.ADAGRAD)
184                 .list(6)
                    .layer(0, new ConvolutionLayer.Builder(5, 5)
186                         .nIn(nChannels)
                            .stride(1, 1)
188                         .nOut(20)
                            .weightInit(WeightInit.XAVIER)
190                         .activation("relu")
                            .build())
192                 .layer(1, new SubsamplingLayer.Builder(
                        SubsamplingLayer.PoolingType.MAX, new int[]{2,
                        2})
                            .build())
194                 .layer(2, new ConvolutionLayer.Builder(5, 5)
                            .nIn(20)
196                         .nOut(50)
                            .stride(2, 2)
198                         .weightInit(WeightInit.XAVIER)
                            .activation("relu")
200                         .build())
                    .layer(3, new SubsamplingLayer.Builder(
                        SubsamplingLayer.PoolingType.MAX, new int[]{2,
                        2})
202                         .build())
                    .layer(4, new DenseLayer.Builder().activation("relu
                        ")
```

```
204                         .weightInit(WeightInit.XAVIER)
                            .nOut(200).build())
206                 .layer(5, new OutputLayer.Builder(LossFunctions.
                        LossFunction.NEGATIVELOGLIKELIHOOD)
                            .nOut(outputNum)
208                         .weightInit(WeightInit.XAVIER)
                            .activation("softmax")
210                         .build())
                    .backprop(true).pretrain(false);
212     }

214     /**
         * Gets the NN configuration of an experimental Network
216      * @return MultiLayerConfiguration.Builder
         */
218     public static MultiLayerConfiguration.Builder get4LayerConf() {
            return new NeuralNetConfiguration.Builder()
220                 .seed(seed)
                    .iterations(iterations)
222                 .regularization(true).l2(0.0005)
                    .learningRate(0.01)
224                 .weightInit(WeightInit.XAVIER)
                    .optimizationAlgo(OptimizationAlgorithm.
                        STOCHASTIC_GRADIENT_DESCENT)
226                 .updater(Updater.NESTEROVS).momentum(0.9)
                    .list(4)
228                 .layer(0, new ConvolutionLayer.Builder(5, 5)
                            .nIn(nChannels)
230                         .stride(1, 1)
                            .nOut(20).dropOut(0.5)
232                         .activation("relu")
                            .build())
234                 .layer(1, new SubsamplingLayer.Builder(
                        SubsamplingLayer.PoolingType.MAX)
                            .kernelSize(2, 2)
236                         .stride(2, 2)
                            .build())
238                 .layer(2, new DenseLayer.Builder().activation("relu
```

```
                    ")
                        .nOut(500).build())
240                 .layer(3, new OutputLayer.Builder(LossFunctions.
                        LossFunction.NEGATIVELOGLIKELIHOOD)
                        .nOut(outputNum)
242                     .activation("softmax")
                        .build())
244                 .backprop(true).pretrain(false);
    }
246
    /**
248  * Gets the NN configuration based on the parameter passed
     * @param mode
250  * @return
     */
252 public static MultiLayerNetwork getSparkNetwork (int mode) {
        //Set up network configuration
254     MultiLayerConfiguration.Builder builder = null;
        if (mode == 0) {
256         builder = get6LayerConf();
        }
258     else {
            builder = get4LayerConf();
260     }

262     new ConvolutionLayerSetup(builder,28,28,1);

264     MultiLayerConfiguration conf = builder.build();
        MultiLayerNetwork net = new MultiLayerNetwork(conf);
266     net.init();
        net.setUpdater(null);    //Workaround for minor bug in 0.4-
            rc3.8
268     return net;
    }
270
    public static void main(String args[]) throws Exception {
272     System.out.println(ManagementFactory.getRuntimeMXBean().
            getName());
```

```
//          Thread.sleep(2);
274         SparkConf sparkConf = new SparkConf()
//                  .setMaster("local")
276                 .setAppName("Mnist Classification Suilabs (Java)")
                    .set("spark.akka.frameSize", "128");
278         JavaSparkContext sc = new JavaSparkContext(sparkConf);
            System.out.println("FRAMESIZE "+sc.getConf().get("spark.
                akka.frameSize"));
280         sc.setLogLevel("WARN");
            Map<String, String> env = System.getenv();
282         int nCores;
            try {
284             nCores = Integer.parseInt(env.get("SPARK_NNODES"))*
                    Integer.parseInt(env.get("SPARK_NWORKERS_PER_NODE")
                    );
            }
286         catch (NumberFormatException e){
                nCores = Integer.parseInt(env.get("
                    SPARK_EXECUTOR_INSTANCES"));
288         }
            env = null;
290         System.out.println("Number of cores: " + nCores);
            for (String arg : args) {
292             System.out.println(arg);
            }
294         int mode = Integer.parseInt(args[0]);
            int records = Integer.parseInt(args[1]);
296         int nTrain = (int) Math.round(records * 0.8);
            int batchTrain = 2000;
298         int nTest = records - nTrain;

300         String imagesPath = args.length == 4 ? args[2]
                    : "file://" + System.getProperty("user.dir") + "/
                        data/train-images-idx3-ubyte";
302         String labelsPath = args.length == 4 ? args[3]
                    : "file://" + System.getProperty("user.dir") + "/
                        data/train-labels-idx1-ubyte";
304
```

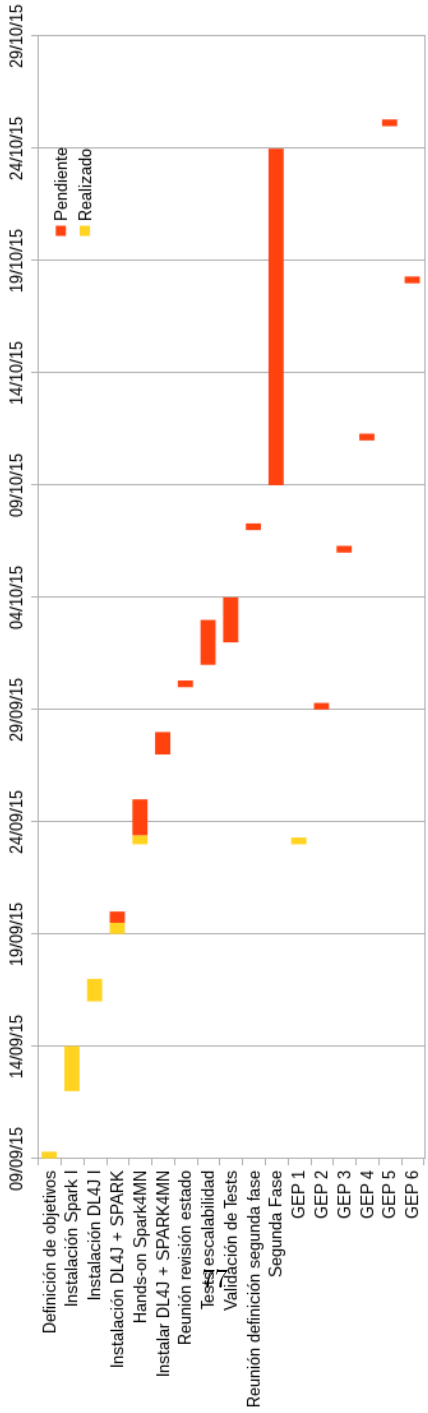
```
        MultiLayerNetwork net = getSparkNetwork(mode);
306
        List<DataSet> test = new ArrayList<DataSet>(nTest);
308
        int totalRecords = records;
310     int step = 0;
        int batchSize = 50;
312     while (records > 0) {
            int minBatch = Math.min(batchTrain, records);
314         int trainBatch = (int) Math.round(minBatch*0.8);
            int testBatch =  minBatch - trainBatch;
316         System.out.println("[INFO] "+records/batchTrain+" batch
                training to go");
            // read next image batch set
318         // batchTrain is always lower than records except in
                the last step
            Iterator<DataSet> diter = readMnistFile(imagesPath,
                labelsPath, minBatch, batchTrain*step);
320         List<DataSet> allData = new ArrayList<DataSet>(records)
                ;
            while (diter.hasNext()) {
322             allData.add(diter.next());
            }
324         diter = null;
            Collections.shuffle(allData, new Random(12345));
326
            Iterator<DataSet> iter = allData.iterator();
328         List<DataSet> train = new ArrayList<DataSet>(minBatch);
            int c = 0;
330         while (iter.hasNext()) {
                if (c++ <= trainBatch) train.add(iter.next());
332             else test.add(iter.next());
            }
334         iter = null;

336         JavaRDD<DataSet> sparkDataTrain = sc.parallelize(train,
                nCores);
    //      Broadcast<JavaRDD<DataSet> > broadcastRDD = sc.broadcast(
```

```
        sparkDataTrain);
338             train = null;
                sparkDataTrain.persist(StorageLevel.MEMORY_ONLY_SER());
340
                //Create Spark multi layer network from configuration
342             SparkDl4jMultiLayer sparkNetwork = new
                    SparkDl4jMultiLayer(sc, net);
                //Train network
344             System.out.println("--- Starting network training ---")
                    ;
                //Run learning. Here, we are training with
                    approximately 'batchSize' examples on each executor
346 //            net = sparkNetwork.fitDataSet(broadcastRDD.value());
                net = sparkNetwork.fitDataSet(sparkDataTrain, batchSize
                    );
348
                // decrease records counter
350             records -= batchTrain;
                step++;
352
                Evaluation eval = new Evaluation();
354             for(DataSet ds : test){
                    INDArray output = net.output(ds.getFeatureMatrix())
                        ;
356                 eval.eval(ds.getLabels(),output);
                }
358 //              List<String> lines = Arrays.asList(eval.stats());
    //              Path file = Paths.get("/tmp/tmp-eval" + step + ".txt
        ");
360 //              Files.write(file, lines);
            }
362
            //Evaluate (locally)
364         Evaluation eval = new Evaluation();
            for(DataSet ds : test){
366             INDArray output = net.output(ds.getFeatureMatrix());
                eval.eval(ds.getLabels(),output);
368         }
```

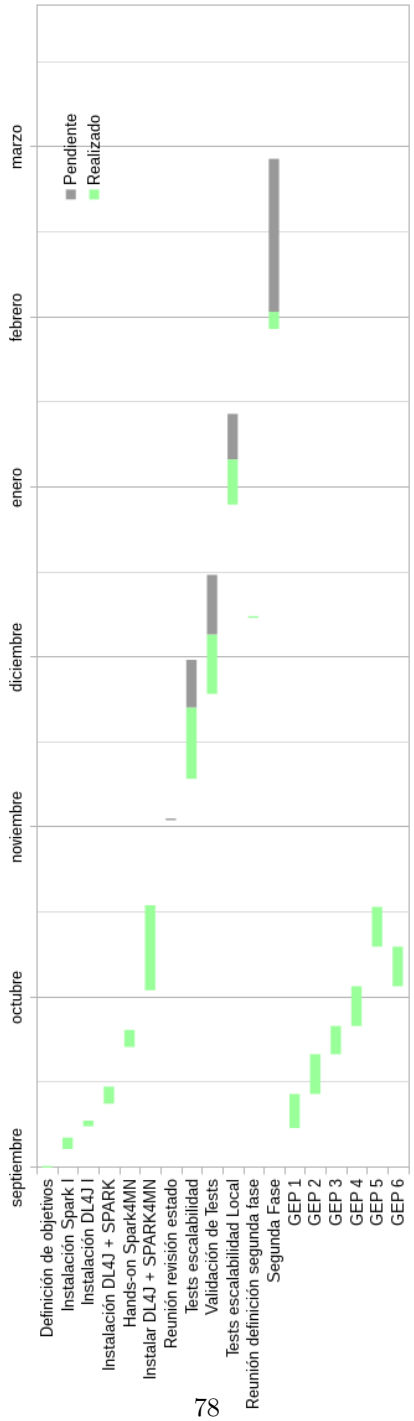
```
            System.out.println(eval.stats());
370         System.out.println("***************Example finished
                ********************");
        }
372 }
```

Apéndice D Diagramas de Gannt

D.1 Planificación inicial



D.2 Revisión de la planificación



Índice de figuras

1 Ejemplo de red neuronal analizando una imagen [31] 12

2 Representación de una neurona en una ANN 13

3 Función escalón unitario 13

4 Función sigmoidea 14

5 Representación de una MLP con 4 capas: una de entrada, dos ocultas y una de salida.[37] 15

6 Impresión de pantalla del software online Trello 21

7 Convolutional Network Layer decomposition[15] 22

8 Ejemplo del resultado de aplicar una capa de submuestreo. Entrada y salida, izquierda y derecha respectivamente[15] 23

9 Muestra aleatoria de la base de datos MNIST[39] 23

10 Diagrama de Gantt revisado. Ver Apendice D.2 30

11 Gráficos de uso de cpu (arriba) y de memoria(abajo) de una ejecución correcta del ejemplo 47

12 Cálculo de la exactitud, precisión y exhaustividad a cada paso del aprendizaje incremental 48

13 Escalabilidad por worker 49

14 DataSize Benchmark 50

15 Escalabilidad con limitación de 4 particiones 51

Índice de cuadros

1 Matriz de confusión de una ejecución con 48000 imágenes de entrenamiento y 12000 de prueba. 24

2 Matriz de confusión identificando falsos negativos, falsos positivos, positivos y negativos 25

3 Calculo de los valores de a, p, r para cada una de la etiquetas . . 26

4 Tareas 27

5 Hitos 27

6 Dependencias de tareas 28

7 Repartición de tareas 35

8 Presupuesto de recursos humanos 35

9 Presupuesto de recursos materiales 36

10 Costes indirectos derivados del consumo eléctrico 37

11 Costes de Imprevistos 37

12 Resumen de Presupuesto 38

13 Margen de presupuesto inicial 39

14 Matriz de sostenibilidad 41