
***Facultad***
***de***
***Ciencias***

# APLICACIÓN DE TÉCNICAS DE MACHINE LEARNING, UN CASO PRÁCTICO

## (Applying Machine Learning Techniques, a Practical Case)

**Trabajo de Fin de Grado**
**para acceder al**

**GRADO EN INGENIERÍA INFORMÁTICA**

**Autor: Rubén Revuelta Briz**

**Director: Pablo Abad Fidalgo**

**Septiembre - 2018**

**Resumen**

La gran cantidad de datos que manejan las empresas en sus procesos de negocio, así como su complejidad, han propiciado la búsqueda de mecanismos capaces de extraer información de interés a partir del ingente volumen de datos y sus complejas relaciones, imposibles de encontrar solamente a través del análisis manual de los datos. Para dicha labor, las herramientas de aprendizaje máquina se presentan como una solución capaz de aprovechar estos volúmenes de datos de forma altamente rentable y eficiente.

El presente trabajo pretende explorar la utilización de técnicas de aprendizaje automático en un entorno industrial real, como mecanismo de mejora del proceso de fabricación de una pieza a lo largo de una cadena de montaje. Dicho proceso de montaje incorpora un número elevado de sensores, capaces de proporcionar un número elevado de datos sobre el proceso de fabricación de cada pieza. Mediante la aplicación de técnicas de Aprendizaje Máquina, conocido comúnmente en inglés como Machine Learning (ML), se pretende descubrir si con los datos obtenidos durante el proceso de fabricación se pueden sacar conclusiones que nos permitan mejorar la eficiencia, detectando aquellas piezas defectuosas en etapas tempranas durante el proceso de fabricación y evitando que estas avancen por la cadena de montaje para ser finalmente desechadas.

Las tareas principales realizadas durante este proyecto han consistido en estudiar los conceptos y procesos involucrados en proyectos de Machine Learning y, más concretamente, los relacionados con redes neuronales, analizar y procesar conjuntos de datos para su posterior utilización, desarrollar y configurar adecuadamente modelos basados en el aprendizaje máquina y evaluar e interpretar los resultados inferidos por el modelo.

*Palabras clave:* Machine Learning, Deep Learning, redes neuronales, TensorFlow.

**Summary**

The large amount of data handled by companies in their businesses as well as their complexity, have prompted the search for mechanisms able to obtain the relevant information from the enormous volume of data and their complex relations, unbearable to handle manually. In order to do this, machine learning tools arise as a solution capable of taking advantage of these volumes of data in a profitable and efficient way.

The present work tries to explore the utilization of automatic learning techniques in an actual industrial environment, as an improvement mechanism in the production of a machinery piece in the assembly line. This assembly procedure has incorporated a great number of sensors, which provide a high amount of data regarding the production of each piece. Through the utilization of ML techniques, the intention is to discover if the data obtained during the fabrication process allow us to improve in efficiency, detecting faulty pieces in early stages of the process and preventing them from advancing through the assembly line, to be finally discarded. The main tasks conducted during this project include studying the concepts and processes involved in Machine Learning projects and, more specifically, those related to neural networks, analyzing and processing data sets for their future utilization, developing and configuring adequately models based in machine leaning and evaluating and interpreting the inferred results.

*Key words:* Machine Learning, Deep Learning, neural networks, TensorFlow.

# Índice

# 1. Introducción

Las nuevas tecnologías han originado una gran transformación en todos los sectores de la sociedad. Las industrias se han visto transformadas debido a la introducción de innovaciones tecnológicas que mejoran sus procesos productivos. En el campo de la salud las nuevas tecnologías han supuesto una herramienta esencial en la investigación de nuevas curas y en la aplicación de tratamientos, y en educación se ha facilitado el acceso y distribución de la información. Para las personas también ha supuesto un cambio radical en su estilo de vida, modificando sus hábitos de consumo y la forma en la que interaccionamos con el entorno y con el resto de la sociedad. Desde solicitar una cita médica hasta hacer la compra por Internet son acciones cotidianas que cualquier persona puede realizar hoy en día a través de un dispositivo móvil.

Este imparable avance tecnológico ha supuesto, durante la última década, un crecimiento exponencial en el número de dispositivos interconectados, disparando la capacidad de generación de datos. Este fenómeno, conocido como "Big Data"[1], ha supuesto una revolución en muchos aspectos de nuestra vida cotidiana donde ya se encuentra presente, tales como en la recomendación de contenido personalizado para el usuario o en la monitorización de constantes vitales a través de dispositivos inteligentes como el reloj inteligente. Las fuentes de generación de datos en entornos Big Data son múltiples, tan dispares como los contenidos multimedia y de redes sociales [2] o los datos generados en grandes proyectos de investigación. Entre todas las fuentes de datos, una de las que está cobrando últimamente una importancia creciente son las aplicaciones relacionadas con el conocido como Internet de las cosas (Internet of Things ó IoT). Las siglas IoT engloban todos aquellos sensores capaces de conectar el mundo físico con Internet, midiendo y almacenando una cantidad ingente de parámetros físicos en productos del consumidor (refrigeradores, cámaras de seguridad, decodificadores), sistemas industriales (cintas transportadoras, equipos de fabricación) y dispositivos comerciales (señales de tráfico, medidores inteligentes).

Las compañías industriales suponen uno de los principales usuarios de las aplicaciones de IoT, intentando que sus equipos y procesos industriales sean más inteligentes y seguros. Este proceso, bautizado con el nombre de Industria 4.0, persigue

la integración de tecnologías de procesamiento de datos, software inteligente y sensores en el proceso productivo, con el objetivo de realizar tareas predictivas, de control, planificación y producción que generen valor añadido al proceso de fabricación. Las empresas de nuestra comunidad autónoma no han sido ajenas a dichos avances tecnológicos, siendo algunas de ellas participantes activas en los procesos de la industria 4.0. Un ejemplo claro es la empresa BSH, fundada en Santander en 1967 y perteneciente al Grupo Bosch desde enero de 2015. Con 6 plantas de fabricación en España y más de 4000 empleados, es la empresa líder en el sector de fabricación de electrodomésticos en España [3]. En su planta de Santander, líder mundial en la fabricación de placas de cocina de gas, se ha realizado una apuesta clara por las nuevas tecnologías, dotando a sus líneas de fabricación más recientes (6 millones de euros de inversión) de un gran número de sensores que permiten medir un elevado número de parámetros relativos al proceso de fabricación de cada una de las piezas producidas.

Su producto más novedoso consiste en el desarrollo y fabricación de una válvula de gas (step valve) que permite graduar el nivel de fuego en valores discretos, proporcionando valores reproducibles de nivel de fuego (pasando del tradicional y ambiguo "a fuego medio" a un rango numérico del 1 al 9). El proceso de fabricación de dicha válvula consiste en el ensamblado de las más de 40 piezas de las que consta a través de una línea de producción, situada en su planta de Santander, con más de 300 pasos de fabricación. Gracias a los diferentes sensores incorporados en dicha línea, BSH mide y almacena más de 50 valores por pieza fabricada, incluyendo información identificativa, medidas de fuerza, profundidad (de atornillado ó perforado), temperatura, humedad, etc. y resultados de test (comprobación de funcionamiento). Durante los primeros meses tras su puesta en marcha, la línea sufrió diversos problemas, siendo uno de los principales el elevado número de piezas defectuosas fabricadas (chatarra). Actualmente, dicho valor de ha reducido a un porcentaje inferior al 5 %, pero el costo anual en chatarra sigue siendo relevante, cercano a los 400.000€.

Actualmente, la empresa busca mecanismos alternativos para sacar partido al volumen de datos producido, siendo una de sus líneas principales la utilización de tecnologías de aprendizaje automático para la detección de problemas en sus líneas de fabricación y reducción del volumen de chatarra. El presente TFG pretende ser

un punto de partida del proyecto descrito, consistiendo sus tareas principales en el aprendizaje en el manejo de herramientas de Machine Learning para poder llevar a cabo una serie de pruebas preliminares con los datos suministrados por la empresa BSH. Disponiendo de los datos de fabricación de más de dos millones de piezas, así como los resultados de los test de validez de las mismas (pruebas de verificación de su correcto funcionamiento), utilizaremos herramientas de aprendizaje basadas en Redes Neuronales profundas (Deep Learning) para intentar encontrar modelos capaces de inferir el resultado de los test de validez a través de los datos obtenidos por los sensores. Se trata de un paso inicial dentro de objetivo más ambicioso, que consistiría en la detección y descarte de piezas en etapas tempranas del proceso de fabricación que con alta probabilidad darán lugar a un producto defectuoso.

## 1.1. Machine y Deep Learning

La inteligencia artificial (artificial intelligence ó AI) y el aprendizaje máquina (Machine Learning ó ML) son técnicas de aprendizaje autónomo por computador que se llevan utilizando desde hace más de 50 años [4], a pesar de lo que se podría pensar dada su reciente popularidad. Desde los primeros programas de ajedrez con capacidad de aprendizaje [5] y las primeras redes neuronales artificiales [6] de los años 50, la evolución hasta la actualidad ha sido increíblemente rápida, siendo capaces en la actualidad de realizar tareas extremadamente complejas a través de dichos mecanismos, tales como reconocimiento de objetos en imágenes [7] o interpretación de gestos humanos [8].

Las herramientas de Machine Learning dotan a los computadores de la capacidad de aprendizaje sin ser explícitamente programados para ello, facilitando que un computador aprenda a realizar una determinada tarea a partir del estudio de un conjunto de ejemplos de entrenamiento. Tras dicho entrenamiento, el computador sería capaz de realizar la tarea para la que ha sido entrenado sobre datos que nunca se le han mostrado (distintos a los utilizados para el entrenamiento).

La gran cantidad de datos de los que se disponen hoy en día ha ocasionado que no sea posible establecer correlaciones entre ellos como se ha venido haciendo hasta el momento. Por ello, el ML surge como una herramienta ideal para establecer relaciones entre los datos que de otra forma no sería fácil. Una de las ventajas más

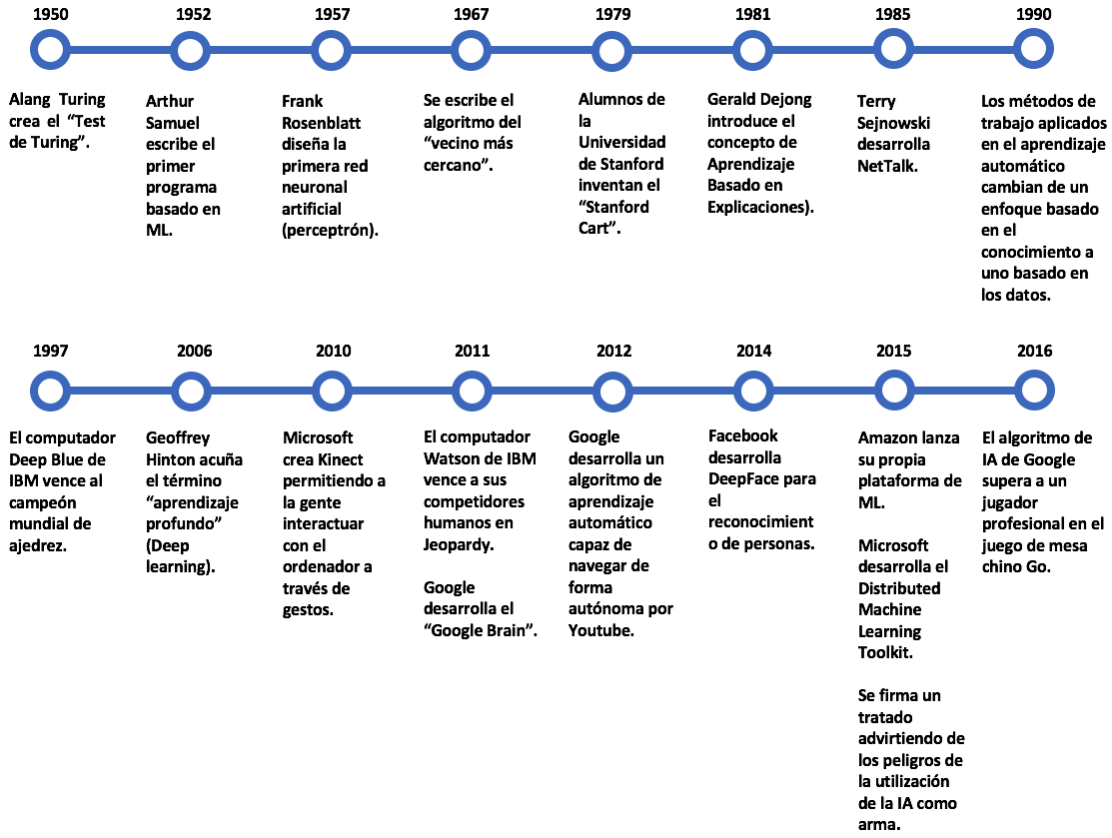


Figura 1: Evolución histórica de la IA, obtenida de la referencia [4].

importantes del ML es la gran capacidad de adaptación en entornos cambiantes [9]. Mientras que los métodos tradicionales de programación necesitan de un continuo mantenimiento para adaptarse a las nuevas necesidades de los datos, los modelos de ML, basándose en su capacidad de aprendizaje son capaces de adaptarse y aprender a partir de los nuevos datos sin necesidad de ser nuevamente programados.

En ML se pueden diferenciar dos tipos de modelos en función del tipo de predicciones realizadas [10]: regresión y clasificación. Los modelos de regresión son capaces de predecir valores continuos como por ejemplo el valor de una casa. En segundo lugar, los modelos de clasificación realizan predicciones sobre un conjunto de valores discreto, como por ejemplo determinar si un correo es spam o no.

En función de si los ejemplos utilizados durante el entrenamiento contienen el resultado que se desea predecir se pueden distinguir dos tipos de entrenamiento:

supervisado y no supervisado. En el aprendizaje supervisado, el conjunto de datos con el que se entrena el modelo contiene el resultado que se quiere predecir en cada ejemplo. A este resultado se le denomina etiqueta.

En el caso opuesto se encuentra el entrenamiento no supervisado [10]. En el entrenamiento no supervisado los ejemplos no se encuentran etiquetados, es decir, cada ejemplo no contiene el resultado que se desea predecir. El objetivo principal de un entrenamiento no supervisado es el de encontrar patrones dentro de un conjunto de datos formado por ejemplos no etiquetados [10]. Por ejemplo, un algoritmo de aprendizaje no supervisado puede ser capaz de agrupar personas en función de sus preferencias musicales.

Las técnicas de entrenamiento supervisado son ampliamente utilizadas tanto en modelos de clasificación como de regresión [9]. Muchos son los algoritmos basados en aprendizaje supervisado: regresión lineal, redes neuronales, árboles de decisión, máquinas de soporte vectorial, etc.

El objetivo de los árboles de decisión es crear un modelo que predigan el valor de una variable determinada mediante el aprendizaje de reglas de decisión simples (if-then-else) inferidas a partir de las características de los datos [11]. Es decir, en función de los valores de los atributos se tomará una decisión u otra continuando por una determinada rama del árbol de decisión hasta llegar a la predicción.

Dado un conjunto de puntos representados en un espacio multidimensional y pertenecientes a diferentes clases, las máquinas de soporte vectorial buscan establecer separadores lineales, también denominados hiperplanos, que separen de forma óptima a los puntos de una clase de las otras [12]. De esta forma, en un espacio bidimensional el hiperplano será una recta que separe el espacio en dos mitades y en un espacio tridimensional se corresponderá con un plano. En este caso, el mejor separador o hiperplano será aquel cuya distancia a las diferentes clases sea máxima .

Uno de los métodos más utilizados en técnicas de ML es la regresión lineal. El concepto general de este modelo es establecer una relación lineal entre los diferentes atributos de los ejemplos y la etiqueta, es decir, el resultado que queremos predecir [10]. Este concepto es muy sencillo y aplicable en gran cantidad de casos, pero no es capaz de satisfacer y dar una solución válida a problemas de gran complejidad

cuyas relaciones no son lineales. En la Figura 2 se muestran distribuciones de datos que no pueden ser bien representadas mediante una relación lineal.

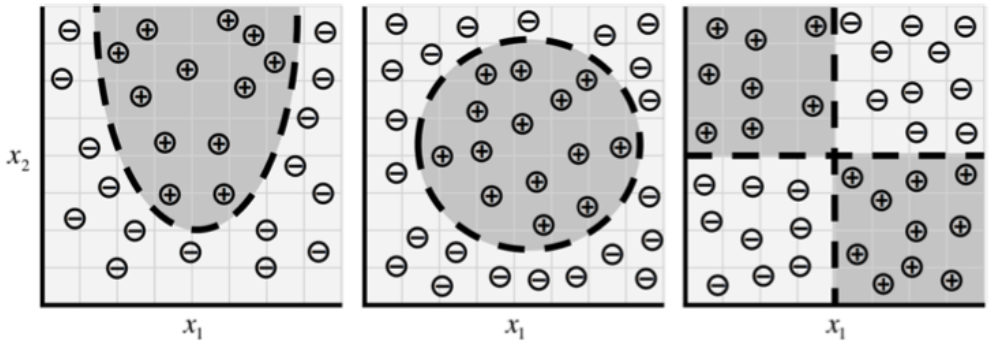


Figura 2: Distribuciones de datos complejas cuyas relaciones no son lineales y requieren de métodos más complejos, obtenida de la referencia [13]

Este tipo de casos tan complejos cuyas relaciones entre los datos no son lineales son muy comunes en problemas relacionados con el reconocimiento de imágenes y texto [7]. Como solución a este tipo de problemas surge el Deep Learning. El Deep Learning es una rama del ML basada en la utilización de redes neuronales que se asemejan a las redes de neuronas utilizadas por nuestro cerebro.

## 1.2. Objetivos y fases del proyecto

El objetivo principal de este trabajo fin de grado es la toma de contacto con los conceptos y herramientas relativos al aprendizaje automático a través de redes neuronales, haciendo uso de las mismas en un caso práctico. A lo largo del trabajo se han recorrido las diferentes etapas de un proyecto de ML, desde la definición, análisis y preparación de un dataset real, pasando por la definición de múltiples modelos predictivos hasta la evaluación de dichos modelos a través de entrenamiento e inferencia para determinar la calidad del predictor. Para ello, se ha aprendido a utilizar herramientas especialmente diseñadas para esta labor como TensorFlow [14] y otras librerías de apoyo escritas principalmente en el lenguaje de programación Python y especialmente dedicadas a la manipulación de datos (como pueden ser Pandas [15] o NumPy [16]). De manera complementaria, se ha hecho necesario adquirir cierta soltura en el uso del lenguaje de programación Python ya que ha

sido el escogido para el desarrollo y despliegue de todas las pruebas realizadas. Este lenguaje es uno de los más habituales para realización de proyectos de ML y permite trabajar de forma relativamente sencilla con la herramienta que se ha utilizado, TensorFlow.

Cronológicamente, el desarrollo del trabajo ha seguido las fases detalladas a continuación:

- La primera etapa del trabajo se ha dedicado al aprendizaje de los primeros conceptos relacionados con el ML, la instalación y toma de contacto con la herramienta TensorFlow y la realización de las primeras pruebas de concepto. Durante esta fase se ha hecho uso del dataset numérico de MNIST, orientado al entrenamiento de sistemas de reconocimiento de imágenes, más concretamente de números escritos a mano. Antes de entrar a la creación de los primeros modelos se ha estudiado la estructura y características del dataset de MNIST. Tras esto, se han creado las primeras configuraciones de redes neuronales y observado su comportamiento a través de los resultados obtenidos.
- Después de haber establecido un primer contacto con las técnicas de ML, en la segunda fase se ha procedido a trabajar con el dataset obtenido en el proceso de fabricación descrito en la sección 1. En esta fase se ha comenzado analizando, filtrando y modificando los datos de partida para realizar labores como la selección de columnas adecuadas, la eliminación de errores de medida y valores extremos, normalización de datos, etc. Esta parte relacionada con el tratamiento de los datos es de vital importancia ya que la calidad de los datos es esencial para que el modelo sea capaz de extraer información de los mismos.
- La tercera fase ha consistido en la creación y configuración del modelo. En este caso se corresponde con la creación de la red neuronal que va a ser entrenada con el dataset anteriormente preparado. Con el modelo ya creado, se han realizado múltiples pruebas de entrenamiento buscando valores de predicción apropiados.
- Finalmente, a través de diferentes medidas de evaluación, se ha comprobado el comportamiento del modelo ante datos que nunca le han sido mostrados.

## 1.3. Estructura del Documento

Las secciones del presente TFG se organizan con la siguiente estructura:

- En el Capítulo 2 se introducen los principales conceptos sobre Redes Neuronales y se describen las herramientas utilizadas para el desarrollo del TFG.
- En el Capítulo 3 se detallan las tareas realizadas para las fases de aprendizaje y toma de contacto con esta área.
- En el Capítulo 4 se detalla el proceso seguido para trabajar en un caso práctico de aprendizaje automático, describiendo el origen de los datos y la definición de objetivos, así como las múltiples pruebas realizadas.
- Se cierra el documento con el Capítulo 5, dedicado a detallar las principales conclusiones obtenidas, así como posibles líneas de trabajo futuras.

# 2. Conceptos y herramientas utilizadas

## 2.1. Desarrollo de un proyecto ML

En ML un modelo representa el conocimiento obtenido por parte del sistema a partir de los datos de entrenamiento [10]. El proceso de obtención de dicho conocimiento sigue una estructura similar en la mayoría de proyectos de ML, cuya descripción en etapas se muestra en la Figura 3.

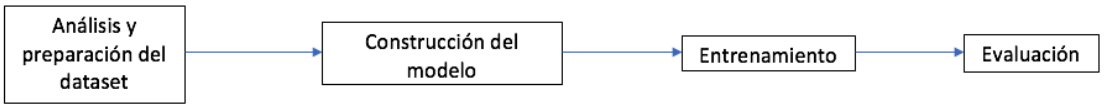


Figura 3: Ciclo vida proyecto ML.

El punto de partida consiste en todos los casos en un conjunto de datos (dataset) formado por un conjunto de ejemplos con los que será entrenado el modelo de ML. Cada uno de estos ejemplos está formado por un conjunto de atributos que se corresponden con las variables de entrada de las que se extrae el conocimiento, y en función del tipo de entrenamiento una etiqueta. Esta etiqueta es la que determina el resultado de un ejemplo (en el caso de un filtro anti-spam: "spam", "no spam"). Las primeras etapas consisten en analizar y preparar los ejemplos que componen el dataset, llevando a cabo tareas como la normalización o pre-procesados más complejos como los que se verán en secciones posteriores. Esta es considerada una de las etapas de mayor importancia debido a que la calidad de los datos tiene un gran impacto sobre el aprendizaje.

Con el dataset ya preparado se procede a la definición y construcción del modelo (regresión lineal, red neuronal, etc.). El entrenamiento del modelo definido consiste en ir alimentando el modelo con los ejemplos del dataset a través de los cuales este será capaz de extraer conocimiento. El objetivo del entrenamiento consiste en reducir la pérdida que indica en qué medida se ha equivocado el modelo al realizar una predicción sobre un ejemplo. Esto quiere decir que un modelo perfecto será aquel cuya pérdida sea igual a cero.

Una de las funciones más conocidas y utilizadas habitualmente para calcular

la pérdida es el error cuadrático medio (ECM) [10]. Esta función viene dada por la fórmula:

$$ECM = \frac{1}{N} \sum \_(x, y) \in D(y - prediccion(x))^2 \quad (1)$$

Esta función calcula el cuadrado de la diferencia entre el valor real (y) y la predicción realizada sobre un ejemplo x, midiendo la "distancia" entre el valor estimado y el resultado real para los valores de pesos actuales. Pese a ser una de las más utilizadas, no significa que sea la única función de pérdida ni la mejor para todas las circunstancias posibles.

En los modelos de ML la pérdida se reduce mediante un proceso iterativo. En cada iteración, al modelo se le alimenta con un número determinado de ejemplos escogidos del conjunto de datos denominado lote (conocido en inglés como batch). Basándose en los ejemplos de este lote el modelo realizará las correspondientes predicciones que serán propagadas a la función de pérdida junto con las etiquetas de cada ejemplo para calcular la pérdida obtenida. En la Figura 4 se muestra este enfoque iterativo para reducir la pérdida en los modelos de ML.

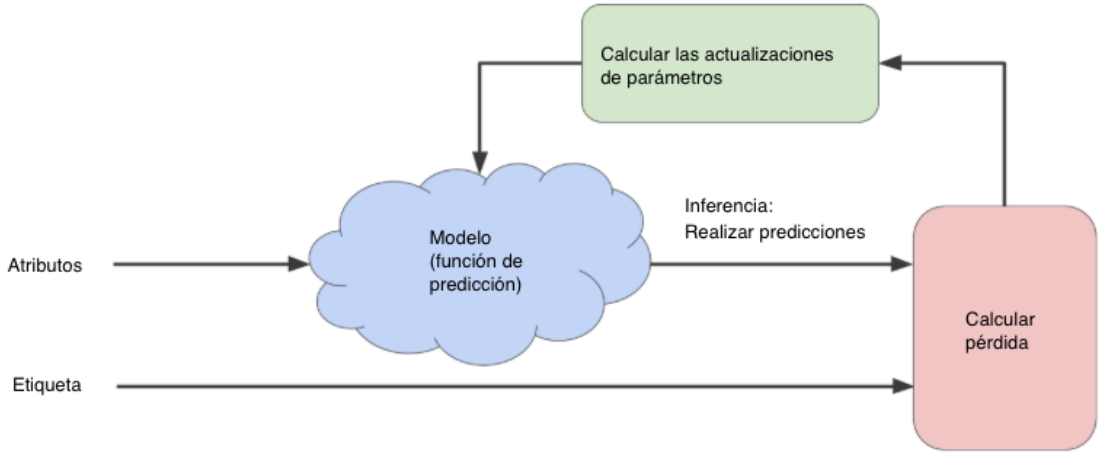


Figura 4: Ciclo iterativo para la reducción de la pérdida en proyectos de ML, obtenida de la referencia [10].

Para modificar los pesos de un modelo de ML de forma iterativa se utiliza el algoritmo del gradiente descendente basado en la derivada de la función de pérdida. El primer paso es inicializar los pesos a un valor aleatorio. En la primera iteración

del entrenamiento, y tras conocer la pérdida, se calculan las derivadas parciales de los pesos en el punto dado por la función de pérdida. Una vez calculadas todas las derivadas parciales de todos los pesos se obtiene lo que se conoce como vector gradiente.

Al tratarse de un vector, este consta de dos características: dirección y magnitud. El vector gradiente siempre apunta en la dirección en que la función crece más rápidamente. De esta manera, para conseguir reducir la pérdida y alcanzar el mínimo global de la función, los pesos son actualizados en la dirección hacia la que apunta el negativo del vector gradiente, es decir, en la dirección en que la función de pérdida decrece más rápidamente. Una vez conocida la dirección queda determinar el siguiente punto a lo largo de la función de pérdida en la que establecer el nuevo valor de los pesos. Para ello se avanza a lo largo de la función de pérdida una determinada proporción de la magnitud del vector gradiente. Esta proporción viene dada por el hiperparámetro conocido como tasa de aprendizaje. El nuevo valor que tomarán los pesos se corresponderá con la suma del anterior peso más la magnitud del vector gradiente multiplicada por la tasa de aprendizaje. Al producto de la tasa de aprendizaje por la magnitud del vector gradiente se le llama tamaño del paso. De esta forma, en cada iteración del entrenamiento se aplica el algoritmo del gradiente descendente hasta llegar a un valor de pérdida que satisfaga los requisitos [10]. En la Figura 5 se muestra un ejemplo gráfico de aplicación de este algoritmo para un modelo con un único peso.

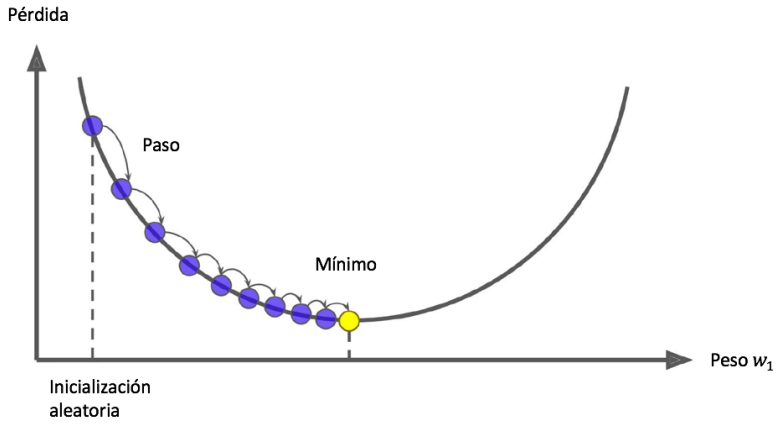


Figura 5: Gradiente descendente, obtenida de la referencia [9].

La tasa de aprendizaje es uno de los parámetros de configuración más importantes en un modelo de ML. Una tasa de aprendizaje muy pequeña origina que el tamaño del paso sea muy pequeño, por lo que el modelo tarde mucho tiempo en converger y alcanzar el mínimo global de la función de pérdida ya que los pesos apenas se habrán actualizado. En el caso contrario, si la tasa de aprendizaje es muy grande lo más probable es que se produzca un efecto rebote a lo largo de la función de pérdida, ya que el tamaño del paso es muy grande, haciendo que nunca se llegue a alcanzar el mínimo deseado [9].

En el caso de las redes neuronales para aplicar el algoritmo del gradiente descendente se utiliza otro algoritmo denominado propagación hacia atrás (conocido habitualmente en inglés como backpropagation) [17]. En redes neuronales, el error cometido por una neurona no depende únicamente de sus pesos sino que dependerá del error heredado de las neuronas de las capas anteriores. Por ello este algoritmo tiene en cuenta la contribución de cada capa en la pérdida generada. En cada paso del entrenamiento se realiza una predicción y se calcula la pérdida obtenida en la capa de salida, es decir, cuanto difiere la predicción realizada del valor real. Tras esto se recorren las capas de forma inversa, de la superior a la inferior, determinando cuanto ha contribuido cada capa a producir dicho error y modificando los pesos de las interconexiones entre las neuronas para reducir la pérdida.

Existen múltiples formas de organizar un entrenamiento. Una de las habituales consiste en dividir el conjunto de ejemplos que forman el dataset en lotes del mismo tamaño e iterar sobre todos los lotes tantas veces como se desee. Cada iteración sobre el conjunto de lotes o ejemplos del dataset se denomina época. El tamaño de los lotes tiene gran impacto sobre el entrenamiento ya que al final de cada iteración se actualizan los pesos del modelo. De esta forma, tamaños de lote más pequeños generaran un mayor número de lotes, es decir, un mayor número de iteraciones.

Tras el entrenamiento se evalúan los resultados obtenidos mediante el uso de diferentes métricas de evaluación que veremos más adelante. Estas métricas son aplicadas sobre un subconjunto del dataset inicial que nunca se le haya mostrado al modelo. Esto se realiza con la finalidad de comprobar que se ha obtenido un modelo que satisfaga los requisitos para poder ser implementado en el sistema final.

## 2.2. Redes neuronales para el Aprendizaje Máquina

A lo largo de la historia el ser humano se ha inspirado en la naturaleza y sus fenómenos para llevar a cabo gran cantidad de innovaciones. De esta forma, la arquitectura del cerebro ha servido de inspiración para intentar alcanzar el objetivo de construir máquinas inteligentes. La primera aproximación de red neuronal artificial estaba basada en la lógica proposicional e inspirada en cómo se comunican entre sí las neuronas presentes en los cerebros animales [9].

Las neuronas presentes en el cerebro humano están compuestas por un cuerpo celular que contiene al núcleo y del cual nacen diferentes ramificaciones denominadas dendritas. Aparte de las dendritas, tal y como se muestra en la Figura 6, del cuerpo celular surge una prolongación mucho más larga que éstas, conocida como axón. Al llegar al final del axón éste se divide en unas ramificaciones conocidas como terminaciones nerviosas que se conectan con las dendritas de otra neurona o directamente con el cuerpo celular. A través de esta conexión las neuronas reciben impulsos eléctricos provenientes de otras neuronas. Cuando una neurona recibe suficientes pulsos eléctricos se activa y genera su propia señal. De esta forma, el cerebro es capaz de realizar complejos cálculos mediante la interconexión de un inmenso número de estas neuronas [9].

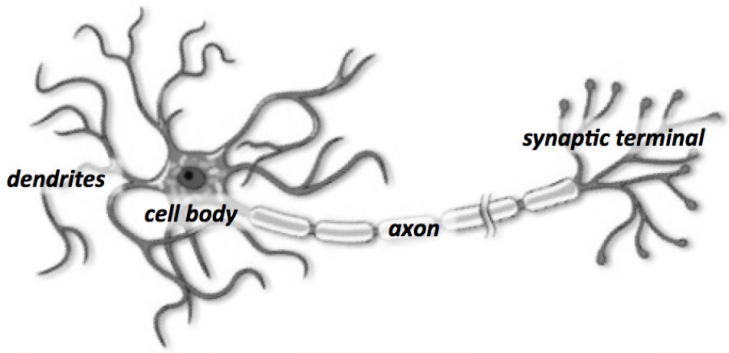


Figura 6: Neurona biológica formada por el axón, las dendritas y las terminaciones nerviosas , obtenida de la referencia [13].

Basándose en la anatomía de las neuronas presentes en nuestro cerebro, las neuronas artificiales o linear threshold units (LTUs), están compuestas por un conjunto de N entradas las cuales son multiplicadas por unos valores, denomina-

dos pesos, asignados de forma independiente a cada entrada. La LTU se encarga de realizar el sumatorio de las entradas multiplicadas por sus pesos (conocido como logits), para posteriormente aplicar una determinada función de activación en función del resultado de la suma. La salida de la LTU es el resultado de la aplicación de dicha función. Por sí sola, una única LTU puede ser utilizada para realizar clasificaciones binarias simples. La LTU computa una combinación lineal de las entradas, mostrando en la salida uno de los dos resultados posibles en función de la superación o no de un cierto valor límite [9]. La Figura 7 muestra la estructura de una de estas neuronas artificiales.

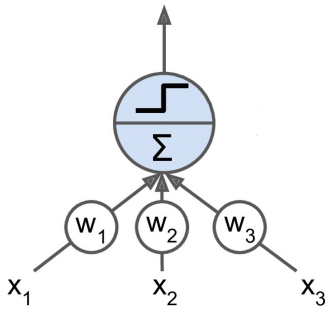


Figura 7: Linear threshold unit, obtenida de la referencia [9].

Continuando con el símil del cerebro humano, donde las neuronas se encuentran organizadas en capas, las LTUs se agrupan en múltiples capas que crean redes neuronales artificiales (Artificial Neural Networks o ANN). Las neuronas de cada capa se encargan de recoger la información de la capa anterior, procesarla y propagarla a la capa inmediatamente superior hasta llegar a la última capa, a cargo de determinar el resultado final. En su formato más básico , las capas de una ANN se pueden dividir en tres tipos: de entrada, intermedias y de salida. La capa inicial se encarga únicamente de recibir los datos de entrada y comenzar su propagación hacia la capa superior. En su camino, las capas intermedias (también denominadas capas ocultas o hidden layers), son las encargadas de realizar el procesamiento de los datos. Cada neurona que se encuentra en una de las capas ocultas tiene tantas entradas como neuronas haya en la capa inmediatamente inferior y tantas salidas como neuronas tenga la capa inmediatamente superior [9]. No existen conexiones entre neuronas de la misma capa o conexiones entre neuronas de una capa superior a una capa inferior. A este tipo de redes se las denomina feed-forward networks [9]

. Cada entrada de cada LTU individual dentro de la ANN tiene un peso asignado, valor que representará la capacidad de la red para resolver un determinado problema. Esto quiere decir que se deberá encontrar un conjunto de valores óptimo para estos pesos que den solución al problema planteado. En la Figura 8 se muestra la estructura hasta ahora descrita.

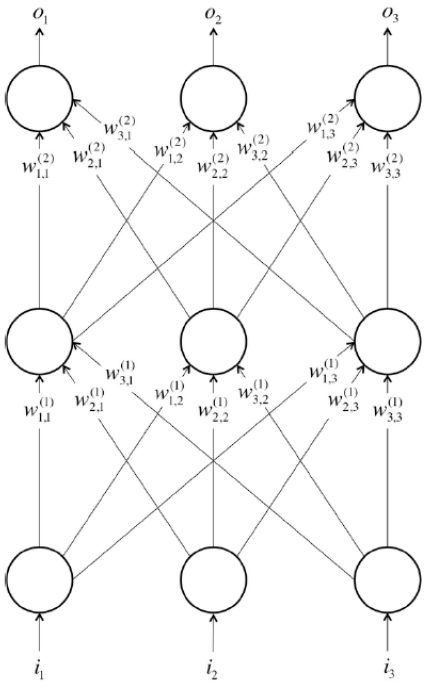


Figura 8: Estructura de una ANN, obtenida de la referencia [13].

### 2.2.1. Funciones de Activación

Las neuronas de una red pueden ser clasificadas según su función de activación. Como hemos comentado anteriormente, se requieren funciones no lineales capaces de dar solución a distribuciones de datos complejas cuyas relaciones no son lineales. Concretamente las funciones de activación más utilizadas son: Sigmoid, Tanh y ReLU.

La función *sigmoid* viene dada por:

$$f(z) = \frac{1}{1 + e^{-z}} \tag{2}$$

donde z es la suma de las entradas multiplicadas por sus correspondientes pesos. Se trata de una función continua y diferenciable que devuelve un valor próximo a 1 cuanto mayor sea z y 0 cuando el valor sea muy pequeño. De forma más generalizada e intuitiva, la función mapea la entrada a una probabilidad comprendida entre 0 y 1 [10].

La función tangente hiperbólica (3), al igual que la función sigmoid, es continua y diferenciable. En este caso devuelve valores entre -1 y 1 lo que ayuda a normalizar las salidas de cada capa en valores entorno al 0 lo que puede ayudar a la convergencia del aprendizaje [13].

$$f(z) = tanh(z) \tag{3}$$

La última de estas tres funciones, la función ReLU (4), es una función continua, pero, al contrario que las anteriores, no diferenciable en z=0 lo que hace que el algoritmo del gradiente descendiente (basado en la derivada) pueda dar lugar a error [9]. Se trata de una función muy utilizada y que, debido a su sencillez, tiene un bajo tiempo de cómputo. En la Figura 9 se puede ver una representación gráfica de las tres funciones descritas hasta el momento.

$$ReLU(z) = max(0, z) \tag{4}$$

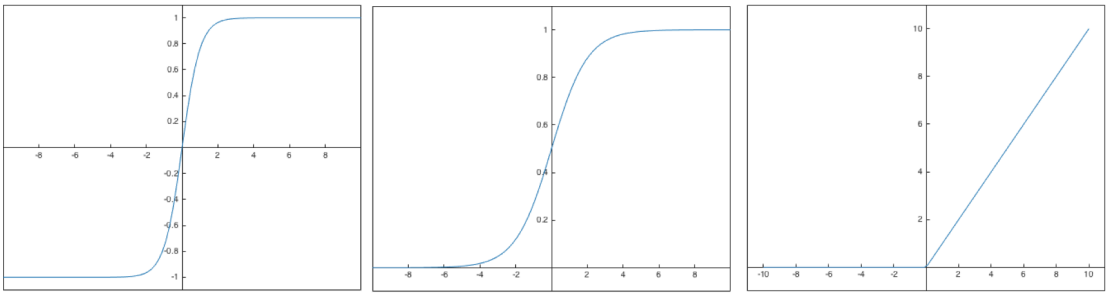

Figura 9: Comenzando desde la izquierda, representación de las funciones de activación *sigmoid*, *tangente hiperbólica* y *ReLU*.

Uno de los problemas más habituales que se abordan con redes neuronales son los de clasificación. En los problemas cuyas clases son exclusivas, es decir, si

un ejemplo pertenece a la clase A no puede pertenecer a ninguna de las otras clases, se emplea un tipo de función de activación denominada softmax [18]. Esta función se encuentra presente en las neuronas que forman la capa de salida. Cada neurona de la capa de salida corresponde a una de las clases, por lo que la capa de salida tiene tantas neuronas como clases haya, y retorna la probabilidad de que el ejemplo corresponda a dicha clase. De esta forma, en la salida se obtiene un vector de probabilidades cuya suma siempre es igual a uno. Una buena predicción será aquella cuyo vector de probabilidades tenga un elemento cercano al 1 y el resto cercanos al 0. Por el contrario, una mala predicción será aquella cuya probabilidad se encuentra distribuida por igual entre todos los elementos del vector.

## 2.3. TensorFlow

Tensorflow es una librería de código abierto creada para el cálculo numérico de alto rendimiento que permite la creación de modelos de ML. Desarrollada por Google, permite de forma sencilla llevar a cabo proyectos sobre distintas plataformas de cálculo (CPUs, GPUs, TPUs) con diferentes capacidades computacionales [14].

En Tensorflow, la representación y realización de los diferentes cálculos se realiza a través de grafos. Un grafo en Tensorflow está formado por una red de nodos, cada uno de los cuales representa una operación, conectados entre sí por sus respectivas entradas y salidas [9].

El nombre de TensorFlow proviene del término tensor que es el objeto principal usado por el framework de TensorFlow como unidad de cálculo. Un tensor es un array multidimensional utilizado por TensorFlow para la entrada y salida de datos. Cada tensor tiene un tipo de dato y dimensiones definidas [14]. De esta forma, cada elemento del tensor tiene el mismo tipo de dato que debe ser previamente definido (punto flotante, entero, etc.). La forma del tensor no tiene por qué ser conocida, sino que puede ser inferida en tiempo de ejecución. Según el número de dimensiones un tensor puede ser un simple valor escalar, un vector, una matriz, etc.

Los programas basados en TensorFlow funcionan mediante la construcción de un grafo en el que se indica cómo se calcula cada tensor basándose en el resto de

tensores y ejecutando dicho grafo para la obtención del resultado. Al proceso de ejecución del grafo para obtener los valores asignados a los tensores se le denomina evaluación.

Una de las características más importantes de TensorFlow es la capacidad de dividir el grafo de ejecución para poder ejecutarlo sobre diferentes arquitecturas en paralelo (CPUs o GPUs). Por otra parte, soporta ejecución distribuida, esto permite ejecutar modelos muy complejos con enormes conjuntos de entrenamiento en un periodo razonable de tiempo dividiendo su ejecución en múltiples servidores.

A la hora de crear un modelo, Tensorflow ofrece diferentes niveles de abstracción (representados en la Figura 10) a los usuarios para poder realizar sus desarrollos [10]. De esta forma, a medida que se desciende en la jerarquía de niveles se gana en precisión y control sobre el modelo, pero aumenta la dificultad a la hora de implementarlo. Una de las ventajas de esta organización jerárquica es que cada nivel se encuentra desarrollado con las APIs de los niveles inferiores lo que hace que descender en la jerarquía sea algo razonablemente más sencillo.

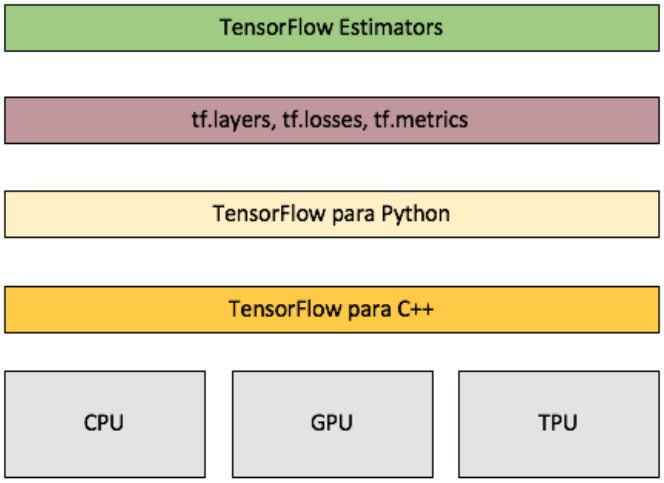


Figura 10: Niveles abstracción Tensorflow obtenidos de la referencia [10].

En el nivel más alto de la jerarquía (tf.estimator) se pueden encontrar modelos predefinidos, como regresiones lineales y redes neuronales, que se encuentran totalmente parametrizados y únicamente requieren del ajuste de sus parámetros. Un ejemplo muy utilizado de modelo predefinido es el tf.estimator.DNNClassifier.

Este clasificador únicamente necesita recibir como parámetros el número de capas que forman la red y el número de neuronas por capa para generar un modelo completamente funcional.

Sin embargo, para este proyecto se ha utilizado el segundo nivel de la jerarquía conformado por librerías que representan componentes específicos que forman los modelos como, métricas de rendimiento, funciones de pérdidas, capas, etc. El uso de este conjunto de librerías nos permite un control más preciso sobre el desarrollo del modelo que usando la API de más alto nivel.

La ejecución de un programa en TensorFlow consta de dos fases. En la primera fase se realiza la construcción del grafo (fase de construcción) que representa el modelo de ML y las operaciones necesarias para entrenarlo. La segunda parte se corresponde con el entrenamiento del modelo mediante la ejecución de un bucle que en cada iteración lleva a cabo un paso del entrenamiento (fase de ejecución).

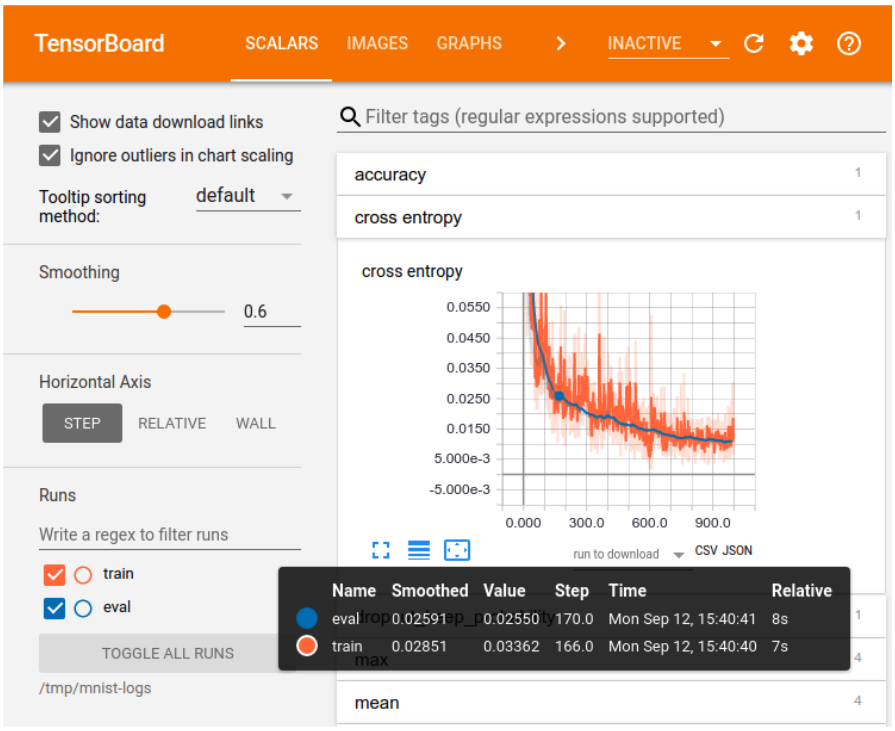


Figura 11: Visualización de métricas en TensorBoard, obtenida de la referencia [10].

El seguimiento y configuración de los modelos puede llegar a ser una ardua tarea debido a la complejidad de los mismos (por ejemplo, el entrenamiento de redes neuronales muy extensas). Por ello, TensorFlow ofrece una herramienta de visualización que facilita las tareas de optimización y corrección de errores de los programas escritos con TensorFlow, denominada TensorBoard [14].Esta utilidad permite visualizar distintas métricas de rendimiento y evaluación, tal y como se muestra en la Figura 11, que pueden ser actualizadas en tiempo de ejecución para observar el transcurso del entrenamiento. Este recurso ayuda a los desarrolladores a entender el comportamiento que está teniendo el modelo durante el aprendizaje. También permite mostrar el grafo de ejecución o mostrar cualquier tipo de dato que se quiera visualizar y pueda ayudar en el desarrollo.

## 2.4. Dataset MNIST

TensorFlow ofrece un conjunto de modelos, ubicados en su repositorio de GitHub [19], que utilizan las APIs de alto nivel de TensorFlow. Estos modelos ya construidos se caracterizan por estar muy optimizados a la vez que son fáciles de entender lo que los hacen ideales para usuarios que quieran llevar a cabo sus primeras pruebas en TensorFlow. A estos modelos se les denomina "oficiales" y reciben soporte por parte de TensorFlow. Esto quiere decir que los modelos disponen de las últimas actualizaciones estables de las APIs de TensorFlow.

Aparte de estos modelos oficiales se pueden encontrar otras colecciones como los denominados "modelos de investigación". Se tratan de modelos implementados en TensorFlow pero que no reciben soporte oficial por su parte ya que son desarrollados y mantenidos por investigadores [19].

Uno de estos modelos oficiales que ofrece TensorFlow es el basado en el dataset de MNIST (Modified National Institute of Standards and Technology) [20]. Este conjunto de datos se trata a su vez de un subconjunto del dataset de NIST (National Institute of Standards and Technology) el cuál está compuesto por 70,000 ejemplos de números escritos a mano, como los mostrados en le Figura 12. Originalmente, en el dataset de NIST las imágenes están representadas por una matriz de 20x20 píxeles. Inicialmente, en estas imágenes solo se podían distinguir los colores blanco y negro. Tras un proceso de normalización, en las imágenes están represen-

tadas por una matriz de 28x28 píxeles en los que se pueden distinguir tonalidades grises [21].

Figura 12: Ejemplos del dataset de MNIST, obtenida de la referencia [21].

De esta forma, cada ejemplo del conjunto de datos tendrá un atributo por píxel que compone la imagen, es decir, 784 atributos (matriz 28x28 píxeles) y una etiqueta que determina el número que se encuentra representado en la imagen.

Como se ha mencionado, MNIST tiene su origen en el conjunto de datos de NIST. En este segundo conjunto se pueden distinguir dos fuentes diferentes de las cuales se obtuvieron los dígitos escritos a mano. Una primera fuente proviene de un grupo de empleados de las oficinas del censo en Estados Unidos. La otra parte de ejemplos fueron recogidos de estudiantes de instituto. Estudiando los datos recogidos se observó que los dígitos recogidos de los funcionarios eran mucho más claros y fáciles de interpretar para el computador que los escritos por estudiantes [20]. Para que un proceso de aprendizaje sea exitoso se requiere que los resultados obtenidos sean independientes del conjunto de entrenamiento y prueba escogidos de entre todo el conjunto de datos [20]. Por ello surge MNIST, cuyo conjunto de entrenamiento está compuesto por 30,000 ejemplos de números escritos por el grupo de funcionarios y otros 30,000 ejemplos escritos por estudiantes. De forma similar, el conjunto de prueba consta de 5,000 ejemplos provenientes de cada fuente lo que suman un total de 10,000 ejemplos destinados al conjunto de prueba. Este conjunto de datos se encuentra disponible para descargar en la página web de MNIST [20].

# 3. Primeros pasos con TensorFlow

## 3.1. Instalación: versión pre-compilada vs. compilación local

El proceso de instalación de TensorFlow no es trivial y requiere tener en cuenta las necesidades a cubrir y los recursos de los que se disponen. En una primera instancia se pueden diferenciar dos tipos de instalaciones: pre-compilada y compilación local. A su vez, en ambas instalaciones TensorFlow ofrece la posibilidad de realizar una instalación especial con soporte para ejecutar los desarrollos realizados sobre unidades de procesamiento gráfico (GPUs) de la compañía NVIDIA. En el presente trabajo todas las pruebas realizadas han sido ejecutadas únicamente sobre la CPU, es decir, sin esta última opción.

La principal ventaja de la instalación pre-compilada es su sencillez y las diferentes opciones que ofrece para llevarla a cabo. En contra, al tratarse de una instalación pre-compilada que no tiene en cuenta la arquitectura de la máquina sobre la que se instala a la hora de la compilación, no contiene mucha de las extensiones que hacen que mejore su rendimiento y que sí están presentes en las instalaciones compiladas de forma local.

Una de estas posibilidades que ofrece TensorFlow es mediante la utilización del instalador de paquetes de Python pip [22]. Esta instalación se puede realizar de dos formas. Una de ellas consiste en la creación de un entorno virtual de Python (Python virtualenv) que aísla el entorno de trabajo del resto del sistema pudiendo tener diferentes entornos virtuales con diferentes versiones de los paquetes instalados sin que estos se interfieran. Otra forma es utilizar pip para realizar una instalación directa sobre el sistema sin ninguna clase de mecanismo que la aísle.

Otra forma que ofrece TensorFlow de llevar a cabo una instalación pre-compilada es mediante la utilización de contenedores. En concreto contenedores Docker. Docker es un proyecto de código abierto para la automatización de la instalación de paquetes software, denominados contenedores, abstrayéndose del sistema operativo que haya debajo [23]. De esta forma, TensorFlow ofrece un contenedor de Docker con el framework de TensorFlow ya instalado junto a todas las dependen-

cias necesarias para que este funcione correctamente. Con este método se consigue un efecto similar al conseguido mediante la creación de entornos virtuales aislando a la instalación del resto de paquetes instalados en el sistema.

La otra alternativa a la de utilizar binarios ya pre-compilados es la de compilar el framework de TensorFlow de forma local. Esta es la opción que se debe elegir si se busca instalar TensorFlow y obtener su mejor rendimiento. De esta forma, además de poder hacer uso de los conjuntos de instrucciones más recientes, si se dispone de un procesador Intel es posible utilizar la librería de Intel Math Kernel Library for Deep Neural Networks (Intel MKL-DNN) la cual optimiza un conjunto de operaciones usadas en procesos de Deep learning [14].

## 3.2. Evaluación del dataset MNIST

Como primera toma de contacto con la creación y entrenamiento de modelos de ML a través del framework de TensorFlow, se utilizará el conjunto de datos numéricos de MNIST. Este dataset se puede encontrar disponible en su página web [20] junto con multitud de referencias a pruebas realizadas con este en las que se pueden observar el resultado que se consiguió obtener en cada prueba y las características del modelo utilizado, así como de su proceso de entrenamiento.

En este caso, nos basaremos en un conjunto de pruebas realizadas por Yann LeCun, Léon Botton, Yoshua Bengio y Patrick Haffner [21]. De entre todas las pruebas presentes en dicho documento, se busca alcanzar la tasa de error obtenida para la prueba realizada con un modelo de dos capas en el que la primera capa se trata de una capa oculta de 300 neuronas y la segunda se corresponde con la capa de salida y consta de tantas neuronas como clases a predecir, en el caso del MNIST los 10 posibles números a inferir. En este experimento logran alcanzar una tasa de error de 4,7 %.

En primer lugar, se obtienen los datos con los que va a ser entrenado y validado el modelo de la página web de MNIST. Una vez que se tienen los datos, se aplica un proceso de normalización basado en los valores máximos y mínimos de cada atributo consiguiendo reducir el rango de valores de cada atributo al intervalo [0,1]. Este proceso ayuda al modelo a reducir la pérdida más rápidamente y a

evitar otra serie de posibles inconvenientes los cuales se entrarán a estudiar en la siguiente sección. En principio, no es necesaria la limpieza del conjunto de datos de cualquier atributo o ejemplo que pudieran dar lugar a error durante la fase de entrenamiento del modelo o que no pudieran servir. Esto se debe a que el conjunto de datos de MNIST viene previamente depurado de este tipo de inconvenientes. De no ser así, esta es una labor esencial que debe hacerse antes de llevar a cabo cualquier proceso de entrenamiento en ML.

Una vez se tienen lo datos preparados se construye una red neuronal con las características descritas anteriormente y se procede a su entrenamiento. Como primera prueba se establece una tasa de aprendizaje de 0,1 para ver cómo se comporta la pérdida y se realiza un entrenamiento de una época con lotes de 50 ejemplos. La función de pérdida utilizada es el ECM presentado anteriormente.

Como medida de evaluación del entrenamiento utilizaremos la tasa de error o mejor dicho, su contrario, la exactitud. La exactitud determina la proporción de predicciones correctas sobre el total de predicciones realizadas. Más adelante se entra a detallar más en profundidad el significado de esta medida y sus implicaciones. Tras aplicar esta medida sobre el conjunto de validación se obtiene una exactitud de en torno al 93,4 % acercándose bastante a nuestro objetivo de 95,3 %. La Figura 13 muestra la evolución de la exactitud medida sobre el conjunto de validación a lo largo del entrenamiento cada vez que se finaliza un paso del aprendizaje.

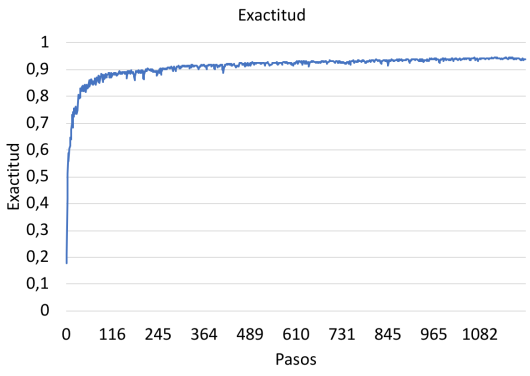


Figura 13: Evolución de la medida de la exactitud medida sobre el conjunto de validación a lo largo del entrenamiento.

A pesar de esto, es conveniente estudiar si existe la posibilidad de que hubiera cierto margen de mejora que nos permitiera alcanzar el objetivo buscado, ampliando el entrenamiento o aumentando la tasa de aprendizaje. En este caso, y debido a que la tasa de aprendizaje establecida es bastante razonable, se opta por aumentar la duración del entrenamiento a dos épocas. En esta segunda prueba se puede ver como al ampliar la duración del entrenamiento se ha conseguido aumentar la exactitud y alcanzar así el objetivo con una exactitud del 95,5 %, es decir, una tasa de error del 4,5 %.

Por último, para comprobar las mejoras de rendimiento introducidas por la versión compilada localmente del framework de TensorFlow, se realizan tres pruebas con las configuraciones usadas en la última prueba pero con los tamaños y dimensiones de red mostrados en la Tabla 1. Para realizar la comparación se mide el tiempo que se tarda en llevar a cabo el entrenamiento para una instalación precompilada y una compilada de forma local, concretamente una instalación llevada a cabo mediante el instalador de paquetes de Python *pip*.

| Capas ocultas | Neuronas por capa |
|---|---|
| 1 | 300 |
| 2 | 300-100 |
| 2 | 500-150 |

Tabla 1: Dimensiones de las redes neuronales utilizadas.

Ambas pruebas realizadas se ha llevado con la versión de TensorFlow con soporte exclusivamente para la ejecución sobre la CPU. En concreto se ha utilizado un procesador modelo Intel(R) Core(TM) i5-5257U CPU @ 2.70GHz.

En la Figura 14 se puede observar como con la versión compilada se consigue una reducción del tiempo de entrenamiento de hasta un 7.92 % para el caso de la red neuronal más compleja. En un principio puede parecer que la ganancia obtenida no es muy grande debido a que la reducción conseguida es de unos pocos segundos. En realidad, los procesos de entrenamiento aplicados a problemas reales son procesos que emplean mucho tiempo de ejecución y en los que una reducción de un 8 % supone una ganancia sustancial.

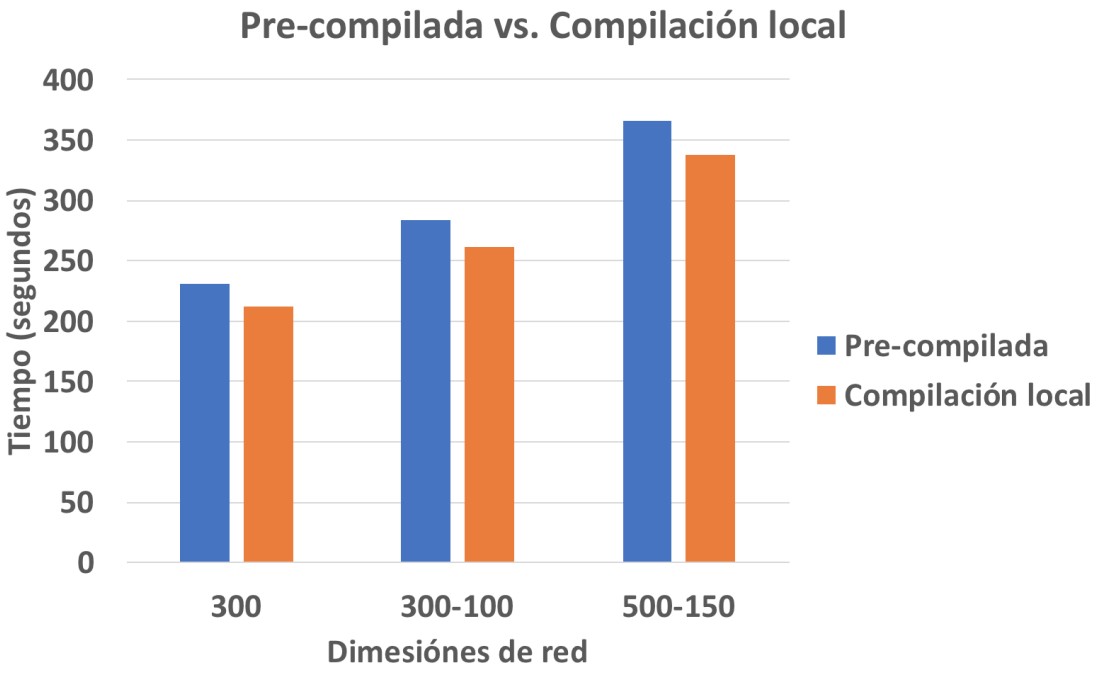


Figura 14: Comparación de los tiempos de ejecución de una instalación pre-compilada contra una compilada localmente.

A partir de ahora, por simplicidad, todas las ejecuciones y pruebas realizadas han sido llevadas a cabo sobre una instalación pre-compilada. En este caso se ha utilizado el gestor de paquetes de Python *pip*.

# 4. Análisis de un dataset real

## 4.1. Dataset: Origen de datos y formato inicial

Los datos sobre el proceso de fabricación a analizar se proporcionan a través de ficheros Excel, con un formato de matriz de datos. Se facilita un fichero por cada mes de producción, con dos hojas de datos distintas. Una de las hojas (etiquetada como SUMMARY), contiene un resumen de los datos mensuales, con valores como el volumen de fabricación y el porcentaje de piezas correctas e incorrectas. La hoja adicional contiene la matriz de datos de fabricación, donde cada fila corresponde con una pieza fabricada mientras que las columnas contienen información sobre diferentes aspectos del proceso de fabricación, con un total de 72 columnas de datos para cada pieza. Dicha información puede ser clasificada en tres tipos distintos, proporcionando a continuación una descripción de cada tipo, así como algunas columnas de ejemplo del dataset.

### 4.1.1. Tipo 1: Información

Se trata de columnas que proporcionan información identificativa sobre la pieza, así como sobre cómo se ha desarrollado el proceso de fabricación. Algunos valores significativos son:

- SER_NO (Serial Number): número de serie de la pieza (identificador único).
- Date: fecha de fabricación, con una precisión de segundo (hora de terminación de la pieza).
- PALLET_NO (Pallet Number): Identificador numérico del pallet que transporta la pieza por la línea de producción.
- MATNR: Identificador de la variante de la pieza que se está fabricando (42 variantes distintas en total).
- TURN_COUNTS: En presencia de algunos fallos, a la pieza se le permite pasar más de una vez por la línea de producción.

- ATS_ERROR_CODE: Código de error y descripción textual del mismo. El valor numérico del código proporciona información como el tipo de fallo ó la etapa de fabricación en la que se detectó.

### 4.1.2. Tipo 2: Medida

Estas columnas son las que realizan medidas de parámetros físicos relativos al proceso de fabricación. Son las más numerosas y miden desde valores ambientales (temperatura y humedad de la sala en el momento de fabricación) hasta procesos tan simples como atornillados (par fuerza + profundidad). La identificación numérica de cada medida proporciona información sobre su posición en la línea de fabricación (un valor numérico bajo implica que se trata de medidas en las primeras etapas). Algunos valores de ejemplo:

- ST256 (2 columnas): Estación 256, si hay dos columnas con la misma etiqueta, implica que se trata de un proceso por parejas, como por ejemplo un atornillado (Fuerza+stroke, fuerza+torque, etc.).
- ST540: Estanqueidad, mide posibles fugas de gas, tanto internas como externas, cuando la válvula se encuentra en posición "apagado".

### 4.1.3. Tipo 3: Test

Al finalizar el proceso de ensamblado del producto, las últimas etapas del proceso de montaje se dedican a realizar múltiples tests para comprobar el funcionamiento de la pieza, realizando medidas en cada uno de ellos (en función del resultado de dichas medidas se determina si la pieza es correcta, dando el valor apropiado a la columna ATS_ERROR_CODE). Algunos ejemplos del tipo de pruebas realizadas son:

- ST530: Caudal de gas mínimo, que indica qué caudal de gas debería proporcionar la válvula en la posición 1.
- ST345 (6 columnas): Con más de dos columnas con la misma etiqueta, múltiples operaciones se realizan en la misma etapa (varios tornillos simultáneos).

## 4.2. Pre-procesado de datos, "limpieza" y normalización

La fase de análisis y preparación de los datos es una de las fases que más importancia tiene y que más tiempo requiere por parte de los desarrolladores. De la calidad de los datos depende que el modelo sea capaz de alcanzar una solución óptima. En este caso, para la manipulación de los datos se ha elegido una librería de código abierto y escrita en Python denominada Pandas [15]. Esta librería provee de un conjunto de estructuras de datos y funciones de análisis específicas para llevar a cabo este tipo de tareas.

El conjunto de datos está formado por ejemplos de piezas fabricadas entre los años 2016 y 2018. El primer paso ha consistido en la lectura de todas las filas de los múltiples ficheros .xls y su posterior introducción en una estructura de datos, propia de la librería Pandas, que facilita el trabajo con ellos. A esta estructura se la conoce como DataFrame. Un DataFrame posee estructura de tabla correspondiendo cada fila a un ejemplo del dataset y las columnas a los atributos de los ejemplos. No es posible trabajar de forma directa con el DataFrame generado incialmente, pues los datos de origen presentan diversas anomalías que requieren ser corregidas antes de servir de dataset de entrenamiento para nuestro modelo de ML. Esta sección describirá todo este proceso de "limpieza", siendo una de las tareas del TFG su automatización a través de tareas de scripting desarrolladas en Python. El código desarrollado para esta labor se puede consultar en el repositorio de GitHub de la referencia [24] bajo el nombre "*dataset.py*".

Para determinar si una pieza ha sido fabricada correctamente o no, se hace uso del atributo ATS_ERROR_CODE. Este atributo muestra un 0 en caso de que la pieza haya sido correctamente fabricada o un código numérico de error cuyos tres dígitos más significativos se corresponden a la etapa en la que se ha producido el error. Esto quiere decir que el ATS_ERROR_CODE hace las funciones de etiqueta permitiéndonos discernir si una pieza ha sido fabricada correctamente o no en función de su valor.

La línea de producción incorpora alguna prueba de test en etapas intermedias del proceso de fabricación. En caso de detectarse un error en una de las etapas intermedias del proceso de fabricación, el resto de etapas posteriores no se llevan a cabo, por lo que los atributos correspondientes a las medidas de esas etapas no rea-

lizadas tendrán como valor 0. Este tipo de errores constituyen una fracción mínima (menor del 5 %) del total de piezas defectuosas, siendo su mayor parte detectadas en las pruebas de test finales. Por esta razón se ha decidido filtrar y descartar estos casos del conjunto de entrenamiento ya que se requiere de ejemplos erróneos que tengan todas las medidas para que el modelo sea capaz de extraer relaciones entre ellas y la existencia de una pieza defectuosa. Para ello se añaden al conjunto de entrenamiento únicamente aquellos ejemplos cuyo ATS_ERROR_CODE es igual a 0 (pieza correcta) o aquellos ejemplos cuyos tres primeros dígitos del código de error se corresponden con etapas de fabricación finales, es decir, la pieza ha completado el proceso de fabricación y el error ha sido detectado en las etapas finales de test.

Una vez realizado el filtrado de piezas erróneas en etapas intermedias, los diferentes códigos de error no revisten de utilidad más que para indicar que la pieza no es válida. Por ello, y debido a que el modelo de predicciones se basa en una clasificación binaria (pieza válida o no válida), se mantiene el valor del atributo ATS_ERROR_CODE a 0 para las piezas correctas y los códigos de error en las incorrectas se sustituyen por 1. De esta forma el modelo observará durante el entrenamiento que una pieza es correcta cuando en el ATS_ERROR_CODE haya un 0 o incorrecta cuando haya un 1. De igual manera realizará las predicciones en base a estos dos valores.

Dentro de la pieza que se fabrica existen hasta 42 modelos diferentes denominados "variantes" cuyas características y atributos difieren entre sí por lo que no es posible entrenar el modelo con diferentes variantes. Por ello, es necesario filtrar el conjunto de ejemplos según la variante que queramos estudiar. Este filtrado se lleva a cabo mediante el atributo MATN que se trata de un código numérico que identifica a la variante. En este caso la variante escogida ha sido la 9001073660 ya que es la variante con mayor número de ejemplos, lo que permite crear un conjunto de datos de entrenamiento más amplio.

Durante el proceso de fabricación, en presencia de algunos fallos, las piezas pueden ser re-testeadas. Esto implica que en el conjunto de datos se den ejemplos con identificadores de piezas repetidos con los problemas que esto conlleva. Estas piezas que han vuelto a pasar por la cadena de fabricación pueden ser detectadas mediante el atributo numérico TURN_COUNTS que determina las veces que ha pasado la pieza por la cadena de montaje. De esta forma se eliminan aquellos ejemplos que

han sido re-testeados, es decir, aquellos ejemplos cuyo valor de TURN_COUNTS sea distinto de 1. Una vez eliminados dichos ejemplos este atributo es eliminado del dataset ya que no reviste de relevancia para el entrenamiento.

Tras el proceso de limpieza llevado a cabo se asumió que los datos que contienen los ejemplos restantes son correctos. Tras varias pruebas y algunas medidas, se observó que esta suposición es incorrecta, ya que el conjunto de datos todavía contenía ejemplos erróneos por diferentes motivos: ejemplos mal etiquetados, valores de atributos incoherentes, valores omitidos o ejemplos duplicados. Para detectar este tipo de errores se han utilizado diferentes tipos de medidas como son la desviación, máximos y mínimos de cada atributo o la media y mediana. Por ejemplo, la desviación nos permite conocer cuánto se alejan los datos con respecto a su valor promedio. Otra medida útil es la de los valores máximos y mínimos de cada atributo. Esto permite detectar valores atípicos que se encuentren fuera del intervalo de valores que usualmente toma un atributo.

Para realizar este análisis la librería *Pandas* ofrece mediante el método "describe()" un conjunto de medidas, como las comentadas anteriormente, aplicables sobre la propia estructura de datos de *Pandas*, *DataFrame*. Estas ayudas, junto con un conocimiento detallado sobre el conjunto de datos, son vitales para detectar y eliminar ejemplos con este tipo de errores.

No siempre todos los atributos que contienen los ejemplos del dataset son útiles a la hora de entrenar un modelo. Por ello se eliminan aquellos atributos que no se consideran útiles evitando que el modelo pueda inferir relaciones erróneas a partir de ellos o que simplemente no revisten de interés para el proyecto que se desea llevar a cabo. Ejemplo de estos atributos pueden ser la fecha de fabricación de la pieza guardada en el atributo "Date" o la descripción textual del error mostrada en el atributo "Description".

Antes de llevar a cabo el proceso de análisis y preparación de los datos se disponía de un total de 1.109.250 ejemplos. Finalmente, tras haber realizado dicho proceso esta cantidad se ha reducido a 261.240 ejemplos, es decir, para una primera prueba se han descartado el 76,45 % de los datos siendo la mayor parte de la reducción producida por la selección de una única variante de pieza tal y como se ha explicado anteriormente. No solo se ha reducido el número de ejemplos de

forma considerable, sino que también se ha visto reducido el número de atributos. Inicialmente cada ejemplo constaba de 72 atributos. Tras esta fase cada ejemplo consta únicamente de 20 atributos más el que determina la validez de la pieza, es decir, la etiqueta. Esto pone en evidencia la importancia y necesidad de llevar a cabo este tipo de procesos no solo para eliminar aquellos ejemplos o atributos que puedan contener errores, sino que también para descartar aquellos que no sean de utilidad para entrenar o probar el modelo.

El proceso de filtrado proporciona un conjunto de datos uniforme y sin resultados anómalos que facilite su utilización como dataset de entrenamiento. Sin embargo, dichos datos presentan todavía valores muy dispares entre columnas, dado que pertenecen a unidades sin relación alguna. Por ejemplo, las medidas de profundidad de atornillado se hacen en milímetros con órdenes de magnitud entre 0,01 y 0,1, mientras que los giros se miden en grados y superan en algunos casos la centena. Esta disparidad influye de manera notable en la convergencia del modelo durante la fase de entrenamiento. Cuando la red neuronal intenta reducir la pérdida obtenida tras una predicción, ésta actualiza los pesos de forma proporcional a los valores de entrada. De esta forma, basándonos el el algoritmo del descenso del gradiente presentado anteriormente,si los atributos de entrada presentan valores muy grandes, los pesos sufrirán cambios muy bruscos lo que no ayuda a la convergencia. El proceso de normalización transforma el rango de valores que puede tomar un atributo a un rango de valores determinado (por ejemplo, de 0 a 1). Normalmente los modelos tienden a converger (reducir el valor de la pérdida producida durante el entrenamiento a un mínimo global) cuando la media de los valores de cada entrada del conjunto de entrenamiento es cercana a cero [17].

Una de las fórmulas de normalización más utilizadas es el conocido como "escalado de atributos" o "escalado de máximos y mínimos " (5). Esta función, tal y como indica su nombre, se basa en los valores experimentales máximo y mínimo de los atributos para realizar una reasignación al intervalo de valores [0,1]. De esta forma, el valor mínimo de un atributo tomará el valor 0 y análogamente el valor más alto de un atributo tomará el 1.

$$X\_normalizado = \frac{X - X\_min}{X\_max - X\_min} \tag{5}$$

El método más rápido para conocer el rango de valores de cada atributo es obtener directamente los valores máximo y mínimo de los mismos. Para ello la librería Pandas ofrece dos funciones, aplicables sobre la estructura de datos DataFrame: min() y max(). De tal forma, se puede observar como el dataset está compuesto por atributos con rangos de valores muy dispares. Esto puede ocasionar que aquellos atributos cuyo rango de valores sea mucho mayor al resto tengan mayor impacto en el entrenamiento del modelo que aquellos atributos con rangos más pequeños.

Mediante la utilización de esta técnica de normalización, aplicada a los atributos de entrada de la red neuronal, se consigue disminuir el impacto que tienen los valores extremos o atípicos sobre el resto de valores. Por otra parte, se logra que el rango de valores de todos los atributos sea el mismo, lo que facilita la realización de cálculos y operaciones durante entrenamiento. Tras la aplicación de la fórmula de normalización todos los valores se encuentran en el intervalo [0,1].

## 4.3. Primera definición del modelo y evaluación

En una primera instancia, para comprobar el correcto funcionamiento del modelo, se genera una red neuronal con dos capas ocultas con 128 y 32 neuronas respectivamente. Por otra parte, se establece un entrenamiento de una única época dividida en lotes de 100 ejemplos, es decir, se recorre una única vez todos los ejemplos del dataset dividiendo los ejemplos en grupos de 100.

Aparte de las dos capas ocultas de 128 y 32 neuronas, se establece una capa más de salida formada por dos neuronas que implementan la función de activación softmax presentada anteriormente. Recordando lo comentado con anterioridad, la función de activación softmax genera un vector de probabilidades con tantos elementos como clases tenga el problema. En nuestro caso, dos clases: pieza válida y pieza no válida. Como función de pérdida, en lugar de utilizar el ECM, utilizaremos la conocida como entropía cruzada (conocida en inglés como cross entropy loss). Esta función es especialmente utilizada cuando la capa de salida está formada por neuronas con funciones de activación del tipo softmax. Esta función calcula la diferencia entre lo que el modelo predice que debe ser la distribución de salida, es decir, la calculada por la función de activación softmax y la que realmente debe

ser [18].

Idealmente se desea establecer una tasa de aprendizaje lo suficientemente grande como para que el modelo converja rápidamente pero que no haga que el valor de la pérdida rebote descontroladamente sin alcanzar el mínimo global. Por ello, primeramente, probamos una tasa de aprendizaje bastante pequeña y la vamos modificando en función de los resultados obtenidos. En un principio establecemos una tasa de aprendizaje de 0.0001 y, tal y como se muestra en la gráfica de la Figura correspondiente a esta tasa de aprendizaje, la pérdida comienza a descender paulatinamente pero este finaliza antes de que el modelo consiga llegar a converger de forma definitiva.

Una primera aproximación sería aumentar la duración del entrenamiento incrementando el número de épocas para dar tiempo al modelo a que converja. El principal inconveniente de este método es el sobreajuste (conocido comúnmente en inglés como overfitting). Este fenómeno se produce cuando el modelo es sobrexpuesto a los ejemplos de entrenamiento lo que origina que sea incapaz de realizar predicciones correctas sobre nuevos ejemplos que no sean los de entrenamiento. Este suceso es muy frecuente cuando se entrena un modelo con tasas de aprendizaje muy pequeñas durante periodos de tiempo muy largos con los mismos ejemplos [25]. En el caso probado anteriormente con MNIST se optó por aumentar la duración del entrenamiento debido a que la tasa de aprendizaje establecida no era tan pequeña como en este caso lo que reduce el riesgo de caer en el problema del sobreajuste. Por ello, en este caso, se realiza una segunda prueba aumentando la tasa de aprendizaje a 0.001 para intentar hacer que el modelo converja más rápidamente. Con esta nueva tasa de aprendizaje se observa que efectivamente el modelo disminuye la pérdida de forma mucho más rápida, pero al llegar a la zona de convergencia, entorno a una pérdida de 0.2, los valores empiezan a fluctuar bruscamente.

Se realiza una tercera prueba estableciendo la tasa de aprendizaje en 0.01. Con este nuevo valor se observa que aún el modelo es capaz de converger más rápidamente consiguiendo que la mayor parte del entrenamiento se mantenga fluctuando en la zona de convergencia sin producir cambios. Teniendo en cuenta este último aspecto, sería posible reducir la duración del entrenamiento ya que el modelo consigue converger muy rápidamente con muy pocas iteraciones y el resto no con-

tribuyen a que se siga reduciendo la pérdida. Por último, se realiza una prueba con una tasa de aprendizaje inusualmente grande para comprobar el efecto que tienen estos valores sobre el aprendizaje. Los resultados obtenidos en esta prueba se corresponden con las apreciaciones en la sección 4.1 sobre los valores extremos de la tasa de aprendizaje. En este caso al tratarse de una tasa extraordinariamente grande presenta valores de pérdida extremos y muy superiores a los de las otras tres pruebas realizadas, sin signos de convergencia alguna. En la Figura 15 se muestran las gráficas de la función de pérdidas de las cuatro pruebas realizas.

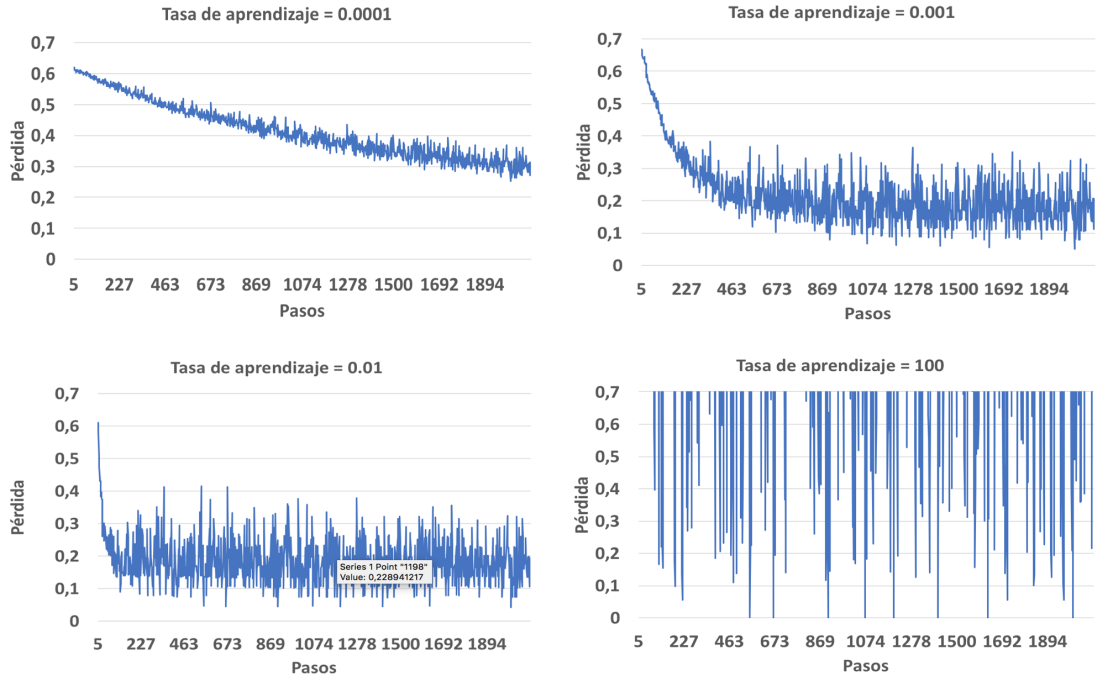


Figura 15: Gráfica de pérdidas de los cuatro experimentos realizados con diferentes tasas de aprendizaje.

Normalmente la primera medida utilizada para evaluar el entrenamiento es la exactitud (en inglés, accuracy) (6). La exactitud determina la fracción de predicciones correctas sobre el total de predicciones realizadas. Como se puede observar, tanto la prueba realizada con una tasa de aprendizaje de 0.001 como la de 0.01 presentan el mismo patrón de divergencia cuando se alcanza el rango de pérdida de 0.2. Por ello, para poder evaluar las predicciones realizadas, se ha decidido aplicar esta medida sobre el modelo con una tasa de aprendizaje de 0.01 por ser el que

más rápido converge. Al aplicar esta medida con predicciones sobre ejemplos que el modelo nunca ha visto, es decir, ejemplos distintos a los mostrados durante la fase de entrenamiento, se obtienen que la exactitud obtenida es de alrededor de un 95 %.

$$Exactitud = \frac{n^{\circ}\ de\ predicciones\ acertadas}{n^{\circ}\ total\ de\ predicciones} \tag{6}$$

En un primer momento es fácil pensar que el proceso de entrenamiento ha tenido éxito y que las predicciones realizadas por el modelo son muy acertadas, ya que la exactitud conseguida es de entorno al 95 %, es decir, acierta 95 de cada 100 predicciones realizadas. La verdad es que si se observan las predicciones realizadas de una manera más detallada se puede ver que todas las predicciones pertenecen a una única clase. En este caso, el modelo predice que todas las piezas son correctas y al haber una mayoría de piezas correctas la exactitud conseguida es muy alta.

Esto podría explicar el comportamiento mostrado en la gráfica de pérdidas. El 100 % de las veces el modelo predice que la pieza no es válida acertando un 95 % lo que genera una pérdida muy baja. Las fluctuaciones en el valor de la pérdida se producen cuando el modelo siempre que se encuentra con una pieza no válida falla lo que hace que se dispare la pérdida generada.

Al estudiar el número de ejemplos que se tiene de cada clase se observa que el número de ejemplos correspondientes a piezas correctas es de 249737 y el de piezas incorrectas es de 11503. Como se puede ver el conjunto de datos está claramente desbalanceado hacia la clase de piezas correctas con un 95,6 % de ejemplos de piezas correctas, lo que coincide justamente con la exactitud obtenida.

En este caso, debido a la disparidad en el número de ejemplos de cada clase, la exactitud no es una medida fiable ya que no refleja la realidad [26]. En los problemas de clasificación binaria surge un nuevo elemento, la matriz de confusión. La matriz de confusión mostrada en la Tabla 2 recoge los cuatro posibles resultados en un problema de clasificación binaria. En la experiencia el término “positivo” hace referencia a la clase minoritaria, en este caso la de piezas no válidas, mientras que el término “negativo” hace referencia a la clase mayoritaria, la compuesta por las piezas válidas.

| | | Predicción | |
|---|---|---|---|
| | | **Positivo** | **Negativo** |
| **Valor real** | **Positivo** | Verdadero Positivo (VP) | Falso Negativo (FN) |
| | **Negativo** | Falso Positivo (FP) | Verdadero Negativo (VN) |

Tabla 2: Matriz de confusión para problemas de clasificación binaria, obtenida de la referencia [26] .

Para poder evaluar el rendimiento del modelo de clasificación en estos casos, se utiliza la matriz de confusión . Los resultados que se pueden obtener a partir de la matriz de confusión son cuatro:

- Verdadero positivo (VP) hace referencia a los ejemplos positivos que han sido correctamente clasificados como positivos.
- Verdadero negativo (VN) denota a aquellos ejemplos correctamente clasificados como negativos.
- Falso positivo (FP), o falsa alarma, se refiere a los ejemplos incorrectamente clasificados como positivos.
- Falso negativo (FN), hace referencia al conjunto de ejemplos positivos que son clasificados como negativos.

A raíz de estos conceptos surgen otras medidas de evaluación que ayudan a detectar la existencia de este tipo de problemas. Dos de estas nuevas medidas son la precisión y la exhaustividad. La *precisión* (7) determina la proporción de predicciones positivas realizadas correctamente. Es decir, de todas las veces que se predice que una pieza no es válida, ya sea correctamente o incorrectamente (VP + FP), cuantas veces o en qué proporción se acierta realmente (VP).

$$Precision = \frac{VP}{VP + FP} \tag{7}$$

Por otra parte, la *exhaustividad* (8), conocida comúnmente en inglés como "*recall*", muestra la proporción de positivos reales que fueron identificados correctamente. Aplicada a este problema, la exhaustividad nos indica la proporción de piezas no válidas detectadas correctamente (VP) de todo el conjunto de piezas

no válidas existentes, es decir, de la suma de las piezas correctamente detectadas (VP) y de las piezas que no son válidas y el modelo ha catalogado como válidas (FN).

$$Exhaustividad = \frac{VP}{VP + FN} \tag{8}$$

Una de las prioridades cuando se trabaja con conjuntos de datos desbalanceados es mejorar la exhaustividad sin que la precisión se vea afectada [27]. Esto no siempre es posible ya que ambos objetivos habitualmente son contrarios, es decir, mejorar uno implica empeorar el otro. Esto se debe a que incrementar el número de verdaderos positivos para mejorar la exhaustividad puede originar que también aumente el número de falsos positivos, lo que perjudica a la precisión.

## 4.4. Corrección desbalanceo de muestras

El problema de desbalanceo de muestras hace referencia a la diferencia en la cantidad de ejemplos disponibles para cada una de las posibles etiquetas (resultado) en un modelo de clasificación. Esto quiere decir que un dataset está desbalanceado cuando la presencia de ejemplos de una clase es mucho mayor a la de las demás.

Una vez detectado el desbalanceo de muestras en el dataset se requieren de mecanismos que puedan dar solución a este nuevo problema. Por lo general las estrategias para manejar esta clase de problemas se suelen clasificar en dos niveles: nivel de datos y nivel de algoritmo [26]. Las estrategias del nivel de datos se basan en ajustar la proporción de ejemplos que hay de cada clase para lograr una distribución equilibrada de los ejemplos entre las diferentes clases. Por otro lado, las estrategias aplicadas a nivel de algoritmo realizan ajustes en el algoritmo de entrenamiento para facilitar el aprendizaje de los ejemplos de la clase más desfavorecida.

Dos de las técnicas más conocidas aplicadas al conjunto de datos son las conocidas como undersampling y oversampling La técnica de undersampling consiste en la eliminación de ejemplos de la clase mayoritaria para conseguir equipararla en número de ejemplos a la clase minoritaria [28]. Existen múltiples formas de aplicar esta técnica, una de las más comunes y famosas debido a su simplicidad es la

conocida como random undersampling. En este caso, para conseguir balancear el dataset, se eliminan de forma aleatoria ejemplos de la clase mayoritaria hasta que se consigue la proporción de ejemplos deseada. Al reducir el número de ejemplos se reduce el tiempo de cómputo y la capacidad de almacenamiento requerida para guardarlos. Pese a esto, una de las mayores desventajas es la probable pérdida de información útil para el entrenamiento.

Aplicado al dataset de BSH, se reduce el número de ejemplos de piezas válidas, es decir, de la clase mayoritaria, hasta que el número de ejemplos de piezas válidas sea únicamente un 15 % mayor al número de ejemplos de piezas no válidas. El conjunto de datos inicial constaba de 261240 ejemplos de los cuales 249737 eran correspondientes a piezas válidas y 11503 a piezas no válidas. Tras aplicar random undersampling el conjunto de datos pasa a estar formado por un total de 24731 ejemplos de los cuales 13228 se corresponden a piezas válidas y 11503 a piezas válidas. Como se puede observar se ha mantenido el número de ejemplos de la clase positiva y reducido el número de ejemplos de la clase negativa en casi un 95 %, es decir, se ha producido una gran pérdida de información.

En segundo lugar, la técnica de oversampling se basa en replicar ejemplos de la clase minoritaria hasta conseguir que ambas clases se encuentren balanceadas [27]. Al igual que en el caso anterior, una de las formas más sencillas de aplicar esta técnica es mediante la replicación aleatoria de ejemplos de la clase minoritaria (random oversampling). Pese a que en este caso no se produce pérdida de información, la aplicación de esta técnica puede producir sobreajuste.

El proceso de oversampling debe ser realizado una vez se hayan separado los ejemplos con los que se va a evaluar el modelo de los ejemplos de entrenamiento. De no ser así se estarían introduciendo ejemplos repetidos en el conjunto de ejemplos de evaluación lo que podría falsear los resultados.

En cuanto a las técnicas aplicadas a nivel de algoritmo, una de las más utilizadas es la conocida en inglés como cost sensitive learning. La idea fundamental de este método es penalizar lo fallos relacionados con la clase positiva, es decir, penalizar en mayor medida los falsos negativos que los falsos positivos [26]. En el caso del conjunto de datos de BSH, si se aplica esta técnica el modelo sufrirá mayor penalización si predice que una pieza es válida y no lo es (falso negativo)

que cuando predice que una pieza no es válida cuando en realidad si lo es (falso positivo). La mayor desventaja de esta técnica es encontrar un valor adecuado para la penalización que consiga corregir el desbalanceo.

## 4.5. Segundo intento de entrenamiento

Tras detectar que el conjunto de datos de BSH se encuentra desbalanceado y presentar tres posibles soluciones a este fenómeno se procede a aplicarlas sobre el modelo creado en la primera prueba de entrenamiento. Se utiliza la misma red neuronal que la utilizada en el primer intento: dos capas de 128 y 32 neuronas con una capa de salida formada por dos neuronas con la función de activación softmax y la función de entropía cruzada como función de pérdida. La tasa de aprendizaje se establece a 0,01 ya que como se pudo observar anteriormente es la que más rápidamente convergía. A continuación, la Tabla 3 muestra los parámetros de las pruebas realizadas a continuación con las tres técnicas explicadas en la sección anterior.

| | Capas | Tasa aprendizaje | N° épocas | Tamaño lote |
|---|---|---|---|---|
| Undersampling | [128, 32] | 0.01 | 10* | 100 |
| Oversampling | [128, 32] | 0.01 | 1 | 100 |
| Cost Sensitive | [128, 32] | 0.01 | 1 | 100 |

Tabla 3: Configuraciones redes neuronales.

Aplicando la técnica de random undersampling, en la Figura 16 , se observa como la pérdida está descontrolada con muy altas fluctuaciones desde el primer momento sin indicios de que vaya a converger en ningún momento.

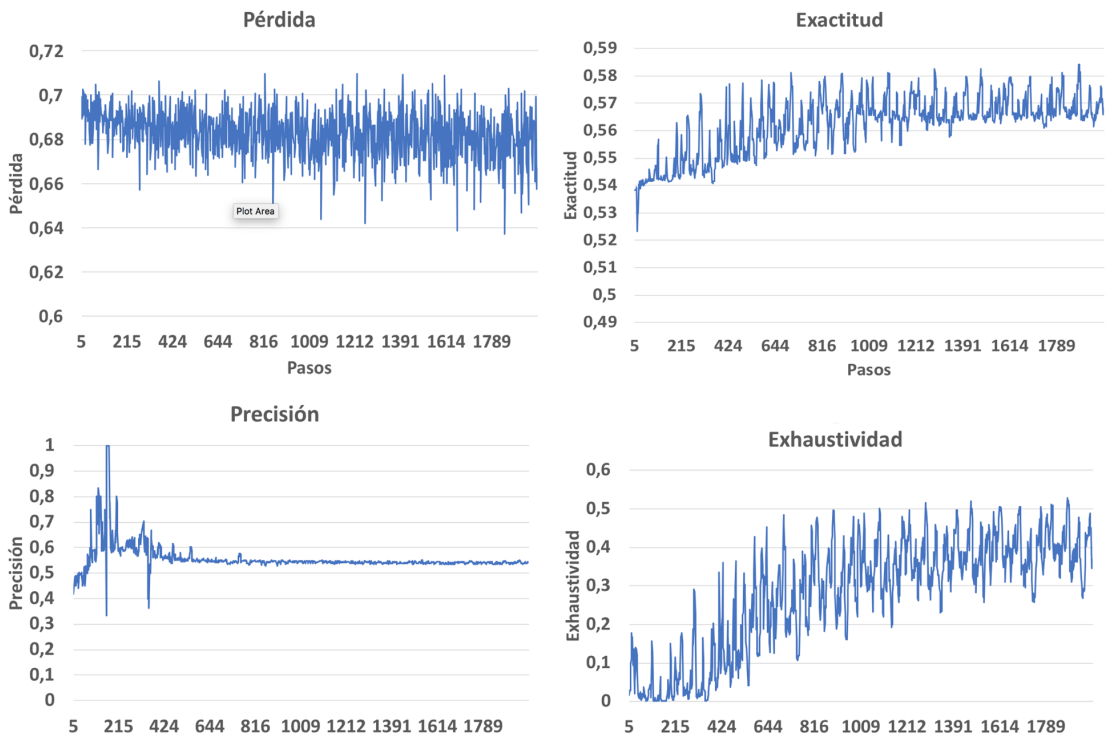


Figura 16: Resultados obtenidos tras aplicar *undersampling*.

Por otra parte, también se puede ver como la exactitud se ha visto reducida a entorno un 57 % lo que muestra que esta vez el modelo sí que está intentando predecir la existencia de piezas no válidas. Esto último se puede verificar aplicando la medida de la exhaustividad. En la tercera gráfica de la Figura 16 se puede ver como esta vez el modelo alcanza una exhaustividad media de un 40 %. Esto quiere decir que el modelo creado únicamente es capaz de identificar correctamente el 40 % de todas las piezas no válidas que componen el conjunto de datos. A su vez, la precisión nos muestra que de todas las piezas que identifica como no válidas únicamente el 54 % lo son en realidad.

Utilizando la técnica opuesta (random oversampling) se consiguen resultados muy similares a los conseguidos con la anterior. En la Figura 17 se puede observar como la pérdida sigue sin dar muestras de convergencia.

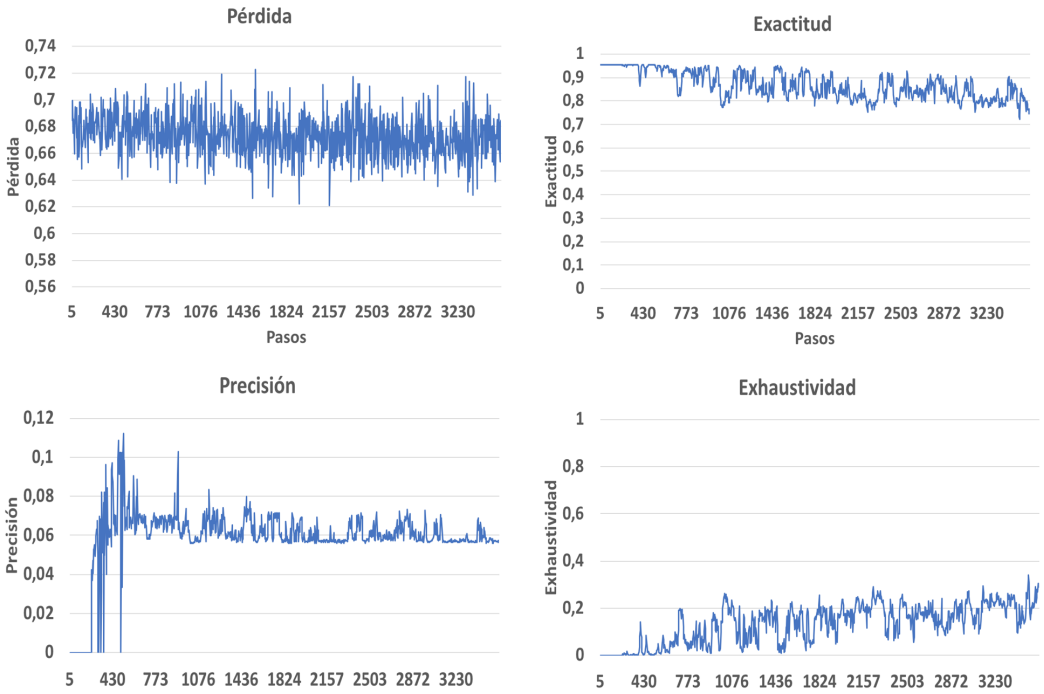


Figura 17: Resultados obtenidos tras aplicar *oversampling*.

Si se comparan juntas las gráficas de exactitud y exhaustividad se puede observar como la exactitud sigue una leve tendencia descendente que es justamente igual a la mostrada por la exhaustividad, pero esta última en sentido ascendente. Es decir, se puede ver como una bajada en un determinado punto de la exactitud produce una subida en el valor de la exhaustividad en el mismo punto y viceversa. Esto puede ser debido a que, en primer lugar, el modelo ha aumentado la frecuencia con la que realiza predicciones de piezas no válidas, sean estas correctas (verdaderos positivos) o no (falsos positivos). Esto supone al final un aumento de la exhaustividad ya que se encuentran más piezas no válidas. Sin embargo, pese a encontrarse más piezas no válidas la precisión nos muestra que la mayoría de predicciones de la clase positiva (de piezas no válidas) son erróneas siendo únicamente el 6 % de estas correctas. Esto se traduce al final en un descenso de la exactitud ya que se catalogan piezas válidas como piezas no válidas.

Por último, la última técnica a probar es la basada en la penalización de las predicciones correspondientes a un falso negativo (cost sensitive learning). En primer lugar, se calcula la proporción o ratio de piezas no válidas existentes en el

conjunto de datos (9). A cada ejemplo se la asigna un determinado peso dado por la función (10). Finalmente, la pérdida producida por cada ejemplo será multiplicada por el peso que tiene asignado. De esta forma, un ejemplo de la clase positiva tendrá mayor peso que un ejemplo de la clase negativa debido al desbalanceo de muestras existentes.

$$Ratio = \frac{n^{\circ}\ piezas\ defectuosas}{n^{\circ}\ total\ de\ piezas} \tag{9}$$

$$f(x) = \begin{cases} 1 - ratio & \text{si } x = pieza\ defectuosa \\ ratio & \text{si } x = pieza\ correcta \end{cases} \tag{10}$$

Una vez asignados los pesos se procede al entrenamiento y se obtienen los resultados representados en las gráficas de la Figura 18 . Primeramente, llama la atención que, con una exactitud tan baja, en torno al 50 %, la pérdida obtenida durante todo el entrenamiento sea mínima. Si se observa la gráfica de la exhaustividad y la precisión, al igual que en el caso del oversampling, alrededor del 95 % de las veces que el modelo predice que una pieza es errónea se equivoca. Esto quiere decir que los ejemplos de piezas válidas generarían una pérdida elevada cuando el modelo la predice como no válida. Sin embargo, debido a la asignación de pesos, la pérdida ocasionada por estos ejemplos es multiplicada por un valor muy pequeño debido al ratio calculado anteriormente lo que reduce considerablemente la pérdida final.

En segundo lugar, se observa como el modelo en los primeros pasos del entrenamiento presenta una exhaustividad muy alta, llegando a alcanzar al principio el 100 %, y de forma progresiva comienza a disminuir hasta más o menos la mitad. Si se observa este aspecto de nuevo junto con la precisión y la exactitud se puede deducir que el modelo al principio del entrenamiento está prediciendo todas las piezas como piezas erróneas. Al tener la precisión tan baja ocurre lo mismo que en la prueba anterior pero esta vez a medida que disminuye la exhaustividad es la exactitud la que aumenta llegando a alcanzar un 50 % cuándo al principio, momento en que la exhaustividad era de un 100 %, era de alrededor de un 4 % (correspondiente únicamente a los aciertos de las piezas erróneas).

Esto no quiere decir que la exactitud sea una medida inversamente proporcional a la exhaustividad, sino que al darse una precisión extremadamente baja da lugar a

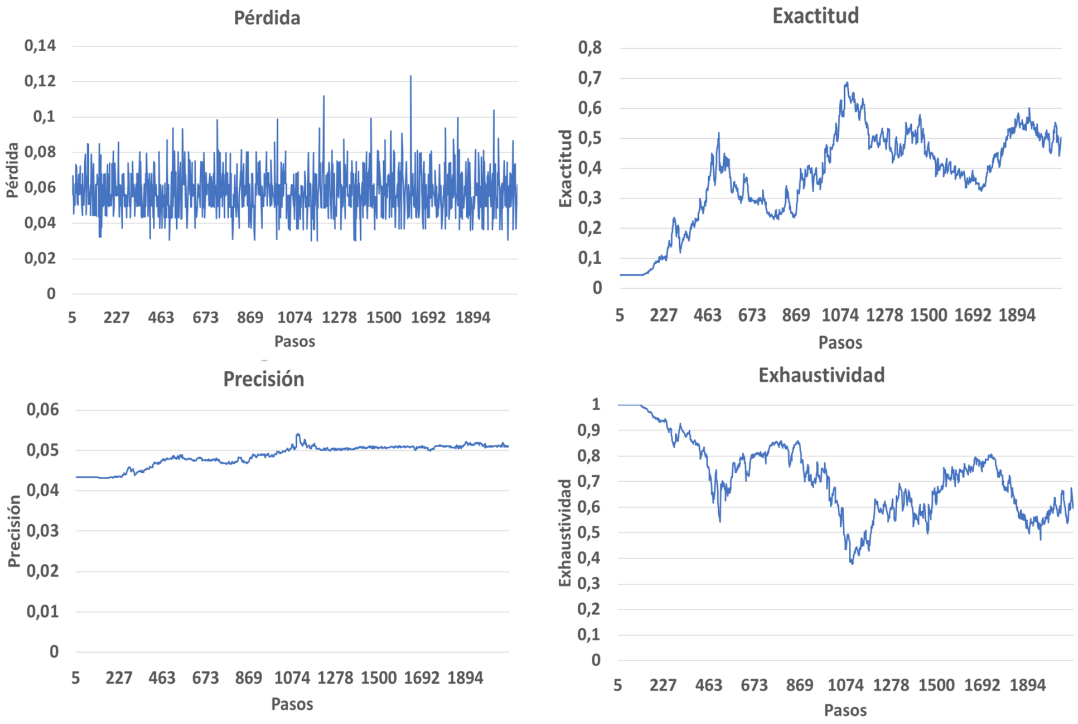


Figura 18: Resultados obtenidos tras aplicar *cost sensitive learning*.

la situación vista en las dos últimas pruebas. Esto pone en evidencia la importancia de tener claro el significado de las diferentes medidas de evaluación y de tener una visión global de todas ellas aplicadas sobre el resultado inferido por el modelo. Pese a que con estas soluciones el modelo ha empezado a realizar predicciones de la clase minoritaria, no se han conseguido resultados que satisfagan los objetivos.

Por último, dado que la aplicación de estas tres técnicas básicas para corregir el desbalanceo de muestras por sí solas no han conseguido que el modelo presente un buen rendimiento a la hora de realizar predicciones, se intenta determinar si la arquitectura de la red está siendo determinante sobre el resultado obtenido. Por ello, se realizan tres pruebas más por cada técnica de desbalanceo de muestras con tres dimensiones de redes neuronales diferentes. Las dimensiones utilizadas son las mostradas en la Tabla 4 .

Una vez establecidas las dimensiones se lleva a cabo el entrenamiento con la misma tasa de aprendizaje, épocas y tamaño de lote de las pruebas anteriores. Tras esto se presenta en la Figura 19 las gráficas correspondientes a la exactitud

| Capas ocultas | Neuronas por capa |
|---|---|
| 2 | 256-64 |
| 2 | 512-128 |
| 3 | 128-64-32 |

Tabla 4: Dimensiones de las redes neuronales utilizadas.

de los nuevos experimentos realizados.

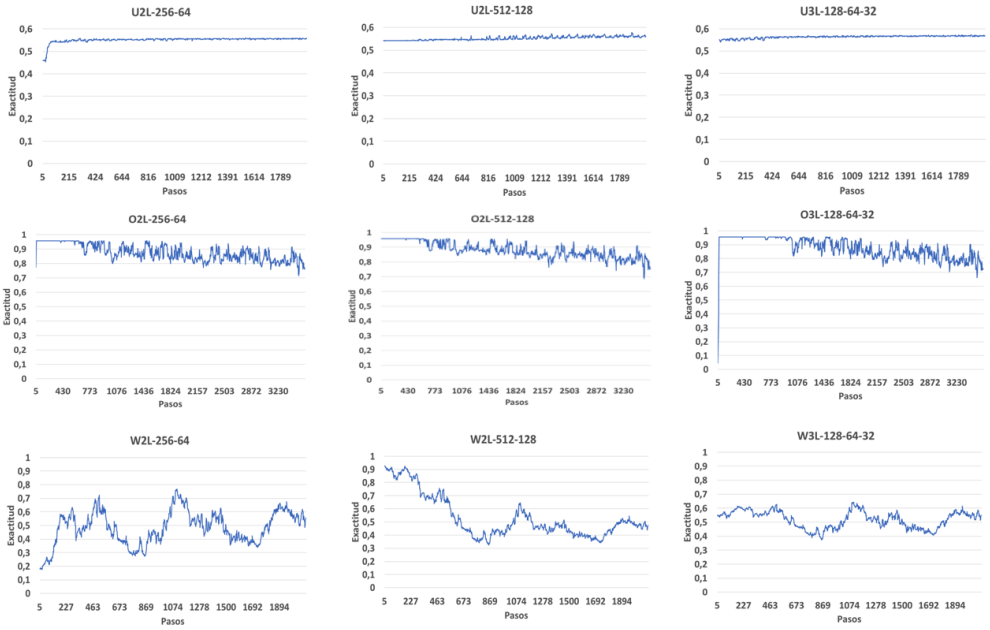


Figura 19: De arriba a abajo: *undersampling* (U), *oversampling* (O), *cost sensitive learning* (W).

Como se puede observar, si se comparan con los resultados de las pruebas realizadas con una red neuronal de dos capas con 128 y 32 neuronas respectivamente, no se consiguen nuevos resultados con ninguna de las arquitecturas propuestas. En este caso, se observa como la exactitud, en función de la técnica aplicada para corregir el desbalanceo, acaba tendiendo siempre hacia los mismos valores indistintamente del número de capas y neuronas por capa. El código desarrollado correspondiente a la creación y entrenamiento del modelo puede ser consultado en el repositorio de GitHub de la referencia [24] bajo el nombre "*train_low.py*".

# 5. Conclusiones

Se ha llevado a cabo una introducción a los conceptos y técnicas de ML y más concretamente a los relacionados con redes neuronales a través de su aplicación en un proyecto real de ML con los datos cedidos por BSH.

Se ha realizado el pre-procesado de los datos de BSH evidenciando la importancia de realizar este tipo de tareas sobre los conjuntos de datos antes de utilizarlos ya que estos distan mucho de ser ideales, como el usado en la prueba de concepto de MNIST.

Se ha propuesto un primer prototipo de modelo basado en redes neuronales que en una primera aproximación no ha sido capaz de cumplir con el objetivo de clasificar las piezas fabricadas durante el proceso de fabricación de BSH. En consecuencia, se ha detectado y presentado el problema del desbalanceo de muestras presente en el conjunto de datos de BSH y se le ha intentado dar solución sin que estas tuvieran un efecto positivo sobre los resultados finales.

Se han evaluado los resultados obtenidos a partir de las diferentes pruebas realizadas mediante la utilización de distintas métricas de evaluación y comprobado la importancia de llevar a cabo un estudio en conjunto de las mismas para el correcto entendimiento de los resultados. En consecuencia, se plantea la posibilidad de que las redes neuronales no sean el mecanismo idóneo para la extracción de información del conjunto de datos de BSH o que los datos presentes en este no revistan de utilidad para el entrenamiento del modelo.

Como trabajo a futuro se plantea la exploración de otros modelos de ML, distintos a las *feed-forward-networks* utilizadas en el presente documento, que puedan ser capaces de extraer conocimiento del conjunto de datos de BSH. Ejemplo de estos son los ya mencionados en el trabajo: regresión lineal, árbol de decisión, máquinas de soporte vectorial, etc. También se plantea la búsqueda de otras técnicas de pre-procesado de datos como puede ser la exploración de nuevas técnicas de normalización que se ajusten más a los datos disponibles. Todo esto sin olvidar el problema evidenciado en este documento sobre el desbalanceo de muestras presente en el conjunto de datos. También, sobre este inconveniente, se propone explorar nuevas técnicas más sofisticadas que den solución a esta problemática.

# Referencias


[1] Amir Gandomi and Murtaza Haider. Beyond the hype: Big data concepts, methods, and analytics. *International Journal of Information Management*, 35(2):137 – 144, 2015.

[2] F. Jiang, C. K. Leung, and A. G. M. Pazdor. Big data mining of social networks for friend recommendation. In *2016 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining (ASONAM)*, pages 921–922, Aug 2016.

[3] Bsh, https://www.bsh-group.com, Septiembre 2018.

[4] A short history of machine learning, https://www.forbes.com/sites/bernardmarr/, Septiembre 2018.

[5] Murray Campbell, A.Joseph Hoane, and Feng hsiung Hsu. Deep blue. *Artificial Intelligence*, 134(1):57 – 83, 2002.

[6] F. Rosenblatt. The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological Review*, pages 65–386, 1958.

[7] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *Nature*, 521:436 EP –, 05 2015.

[8] S. Bhattacharya, B. Czejdo, and N. Perez. Gesture classification with machine learning using kinect sensor data. In *2012 Third International Conference on Emerging Applications of Information Technology*, pages 348–351, Nov 2012.

[9] Aurélien Geron. *Hands-On Machine Learning with Scikit-Learn and TensorFlow*. O'REILLY, 2017.

[10] Machine learning crash course, https://developers.google.com/machine-learning/crash-course/ml-intro, Septiembre 2018.

[11] Scikit, http://scikit-learn.org/stable/, Agosto 2018.

[12] Enrique J. Carmona Suárez. Tutorial sobre máquinas de vectores soporte (svm). *Universidad Nacional de Educación a Distancia (UNED)*, 11 Julio 2014.

[13] Nikhil Buduma. *Fundamentals of Deep Learning*. O'REILLY, 2017.

[14] Tensorflow, https://www.tensorflow.org, Septiembre 2018.

[15] Pandas, http://pandas.pydata.org, Septiembre 2018.

[16] Numpy, http://www.numpy.org, Septiembre 2018.

[17] Yann LeCun, Leon Bottou, Genevieve B. Orr, and Klaus Robert Müller. *Efficient BackProp*, pages 9–50. Springer Berlin Heidelberg, Berlin, Heidelberg, 1998.

[18] The softmax function and it's derivative, https://eli.thegreenplace.net/2016/the-softmax-function-and-its-derivative/, Octubre 2016.

[19] Github tensorflow, https://github.com/tensorflow/tensorflow, Septiembre 2018.

[20] The mnist database of handwritten digits, http://yann.lecun.com/exdb/mnist/, Septiembre 2018.

[21] Yoshua Bengio Yann LeCun, Léon Bottou and Patricl Haffner. Gradient-based learning applied to document recognition. *IEEE*, 1998.

[22] Pip, https://pypi.org/project/pip/, Septiembre 2018.

[23] Docker, https://www.docker.com, Septiembre 2018.

[24] Repositorio tfg, https://github.com/rrevuelta/tfg-machine-learning.git, Septiembre 2018.

[25] I. Bilbao and J. Bilbao. Overfitting problem and the over-training in the era of data: Particularly for artificial neural networks. In *2017 Eighth International Conference on Intelligent Computing and Information Systems (ICICIS)*, pages 173–177, Dec 2017.

[26] Aida Ali, Siti Mariyam Shamsuddin, and Anca Ralescu. Classification with class imbalance problem: A review. 7:176–204, 01 2015.

[27] Nitesh V. Chawla. *Data Mining for Imbalanced Datasets: An Overview*, pages 853–867. Springer US, Boston, MA, 2005.

[28] Satwik Mishra. Handling imbalanced data: Smote vs. random undersampling. *International Research Journal of Engineering and Technology (IRJET)*, Agosto 2017.