UNIVERSIDAD NACIONAL DE EDUCACIÓN A DISTANCIA
ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA INFORMÁTICA

MÁSTER EN INVESTIGACIÓN EN INTELIGENCIA ARTIFICIAL

Trabajo de fin de máster

Comparación de MOOCs y Librerías de Aprendizaje Profundo: Contenidos, Funcionalidad y Rendimiento

Adrián Ciordia Galar

Dirigido por: Dra. Raquel Sánchez Cauce
Prof. Dr. Francisco Javier Díez Vegas

Curso: 2019-2020: 2ª Convocatoria

Resumen

Este proyecto se centra en ofrecer una guía que ayude a los interesados a introducirse en el aprendizaje profundo. Hemos estudiado varios MOOCs sobre este tema y se han comparado exhaustivamente los contenidos y la metodología de dos de ellos, el *Programa Especializado Aprendizaje Profundo* de Coursera y *fastai course v3*. También hemos realizado una comparación experimental de dos de las librerías de aprendizaje profundo más populares hoy en día, Keras y fastai, que se estudian en cada uno de estos cursos, respectivamente, con el objetivo de mostrar sus diferencias en funcionalidad y rendimiento.

Abstract

This project focuses on offering a guide for helping students to introduce themselves to deep learning. We have studied several MOOCs on this subject and exhaustively compared the contents and methodology of two of them, Coursera's *Deep Learning Specialization* and *fastai course v3*. We have also made an experimental comparison of two of nowadays most popular deep learning libraries, Keras and fastai, studied in each of these courses, respectively, in order to show their differences in functionality and performance.

Índice general

Índice de figuras

Índice de tablas

Capítulo 1

Introducción

Terminología

En el momento de escribir este documento no existe una traducción ampliamente aceptada por la comunidad científica para parte de la terminología usada. Mientras que algunos de estos términos los hemos traducido ex profeso, otros los hemos dejado en inglés por motivos de claridad. En el apéndice B se puede encontrar una sección donde se correlaciona la terminología en ambos idiomas.

1.1 Motivación

En los últimos años ha habido un creciente interés en el desarrollo y la aplicación del aprendizaje profundo (*deep learning*, DL), un área del aprendizaje automático (*machine learning*, ML) especializada en el uso de redes neuronales artificiales (RNAs) compuestas por un gran número de capas. Este interés está justificado por el éxito sin precedentes que ha tenido en el tratamiento de datos no estructurados, en comparación con otros algoritmos de aprendizaje automático. Ha destacado en áreas cómo la visión artificial (*computer vision*, CV) o el procesamiento del lenguaje natural (PLN, *natural language processing* o NLP), donde ha mejorado la calidad de los resultados con respecto a algoritmos clásicos. Aun así, actualmente sólo se han dado los primeros pasos en esta disciplina; es muy probable que en un futuro próximo se produzcan importantes mejoras en su rendimiento y se amplíen sus áreas de aplicación.

El fuerte desarrollo que ha experimentado el aprendizaje profundo ha llevado a la aparición de multitud de librerías y entornos para facilitar la construcción y el entrenamiento de modelos de este tipo. Algunas de las más conocidas son TensorFlow, PyTorch, CNTK, DL4J, Keras y fastai. Cada una de ellas presenta sus propias características, que la hacen única frente a las demás, y es responsabilidad del usuario escoger aquella que mejor se ajuste a sus necesidades.

Pese a la actual popularidad del aprendizaje profundo no resulta fácil introducirse en esta disciplina. Aunque es posible encontrar libros didácticos sobre el tema, cursos impartidos por universidades e incluso sitios web con tutoriales, una de las formas más habituales hoy en día de dar los primeros pasos en el aprendizaje profundo es mediante

la realización de algún MOOC (*massive open online course*). Los MOOCs son cursos impartidos por internet pensados para ser realizados por multitud de estudiantes al mismo tiempo. Están principalmente basados en lecciones grabadas en vídeo, aunque suelen ofrecer material didáctico adicional, como lecturas o ejercicios para practicar los conocimientos adquiridos. Además, suelen tener un alto componente de interactividad, de modo que los estudiantes se relacionan entre sí y con el equipo docente mediante foros o medios similares.

Sin embargo, no existe un manual claro que indique qué cursos realizar ni en qué orden. Además, muchos de estos cursos suelen coincidir en gran parte de su temática, aunque difieren en la forma de impartirla, y a veces tienen contenidos exclusivos que no se dan en otros. Con este trabajo pretendemos dar una guía de estudio que (1) oriente al alumno sin ningún conocimiento previo de la materia sobre cómo introducirse en el aprendizaje profundo, (2) le ayude a escoger aquel curso cuya metodología se adapte mejor a sus hábitos de estudio y (3) le muestre los contenidos exclusivos de cada curso que puedan hacer decantarse entre uno u otro en función de sus intereses.

1.2 Objetivos

Los objetivos de este trabajo son los siguientes:

- Dar al lector indicaciones que le permitan seleccionar algún MOOC para introducirse en el mundo del aprendizaje profundo en función de sus necesidades. Hemos examinado siete MOOCs de aprendizaje profundo, pero nuestro estudio se ha centrado en los dos más relevantes en la actualidad: (1) el *Programa Especializado de Aprendizaje Profundo*, de Andrew Ng et al. en Coursera[1], y (2) *fastai course v3*, de Jeremy Howard et al[2].

- Que el lector sea capaz de seleccionar aquellas librerías de aprendizaje profundo que mejor se adecuen a las necesidades de sus futuros proyectos. Se han estudiado dos de las más usadas hoy en día: Keras[3] y fastai[4]. Ambas se estudian en los dos cursos analizados, respectivamente.

Además, al describir cada curso hemos incluido referencias tanto a las publicaciones originales de los algoritmos más importantes, como a diferentes materiales didácticos donde ampliar los conocimientos ofrecidos en los cursos.

1.3 Metodología

Este trabajo se ha llevado a cabo en dos fases. La primera se ha centrado en la comparación de diferentes MOOCs de aprendizaje profundo, mientras que la segunda ha consistido en un análisis experimental para comparar dos de las librerías de desarrollo explicadas en los MOOCs indicados en la sección anterior.

[1] `https://www.coursera.org/specializations/deep-learning`.
[2] `https://course.fast.ai`.
[3] `https://keras.io`.
[4] `https://www.fast.ai`, `https://docs.fast.ai`.

Hemos recogido las principales características de siete cursos online de aprendizaje profundo y, posteriormente, hemos cursado los dos ya citados. Una vez realizados, hemos descrito los contenidos que ofrecen y los hemos comparado entre sí.

Por otro lado, hemos comparado experimentalmente dos de las librerías de aprendizaje profundo más usadas hoy en día: Keras y fastai. Los experimentos se han centrado en el entrenamiento de varios clasificadores multi-clase de imágenes, uno de los problemas más típicos hoy en día en el aprendizaje profundo. Hemos analizado la funcionalidad que ofrece cada una de estas librerías para acometer esta tarea y, además, hemos estudiado la eficacia y la eficiencia en ambos casos.

1.4 Organización del documento

El resto del documento se organiza de la siguiente manera: el capítulo 2 revisa el estado del arte, ofreciendo un resumen de la historia y la evolución de las redes neuronales artificiales desde sus inicios a mediados de siglo XX hasta los últimos avances en el aprendizaje profundo; en el capítulo 3 se realiza la comparación entre los MOOCs de aprendizaje profundo y se proporcionan referencias con las que ampliar el conocimiento ofrecido en ellos; en el capítulo 4 se comparan experimentalmente las librerías Keras y fastai centrándose en un problema de clasificación de visión artificial; y en el capítulo 5 se dan las conclusiones y el trabajo futuro.

Adicionalmente, el apéndice A contiene algunas gráficas obtenidas durante los entrenamientos realizados en el análisis experimental de las librerías, y el apéndice B recoge la traducción de la terminología.

1.5 Aspectos éticos

No hemos encontrado ninguna implicación ética de este trabajo digna de reseñar.

Capítulo 2

Estado del arte

2.1 Nacimiento de la inteligencia artificial

Aunque a lo largo de la historia ha habido intentos de replicar el comportamiento humano mediante la construcción de diversos artefactos artificiales [Cave y Dihal 2018; Moran 2006], no es hasta mediados del siglo XX cuando la ciencia empezó a abordar este objetivo de forma activa. Tomando como base la teoría de la computación de Alan Turing[1] e inspirándose en otras disciplinas como la neurobiología, la cibernética o la teoría de la información, se estableció la hipótesis de que las máquinas pudieran reflejar cierto comportamiento inteligente. Siguiendo esta línea de trabajo, McCulloc y Pitts [1943] definieron el primer modelo de *redes neuronales artificiales* (RNAs) de la historia y sugirieron su capacidad para aprender. Unos años más tarde Donal Hebb [1949] describió las bases de este aprendizaje, que más adelante sería conocido como aprendizaje hebbiano. Este trabajo estableció el punto de partida para el aprendizaje no supervisado en RNAs.

Alan Turing [1950] publicó el artículo "Computing Machinery and Intelligence", donde definió su famoso test y, además, estableció muchas de las ideas que posteriormente sentarían las bases de la *Inteligencia Artificial* (IA). Ese mismo año, dos estudiantes de la Universidad de Harvard, llamados Marvin Minsky y Dean Edmonds, construyeron el primer ordenador que simulaba una red neuronal, al que denominaron SNARC (*Stochastic Neural Analog Reinforcement Calculator*). En 1952, Arthur Samuel [1959][2] presentó un programa capaz de jugar a las damas y de mejorar conforme lo hacía. Ese mismo año, dicho investigador acuñó el término "*machine learning*" (*aprendizaje automático*). En la conferencia de Darthmouth de 1956 fue donde se concibió oficialmente la IA como una disciplina propia. Dicho término fue establecido por John McCarthy, uno de los organizadores de la conferencia.

2.2 Desarrollo de las redes neuronales

A pesar de los tempranos intentos de crear una máquina capaz de aprender, esta aproximación al modelado del conocimiento fue relegada a un segundo plano durante años.

[1] Algunos de los eventos citados en el estado del arte se han extraído de o inspirado en [Russell y Norvig 2009; Schmidhuber 2014; Chollet 2018].

[2] El trabajo fue desarrollado en 1952 pero publicado en 1959.

Hasta finales de la década de 1980 el paradigma dominante fue la IA simbólica, basada en reproducir comportamientos inteligentes mediante la programación explícita de conjuntos de reglas para manejar el conocimiento.

Mientras tanto, siguieron las investigaciones sobre el aprendizaje automático y el paradigma conexionista. Poniendo el foco en el desarrollo de las RNAs y el aprendizaje supervisado, Frank Rosenblatt [1958, 1962] se inspiró en los trabajos previos de Warren McCulloc, Walter Pitts, Donal Hebb y otros para crear el *perceptrón*, la base de las RNAs modernas. Según varios autores [Nagy 1991; Olazaran 1996; Nilsson 2010], fue el propio Rosenblatt quien empezó a experimentar con *perceptrones multicapa*. Posteriormente, Alexey Ivakhnenko et al. [1965; 1967; 1971] desarrollaron una serie de RNAs entrenadas mediante el *Group Method of Data Handling* (GMDH), que fueron probablemente las primeras *redes neuronales profundas* (*deep neural networks*, DNNs) basadas en perceptrones. Ivakhnenko [1971] describió una *deep GMDH NN* de 8 capas.

Seppo Linnainmaa [1970] definió por primera vez un método general para la diferenciación automática, el componente clave que permite entrenar RNAs mediante aprendizaje supervisado. Posteriormente, este mecanismo recibió el nombre de *retropropagación del error* (*backpropagation*)[3] y contribuyó al resurgimiento del interés por las RNAs. Paul Werbos [1974] fue el primero en usar este método para entrenar RNAs y, posteriormente, esta forma de entrenamiento se popularizó tras la publicación de Rumelhart et al. [1986].

Kunihiko Fukushima [1979, 1980] creó el Neocognitron, la primera *red neuronal convolucional* (*convolutional neural network*, CNN o ConvNet), un modelo inspirado en la neurofisiología para resolver el problema de reconocimiento de caracteres escritos. Posteriormente, Yann LeCun et al. [1989] aplicaron el algoritmo de retropropagación para entrenar una RNA similar al Neocognitron; es decir, fue la primera vez que se entrenó una ConvNet usando dicho algoritmo. Weng et al. [1992] publicaron una variación del Neocognitron, denominada Cresceptron, capaz de adaptar su topología durante el entrenamiento. Además, en esta nueva arquitectura usaron por primera vez capas *max pooling*, las cuales reducen las dimensiones de las activaciones seleccionando el elemento de valor máximo de cada subconjunto del mapa de características definido por un filtro o kernel.

Hopfield [1982] presentó las redes que llevan su nombre, un tipo temprano de red neuronal recurrente (*recurrent neural network*, RNN), aunque ya habían sido descritas previamente en [Little 1974]. Posteriormente, Sepp Hochreiter y Jürgen Schmidhuber [1997] introdujeron las *Long Short-Term Memory* (LSTM), una nueva arquitectura de RNNs que podía capturar mucho mejor las dependecias a largo plazo. En [Schuster y Paliwal 1997; Schuster 1999] se presentaron las *bidirectional RNNs* (BRNNs), redes diseñadas para procesar secuencias de datos cuyo inicio y final se conoce desde un principio.

Sepp Hochreiter [1991] descubrió el *problema del desvanecimiento y la explosión de los gradientes* (*vanishing/exploding gradients problem*), la principal causa que impidió durante años el entrenamiento de DNNs mediante retropropagación del error o el descenso

[3]Se puede encontrar más información sobre el descubrimiento de la retropropagación del error en `http://people.idsia.ch/~juergen/who-invented-backpropagation.html`

del gradiente. Aunque identificó el problema, no se dio con una solución aceptable hasta casi dos décadas más tarde.

Durante este tiempo también se fueron desarrollando otros mecanismos muy influyentes dentro del aprendizaje automático, pero ajenos al mundo de las RNAs supervisadas, como los *mapas autoorganizados* (*self-organizing maps*, SOMs) [Kohonen 1982], los *random forest* [Ho 1995], o las *máquinas de vectores soporte* (*support vector machines*, SVMs) [Cortes y Vapnik 1995], por ejemplo.

2.3 Auge del aprendizaje profundo

Aparte de la evolución de los algoritmos, hay dos factores adicionales responsables del éxito del aprendizaje profundo en los últimos años. El primero de ellos ha sido la recolección masiva de datos gracias a la popularización del uso de internet. Las DNNs necesitan de una gran cantidad de datos y este fenómeno ha permitido crear conjuntos de datos apropiados para la experimentación. El segundo factor es el desarrollo tecnológico de las GPUs, actualmente el hardware usado habitualmente para el entrenamiento de DNNs, debido a que aceleran enormemente este proceso con respecto a las CPUs. Las GPUs fueron diseñadas originalmente para el procesamiento gráfico y su desarrollo está íntimamente ligado a la industria del videojuego y a los sistemas profesionales de diseño y visualización.

Yann LeCun et al. [1998] propusieron LeNet-5, que es considerada como la primera ConvNet moderna. En dicho trabajo, además, se definió el conjunto de datos MNIST, basado en el problema de reconocimiento de códigos postales presentado en [LeCun et al. 1989], que se ha usado como estándar de comparación para determinar la calidad de nuevos algoritmos durante más de dos décadas. Posteriormente, Jia Deng et al. [2009] crearon ImageNet, un conjunto de datos con alrededor de 14 millones de imágenes que actualmente es considerado como el problema estándar de clasificación de imágenes.

Yoshua Bengio et al. [2003] presentaron el primer mecanismo para obtener una representación distribuida de un conjunto de palabras en un espacio vectorial conocido como *word embeddings*. El algoritmo presentado en su trabajo hace uso del entrenamiento de un modelo de lenguaje (*language model*) para obtener dicha representación. En trabajos posteriores este mecanismo se fue simplificando, por ejemplo, [Mikolov et al. 2013a,c; Pennington et al. 2014]. Graves et al. [2006] propusieron la clasificación conexionista temporal (*Connectionist Temporal Classification*, CTC), un mecanismo para predecir secuencias de etiquetas a partir de entradas no segmentadas[4], que es muy usado actualmente junto con RNNs para tareas como el reconocimiento del habla.

Steinkraus et al. [2005] presentaron por primera vez una RNA de 2 capas acelerada mediante una GPU, y Chellapilla et al. [2006] hicieron lo mismo para una ConvNet. NVIDIA publicó CUDA en 2007, una plataforma para cálculo en paralelo y API para computación en GPUs, ampliamente usada actualmente para el aprendizaje profundo.

[4]Es decir, las entradas no están divididas en secuencias de elementos donde cada uno de ellos coincide exactamente con su correspondiente etiqueta de la secuencia de salida.

Cireşan et al. [2011a,b] presentaron un sistema compuesto por varias ConvNets capaz de superar por primera vez las capacidades humanas en un problema de clasificación de imágenes. Concretamente se trataba de reconocer señales de tráfico para un hipotético vehículo de conducción autónoma. Alex Krizhevsky et al. [2012] crearon AlexNet, una ConvNet en la que se usó por primera vez la función de activación ReLU, que consiste en poner a cero aquellas activaciones cuyo valor es negativo. Además, este modelo consiguió los mejores resultados en la competición de ImageNet[5] de dicho año, suponiendo un punto de inflexión en el uso de ConvNets para visión artificial. Girshick et al. [2013] propusieron R-CNN, una ConvNet para detección de objetos muy influyente; usa un algoritmo de segmentación para preprocesar las imágenes y detectar las áreas más prometedoras donde localizar los objetos de interés. Tsung-Yi Lin et al. [2014] crearon el conjunto de datos COCO para Microsoft, que es actualmente el más usado para la experimentación sobre detección de objetos en imágenes. Christian Szegedy et al. [2014], un grupo compuesto principalmente por investigadores de Google, presentaron *Inception Network* (GoogLeNet), una ConvNet que se caracteriza por aplicar diferentes operaciones paralelas (convoluciones de diferentes tamaños) a las activaciones de entrada de una capa y concatenar los resultados para obtener las de salida.

Sutskever et al. [2014] y Cho et al. [2014a] propusieron el *RNN Encoder-Decoder*, una arquitectura diseñada para tratar con secuencias de entrada y salida de diferente longitud. Se basa en usar una RNN para codificar la entrada y otra para generar la secuencia de salida en función de dicha codificación. RNN Encoder-Decoder codifica la secuencia de entrada en un vector de características que a veces no es capaz de capturar toda la información necesaria de ésta. Por ello, Dzmitry Bahdanau et al. [2014] diseñaron el *Attention Model*, una RNN que emplea diferentes porciones de la secuencia de entrada para generar cada uno de los elementos de la de salida.

Kaiming He et al. [2015a] crearon una ConvNet revolucionaria denominada *Residual Network* (ResNet). Esta nueva arquitectura añadía conexiones "alternativas" entre capas no consecutivas, llamadas *skip connections*, que permitían entrenar DNNs con muchas más capas que hasta entonces. Con este modelo ganaron la competición de ImageNet de dicho año, entre otras. Joseph Redmon et al. [2015] introdujeron el modelo *You Only Look Once* (YOLO), una de las ConvNets para detección de objetos más influyentes hoy en día. Olaf Ronneberger et al. [2015] presentaron U-Net, una ConvNet que usa *skip connections* para comunicar un codificador con un decodificador. Aunque originalmente estaba pensada para la segmentación de imágenes médicas, actualmente es el modelo más eficaz para segmentar cualquier tipo de imágenes. ResNet tuvo un gran impacto en la comunidad del aprendizaje profundo y la visión artificial. Por ello, desde su aparición se han ido proponiendo diferentes mejoras y variaciones de esta arquitectura. Por ejemplo, Gao Huang et al. [2016] desarrollaron la *Densely Connected Convolutional Network* (DenseNet), una modificación que concatena las activaciones de las *skip connections* en vez de sumarlas. Por otro lado, Tong He et al. [2018] propusieron tres importantes variaciones del bloque ResNet básico que, al combinarse, daba lugar a un modelo más eficaz que el original.

Como ya hemos comentado, el problema del desvanecimiento y la explosión de los

[5]ILSVRC (*ImageNet Large Scale Visual Recognition Challenge*) [Russakovsky et al. 2015] es una competición que evalúa algoritmos de detección de objetos y de clasificación de imágenes con el objetivo de comparar el trabajo de los investigadores y medir el progreso de la visión artificial en estas áreas.

gradientes fue uno de los principales escollos que impedía el entrenamiento de DNNs. Xavier Glorot y Yoshua Bengio [2010] se dieron cuenta de que los valores iniciales que toma una DNN influyen enormemente en dicho problema y tienen un impacto crucial en el éxito de su entrenamiento. Por ello, propusieron la *inicialización de Xavier* (*Xavier initialization*), un mecanismo que establece adecuadamente estos valores y permite entrenar redes con mayor número de capas. Más adelante, Kaiming He et al. [2015b] presentaron una modificación de este mecanismo de inicialización para adaptarlo a DNNs que usan funciones de activación ReLU. Éste acabó recibiendo el nombre de *inicialización de Kaiming* (*Kaiming initialization*). Posteriormente, Mishkin et al. [2016] introdujeron la *Layerwise Sequential Unit Variance* (LSUV), un nuevo mecanismo de inicialización basado en reescalar los parámetros de cada capa para que sus activaciones de salida estén normalizadas. Se diferencia de los anteriores en que establece los valores iniciales en función del modelo concreto sobre el que actúa y de la media y desviación típica de las activaciones de salida de cada capa. De esta forma, ajusta los parámetros del modelo para que dichas activaciones tengan aproximadamente media 0 y desviación típica 1.

En la última década también ha habido una importante evolución de los algoritmos de optimización usados para entrenar DNNs. Geoffrey Hinton, Nitish Srivastava y Kevin Swersky propusieron en 2012 el algoritmo de optimización *RMSprop* durante la lección 6e de su MOOC *Neural Networks for Machine Learning*[6], en Coursera. Posteriormente, Diederik P. Kingma y Jimmy Lei Ba [2014] crearon *Adam*, una modificación del descenso del gradiente que combina *momentum* con RMSprop y es muy usada hoy en día. Años más tarde, You et al. [2019] presentaron LAMB, una modificación de Adam.

Otro factor clave en el éxito de las DNNs ha sido el desarrollo de nuevas técnicas de regularización, las cuales ayudan a generalizar el problema contenido en los datos y que el modelo funcione correctamente con datos diferentes a los usados para su entrenamiento. Nitish Srivastava et al. [2014] propusieron el *dropout*, que consiste en eliminar aleatoriamente unidades de la RNA para evitar que partes concretas de la misma se especialicen en reconocer patrones específicos contenidos en los datos de entrenamiento. Posteriormente, Christian Szegedy et al. [2015] introdujeron el *label smoothing*, que consiste en introducir una ligera incertidumbre en las predicciones de la RNA, y Hongyi Zhang et al. [2017] presentaron otra, denominada *mixup*, basada en realizar una combinación lineal de dos muestras.

Aparte de la evolución de los optimizadores, de los mecanismos de regularización y de los propios modelos, han aparecido otro tipo de técnicas que han contribuido en mayor o menor medida al avance del aprendizaje profundo. Por ejemplo, Sergey Ioffe y Christian Szegedy [2015] presentaron la *normalización por lotes* (*batch normalization*, BatchNorm), un mecanismo que permite ajustar la media y la desviación típica de las activaciones internas de una RNA, facilitando así el entrenamiento de la misma. Así mismo, Leslie Smith [2015, 2018] ha contribuido a este desarrollo proponiendo dos mejoras: (1) *learning rate finder*, un mecanismo que estima a priori un valor de la tasa de aprendizaje para entrenar adecuadamente cada DNN, y (2) *one cycle*, un planificador de hiperparámetros que acelera los entrenamientos de DNNs. La aplicación de DNNs a problemas con datos estructurados es un área que actualmente no se ha explorado exhaustivamente. Sin embargo, Cheng

[6]Diapositivas de la lección 6 del curso online *Neural Networks for Machine learning*: `https://www.cs.toronto.edu/~tijmen/csc321/slides/lecture_slides_lec6.pdf`

Guo y Felix Berkhahn [2016] introdujeron el uso de *embeddings* para representar variables categóricas, una técnica muy usada hoy en día que supuso un gran avance en dicha dirección.

El camino recorrido que ha derivado en las DNNs actuales es muy extenso y está compuesto por el trabajo de una gran cantidad de personas. En este capítulo hemos presentado solamente un breve resumen. Es muy recomendable que quién desee conocer con más detalle el estado del arte del aprendizaje profundo y su historia lea [Schmidhuber 2014].

2.4 Librerías de aprendizaje profundo

Durante este tiempo se han ido desarrollando diferentes librerías de aprendizaje profundo. Todas ellas suelen tener algún sistema de diferenciación automática (*automatic differentiation*) que permite calcular las operaciones del paso hacia atrás (*backward pass*) de forma automática a partir del paso hacia delante (*forward pass*) definido por el grafo de computación (*computation graph*) del modelo a entrenar. El desarrollo de este mecanismo ha facilitado la experimentación y aplicación del aprendizaje profundo debido a que simplifica enormemente los cálculos que el usuario tiene que realizar, especialmente para la actualización de los parámetros de un modelo durante su entrenamiento.

Collobert et al. [2002] publicaron Torch[7], una librería implementada con Lua, C, C++ y CUDA, que utiliza LuaJIT como interfaz de usuario. Jia et al. [2014] desarrollaron Caffe[8] dentro de BAIR (*Berkeley Artificial Intelligence Research*) de la Universidad de California, Berkeley. Está escrita en C++ con una interfaz de usuario en Python. Bajo el amparo de Facebook, se publicó Caffe2 en 2017 ampliando la funcionalidad que ofrecía la versión anterior. FAIR (*Facebook AI Research*) presentó PyTorch [Paszke et al. 2019][9] en 2016, una nueva librería basada en Torch. Está escrita en C++ y CUDA, pero utiliza Python como interfaz de usuario. En 2018 las librerías Torch y Caffe2 se fusionaron con PyTorch abandonando sus respectivos desarrollos en favor de esta última.

MILA (*Montreal Institute for Learning Algorithms*) de la Universidad de Montreal publicó Theano[10] en 2007. Al igual que otras de las librerías citadas, está desarrollada con C y CUDA, pero utiliza Python para construir la API. MILA dejó de mantener esta librería en 2017. En 2014 se presentó DL4J (*Deep Learning for Java*)[11], una librería de aprendizaje profundo para ser usada con los lenguajes Java y Scala, aunque también está desarrollada con C, C++ y CUDA. Su desarrollo y mantenimiento pasó a manos de la Eclipse Foundation en 2017. Apache Software Foundation desarrollo MXNet[12] y la publicó en 2014. A diferencia de otras librerías de aprendizaje profundo, soporta una gran cantidad de lenguajes de programación como C++, Python, Java, Julia, Matlab, JavaScript, Go, R, Scala, Perl o Wolfram Language. Google Brain desarrolló TensorFlow

[7] `http://torch.ch`.
[8] `https://caffe.berkeleyvision.org`.
[9] `https://pytorch.org`.
[10] `http://www.deeplearning.net/software/theano`.
[11] `https://deeplearning4j.org`.
[12] `https://mxnet.apache.org`.

[Abadi et al. 2016][13] para el uso interno de Google y, posteriormente, fue publicada en 2015 para uso de terceros. Está implementada con C++ y CUDA, con una API en Python. En 2018 se hizo público que TensorFlow estaba siendo reimplementada por completo en el lenguaje de programación Swift[14] bajo la dirección de Chris Lattner, el creador de dicho lenguaje. Este nuevo proyecto recibió el nombre de *Swift for TensorFlow*[15]. Microsoft presentó CNTK (*Microsoft Cognitive Toolkit*)[16] en 2016, su propia librería de aprendizaje profundo que puede ser usada en Python, C# o C++. Ese mismo año también se presentó PaddlePaddle (*PArallel Distributed Deep LEarning*)[17], otra librería con API en Python.

Además de este tipo de librerías desarrolladas a partir de componentes de bajo o medio nivel como C o CUDA, existen de otro tipo que proporcionan funcionalidad de alto nivel tomando como base otras librerías de aprendizaje profundo. En 2015 François Chollet presentó Keras[18], la cual proporciona una interfaz común en Python para librerías como Theano, TensorFlow o CNTK y cierta funcionalidad superior para facilitar el entrenamiento y uso de DNNs. A partir de 2019, tras la publicación de TensorFlow 2.0, Keras pasó a formar parte del núcleo de esta librería, descartando el resto de librerías con las que era compatible anteriormente. En 2017, Jeremy Howard, Rachel Thomas y Sylvain Gugger publicaron fastai [Howard y Gugger 2020][19]. Esta librería toma PyTorch como base y desarrolla funcionalidad de alto nivel para resolver problemas de visión artificial, procesamiento del lenguaje natural, datos tabulados o filtrado colaborativo.

2.5 MOOCs

Un MOOC (*Massive Open Online Course*)[20] es un curso impartido a través de internet abierto a la participación de un gran número de estudiantes. Este concepto se inspira en la educación a distancia que usaba como medio didáctico el correo postal y las emisiones por radio o televisión. El paradigma cambió en la década de los 2000 debido a la popularización de internet, pasando a ser este medio la principal vía para la comunicación. En 2008, George Siemens y Stephen Downes impartieron el curso *Connectivism and Connective Knowledge* (CCK08) en la Universidad de Manitoba con 25 estudiantes presenciales, y otros 2200 de forma online y gratuita. Éstos podían acceder a los contenidos mediante RSS, un formato basado en XML para difundir contenido por internet, y comunicarse entre sí a través de diferentes aplicaciones web. Esta novedosa forma de abordar la docencia inspiró a Dave Cormier para acuñar el término MOOC.

Los cursos centrados en la conectividad y la interacción entre los participantes, como el citado previamente, acabaron por denominarse cMOOCs. Por otro lado, aparecieron los xMOOCs, con una aproximación más tradicional y centrados en la obtención de algún

[13] `https://www.tensorflow.org/`.

[14] `https://blog.tensorflow.org/2018/04/introducing-swift-for-tensorflow.html`.

[15] `https://www.tensorflow.org/swift`

[16] `https://docs.microsoft.com/en-us/cognitive-toolkit`.

[17] `https://www.paddlepaddle.org.cn`, `https://github.com/PaddlePaddle/Paddle`.

[18] `https://keras.io`.

[19] `https://www.fast.ai`, `https://docs.fast.ai`.

[20] Parte de la información de esta sección se ha extraído o inspirado principalmente en `https://en.wikipedia.org/wiki/Massive_open_online_course` y `https://www.mcgill.ca/maut/current-issues/moocs/history`.

tipo de certificado. Tienen una estructura más jerárquica en la que el profesor expone las lecciones a una audiencia masiva y la intercomunicación se limita a resolver dudas de los estudiantes. Es habitual que se requiera del pago de una tarifa para poder acceder a los contenidos.

En 2012 se produjo un punto de inflexión con la aparición de los proveedores de MOOCs, plataformas online especializadas para la difusión de este tipo de cursos. De esta forma, los cursos no son mantenidos directamente por las instituciones educativas sino que comienzan a ser distribuidos a través de estos intermediarios, generalmente comerciales. Debido al éxito de varios cursos online impartidos en la Universidad de Stanford durante el año 2011, algunos de los profesores que participaron en ellos se animaron a fundar sus propias plataformas de distribución de MOOCs. Sebastian Thrun, responsable de uno de estos cursos junto a Peter Norvig, creó Udacity[21] en febrero de 2012 en colaboración con David Stavens y Mike Sokolsky. De forma similar, los profesores de Stanford Andrew Ng y Daphne Koller fundaron Coursera[22] en abril del mismo año. Por otro lado, el MIT (*Massachusetts Institute of Technology*) y la Universidad de Harvard se aliaron para fundar edX[23] en mayo de 2012 tomando como base la plataforma para distribución de MOOCs MITx.

En el seno de estas plataformas comerciales de difusión de MOOCs ha ido apareciendo una panoplia de cursos sobre aprendizaje profundo. Actualmente se pueden destacar el *Programa Especializado Aprendizaje Profundo* y *Applied AI with Deep Learning* de Coursera; *Deep Learning Nanodegree* de Udacity; e *IBM's Deep Learning Professional Certificate* de edX. Hay quien no usa este tipo de plataformas para difundir sus MOOCs. Un ejemplo de ello son los cursos ofrecidos por fastai[24], cuyas lecciones en vídeo se distribuyen a través de YouTube y utilizan foros y blogs para mantener la comunicación e interactividad entre los estudiantes y el equipo docente.

[21] `https://www.udacity.com`.

[22] `https://www.coursera.org`.

[23] `https://www.edx.org`.

[24] fastai course v3: `https://course.fast.ai`, Introduction to Machine Learning for Coders: `https://www.fast.ai/2018/09/26/ml-launch`, Computational Linear Algebra: `https://www.fast.ai/2017/07/17/num-lin-alg`, A Code-First Introduction to Natural Language Processing: `https://www.fast.ai/2019/07/08/fastai-nlp/`.

Capítulo 3

Análisis de los cursos de aprendizaje profundo

3.1 Análisis general de los MOOCs de aprendizaje profundo

Una de las formas más populares de introducirse al aprendizaje profundo hoy en día es mediante MOOCs, es decir, a través de cursos online cuyas lecciones vienen generalmente impartidas mediante vídeos. Adicionalmente, estos cursos suelen proporcionar recursos secundarios, como foros para preguntar y debatir, transcripciones de las lecciones, referencias a artículos o documentos de interés, etc.

Actualmente hay multitud de este tipo de cursos sobre aprendizaje profundo. Hemos llevado a cabo una investigación para encontrar los más relevantes de entre toda la oferta disponible. En el transcurso de ésta, nos hemos centrado principalmente en las plataformas de difusión de MOOCs más usadas en la actualidad: Coursera, Udacity y edX. La tabla 3.1 muestra la información básica sobre aquellos seleccionados.

Conocimientos previos Generalmente, todos ellos tienen como requisitos que el alumno posea ciertos conocimientos (1) de programación, preferiblemente en Python, y (2) de álgebra lineal básica. Sin embargo, estos requisitos a veces se tratan dentro de los contenidos de los propios cursos. Algunos MOOCs precisan de conocimientos previos adicionales más específicos como, por ejemplo, *Intro to Deep Learning with Python* requiere que el alumno conozca previamente las librerías de Python NumPy[1] y Matplotlib[2], o *IBM's Deep Learning Professional Certificate* precisa ciertos conocimientos de derivadas parciales.

Contenidos En todos los cursos se imparte cierto contenido común: (1) fundamentos de RNAs (2) ConvNets, para resolver problemas de visión artificial; y (3) RNNs, generalmente aplicadas a problemas de PLN; aunque algunos de ellos imparten materias menos comunes. Por ejemplo, *IBM's Deep Learning Professional Certificate* y el *Programa Especializado Aprendizaje profundo* ofrecen lecciones sobre la gestión de un proyecto de aprendizaje

[1] `https://numpy.org`.
[2] `https://matplotlib.org`.

automático o de aprendizaje profundo; o *fastai course v3* aborda problemas de datos tabulados con DNNs.

curso (plataforma)	**requisitos**	**características**	**contenido**
Programa Especializado Aprendizaje Profundo (Coursera)	– Conocimientos matemáticos básicos. – Cierta experiencia de programación.	– Duración: 16 semanas. – Basado en NumPy, TensorFlow y Keras. – Avalado por deeplearning.ai. – 44€/mes	– Fundamentos de redes neuronales. – Gestión de proyectos de aprendizaje automático. – ConvNets y RNNs.
fastai course v3	– Conocimientos matemáticos "de instituto". – Un año de experiencia de programación.	– Duración: 14 semanas aprox. – Dividido en dos partes. – Basado en PyTorch, fastai y Swift for TensorFlow. – Impartido por los desarrolladores de fastai. – Gratuito.	– Fundamentos de redes neuronales. – Modelos para visión artificial, PLN, sistemas de recomendación y datos tabulados. – Implementación de algoritmos novedosos.
Applied AI with DeepLearning (Coursera)	– Conocimientos de álgebra lineal. – Conocimientos de programación, preferiblemente en Python.	– Duración: 4 semanas, entre 4 y 6 horas/semana. – Basado principalmente en Keras y TensorFlow. Adicionalmente, usa PyTorch, DL4J y Apache SystemML. – Impartido por IBM.	– Fundamentos de álgebra lineal y redes neuronales. – Múltiples DNNs para PLN, visión artificial y otras áreas.
Intro to TensorFlow for Deep Learning (Udacity)	– Álgebra básica. – Conocimientos de la sintaxis de Python (variables, funciones, clases) y programación orientada a objetos.	– Duración: 2 meses aprox. – Basado en TensorFlow. – Avalado por Google y TensorFlow. – Gratuito	– Fundamentos de aprendizaje automático. – ConvNets, aprendizaje por transferencia, RNNs.
Intro to Deep Learning with PyTorch (Udacity)	– Se recomiendan conocimientos básicos de álgebra lineal y cálculo. – Experiencia con Python y librerías de procesamiento de datos, como NumPy y Matplotlib.	– Duración: 2 meses aprox. – Basado en PyTorch. – Avalado por Facebook AI. – Gratuito.	– ConvNets, *neural style transfer*. – RNNs, clasificación en PLN.
Deep Learning Nanodegree (Udacity)	– Conocimientos básicos de programación en Python.	– Duración aproximada de 4 meses, 12 horas/semana. – Impartido en colaboración con Amazon Web Services y Facebook AI. – Basado en PyTorch y NumPy. – 359 €/mes aprox.	– Fundamentos de redes neuronales. – ConvNets, RNNs. – GANs.
IBM's Deep Learning Professional Certificate (edX)	– Derivadas parciales. – Conocimientos de programación en Python. – Conocimientos de aprendizaje automático con Python.	– Duración: 26 semanas, de 2 a 4 horas/semana. – 5 cursos. – Basado en Keras, PyTorch y TensorFlow. – Impartido por miembros de IBM. – Gratuito. Si se desean los certificados hay que pagar.	– Fundamentos de redes neuronales. – ConvNets y RNNs. – Desarrollo de un proyecto de aprendizaje profundo para un problema real.

Tabla 3.1: Cursos online sobre aprendizaje profundo. Acrónimos y abreviaturas: PLN (procesamiento del lenguaje natural), ConvNets (*convolutional neural networks*), RNNs (*recurrent neural networks*), GANs (*generative adversarial networks*).

Librerías utilizadas y prácticas de programación En la mayoría de los cursos se realizan prácticas en Python usando alternativamente las librerías TensorFlow o PyTorch, las cuales ofrecen la funcionalidad básica sobre la que construir aplicaciones de aprendizaje profundo. Algunos de ellos usan NumPy para explicar los fundamentos del cálculo y la programación que sustentan las librerías citadas anteriormente, como en el *Programa Especializado Aprendizaje Profundo* o en *Deep Learning Nanodegree*; y otros introducen otras librerías menos comunes en este tipo de MOOCs, como DL4J en *Applied AI with DeepLearning*. Además, la mayoría de MOOCs enseñan a usar alguna librería de alto nivel, normalmente Keras, aunque *fastai course v3* utiliza la librería creada por el propio equipo docente.

Duración La duración de los cursos es muy variable; normalmente estando entre 4 semanas (6 horas/semana), como *Applied AI with DeepLearning*, y 4 meses (12 horas/semana), como *Deep Learning Nanodegree*; aunque la duración real del curso dependerá de los conocimientos previos del alumno y del tiempo de estudio que dedique cada día.

Precio, evaluación y certificación Muchos de los cursos ofrecen algún tipo de certificación para probar que se han superado con éxito, como *Deep Learning Nanodegree*, *IBM's Deep Learning Professional Certificate* o el *Programa Especializado Aprendizaje Profundo*. Sin embargo, este tipo de certificación suele requerir el pago de una determinada tarifa. Por ejemplo, el *Programa Especializado Aprendizaje Profundo* cuesta 44€ al mes.

3.2 Selección de dos cursos para el análisis en detalle

De entre todos estos MOOCs catalogados hemos seleccionado dos de ellos para estudiarlos en profundidad. Hemos elegido el *Programa Especializado Aprendizaje Profundo* de Coursera debido a que es uno de los más conocidos dentro de este área y suele ser muy recomendado por la calidad de sus lecciones. Por otro lado, hemos seleccionado *fastai course v3* porque está impartido por los creadores de la librería *fastai*, la cual es famosa por su eficacia y por mantenerse actualizada con los últimos algoritmos del estado del arte. Además, el profesor principal de este curso, Jeremy Howard, es uno de los usuarios de Kaggle[3] que ha cosechado más éxito en las competiciones de esta plataforma. A continuación se explican las características básicas de ambos MOOCs:

1. *Programa Especializado Aprendizaje Profundo* (programa de Ng et al.): Es una especialización que consta de 5 cursos que siguen una aproximación tradicional al aprendizaje mediante bloques de teoría y práctica. Está avalada por deeplearning.ai[4] e impartida por Andrew Ng, Kian Katanforoosh y Younes Bensouda Mourri.

2. *fastai course v3* (curso de Howard et al.): Es un curso dividido en dos partes e impartido por Jeremy Howard en colaboración con Rachel Thomas, Sylvain

[3]Kaggle es una comunidad online en la que los usuarios comparten conjuntos de datos de diversos problemas de aprendizaje automático, colaboran entre sí y, también, se organizan competiciones para resolver algunos de estos problemas. `https://www.kaggle.com`.

[4]deeplearning.ai es una compañía creada por Andrew Ng centrada en la docencia sobre IA con el objetivo de hacerla accesible a personas de todo el mundo. `https://www.deeplearning.ai/`

Gugger y Chris Lattner. Se basa en la experimentación y la programación como mecanismo de aprendizaje. La primera parte sigue una aproximación "de arriba a abajo" (*top-bottom*), la cuál parte explicando los conceptos más abstractos (en la primera lección ya se resuelve un problema de clasificación de imágenes) y, posteriormente, éstos se refinan para ir introduciendo la fundamentación teórica. Por el contrario, la segunda parte sigue la aproximación contraria, implementando primero los mecanismos básicos y, posteriormente, integrándolos poco a poco para desarrollar algoritmos más complejos, con el objetivo de construir una librería de aprendizaje profundo similar a fastai.

Para ambos cursos es necesario tener un conocimiento básico de programación en Python. El libro de Downey [2015] puede servir como introducción a dicho lenguaje tanto para aquellas personas que nunca han programado, como para aquellas que tienen experiencia de programación en otros lenguajes. Así mismo, para profundizar en los conocimientos sobre Python, la documentación sobre la librería estándar [Python Software Foundation 2020a] y el tutorial oficial del lenguaje [Python Software Foundation 2020b] pueden ser de utilidad.

A continuación ofrecemos un resumen de cada uno de estos cursos. Cabe decir que ambos cursos son susceptibles de actualización o es posible que salgan nuevas versiones tras escribir estas líneas, por lo que la información aquí expuesta podría no reflejar la realidad de los mismos en un futuro. De hecho, el curso de Howard et al. ya ha sufrido dos actualizaciones desde su inicio, siendo "v3" (contenida en *fastai course v3*) la última versión, publicada en 2019.

El resto del capítulo se organiza de la siguiente manera: en las secciones 3.3 y 3.4 analizamos los cursos de Ng et al. y de Howard et al., respectivamente, y en la sección 3.5 se realiza una comparación entre ambos.

3.3 Programa de Ng et al. en Coursera

El *Programa Especializado Aprendizaje Profundo* (programa de Ng et al.)[5], creado en 2017, es un conjunto de cursos impartidos en Coursera[6] por Andrew Ng, Kian Katanforoosh y Younes Bensouda Mourri, avalado por deeplearning.ai. Consta de 5 cursos:

1. *Neural Networks and Deep Learning* (4 semanas).

2. *Improving Deep Neural Networks: Hyperparameter tuning, Regularization and Optimization* (3 semanas).

3. *Structuring Machine Learning Projects* (2 semanas).

4. *Convolutional Neural Networks* (4 semanas).

5. *Sequence Models* (3 semanas).

[5] `https://www.coursera.org/specializations/deep-learning`
[6] `https://coursera.org/`

Cada uno de estos cursos está dividido en "semanas", organizando las lecciones según lo que la persona interesada debería estudiar cada 7 días, aunque dicha organización es meramente orientativa. El principal medio de impartir la parte teórica es a través de vídeos, que a veces se complementan con lecturas. La parte práctica consiste en unos ejercicios de programación al final de cada semana. Adicionalmente, también hay cuestionarios para evaluar los conocimientos teóricos. Respondiendo los cuestionarios y los ejercicios de programación se puede optar a obtener un certificado de haber superado con éxito el curso.

Para los ejercicios de programación Coursera proporciona su propia plataforma de Jupyter Notebook[7], por lo que no es necesario llevar a cabo ninguna instalación en el ordenador personal ni contratar un servicio de computación en la nube. Para realizar estas tareas se utiliza el lenguaje de programación Python, junto con las librerías NumPy [Oliphant 2006][8], TensorFlow [Abadi et al. 2016][9] y Keras[10], principalmente. De TensorFlow se usa la versión 1. Recientemente ha salido la versión 2, pero dado que esta última toma prestada la sintaxis de Keras, que se estudia en el curso, la transición a TensorFlow 2.0 no tiene por qué ser muy complicada. En [Rosenbrock 2019] se indican las principales diferencias entre ambas versiones y se describe cómo hay que usar actualmente Keras a través de TensorFlow 2.0.

Otros recursos que ofrece la especialización son: (1) un foro donde realizar preguntas, discusiones técnicas, notificación de errores, etc; y (2) direcciones de correo donde contactar directamente con los responsables del programa.

Los requisitos mínimos para cursarlo son:

1. ciertas habilidades básicas de programación, especialmente en Python (bucles, condicionales, diccionarios, etc.);
2. conocimientos básicos sobre aprendizaje automático (aprendizaje supervisado, representación de un conjunto de datos mediante una matriz, etc.); y
3. conocimientos básicos sobre álgebra lineal (multiplicación de matrices, operaciones con vectores, etc.).

Para realizar la especialización hay que registrarse en Coursera e inscribirse en la misma, con un coste de 44€ ($49) al mes, aunque podría estar sujeta a cambios en el futuro. Dada la duración estimada de la especialización (4 meses aprox.) y la tarifa indicada, el coste total de realizarla serían unos 176€ (entre 22 y 44€ por curso, dependiendo de cuánto tarde el estudiante en completarlo).

3.3.1 Curso 1: "Neural Networks and Deep Learning"

Resumen del contenido Este primer curso establece los fundamentos básicos del aprendizaje profundo, empezando por definir qué es una RNA y el contexto de esta

[7] Jupyter Notebook es un entorno web de desarrollo orientado a la experimentación interactiva sobre ciencia de datos. Los principales lenguajes de programación que soporta son Julia, Python y R.

[8] `https://numpy.org/`

[9] `https://www.tensorflow.org/`

[10] `https://keras.io`

disciplina dentro del aprendizaje automático. Las RNAs estudiadas en este curso pertenecen al denominado aprendizaje supervisado, es decir, "aprenden" usando pares de configuraciones $(\mathbf{x}, \mathbf{y})$ donde $\mathbf{x}$ es la entrada e $\mathbf{y}$ indica el valor correcto que debe devolver la red. Así mismo, pueden trabajar potencialmente con dos grandes grupos de datos: (1) datos estructurados y (2) datos no estructurados. Los primeros son aquellos que vienen en formato de "tabla", como en una base de datos o una hoja de cálculo, mientras que los segundos son aquellos que no tienen la información organizada, como archivos de audio, imágenes o texto. Históricamente, el aprendizaje automático ha tenido mejores resultados en problemas con datos estructurados, mientras que el tratamiento de los datos no estructurados ha sido mucho más difícil. Un punto clave del aprendizaje profundo es que es capaz de tratar más satisfactoriamente éstos que los algoritmos de aprendizaje automático clásicos.

Un aspecto fundamental de una RNA es la arquitectura que la define, es decir, cómo se organizan sus estructuras internas. Como punto de partida para entender estas arquitecturas y los elementos internos que las componen, se estudia un modelo de regresión logística, debido a que se asemeja a un perceptrón, el modelo más simple de RNA. Ambos procesan un conjunto de entradas aplicando primero una función lineal y después una no lineal, produciendo un único valor numérico de salida. Es también fundamental conocer los dos tipos de variables en que se divide cualquier RNA: (1) *parámetros*, es decir, los valores entrenables que definen la función que aproxima la red; y (2) *activaciones*, que consisten en las entradas de la red, junto con los resultados intermedios y finales del proceso de computación.

Definir únicamente la arquitectura de un modelo no es suficiente para resolver un problema. Por ello, en el curso se explica el proceso de aprendizaje (ajuste de los parámetros) al que hay que someter a dicho modelo para que aproxime la función que modeliza los datos. Este proceso consta de tres elementos básicos:

1. una *función de pérdida*, que mide el error entre la salida de la red y el valor correcto que debería devolver,
2. el cálculo de los *gradientes* de la función de pérdida con respecto a los parámetros de la red y
3. un *optimizador*, que se encarga de modificar los parámetros de la red usando los valores de los gradientes previamente calculados, reduciendo así el error cometido por la red durante el proceso.

Cabe decir que las librerías de aprendizaje profundo suelen tener un mecanismo para calcular de forma automática los gradientes, basado en un grafo de computación. Aun así, en este curso se realizan estos cálculos manualmente para una mejor comprensión de este proceso. La función de pérdida usada en un modelo depende del problema que se desea resolver. Como punto de partida, en el curso se estudia la función de *entropía cruzada binaria* (*binary cross entropy*), que permite resolver problemas de clasificación compuestos por dos clases.

El optimizador es otro componente fundamental para el entrenamiento. Durante el curso se estudia el algoritmo de optimización básico para el entrenamiento de RNAs, denominado descenso del gradiente (*gradient descent*, GD). Éste usa los gradientes previamente

calculados para modificar los parámetros del modelo, con el objetivo de ir reduciendo el error global de la red. Este algoritmo se configura mediante un hiperparámetro, denominado *tasa de aprendizaje*, que determina la magnitud de la variación de los parámetros cada vez que se actualizan durante el entrenamiento. Su correcta configuración es fundamental para realizar con éxito un entrenamiento. Si el valor de la tasa de aprendizaje es muy alto, las variaciones del modelo se vuelven muy bruscas, impidiendo que el descenso del gradiente pueda alcanzar un error mínimo. En cambio, si la tasa de aprendizaje es muy baja, el proceso de optimización se vuelve muy lento.

Por otro lado, en el curso se hace especial hincapié en el concepto de vectorización, una técnica de programación que permite deshacerse de bucles explícitos (1) agrupando las variables en vectores, matrices o tensores y (2) usando operaciones diseñadas para trabajar con estructuras de este tipo. Esta forma de programar reduce significativamente el coste computacional de los algoritmos, dando lugar a tiempos de ejecución aceptables para muchos problemas.

Seguidamente, partiendo de la regresión logística estudiada, se introduce el concepto de unidad neuronal, el componente básico de una RNA. Cada RNA se compone de multitud de unidades de este tipo. Por ello, el curso detalla cómo estructurar internamente las redes usando dichas unidades. Primero, especifica cómo construir una capa de una RNA combinando unidades neuronales, es decir, conectando todas las entradas de la capa a cada unidad y tomando los valores de salida de cada una de éstas como la salida de la propia capa. Este tipo de capa, en el contexto de aprendizaje profundo, se suele denominar *capa completamente conexa* (*fully connected layer*) o *densa* (*dense layer*). Una RNA puede estar compuesta por varias capas de este tipo. Por ello, posteriormente, se explica cómo concatenar capas para construir redes más complejas. Habitualmente, se denomina a la última capa de una red como *capa de salida* y todas las anteriores como *capas ocultas*. A veces las variables de entrada de la red se consideran como una capa especial, denominada *capa de entrada*. Generalmente, si una de estas redes tiene tres o más capas se puede considerar "profunda".

Una de las claves por las que las RNAs son capaces de aproximar funciones complejas es precisamente por el uso que hacen de funciones no lineales durante su computación. De no utilizarlas, cualquier red quedaría reducida a una simple función lineal. Por ello, en el curso se estudia el concepto de *función de activación*, explicando cuatro de las más importantes: (1) función sigmoide, (2) tangente hiperbólica (tanh), (3) unidad lineal rectificada (*Rectified Linear Unit*, ReLU), y (4) leaky ReLU. Actualmente, la más usada para las capas ocultas es algún tipo de función ReLU. En cambio, para la capa de salida, se escoge una función que depende del problema que se desea resolver. Hemos señalado dos elementos de una red que dependen del tipo de tarea: (1) la función de activación de la capa de salida y (2) la función de pérdida. Para un profesional del aprendizaje profundo es fundamental conocer qué combinaciones de estos dos elementos conviene usar para cada tipo de problema. En cursos posteriores se estudian con más detalle estas relaciones.

Recomendaciones para el estudio del curso 1 Este curso no tiene como requisito estar familiarizado con el cálculo de derivadas o gradientes, ya que se estudian durante el transcurso del mismo; pero dado que son una parte fundamental en el entrenamiento de

RNAs, aquellas personas interesadas en adquirir los conocimientos básicos sobre derivación pueden encontrar en [Khan Academy 2017] mucho material sobre este tema. Sin embargo, tener conocimientos básicos sobre derivadas y gradientes no es suficiente para entender los cálculos realmente realizados durante el entrenamiento de una RNA. Para adquirir más conocimientos sobre el cálculo de derivadas necesario para el aprendizaje profundo, [Parr y Howard 2018; Barnes 2006] pueden ser un buen punto de partida. Como alternativa al curso, en [McDonald 2017; Pandey 2019] se explica de manera introductoria el algoritmo de optimización del descenso del gradiente. Para personas sin conocimientos de programación o programadores que no han usado nunca la vectorización y las operaciones que implica, éstas pueden resultar algo confusas al principio. Por ello, en [SciPy community 2019a,b; Varoquaux et al. 2017] se puede encontrar información adicional sobre operaciones de vectores, como *broadcasting* y *slicing* en NumPy, entre otras. Éstas reglas suelen ser las mismas o parecidas para cualquier librería de aprendizaje profundo. En [Sharma 2017; Sharma 2019] se pueden encontrar explicaciones alternativas sobre algunas de las funciones de activación más usadas y en [Ronaghan 2018] se hace un resumen sobre qué funciones de activación y funciones de pérdida usar para ciertos problemas típicos de aprendizaje profundo.

Referencias al curso de Howard et al. En la lección 8 (sección 3.4.8 de esta memoria) se implementan los pasos hacia delante (*forward pass*) y hacia atrás (*backward pass*) de una RNA de dos capas usando PyTorch. En la lección 9 (sección 3.4.9) se implementa el bucle de entrenamiento básico.

3.3.2 Curso 2: "Improving Deep Neural Networks: Hyperparameter tuning, Regularization and Optimization"

Resumen del contenido El primer punto que se estudia en este curso es la división de los datos en los dos o tres subconjuntos necesarios para construir un modelo: (1) *conjunto de entrenamiento* (*training set*), (2) *conjunto de desarrollo* (*development set*), también denominado *conjunto de validación* (*validation set*), y (3) *conjunto de prueba* (*test set*). Se indica la importancia de incluir la mayoría de los datos en el primero, ya que éstos son los que modifican realmente los parámetros del modelo. Además, los datos de los otros dos deben proceder de la misma distribución[11]. Esta distribución ha de ser lo más semejante posible a la del entorno de producción, dado que tanto el conjunto de validación como el de test se usan para comprobar la calidad del modelo.

A continuación se introduce el *compromiso entre preferencia y varianza* (*bias-variance tradeoff*), también conocido como el *problema de preferencia-varianza* (*bias-variance problem*)[12]. Básicamente consiste en encontrar un equilibrio tal que el modelo es capaz de aprender los datos de entrenamiento pero, a su vez, consigue generalizar el problema de forma que puede procesar correctamente datos no incluidos en el conjunto de

[11] El concepto de "distribución" procede de la probabilidad y la estadística. En este contexto, una distribución es sinónimo de origen de datos. Por ejemplo, imágenes tomadas en un mismo entorno y contexto, textos procedentes del mismo corpus bibliográfico, etc. La similitud de las distribuciones de los datos depende del problema concreto.

[12] En este contexto, "bias", que también se podría traducir como "inclinación" o "prejuicio", se refiere a la preferencia a priori de unos modelos frente a otros en el aprendizaje.

entrenamiento. Usando los conjuntos de entrenamiento y de validación se realiza un *análisis de preferencia-varianza* (*bias-variance analysis*) para determinar el estado en que se encuentra el modelo, comparando el error de entrenamiento con el error de validación. Hay tres estados posibles: (1) *preferencia elevada* (*high bias*) o *sub-ajuste* (*underfitting*), en el que el modelo aprende poco de los datos de entrenamiento; (2) *varianza elevada* (*high variance*) o *sobreajuste* (*overfitting*), en el que el modelo generaliza mal los resultados; y (3) un *equilibrio correcto entre preferencia y varianza*, que es el estado ideal. Se estudia cómo realizar un análisis de preferencia-varianza con el que determinar en qué punto se encuentra el modelo. Posteriormente se exponen de forma general diferentes técnicas para reducir la preferencia y la varianza de una RNA.

Durante el curso se estudian diferentes técnicas de regularización, que son un conjunto de estrategias para prevenir el sobreajuste del modelo:

1. *Regularización L2*, también llamada *declive de los pesos* (*weight decay*): Consiste en sumar a la función de coste los parámetros del modelo ponderados por un nuevo hiperparámetro. Al optimizar esta nueva función se reducen los valores de algunos de estos parámetros, haciendo que la red pierda parte de su capacidad de aprendizaje.

2. *Eliminación de enlaces* (*dropout*) [Srivastava et al. 2014]: En cada iteración del algoritmo de aprendizaje se eliminan aleatoriamente ciertas neuronas de la red, de forma que cada lote de datos[13] se usa únicamente para ajustar los parámetros de las neuronas restantes. Esto evita que partes concretas de la red se especialicen en reconocer patrones específicos de los datos.

3. *Data augmentation*: Se basa en generar de forma artificial nuevas muestras aplicando transformaciones aleatorias a los datos reales. Estas variaciones se realizan de forma que siga siendo válido el etiquetado original. Por ejemplo, al girar una imagen, el objeto representado sigue siendo el mismo aunque su perspectiva ha cambiado.

4. *Early stopping*: Consiste en detener el entrenamiento en un punto intermedio del mismo. El objetivo es que esta parada se produzca antes de que el error de validación[14] empiece a ser significativamente mayor que el de entrenamiento.

Después se estudia la normalización de los datos de entrada, que consiste en sustraerles su media y dividir el resultado por la desviación típica. Este proceso hace que los entrenamientos sean más rápidos debido a que se reduce la dispersión de los datos. Hay que usar los estadísticos (media y desviación típica) del conjunto de entrenamiento para normalizar también los conjuntos de desarrollo y prueba. Si no se hace, los datos de validación y prueba no estarán en la "posición" adecuada dentro del espacio vectorial para el que se ha entrenado la red y no serán procesados correctamente. Esto también es aplicable a los datos en el entorno de producción.

A continuación el curso introduce el *problema del desvanecimiento y la explosión de los gradientes*, una de las principales dificultades que impedía hasta hace pocos años el entrenamiento de DNNs. Éste consiste en que, dependiendo de los valores de los parámetros de la red, las activaciones y los gradientes o (1) tienden a cero o (2) se vuelven tan grandes

[13] Un lote es un subconjunto de muestras del conjunto de entrenamiento. Ver el *descenso del gradiente por mini-lotes* más adelante en este mismo tema para más información.

[14] El error que mide la función de pérdida al evaluar la red el conjunto de validación.

que desbordan la memoria asignada a las variables. Estos resultados no deseados se producen al procesar multitud de capas con valores inadecuados en sus parámetros. Hay varios mecanismos de inicalización de parámetros que permiten solventar el problema. En este curso se estudian dos: la *inicialización de Xavier* (*Xavier initialization*) [Glorot y Bengio 2010], y la *inicialización de Kaiming* (*Kaiming initialization*) [He et al. 2015b].

También se estudian algunos algoritmos que permiten minimizar la función de coste, conocidos como optimizadores. En el curso 1 (sección 3.3.1) ya se ha estudiado el algoritmo básico, el descenso del gradiente. Aquí se explican variaciones de éste:

1. *Descenso del gradiente estocástico* (*stochastic gradient descent*, SGD): Los parámetros del modelo se actualizan para cada una de las muestras del conjunto de entrenamiento. Permite entrenar usando conjuntos con muchas muestras, ya que no necesita cargarlas todas en memoria al mismo tiempo. Converge más lentamente que el algoritmo original, ya que los gradientes son actualizados por cada muestra y no para todo el conjunto de entrenamiento.

2. *Descenso del gradiente por mini-lotes* (*mini-batch gradient descent*): Es un punto intermedio entre el descenso del gradiente original y el SGD. Divide aleatoriamente el conjunto de datos en lotes de longitud fija. Los parámetros del modelo se actualizan para cada uno de los lotes. Permite un uso más eficiente de la memoria que el descenso del gradiente y converge más rápido que SGD.

3. *Descenso del gradiente con momento*[15] (*gradient descent with momentum*), *RMSprop*[16], y *Adam*[17]: Estos optimizadores utilizan el *promedio móvil ponderado exponencialmente* (*exponentially weighted moving average*, EWMA) para calcular los valores medios de los gradientes de la función de coste con respecto a los parámetros del modelo y los usan para actualizar los valores de los parámetros. Esto hace que el "camino" hacia el óptimo global sea más directo, permitiendo valores para la tasa de aprendizaje más grandes y, por ello, acelerando la convergencia. Uno de los optimizadores más eficientes de la actualidad es Adam [Kingma y Ba 2014], que combina *momentum* y RMSprop.

Posteriormente el curso explica una técnica denominada *learning rate decay*, que consiste en ir reduciendo la tasa de aprendizaje a medida que avanza el entrenamiento. Los valores altos de este hiperparámetro al principio del entrenamiento permiten explorar el espacio de búsqueda con más rapidez y, al final del mismo, los valores bajos ayudan a realizar los ajustes finos para acercarse lo más posible al mínimo global.

A continuación se estudia la *normalización por lotes* (*batch normalization*, BatchNorm) [Ioffe y Szegedy 2015], otra de las técnicas fundamentales del aprendizaje profundo. Se basa en ajustar la media y la desviación típica de las activaciones internas de una red usando parámetros entrenables para determinar los mejores valores para estos estadísticos. Es decir, se cambian la media y la desviación típica de las activaciones de entrada de cada capa BatchNorm a unos valores determinados durante el entrenamiento del modelo.

[15] En este contexto, el "momento" es una metáfora del momento lineal de la física, que es proporcional a la velocidad.

[16] El término "RMSprop" viene de *Root Mean Square backpropagation.*

[17] El término "Adam" viene de *adaptive moment estimation.*

Sirve para minimizar el impacto del desplazamiento de las covariables (*covariate shift*)[18] y tiene cierto efecto de regularización. En definitiva, hace la red más robusta, y simplifica y acelera el entrenamiento.

Después, el curso introduce un nuevo tipo de problema denominado *clasificación multi-clase*, en el que hay tres o más clases presentes. Como ya se ha visto, cada tipo de problema necesita una combinación concreta de función de activación para la última capa y una función de pérdida. En este caso, lo más habitual es usar softmax[19] como función de activación y la *entropía cruzada* (*cross entropy*)[20] como función de pérdida.

Recomendaciones para el estudio del curso 2 Para conseguir entrenar correctamente una DNN es fundamental entender bien en qué consiste el compromiso entre preferencia y varianza. En [Singh 2018; Fortmann-Roe 2012] se dan introducciones alternativas sobre el tema, que pueden servir como complemento a la explicación del curso. La alta complejidad de las DNNs suele implicar riesgo de sobreajuste, por lo que es necesario entender y saber trabajar con técnicas de regularización. En [Jain 2018] se explican la regularización L1 (que no se estudia en la especialización de Ng et al.) y L2, *dropout*, *data augmentation* y *early stopping*, aplicando estas técnicas a un caso práctico con Keras. El componente clave sobre el que pivotan las modificaciones de los optimizadores del descenso del gradiente con momento, RMSprop y Adam es EWMA. En [Mahapatra 2018] se explican los fundamentos de esta técnica.

BatchNorm fue propuesta por Ioffe y Szegedy [2015], aunque a posteriori se demostró que la justificación matemática que ofrecieron estos autores era incorrecta. Santurkar et al. [2018] lograron una explicación más acertada. Una de las finalidades de BatchNorm es reducir el desplazamiento de las covariables pero este concepto no se explica en el curso de Ng et al. En [Izadi 2017; Kopczyk 2019] se introduce dicha idea.

Referencias al curso de Howard et al. En la lección 2 (sección 3.4.2 de esta memoria) se introduce el análisis de preferencia-varianza y se lleva a cabo una implementación del descenso del gradiente. Además de la clasificación multi-clase existe la clasificación multi-etiqueta, que consiste en asignar varias etiquetas a una misma muestra. Esta tarea se explica en la lección 3 (sección 3.4.3). En la lección 5 (sección 3.4.5) se introduce la regularización L2 (declive de los pesos); su implementación se muestra en la lección 11 (sección 3.4.11). Además, en la lección 5, se explican los optimizadores SGD, *momentum*, RMSprop y Adam, mientras que en la lección 11 se implementa cada uno de ellos tomando como base un optimizador genérico. En esta última lección también se presenta un

[18] En aprendizaje automático, el desplazamiento de las covariables consiste en la diferencia de distribución de las variables de entrada entre los datos de entrenamiento y prueba. Esto suele llevar a un incremento del error al evaluar el conjunto de prueba, en comparación con los datos de entrenamiento. Por ejemplo, si los datos de entrenamiento únicamente contienen coches rojos y los datos de prueba sólo coches verdes, es muy probable que el error cometido con estos últimos sea mayor debido a que durante el entrenamiento no se ha contemplado que "los coches pueden ser verdes".

[19] La función softmax toma como entrada un vector con n números reales y lo normaliza transformándolo en una distribución de probabilidad consistente en n probabilidades proporcionales a los exponenciales de los valores de entrada.

[20] También conocida como *verosimilitud logarítmica negativa* (*negative log likelihood*).

nuevo optimizador, no estudiado en el curso de Ng et al., denominado LAMB. En la lección 5 también se explica la tarea de clasificación multi-clase, junto con las funciones softmax y entropía cruzada, mientras que en la lección 9 (sección 3.4.9) se realiza una implementación de dichas funciones. En la lección 10 (sección 3.4.10) se dan más detalles sobre el funcionamiento de la función softmax. En la lección 6 (sección 3.4.6) se presentan las técnicas de regularización *dropout* y *data augmentation*. Además, se explica BatchNorm, mientras que su implementación se detalla en la lección 9, junto a otros mecanismos para la normalización por lotes. En la lección 11 se realiza una implementación de *data augmentation* para problemas de visión artificial y en la lección 12 se introducen dos mecanismos de regularización no explicados en este curso, llamados *mixup* y *label smoothing*. En la lección 9 (sección 3.4.9) se implementan las inicializaciones de Xavier y de Kaiming y, además, se estudian variaciones de éstas. En la lección 11 se continúa con el problema de la inicialización de los parámetros, presentando un nuevo mecanismo llamado LSUV.

3.3.3 Curso 3: "Structuring Machine Learning Projects"

Resumen del contenido El desarrollo de una aplicación de aprendizaje automático no es una tarea fácil debido a que hay muchos objetivos dependientes entre sí. Por ello, en este curso se estudian metodologías y técnicas que permiten detectar las fuentes de error más importantes y actuar para reducirlas. La primera estrategia que se estudia se denomina *ortogonalización* (*orthogonalization*). Consiste en abordar secuencialmente cada uno de los objetivos, de forma que las modificaciones que se realicen en el sistema para alcanzar un objetivo no afecten negativamente a los demás. Por otro lado, se indica la importancia de poder medir la calidad de los modelos mediante una única métrica, de forma que sea fácil compararlos entre sí. Se estudia la *métrica F1*, que reúne en un mismo valor tanto la precisión (*precission*) como la exhaustividad (*recall*) del modelo. También se dan alternativas para cuando no es posible determinar la calidad de los modelos con una única métrica.

A continuación el curso profundiza en la idea de controlar la distribución de la que proceden los datos de entrenamiento, validación y prueba, introducida ya en el curso 2. Para el conjunto de entrenamiento, se suelen utilizar diferentes fuentes, ya que las DNNs necesitan una gran cantidad de datos para ser entrenadas. En cambio, es fundamental que los datos para los conjuntos de validación y prueba procedan de las mismas fuentes, lo más parecidas posibles al entorno de producción del sistema. Esto se debe a que son los que marcan el objetivo final del sistema, es decir, el problema que realmente tiene que resolver. Por ejemplo, si se desea crear un sistema de reconocimiento facial para dar acceso a un determinado lugar, podemos usar imágenes de personas obtenidas a través de internet para construir nuestro conjunto de entrenamiento. En cambio, es conveniente que los datos de validación y prueba se capten con un modelo de cámara y en unas condiciones de iluminación lo más similares al entorno de producción.

En el curso se estudian dos niveles de calidad de los resultados, que se establecen como objetivo a alcanzar en el desarrollo de un sistema de aprendizaje automático:

1. *Rendimiento a nivel humano*: Se considera como aceptable el error que cometería una persona al intentar resolver el problema.

2. *Bayes optimal error*: En el caso de que el sistema pueda superar la precisión humana,

hay un cierto límite de rendimiento teórico que nunca va a poder superar.

Dependiendo del problema, se puede establecer como objetivo uno de estos dos niveles de error. Determinar adecuadamente este límite de error o rendimiento es fundamental para un correcto análisis de preferencia-varianza.

Posteriormente se estudia el *análisis manual del error*, una técnica que permite detectar aquellos tipos de muestras con las que el sistema suele equivocarse más. Al centrarse en corregir el error que éstas provocan, se puede optimizar el sistema.

Todas las técnicas estudiadas previamente se aglutinan en el uso de una metodología similar a un desarrollo iterativo e incremental, o *desarrollo ágil*, que se resume en (1) identificar la fuente de error más importante, (2) focalizar el esfuerzo para reducirla, y (3) volver al primer paso hasta que el error sea suficientemente pequeño.

La diferencia entre las distribuciones de las que proceden los datos de entrenamiento y los de validación y prueba afecta al análisis de preferencia-varianza de forma que es muy difícil interpretar cuál es la fuente de error. En el curso se estudia cómo realizar correctamente dicho análisis usando un cuarto conjunto para validar el modelo con datos procedentes de la misma distribución que el conjunto de entrenamiento. Así mismo, se define un procedimiento con el que resolver la discrepancia entre los datos de entrenamiento y los de validación y prueba.

Posteriormente, se estudia el *aprendizaje por transferencia* (*transfer learning*), una técnica que permite usar el conocimiento aprendido en un problema como punto de partida para resolver un problema distinto. Se basa en reutilizar los parámetros de una red como valores iniciales de la segunda. Para ello es necesario que ambos problemas traten con datos del mismo tipo: audio, texto, imágenes, etc. El procedimiento básico del aprendizaje por transferencia consiste en:

1. Cargar un determinado modelo y ajustar su entrada y salida (cabeza[21]) para que se adapten a los requerimientos del problema a resolver (dimensiones de las imágenes de entrada, número de clases, etc.).

2. Inicializar el cuerpo del modelo usando un conjunto de pesos pre-entrenados. Estos pesos proceden de una red previamente entrenada que resuelve un problema similar al que se quiere abordar. La nueva cabeza de la red se inicializa aleatoriamente.

3. "Congelar" (*freeze*) el cuerpo del modelo y entrenar únicamente la cabeza o salida. Es decir, se configuran las capas del cuerpo para que sus parámetros no se modifiquen durante el entrenamiento. De esta forma únicamente se ajustan las capas que se han inicializado aleatoriamente.

4. "Descongelar" (*unfreeze*) todas las capas del modelo y llevar a cabo un entrenamiento que realice los ajustes finos. De esta forma se modifican los parámetros de todas

[21]No hay una terminología oficial para referirse a las diferentes partes de una DNN, aunque es ampliamente usado el término cabeza (*head*) para referirse a las últimas capas de una red, aquellas encargadas de adaptar las activaciones al formato de salida esperado (un vector de longitud igual al número de clases, por ejemplo). El resto de capas de la red se suelen denominar cuerpo (*body*). Adicionalmente, hay quien llama raíz (*stem*) a las primeras capas del cuerpo, las más cercanas a la entrada de la red.

las capas, incluidos los pesos pre-entrenados, para que se adapten mejor al nuevo problema.

Seguidamente, se introduce el *aprendizaje multi-tarea*, que consiste en entrenar una única red para desempeñar varias tareas de forma simultánea. Por ejemplo, para un vehículo con conducción autónoma, una única red podría ser capaz de detectar peatones, coches, señales de tráfico, semáforos, etc. Hipotéticamente, cada una de éstas tareas debería ayudar a la realización de las demás.

Por último, se explica el *procesamiento de múltiples etapas* (*multiple stages processing*) y cómo el aprendizaje profundo ha sustituido este tipo de sistemas por una única red. Esto se denomina *aprendizaje profundo extremo-a-extremo* (*end-to-end deep learning*) y tiene la ventaja de aunar el proceso en un único modelo. Por ejemplo, antiguamente en visión artificial, para conseguir un clasificador de imágenes, era necesario encadenar diferentes extractores de características y algoritmos de aprendizaje automático. En cambio, hoy en día, una única DNN puede realizar el mismo trabajo de forma más eficaz y eficiente. También se estudia la aproximación intermedia en la que se realizan varias etapas con DNNs y, opcionalmente, otros algoritmos de aprendizaje automático. Esto suele simplificar el problema debido a que (1) la suma de la complejidad de las etapas es menor que el problema global, y (2) hay muchos más datos para resolver los sub-problemas de las etapas que para el problema global.

Recomendaciones para el estudio del curso 3 Muchos de los temas tratados durante este curso se refieren a la planificación y gestión de un proyecto de aprendizaje automático. Por ello, es recomendable llevar a cabo un pequeño proyecto de prueba en el que poner en práctica las pautas indicadas en el curso. Por ejemplo, el alumno podría obtener algún conjunto de datos de libre acceso a través de internet e intentar construir un modelo capaz de solucionar su problema asociado. Quizás para practicar más sobre el aprendizaje profundo sería conveniente conocer previamente las ConvNets, actualmente las redes más desarrolladas en el aprendizaje profundo, que se estudian en el curso 4, y realizar el proyecto de prueba basado en redes de este tipo.

En [Ping Shung 2018] se explican las métricas precisión, exhaustividad y F1. Dado que en este curso no se realiza ninguna práctica de aprendizaje por transferencia, recomendamos consultar [Ananthram 2018], que explica cómo aplicarlo utilizando Keras.

Referencias al curso de Howard et al. En la lección 12 (sección 3.4.12) se implementa un proceso de aprendizaje por transferencia para problemas de visión artificial y otro distinto para clasificación de textos. En este último caso se usa un procedimiento llamado ULMFiT (*Universal Language Model Fine-Tuning for text classification*).

3.3.4 Curso 4: "Convolutional Neural Networks"

Resumen del contenido En este curso se estudian las *redes neuronales convolucionales* (*convolutional neural networks*, ConvNets o CNNs) que engloban un conjunto de arquitecturas neuronales utilizadas para resolver problemas de visión artificial. Lo primero

que se explica es la operación de convolución[22], de la que estas redes reciben su nombre. Tradicionalmente, éstas usan un filtro con valores fijos para procesar la entrada; la innovación del aprendizaje profundo es convertir dichos valores en parámetros ajustables, de forma que es el proceso de entrenamiento el que se encarga de encontrar los filtros adecuados. Además del filtro, estas operaciones constan de dos hiperparámetros configurables[23] que modifican su comportamiento: (1) *relleno* (*padding*), que establece un marco de cierto grosor (similar al marco de una fotografía), habitualmente compuesto por ceros, alrededor de las activaciones de entrada; y (2) *zancada* (*stride*), que indica el tamaño del paso con el que el filtro recorre las activaciones de entrada. Generalmente, si se usa relleno es para conseguir que las dimensiones de la salida sean iguales a las de la entrada.

Las ConvNets tienen tres tipos de capas básicas:

1. *Capa convolucional* (*convolutional layer*): Está formada por múltiples filtros entrenables con los que se realizan operaciones de convolución.

2. *Pooling layer*: Se encarga de reducir el tamaño de las activaciones con el objetivo de acelerar la computación y hacer algunas de las características detectadas más robustas. Constan de un filtro que recorre las activaciones y reduce los valores de cada posición. Dependiendo de la operación de reducción que usan, se clasifican en dos tipos: (1) *max pooling*, que calcula el valor máximo; o (2) *average pooling*, que calcula el valor medio.

3. *Capa completamente conexa* (*fully connected layer*, *dense layer*): Conecta todas las entradas con todas las unidades neuronales de la misma. Es la capa estándar que usan las RNAs convencionales.

Combinando capas de estos tres tipos se construyen la mayoría de ConvNets. En ConvNets complejas se usan, además, otro tipo de estructuras.

Aun usando únicamente los tres tipos básicos de capas, no es fácil diseñar una arquitectura. En este curso se estudian ciertos modelos habituales de ConvNets:

1. *LeNet-5* [LeCun et al. 1998]: Uno de los primeros diseños de ConvNet. Es una red sencilla para los estándares actuales, pero introdujo ciertos patrones que hoy en día siguen en vigor.

2. *AlexNet* [Krizhevsky et al. 2012]: Es similar a LeNet-5 pero más compleja. Utiliza la función de activación ReLU. La aparición de esta red convenció a la comunidad académica de que el aprendizaje profundo realmente funciona para problemas de visión artificial debido a su éxito en la competición de ImageNet de 2012.

3. *VGG-16* [Simonyan y Zisserman 2015]: Simplifica el diseño de ConvNets usando únicamente dos tipos de capas con su configuración prefijada, reduciendo el número de hiperparámetros a ajustar.

[22]En cierta bibliografía matemática a esta operación se la denomina *correlación cruzada*. En dicho contexto, para realizar una convolución hay que rotar el filtro a la vez en el eje vertical y horizontal previamente. Por convención, en el aprendizaje profundo no se realiza dicho paso previo, pero aun así se siguen denominando convoluciones.

[23]Hay un tercer hiperparámetro, denominado *dilatación* (*dilation*) que no se estudia en el curso. En caso de ser usado, expande el filtro separando los parámetros en las dimensiones de altura y anchura de forma equidistante y rellenando las posiciones intermedias con ceros.

4. *Residual Network* (ResNet) [He et al. 2015a]: Introduce el concepto de *skip connection* que permite entrenar redes más profundas. Esta *skip connection* consiste en un "camino secundario" de computación que conecta las entradas de un bloque (un conjunto de capas consecutivas) a las salidas del mismo, sumándolas.

5. *Inception Network* [Szegedy et al. 2014]: Se basa en aplicar múltiples capas convolucionales y de *pooling* a las mismas activaciones de entrada y concatenar sus resultados para generar la salida. También usan unas capas especiales, denominadas convoluciones 1×1 [Lin et al. 2013], para reducir el número de canales de un volumen de activaciones. A estas capas también se las denomina *network in network* debido a que actúan como una RNA interna.

Estos 5 tipos de redes están diseñados para resolver tareas de clasificación de imágenes, pero existen otros problemas dentro de la visión artificial a los que se han aplicado ConvNets con éxito. El curso explica la tarea de *detección de objetos*, que consiste en localizar los diferentes elementos de interés que aparecen en una imagen. Antes de abordar este problema, el curso introduce ciertos conceptos necesarios como: (1) la tarea de *clasificación con localización*, que consiste en localizar un único objeto en una imagen; (2) *bounding box*, un rectángulo que se usa para señalar un objeto detectado; (3) *sliding windows detection*, un algoritmo que permite detectar múltiples objetos en una imagen; y (4) la implementación convolucional de *sliding windows detection* [Sermanet et al. 2014], mucho más eficiente que la versión original. Posteriormente, en el curso se estudia el algoritmo *You Only Look Once* (YOLO) [Redmon et al. 2015; Redmon y Farhadi 2016, 2018], el cual es capaz de ajustar las *bounding boxes* alrededor de los objetos con mayor precisión y de forma más eficiente que los algoritmos anteriormente presentados. Alternativamente, el curso también presenta el algoritmo conocido como *region proposal* (R-CNN) [Girshick et al. 2013; Girshick 2015; Ren et al. 2016], que ha sido muy influyente dentro de la literatura de detección de objetos, aunque es algo menos eficiente que YOLO.

Posteriormente, se presentan en el curso dos nuevas tareas relacionadas:

1. *Autenticación facial* (*face verification*) (problema $1:1$): Dada una imagen de un rostro y un identificador, el sistema se encarga de determinar si el rostro pertenece a la persona representada por el identificador.

2. *Reconocimiento facial* (*face recognition*) (problema $1:K$): Dada una imagen del rosto de una persona, el sistema determina si se corresponde con alguno de los K rostros almacenados en su base de datos. En caso contrario, no se reconoce al sujeto.

Para ambas tareas se usa una ConvNet que codifica cada imagen en un vector de características. Los vectores de dos imágenes se comparan mediante una función de similitud o métrica de distancia para determinar si los rostros que contienen pertenecen a la misma persona. Una arquitectura usada habitualmente para estos problemas es *siamese network* [Taigman et al. 2014]. Además, el curso explica la función de pérdida denominada *triplet loss function* [Schroff et al. 2015], necesaria para entrenar una red que resuelva este problema.

La ultima tarea que se estudia en este curso es radicalmente distinta a las vistas anteriormente. Se denomina *neural style transfer* [Gatys et al. 2015], y consiste en extraer el estilo artístico de cierta imagen (una obra pictórica, por ejemplo) y aplicarlo a otra. De

esta forma se obtiene una tercera imagen en la que aparece el contenido de la segunda pero "dibujado" con el estilo artístico de la primera.

Recomendaciones para el estudio del curso 4 Las ConvNets para clasificación de imágenes son el germen del éxito actual del aprendizaje profundo y es conveniente entender lo mejor posible cómo funcionan. Durante el curso se expone mucha información sobre éstas a lo largo de varios vídeos. Sin embargo, a veces es útil tener a mano una versión reducida que explique los puntos clave. [Saha 2018; Ingargiola 2019] son dos guías bastante completas y concisas sobre estas redes. En [Skalski 2019] se explican los cálculos básicos de las capas convolucionales y de *pooling*, tanto para el paso hacia delante como hacia atrás. Adicionalmente, Solai [2018] explica más extensamente los cálculos del paso hacia atrás de las capas convolucionales.

Zeiler y Fergus [2013] explican visualmente cuáles son las características que capturan las sucesivas capas convolucionales de una ConvNet, es decir, muestran qué fragmentos de la imagen son detectados en cada una de ellas. Las ResNets son uno de los tipos de ConvNets más usados actualmente, aunque han ido sufriendo cambios respecto a su diseño original. He et al. [2018] muestran un conjunto de importantes mejoras para esta arquitectura.

En [Johnson 2015] se puede encontrar una implementación de *neural style transfer* [Gatys et al. 2015] junto a algunos resultados obtenidos. Además, Johnson et al. [2016] introducen las *perceptual losses*, unas nuevas funciones de pérdida basadas en una red VGG-16, que permiten resolver problemas de este tipo más rápidamente. Este tipo de función se usa también en algunas implementaciones de *generative models* (véase la lección 7 de fastai course v3, sección 3.4.7).

Referencias al curso de Howard et al. La lección 1 (sección 3.4.1) presenta un ejemplo de clasificación de imágenes usando la librería fastai. La lección 3 (sección 3.4.3) estudia la regresión de imágenes e introduce un problema no visto en el curso de Ng et al.: la segmentación de imágenes. La lección 6 (sección 3.4.6) explica las operaciones de convolución. La lección 7 (3.4.7) estudia la estructura básica de una ConvNet y, posteriormente, de una ResNet. Además, introduce una variación de esta última, denominada DenseNet, que en vez de sumar la *skip connection* a la salida del bloque, la concatena. En la misma lección se estudian (1) las U-Nets, un tipo de arquitectura que sirve para resolver problemas de segmentación de imágenes, (2) las redes generativas antagónicas (GANs) y (3) unas nuevas funciones de pérdida denominadas *perceptual losses* para la generación de imágenes novedosas. En la lección 10 (sección 3.4.10) se codifica una ConvNet simple, mientras que en la lección 11 (sección 3.4.11) se hace lo propio con una versión mejorada de la ResNet, denominada XResNet.

3.3.5 Curso 5: "Sequence Models"

Resumen del contenido Los modelos vistos en los cursos anteriores de esta especialización tratan cada muestra de datos de forma independiente de las demás. Por ejemplo, cualquier ConvNet estudiada en el curso 4 procesa cada imagen de entrada por separado

de las demás, es decir, no busca patrones que relacionen las imágenes que se presentan a la red en instantes de tiempo contiguos. Sin embargo, existe otro tipo de datos que vienen en forma de secuencia, es decir, en los que el orden en que se presentan tiene importancia. Ejemplos de este tipo de datos pueden ser: clips de audio, vídeo, texto, secuencias de ADN. Este curso estudia cómo tratar con datos secuenciales mediante aprendizaje profundo. Se centra especialmente en el *procesamiento del lenguaje natural* (PLN, *natural language processing*, NLP), que es el área en la que más se ha experimentado con este tipo de datos en el aprendizaje profundo, tanto usando textos escritos como discursos orales.

Primero se explica el preprocesamiento básico de textos para PLN. De forma general, consta de los siguientes pasos: (1) crear un *vocabulario* que contenga las palabras o *tokens* de interés ordenados alfabéticamente; y (2) usar el vocabulario para codificar las palabras o *tokens* que contiene usando *one-hot encoding*, vectores de la misma longitud que el vocabulario en los que todas sus posiciones toman valor cero excepto aquella correspondiente a la palabra en cuestión, que toma valor uno.

Como las redes estudiadas en los cursos anteriores no sirven para tratar con datos secuenciales, en este nuevo curso se introducen las *Recurrent Neural Networks* (RNNs), que están específicamente diseñadas para trabajar con secuencias. Siguiendo a Karpathy [2015], se especifican cuatro arquitecturas básicas, que dependen del tipo de entrada y de salida: (1) *many-to-many*, en la que tanto la entrada como la salida son secuencias; (2) *many-to-one*, en las cuales sólo la entrada es una secuencia; (3) *one-to-many*, en que únicamente la salida es una secuencia; y (4) *one-to-one*, en que ni la entrada ni la salida son secuencias, y por tanto son RNAs convencionales.

A continuación, en el curso se estudia el *modelado del lenguaje* (*language modeling*), una de las tareas más elementales e importantes dentro del PLN. Un *modelo de lenguaje* (*language model*) indica la probabilidad de que una frase o texto esté bien formada con respecto al contexto en que se encuentra. Usando un gran corpus de texto se puede entrenar una RNN para crear un modelo de lenguaje. Dicha red contendrá una gran cantidad de conocimiento sintáctico y semántico que puede ser aplicado posteriormente en otras tareas usando aprendizaje por transferencia.

La unidad neuronal básica de las RNNs no captura bien las dependecias a largo plazo. Por ello, en el curso se estudian otras dos mucho más potentes: (1) *Gated Recurrent Unit* (GRU) [Cho et al. 2014b; Chung et al. 2014], y (2) *Long Short-Term Memory* (LSTM) [Hochreiter y Schmidhuber 1997]. Esta última, aunque es más antigua, captura dependencias a mayor distancia que GRU, pero con un mayor coste computacional.

Todas estas RNNs únicamente tienen en cuenta la información anterior de la secuencia para realizar las predicciones, pero en ciertas situaciones puede ser interesante poder usar tanto la información precedente como la posterior. Por ello, en el curso se estudian las *Bidirectional RNNs* (BRNNs), que son capaces de procesar de esta forma los datos secuenciales.

Los modelos de unidades neuronales RNN, GRU, LSTM y BRNN conforman una única capa de la red. Ésta puede ser suficiente para muchos problemas basados en datos secuenciales. Pero, para otros casos, el curso estudia las *Deep RNNs*, que concatenan dos

o más unidades neuronales formando modelos multicapa.

Previamente se ha estudiado la codificación de palabras denominada *one-hot encoding*. Esta representación tiene la desventaja de que trata cada palabra de forma independiente y no permite capturar las relaciones o dependencias entre ellas. En el curso se estudia una representación que no tiene estos problemas, denominada *word embedding*, en la que las palabras se representan mediante vectores de características, de forma que aquellas que capturan conceptos similares están cerca unas de otras dentro del espacio, mientras que los conceptos dispares están alejados entre sí. También se explica cómo reducir la dimensionalidad de esta codificación usando el algoritmo t-SNE [van der Maaten e Hinton 2008]. Tras realizar esta reducción, es posible dibujar las palabras en gráficos bidimensionales o tridimensionales, lo que permite observar fácilmente las relaciones espaciales entre las palabras. Además, se introduce el uso de *word embeddings* para resolver problemas de *razonamiento analógico* (*analogical reasoning*) [Mikolov et al. 2013b]. La codificación con *embeddings* hace que las RNNs sean capaces de analizar más eficazmente los textos y es uno de los puntos clave del éxito del aprendizaje profundo en PLN.

Para obtener los *word embeddings* de un vocabulario, lo que realmente se construye es una *embeding matrix*, es decir, se entrena conjuntamente una matriz que contiene los vectores para cada una de las palabras del vocabulario. En el curso se estudian diferentes técnicas para entrenar esta matriz, desde las más antiguas, hasta versiones más modernas:

1. Entrenamiento de un *modelo de lenguaje* para construir la *embedding matrix* [Bengio et al. 2003].

2. *Word2Vec* y *skip-gram model* [Mikolov et al. 2013a]: Una simplificación del método anterior en la que la arquitectura de la red usada es más sencilla.

3. *Negative sampling* [Mikolov et al. 2013c]: Mejora la eficiencia del método anterior transformando el problema en uno de clasificación binaria.

4. *GloVe* (*global vector for word representation*) [Pennington et al. 2014]: Un mecanismo aún más simple que usa la frecuencia con que dos palabras aparecen juntas dentro de un mismo contexto, para entrenar un modelo similar a una regresión lineal.

Un problema grave de la sociedad actual son los prejuicios que tienen las personas en cuestión de género, etnia, etc. Éstos quedan reflejados en los textos que escriben y, a su vez, son capturados por los *word embeddings*. El uso de estas codificaciones con sesgos en sistemas de IA es un problema muy importante ya que dichos prejuicios también influyen en las decisiones que toman. Dada la gravedad de este problema, en el curso se estudia un mecanismo para eliminar en la medida de lo posible, o al menos minimizar, el impacto de dichos prejuicios en la codificación de las palabras, basado en [Bolukbasi et al. 2016].

Posteriormente, el curso presenta en mayor profundidad los *modelos de secuencia-a-secuencia* (*sequence-to-sequence models*, *many-to-many*), es decir, aquellos cuya entrada y salida son secuencias de datos, centrándose en aquellos en los que dichas secuencias tienen longitudes distintas. Se estudian básicamente dos arquitecturas:

1. *RNN Enconder-Decoder* [Sutskever et al. 2014; Cho et al. 2014a]: Consta de (1) un codificador que se encarga de representar la secuencia de entrada mediante un

vector de características, y (2) un decodificador que transforma dicha representación en la secuencia de salida.

2. *Attention Model* [Bahdanau et al. 2014]: El modelo anterior no es capaz de memorizar ni capturar todas las características de las secuencias de datos largas. El *attention model* resuelve este problema tratando la secuencia de entrada por partes, es decir, el modelo únicamente "presta atención" a una pequeña parte de la secuencia de entrada para predecir cada elemento de la secuencia de salida.

Este tipo de modelos se usan habitualmente en problemas de *traducción automática* (*machine translation*). Pero para resolver esta tarea es necesario optimizar la calidad de las traducciones obtenidas maximizando la probabilidad condicional de dicha traducción dado el texto original. Se estudia cómo resolver este problema mediante el uso de una *beam search*, un tipo de búsqueda heurística. Por otro lado, en una tarea de traducción automática, es muy posible que para cierto texto haya dos o más traducciones correctas. Esto supone un problema a la hora de medir la precisión del sistema. En el curso se estudia la métrica *Bleu Score* (*bilingual evaluation understudy*) [Papineni et al. 2002], que facilita este proceso.

Estos modelos de secuencia-a-secuencia no sólo se pueden usar para traducción automática. Existe una tarea llamada *generación automática de pies de foto* (*image captioning*) en la que se obtiene automáticamente una descripción textual para una imagen dada. La idea básica es sustituir el codificador de la red por una ConvNet. En el curso se citan versiones basadas tanto en RNNs Encoder-Decoder [Mao et al. 2014; Vinyals et al. 2014; Karpathy y Fei-Fei 2014], como en Attention Models [Xu et al. 2015], para resolver esta tarea.

Los modelos de secuencia-a-secuencia no sólo han resultado exitosos a la hora de procesar texto, sino que también han destacado a la hora de tratar datos de audio. Como introducción al tema se estudia la tarea de *reconocimiento del habla* (*speech recognition*), que consiste en obtener la transcripción escrita del discurso contenido en un clip de audio. El curso explica cómo resolver esta tarea usando un Attention Model. Además, se detalla una función de coste comúnmente usada para este propósito denominada *clasificación conexionista temporal* (*Connectionist Temporal Classification*, CTC) [Graves et al. 2006]. El desarrollo de un sistema eficaz para el reconocimiento del habla puede ser muy complejo. Por ello, el curso introduce otro problema parecido, llamado *detección de palabras de activación* (*trigger word detection*), que es más sencillo y más asequible a la hora de empezar a experimentar con audio. Éste consiste en determinar si un clip contiene una palabra clave determinada. Modelos de este tipo se usan en dispositivos como *Amazon Echo*, *Baidu DuerOS*, *Apple Siri* o *Google Home* para "despertarlos" al articular su palabra clave. Este problema todavía es reciente y no hay un consenso en la comunidad sobre cuál es el mejor algoritmo para resolverlo.

Recomendaciones para el estudio del curso 5 En este curso se estudian los *one-hot vectors* y los *embeddings* como mecanismos para codificar palabras de un texto, pero actualmente también es habitual usarlos para codificar variables categóricas [Yadav 2019; Mishra 2019], es decir, para representar en un espacio vectorial variables que pueden tomar un único valor de entre un conjunto fijo y limitado de ellos. Un ejemplo de variable categórica es "mes", que sólo puede tomar un valor de entre un conjunto de 12, los meses

que contiene un año. En [Guo y Berkhahn 2016] se presenta por primera vez el uso de *embeddings* para representar este tipo de variables.

Como complemento a las referencias académicas sobre LSTMs, en [Olah 2015] se puede encontrar una explicación más didáctica. Además, en [Chen 2016; Mallya 2019] se puede profundizar en los detalles del paso hacia atrás de estas unidades neuronales.

El cálculo desde cero de *word embbedings* puede exigir mucho tiempo de computación, pero es posible obtener estos vectores ya precalculados. Jeffrey Pennington, Richard Socher y Christopher D. Manning ofrecen en su sitio web[24] varios conjuntos de *word embeddings* previamente calculados con su algoritmo, GloVe [Pennington et al. 2014], que pueden ser descargados para ser usados en los experimentos que se deseen. Además, en el mismo sitio se encuentra el código fuente de dicho algoritmo.

Referencias al curso de Howard et al. Las lecciones 3 y 4 de este curso (secciones 3.4.3 y 3.4.4) estudian cómo resolver un problema de clasificación de sentimientos con la librería fastai, aplicando aprendizaje por transferencia. En la lección 4 también se estudia cómo usar *embeddings* para representar variables categóricas y se aplica a los problemas de procesamiento de datos tabulados y filtrado colaborativo[25]. En la lección 5 (sección 3.4.5) se profundiza en la tarea de filtrado colaborativo y el uso de *embeddings*. En la lección 12 (sección 3.4.12) se implementa la unidad LSTM y una versión mejorada de la misma, denominada AWD-LSTM. Además, se estudia cómo realizar aprendizaje por transferencia para problemas de clasificación de textos mediante el método ULMFiT.

3.4 Curso de Howard et al. en fast.ai

Fastai course v3 (curso de Howard et al.)[26] es un MOOC sobre aprendizaje profundo publicado en 2019 por los creadores de la librería fastai [Howard y Gugger 2020][27]. Está impartido principalmente por Jeremy Howard, con la ayuda de Rachel Thomas y Sylvain Gugger, aunque las dos últimas lecciones están explicadas con la colaboración de Chris Lattner, el principal responsable de *Swift for TensorFlow*[28], debido a que en ellas se introduce dicho sistema.

Este curso se divide en dos partes:

1. *Practical Deep Learning for Coders*: Consta de 7 lecciones, basadas en el uso de fastai para comprender los rudimentos básicos del aprendizaje profundo y alcanzar un nivel avanzado en el uso de algoritmos del estado del arte. Utiliza una aproximación "de arriba a abajo" al aprendizaje, es decir, primero se establecen las ideas generales para, posteriormente, ir refinándolas.

[24] `https://nlp.stanford.edu/projects/glove`.

[25] El filtrado colaborativo es el núcleo de los sistema de recomendación, los cuales, utilizan los datos de sus usuarios para capturar los gustos de determinados perfiles de personas y, después, los usan para recomendar ítems, como libros, películas o música, a otras con gustos similares.

[26] `https://course.fast.ai`.

[27] `https://www.fast.ai`, documentación: `https://docs.fast.ai`.

[28] `https://www.tensorflow.org/swift`, Swift: `https://swift.org`.

2. *Deep Learning from the Foundations*: Consta de 7 lecciones adicionales basadas en implementar desde cero un subconjunto de las funcionalidades de fastai y PyTorch, usando tanto Python/PyTorch como Swift for TensorFlow. De esta forma, se profundiza tanto en los conocimientos teóricos como en las habilidades técnicas para desarrollar los algoritmos de aprendizaje profundo que forman parte del estado del arte. También tiene el objetivo de que la persona interesada aprenda a leer artículos científicos sobre aprendizaje profundo, a implementar por su cuenta los algoritmos descritos en ellos, realizar sus propias modificaciones y, en definitiva, no tener que depender del código de terceras personas.

El curso se basa en los principios educativos explicados en [Perkins 2009] que en este contexto se resumen en: (1) aprender experimentando, (2) ver el aprendizaje profundo globalmente, (3) partir de los conceptos generales para, posteriormente, ir refinando los detalles, y (4) avance constante, es decir, continuar con la siguiente lección aun no habiendo comprendido en su totalidad la anterior y volver a ella posteriormente con los nuevos conocimientos aprendidos.

Para realizar el curso no es necesario registrarse en ningún sitio ni pagar ninguna tarifa. Tan sólo hay que acceder al sitio web para ver los vídeos y descargarse el material didáctico. En total son 14 vídeos, uno por cada lección del curso, que vienen acompañados por referencias relevantes. El material didáctico está compuesto principalmente por los propios Jupyter Notebooks[29] que Jeremy Howard y Chris Lattner usan en sus explicaciones. Es interesante que, para la segunda parte del curso, todos los *notebooks* están implementados tanto en Python/PyTorch como en Swift for TensorFlow, por lo que se puede comparar el código en ambos lenguajes y marcos de trabajo. Así mismo, para algunas lecciones también se proporcionan las diapositivas usadas en los vídeos u otro tipo de material adicional.

Los requisitos mínimos del curso son:

1. Al menos un año de experiencia en programación, preferiblemente en Python.

2. Conocimientos matemáticos "de instituto".

La mayor parte del curso se basa en el lenguaje de programación Python. En la sección 3.1 ofrecemos algunas referencias útiles para el aprendizaje de este lenguaje. Por otro lado, en [Apple Inc. 2019] se puede encontrar una introducción exhaustiva al lenguaje de programación Swift.

3.4.1 Lección 1: "Image classification"

Resumen del contenido En esta lección se introduce la primera parte del curso indicando, de forma general, qué es lo que se va a estudiar y la metodología que se va a seguir. Así mismo, se da una breve explicación sobre qué es y cómo usar Jupyter Notebook y, de la misma forma, se presenta la librería fastai.

[29] Los Jupyter Notebooks de fastai course v3 se pueden descargar desde `https://github.com/fastai/course-v3/tree/master/nbs`.

Como aproximación inicial al aprendizaje profundo se resuelve un problema de *clasificación de imágenes* (*image classification*) con el conjunto de datos Oxford-IIIT Pet [Parkhi et al. 2012], es decir, se clasifican una serie de imágenes de perros y gatos en sus correspondientes razas. Se usa una *red neuronal convolucional* (*convolutional neural network*, ConvNet, CNN) denominada *Residual Network* (ResNet). Durante la lección se aprende a (1) cargar el conjunto de datos, (2) entrenar la red y (3) evaluar los resultados, todo ello usando la librería fastai. De esta forma, se observa de una sola vez el esquema general de un problema de aprendizaje profundo. Para el entrenamiento se utiliza también *aprendizaje por transferencia* (*transfer learning*).

Una vez realizado el entrenamiento de la red, basándose en [Zeiler y Fergus 2013], explica de forma intuitiva qué es lo que realiza internamente una ConvNet. Se muestran imágenes de los filtros que contienen las capas convolucionales junto con ejemplos de los patrones que son capaces de detectar.

Por otro lado, se realiza una primera toma de contacto con el concepto de *learning rate finder*, un mecanismo que permite estimar un valor adecuado para la *tasa de aprendizaje* (*learning rate*), que es uno de los puntos claves para realizar un correcto entrenamiento de la red. Además, se indica que fastai utiliza un planificador de hiperparámetros denominado *one cycle* (ver la sección 2.3) durante los entrenamientos, una técnica que permite ajustar la red de forma más rápida y eficaz. Ambos conceptos no se detallan en esta lección, sino que son explicados más profundamente en lecciones más avanzadas del curso.

Recomendaciones para el estudio de la lección 1 Debido a la naturaleza introductoria de la lección, es muy probable que tras la finalización de la misma no se comprendan del todo ciertos conceptos clave explicados en ella. Aunque en lecciones posteriores se estudian en más profundidad, aquí se citan algunas referencias para complementar las ideas introducidas. En [Saha 2018; Ingargiola 2019] se pueden encontrar dos introducciones bastante completas a las redes neuronales convolucionales. En [Marcelino 2018; Sarkar 2018] se detalla más extensamente la idea de aprendizaje por transferencia.

En [Smith 2015, 2018] se explican el mecanismo *learning rate finder* y el planificador de hiperparámetros *one cycle*, respectivamente.

Comparación con el programa de Ng et al. A diferencia del programa de Ng et al., en esta primera lección de fastai se introducen directamente las ConvNets sin haber estudiado previamente las RNAs en que se basan. El curso 1 de Ng et al. (sección 3.3.1) estudia exhaustivamente y con una aproximación más teórica los fundamentos de las RNAs, mientras que el curso 4 (sección 3.3.4) hace lo propio con las ConvNets. Por otro lado, el curso 3 (sección 3.3.3) da una explicación introductoria sobre el aprendizaje por transferencia, aunque no realiza una exposición muy amplia ni lleva a cabo ningún estudio práctico. En cambio, el aprendizaje por transferencia tiene una importancia capital en el curso de Howard et al. y está presente en la mayoría de sus lecciones.

3.4.2 Lección 2: "Data cleaning and production; SGD from scratch"

Resumen del contenido Para resolver un problema de clasificación de imágenes lo primero que hay que hacer es obtener un conjunto de imágenes etiquetadas. Por ello, en esta lección se detalla un mecanismo para crear conjuntos de este tipo descargando imágenes desde *Google Images*. Así mismo, se indica cómo limpiar el conjunto de aquellas imágenes que no son útiles para el entrenamiento o que lo perjudican. Además, se explica cómo dividir estos datos en *conjunto de entrenamiento* (*training set*) y *conjunto de validación* (*validation set*). Una vez construido el clasificador, se dan indicaciones de cómo poner dicho modelo en producción construyendo una aplicación web.

Se realizan varios *análisis de preferencia-varianza* (*bias-variance analyses*) en los que se estudian diferentes escenarios con distintos valores de *error de entrenamiento* (*training loss*) y *error de validación* (*validation loss*), explicando cuáles son los problemas más habituales y sus posibles soluciones. Dos de ellos que se destacan son:

1. El *sobreajuste* (*overfitting*) del modelo.
2. Encontrar un valor adecuado para la tasa de aprendizaje.

En un escenario perfecto, en el que la red se ha entrenado correctamente, los errores de entrenamiento y validación tienen ambos que ser bajos y relativamente iguales.

Introduce los primeros conceptos teóricos sobre regresión lineal y cómo vectorizar el código fuente de dicho modelo usando la multiplicación de matrices, junto con otro tipo de operaciones con vectores. Posteriormente se explica el optimizador *descenso del gradiente estocástico* (*stochastic gradient descent*, SGD) como mecanismo para ajustar los parámetros de dicho modelo. Se realiza una implementación exhaustiva tanto del modelo como del algoritmo SGD usando PyTorch.

Recomendaciones para el estudio Con el análisis de preferencia-varianza se desea controlar el *compromiso entre preferencia y varianza* (*bias-variance tradeoff*). En [Singh 2018] se puede encontrar una breve introducción a este problema. Para lograr este objetivo es fundamental dividir correctamente los datos en los conjuntos de entrenamiento y validación. En [Thomas 2017] se explica cómo realizar adecuadamente dicha división.

En [Menon 2018] se puede encontrar una explicación alternativa sobre el entrenamiento de un modelo de regresión lineal usando el descenso del gradiente.

Comparación con el programa de Ng et al. En el curso 3 de Coursera (sección 3.3.3) aparece información más detallada sobre el análisis de preferencia-varianza, cómo interpretarlo y qué soluciones aplicar para reducir los diferentes errores del modelo. En el curso 1 (sección 3.3.1) se explica la regresión logística, que es parecida a la regresión lineal explicada en esta lección. Ng et al. explican más profundamente el funcionamiento y construcción de dicho modelo que en la lección 2 de fastai. En el curso 2 (sección 3.3.2) se detalla el algoritmo del descenso del gradiente, junto con su modificación SGD.

3.4.3 Lección 3: "Data blocks; Multi-label classification; Segmentation"

Resumen del contenido En esta lección se introduce la *data block API*[30] de fastai, es decir, una interfaz que permite cargar y preparar los datos para el entrenamiento de forma sencilla. Todos los datos quedan finalmente encapsulados en un único objeto `DataBunch` de fastai, encargado de gestionarlos durante el entrenamiento.

Posteriormente se introducen tres nuevos problemas de visión artificial:

1. *Clasificación multi-etiqueta* (*multi-label classification*): Es una extensión de los problemas de clasificación donde es posible asignar múltiples etiquetas a una misma muestra del conjunto de datos. Para entender este tipo de problema se entrena un modelo capaz de clasificar el conjunto planet[31] compuesto por imágenes de la selva del Amazonas tomadas por satélite.

2. *Segmentación de imágenes* (*image segmentation*): Consiste en, dada una imagen, separar las áreas de la misma que pertenecen a diferentes clases de interés. Concretamente, se usa el conjunto de datos CamVid [Brostow et al. 2008], que contiene imágenes tomadas por un coche en las que se separan los diferentes elementos de interés de una vía. Para resolver este problema se utiliza un tipo especial de ConvNet denominada U-Net [Ronneberger et al. 2015].

3. *Regresión de imágenes* (*image regression*): Consiste en obtener ciertos valores numéricos de interés sobre una imagen. Para su estudio se utiliza el conjunto de datos Biwi Kinect Head Pose Database [Fanelli et al. 2013], compuesto por imágenes de personas que tienen asociadas las coordenadas del punto que señala aproximadamente el centro de su cabeza.

Tras estudiar estos problemas de visión artificial, la lección introduce un problema de PLN. Concretamente se estudia la *clasificación de sentimientos* (*sentiment classification*), que consiste en clasificar una serie de textos según la naturaleza positiva o negativa de su contenido. Se usa el conjunto IMBd Large Movie Review Dataset [Maas et al. 2011], que contiene una serie de críticas sobre películas, etiquetadas como positivas o negativas. El problema se resuelve primero entrenando un *modelo de lenguaje* (*language model*) y posteriormente aplicando aprendizaje por transferencia para construir el clasificador.

Por último, se explica la relación de las DNNs con el *teorema de aproximación universal* (*universal approximation theorem*)[32], cuya idea básica consiste en que, si se concatenan suficientes operaciones lineales y no lineales, es posible ajustar razonablemente cualquier función. En el contexto de las DNNs, las operaciones no lineales vienen dadas por las *funciones de activación* que se usan entre operaciones lineales, siendo la *unidad lineal rectificada* (*Rectified Linear Unit*, ReLU) y sus variaciones las más habituales hoy en día.

[30] `https://docs.fast.ai/data_block.html`
[31] `https://www.kaggle.com/c/planet-understanding-the-amazon-from-space/overview`.
[32] `https://en.wikipedia.org/wiki/Universal_approximation_theorem`.

Recomendaciones para el estudio de la lección 3 Para entender el teorema de aproximación universal puede ser útil el capítulo 4, titulado *a visual proof that neural nets can compute any function*, de [Nielsen 2019]. Aunque ReLU es la función de activación más usada en las capas ocultas de una red, hay una amplia variedad de alternativas. En [Sharma 2017; Sharma 2019] se estudian más profundamente este tipo de funciones.

Comparación con el programa de Ng et al. Ninguno de los nuevos problemas de visión artificial introducidos en esta lección de fastai se explican en la especialización de Ng et al. Por otro lado, el curso 5 (sección 3.3.5) puede servir de apoyo para entender mejor el problema de clasificación de sentimientos y cómo resolverlo mediante RNNs, debido a que da una explicación más teórica y extensa de este tipo de redes. En el curso 1 (sección 3.3.1) también se explican varias de las funciones de activación más típicas, aunque no se detallan tanto como en la lección 3 de fastai, ni se explica su relación con el teorema de aproximación universal.

3.4.4 Lección 4: "NLP; Tabular data; Collaborative filtering; Embeddings"

Resumen del contenido En esta lección se profundiza en el problema de clasificación de sentimientos visto en la lección anterior, centrándose especialmente en el proceso de aprendizaje por transferencia. Los pasos llevados a cabo para ello son los siguientes:

1. Se entrena un modelo de lenguaje usando el conjunto de datos WikiText-103 [Merity et al. 2016]. De esta forma, se obtiene una base de conocimiento enorme para PLN, que luego puede aplicarse a otro tipo de problemas.

2. Se refina el modelo para crear otro ajustado al conjunto de datos IMDb.

3. Se refina el modelo anterior para generar el clasificador para IMDb.

Posteriormente, se introducen dos nuevos tipos de problemas:

1. Procesamiento de *datos tabulados* (*tabular data*) o *datos estructurados* (*structured data*) con DNNs: Aunque actualmente no se ha explorado mucho, es posible usar el aprendizaje profundo para resolver tareas que usan este tipo de datos. Hoy en día es más habitual usar técnicas como *random forest*. Se estudia también el preprocesamiento de este tipo de datos.

2. *Filtrado colaborativo* (*collaborative filtering*): Es el núcleo de los sistemas de recomendación (*recommender systems*), es decir, aquellos que sugieren a sus usuarios ciertos ítems, como libros, películas, noticias, música, etc, comparando sus elecciones previas con las de otros usuarios. Los datos usados para estos problemas suelen estar organizados en variables categóricas, por ejemplo, una variable "película" que contiene el identificador de un largometraje, seleccionado de entre un conjunto finito de ellos. Por ello, se estudia el uso de *embeddings* para representar dichas variables. Se experimenta con esta tarea usando el conjunto de datos MovieLens [GroupLens 2019].

Al final de la lección se explica la estructura general que tiene cualquier DNN, haciendo hincapié en las multiplicaciones de matrices (operaciones lineales) y las funciones de activación (operaciones no lineales). Además, se indican los dos grandes tipos en los que se dividen las variables de un modelo: los *parámetros* y las *activaciones* (ver la sección 3.3.1 para una explicación más detallada).

Recomendaciones para el estudio de la lección 4 Como lectura complementaria sobre el uso de DNNs para procesar datos tabulados es recomendable [Thomas 2019a]. Para profundizar en la representación de variables categóricas mediante *embeddings*, puede resultar interesante [Guo y Berkhahn 2016]. En [He et al. 2017] se explica más exhaustivamente un modelo neuronal para filtrado colaborativo.

Comparación con el programa de Ng et al. En el curso 5 (sección 3.3.5) se explican los fundamentos de los modelos secuenciales usados para PLN y de los *word embeddings*, los cuales no se estudian en esta lección. Sin embargo, en dicho curso no se explica el proceso de aprendizaje por transferencia para PLN usado en esta lección. En el programa de Ng et al. tampoco se estudian los problemas de procesamiento de datos tabulados con DNNs, ni de filtrado colaborativo. Sin embargo, en el curso 1 (sección 3.3.1) se explican detalladamente los rudimentos básicos de las RNAs en las que se fundamentan los modelos para resolver estos problemas.

3.4.5 Lección 5: "Back propagation; Accelerated SGD; Neural net from scratch"

Resumen del contenido En esta lección se sigue explicando el proceso de aprendizaje por transferencia centrándose en: (1) cómo adaptar la salida de una red a aquella requerida por el nuevo problema, (2) en qué consiste "congelar" las capas del modelo en dicho proceso, (3) el proceso de ajuste fino y el uso de *discriminative learning rates* para variar menos los parámetros de las capas iniciales que los de las finales durante el entrenamiento.

Por otro lado, se profundiza más en el estudio de filtrado colaborativo y el uso de *embeddings* mediante la experimentación con el conjunto MovieLens. Se estudia cómo operar con *embeddings* y cuál es su significado semántico.

Posteriormente se pasa a explicar el problema de preferencia-varianza, junto a la técnica de regularización del declive de los pesos (*weight decay*) o regularización L2 (*L2 regularization*), que es históricamente el método básico para prevenir el sobreajuste de un modelo. Ésta consiste en añadir a la función de pérdida la suma de los cuadrados de los parámetros. De esta forma, cuanto mayores sean los valores de los parámetros, peor se considera el modelo, premiando de esta manera modelos más simples.

A continuación se estudian varios *optimizadores*. Se parte del SGD, ya estudiado. Se introduce el concepto clave del *promedio móvil ponderado exponencialmente* (*exponentially weighted moving average*, EWMA), que permite el cálculo de un valor medio y su actualización a medida que aparecen nuevos elementos de interés. Esto permite crear

optimizadores más rápidos. Se estudian el descenso del gradiente con momento (*gradient descent with momentum*), RMSprop y Adam [Kingma y Ba 2014].

Posteriormente se explica el planificador de hiperparámetros *one cycle* [Smith 2018].

Se introduce un nuevo tipo de problema, la *clasificación multi-clase* (*multi-class classification*). Para este problema, y cualquier otro, es fundamental la definición de la función de activación de la última capa y la función de pérdida que se usa. Para la clasificación multi-clase, se usa la función de activación denominada *softmax*, que define una distribución de probabilidad sobre las clases para cada una de las muestras; es decir, la suma de los valores de salida de la red es siempre 1. Tomando el valor máximo del vector de salida se obtiene la predicción de la muestra. Por otro lado, la función de pérdida que se usa es la *entropía cruzada* (*cross-entropy*), también denominada *verosimilitud logarítmica negativa* (*negative log likelihood*), que otorga un valor muy bajo cuando la etiqueta predicha coincide con la real y muy alto en caso contrario.

Usando parte de lo estudiado anteriormente, se construye una red neuronal desde cero para resolver el problema de clasificación dado por el conjunto de datos MNIST [LeCun et al. 1998].

Recomendaciones para el estudio de la lección 5 Para un estudio más profundo del uso de *embeddings* para representar variables categóricas se puede consultar el artículo ya citado [Guo y Berkhahn 2016]. Fastai usa bastante este mecanismo para problemas de filtrado colaborativo o con datos tabulados. Por otro lado, es muy habitual el uso de *embeddings* para representar palabras en el contexto de PLN. En [Mikolov et al. 2013a] se explican los algoritmos Word2Vec y skip-gram para obtener este tipo de representaciones. Alternativamente, en [Pennington et al. 2014] se explica GloVe, un algoritmo más simple pero igual de potente para el mismo fin.

La publicación de Mahapatra [2018] puede servir como apoyo a la explicación de la lección sobre EWMA. Ruder [2016] presenta una revisión exhaustiva de los diferentes algoritmos de optimización basados en el descenso del gradiente. Aunque se introducen muchos más optimizadores que los estudiados en esta lección, puede servir para dar una visión más amplia sobre estos y más profunda sobre aquellos ya estudiados.

Cada tipo de problema abordado mediante aprendizaje profundo consta de una combinación específica de función de activación para la capa de salida y una función de pérdida. En [Ronaghan 2018] se da un resumen sobre cuáles de ellas usar para cada uno de los problemas habituales.

Comparación con el programa de Ng et al. En el curso 3 (sección 3.3.3) se puede encontrar una explicación teórica sobre el aprendizaje por transferencia, aunque no se realiza ninguna práctica al respecto como en esta lección. En el curso 5 (sección 3.3.5) se detalla el uso y obtención de *embeddings*, pero únicamente para representar palabras en el contexto de PLN y no se estudia cómo usarlos para representar variables categóricas. En el curso 2 (sección 3.3.2) se estudian más profundamente que en esta lección los fundamentos

teóricos de: (1) el declive de los pesos junto a otras técnicas de regularización; (2) cada uno de los optimizadores habituales, desde el descenso del gradiente hasta Adam; y (3) los problemas de clasificación multi-clase, junto con la función de activación softmax y la función de pérdida entropía cruzada. Como apoyo a la implementación de la red neuronal desde cero, se puede usar la información proporcionada en el curso 1 (sección 3.3.1), que es más detallada que la presentada en esta lección.

3.4.6 Lección 6: "Regularization; Convolutions; Data ethics"

Resumen del contenido En esta lección se introducen los conceptos de *transformación* y *preprocesamiento*. Ambos suponen un tratamiento de los datos previo a su utilización, pero se diferencian en que la primera se ejecuta cada vez que un dato se introduce en el modelo durante el entrenamiento, mientras que la segunda se ejecuta una única vez antes de dicho entrenamiento.

Se estudian más técnicas de regularización: (1) *eliminación de enlaces* (*dropout*) [Srivastava et al. 2014] y (2) *data augmentation*. Y posteriormente se introduce la *normalización por lotes* (*batch normalization*, BatchNorm) [Ioffe y Szegedy 2015].

A continuación se explica la operación de *convolución*, aquella de la que las ConvNets reciben su nombre. Se explica su semejanza con la multiplicación de matrices y que, al igual que ésta, constituye realmente una función lineal. Se explica como transforma las activaciones de entrada usando un filtro y, además, en qué consiste el *relleno* (*padding*) y la *zancada* (*stride*) en este tipo de operaciones.

Al final de la lección se da una presentación sobre *ética dentro del contexto del aprendizaje profundo y la ciencia de datos*, donde se subraya la importancia de evitar el sesgo en los datos y la utilización inadecuada de sistemas inteligentes. Se exponen problemas reales en los que la falta de principios éticos han resultado en situaciones terriblemente nocivas, si no en conductas directamente criminales o delictivas. Se exponen ciertas lineas de actuación para evitar este tipo de problemas invitando a que los futuros profesionales las tengan en cuenta durante el desarrollo de sus proyectos.

Recomendaciones para el estudio de la lección 6 En la documentación de fastai se pueden encontrar diferentes ejemplos de transformaciones para *data augmentation*[33] en el contexto de la visión artificial. Al ser ejemplos visuales, ayudan a entender fácilmente en qué consiste esta técnica y, además, se proporciona código fuente con el que se puede experimentar.

Aunque en [Ioffe y Szegedy 2015] se publica por primera vez BatchNorm, su explicación matemática no es correcta. En [Santurkar et al. 2018] se da otra más acertada.

Para más información sobre ética en la IA se pueden consultar las tres exposiciones sobre el tema realizadas por Rachel Thomas [2018a,b, 2019b], una de los co-fundadores

[33] fastai: Lista de transformaciones para *data augmentation* en visión artificial: `https://docs.fast.ai/vision.transform.html`.

de fastai, y en las que se basa lo expuesto en esta lección.

Comparación con el programa de Ng et al. Aunque en el programa de Ng et al. se estudia el preprocesamiento de los datos, no se hace hincapié en la diferencia entre éste y las transformaciones, como en esta lección y en el resto del curso de Howard et al. En el curso 2 (sección 3.3.2) se puede encontrar información teórica más detallada que la presentada en esta lección sobre las técnicas de regularización estudiadas y BatchNorm. En las primeras lecciones del curso 4 (sección 3.3.4) se introducen las ConvNets y se explican las operaciones y capas básicas que las componen. Se estudian sus detalles de forma más exhaustiva que en esta lección. En el curso 5 (sección 3.3.5) se explica un mecanismo, que no se estudia en todo el curso de Howard et al., para eliminar, o al menos reducir, el sesgo en *word embeddings*, una forma de evitar que los prejuicios de las personas, plasmados en textos, pasen a formar parte de un sistema inteligente.

3.4.7 Lección 7: "ResNets from scratch; U-Net; Generative (Adversarial) Networks"

Resumen del contenido En esta lección se estudia la construcción de una ConvNet desde cero usado PyTorch. Se explica el bloque básico de una ConvNet, compuesto por (1) una capa convolucional, (2) una capa BatchNorm, y (3) una función de activación ReLU. Repitiendo bloques de este tipo se construye una ConvNet básica, aunque más allá de cierto número de capas, la eficacia de la red no mejora, a causa del problema del desvanecimiento y la explosión de los gradientes. Por ello, a continuación se explican las ResNets [He et al. 2015a], una evolución de las ConvNets que resuelve este problema. Dichas redes están compuestas por agrupaciones de capas consecutivas denominadas *ResNet Blocks*. Lo que hace especial a este tipo de redes es la inclusión de una *skip connection* dentro de cada bloque que suma sus activaciones de entrada a las de salida. La presencia de estas conexiones hace que las ResNets con un número mayor de capas que otras ConvNets puedan entrenar satisfactoriamente. Además, se introducen las *densely connected convolutional networks* (DenseNets) [Huang et al. 2016], que son similares a las ResNets, salvo que las *skip connections* concatenan las activaciones de entrada con las de salida en vez sumarlas.

Posteriormente se estudia el modelo U-Net [Ronneberger et al. 2015], usado previamente en la lección 3 para segmentación de imágenes. Una ConvNet es básicamente un codificador que va reduciendo las dimensiones de las activaciones para extraer las características de interés, pero las U-Nets constan además de un decodificador que transforma el vector de características, aumentando sus dimensiones, para conseguir que la salida de la red tenga las mismas dimensiones que la entrada. Sin embargo, el decodificador por sí mismo no es capaz de reconstruir adecuadamente la salida. Por ello, las U-Nets usan *skip connections* para unir los bloques equivalentes del codificador y el decodificador, de forma que a estos últimos les llega información de las activaciones originales para poder reconstruir adecuadamente la imagen de salida.

Por último, se explican los *modelos generativos*, redes que habitualmente se usan para restauración de imágenes o para la generación de imágenes completamente nuevas.

Concretamente, se estudian las *redes generativas antagónicas* (*generative adversarial networks*, GANs), que usan otra red adicional como función de pérdida, denominada *discriminadora* (*discriminator*) o *crítica* (*critic*), que intenta discernir cuáles de las imágenes son originales y cuáles son generadas. El objetivo es engañar a la discriminadora de forma que clasifique como imágenes originales aquellas generadas artificialmente. En la lección se propone una alternativa a las GANs basada en usar una U-Net junto a una función de pérdida denominada *perceptual losses* [Johnson et al. 2016] (*feature losses* en fastai). Ésta función de pérdida se basa en una red VGG-16 de la que se extrae un mapa de características con el que se puede determinar si la imagen generada es lo suficientemente buena.

Al final de la lección se da una ligera introducción teórica a las *recurrent neural networks* (RNNs).

Recomendaciones para el estudio de la lección 7 Las ResNets son uno de los tipos de ConvNets más exitosos hoy en día. Esto ha supuesto la aparición de múltiples variaciones de su diseño desde la presentación del modelo original. En [Fung 2017] se puede encontrar un breve resumen tanto de la arquitectura original de ResNet como sus modificaciones más comunes.

Sankesara [2019] ofrece un breve tutorial sobre U-Nets, donde explica a nivel teórico tanto este modelo como el problema de segmentación de imágenes. Además, se incluye una implementación de U-Net en PyTorch.

Karras et al. [2018] presentan StyleGAN, una arquitectura híbrida entre GANs y *neural style transfer*, que supuso un hito en la generación artificial de imágenes hiperrealistas de caras de personas.

Comparación con el programa de Ng et al. En el programa de Ng et al. no se estudian ni las DenseNets, ni las U-Nets, ni tampoco las GANs. En el curso 4 (sección 3.3.4) se explican tanto la arquitectura básica de una ConvNet como la de una ResNet, haciendo más hincapié en los detalles teóricos en comparación con lo expuesto en esta lección. En ese mismo curso también se estudia *neural style transfer*, una técnica distinta a las GANs para generar imágenes artificiales.

3.4.8 Lección 8: "Matrix multiplication; forward and backward passes"

Resumen del contenido Con esta lección empieza la segunda parte del curso, que consiste en implementar parte de la funcionalidad de fastai y PyTorch, como medio para entender más detalladamente el aprendizaje profundo.

Primero, se explica la operación de *multiplicación de matrices* y cómo se implementa. Se parte de una implementación con bucles explícitos hasta llegar al operador proporcionado por PyTorch, la versión más eficiente de realizar esta operación. En el proceso se estudia

cómo la vectorización y el uso de *broadcasting* aceleran la ejecución del código.

Posteriormente, se estudian los *pasos hacia delante* y *hacia atrás* mediante la implementación de un modelo simple de dos capas. Se parte de la inicialización de la red usando la *inicialización de Kaiming* [He et al. 2015b]. A continuación se implementa la función de pérdida del MSE. Posteriormente, se codifica manualmente la computación de los gradientes de la función de pérdida y de cada capa respecto a sus variables de entrada, y se usan para implementar el *paso hacia atrás* de la red. Finalmente, se refactoriza el modelo usando clases y utilizando la funcionalidad de PyTorch.

Recomendaciones para el estudio de la lección 8 A aquellas personas que no hayan programado previamente usando vectorización y las operaciones que implica, esta lección les puede resultar algo confusa al principio. PyTorch usa las mismas reglas que NumPy, por lo que la documentación de esta última librería es muy útil al respecto. En [SciPy community 2019a,b] se puede encontrar la documentación sobre *indexing*, *slicing* y *broadcasting*. Además, en [Varoquaux et al. 2017] hay un tutorial exhaustivo sobre NumPy y cómo usar estas operaciones sobre vectores y matrices.

Aunque la mayoría de las librerías de aprendizaje profundo realizan automáticamente los cálculos del *paso hacia atrás*, es interesante que un profesional del aprendizaje profundo sepa cómo implementarlas explícitamente. En [Khan Academy 2017] hay un tutorial sobre el cálculo de derivadas básico. Adicionalmente, [Parr y Howard 2018; Barnes 2006] pueden ser un buen punto de partida para profundizar en el cálculo matricial y de derivadas necesario para el aprendizaje profundo.

Comparación con el programa de Ng et al. En el curso 1 (sección 3.3.1) se puede encontrar una aproximación más teórica sobre cómo funciona una RNA básica, la vectorización y los *pasos hacia delante* y *hacia atrás* de un grafo de computación.

3.4.9 Lección 9: "Loss functions, optimizers, and the training loop"

Resumen del contenido Primero se profundiza en la inicialización de los parámetros del modelo y la importancia de llevarla a cabo correctamente, ya que la falta de un mecanismo de inicialización adecuado fue la principal barrera que impidió entrenar DNNs durante décadas. Se centra principalmente en la *inicialización de Xavier* [Glorot y Bengio 2010], y la *inicialización de Kaiming* [He et al. 2015b], junto a algunas variaciones de éstas.

Posteriormente, se realiza una implementación de la función de activación *softmax* y de la función de pérdida *entropía cruzada*, centrándose en la eficiencia de las mismas. Esta combinación de funciones se suele usar para problemas de *clasificación multi-clase*.

A continuación se pasa a implementar el bucle de entrenamiento básico fundamentado en el *descenso del gradiente por mini-lotes*. Siguiendo el diseño de fastai, se implementa la validación del modelo dentro del bucle de entrenamiento.

Un punto clave en el entrenamiento de una DNN es la correcta gestión del conjunto de datos para poder seleccionar las muestras adecuadamente o para que la memoria no se desborde, por ejemplo. Fastai contiene una API dedicada a esta labor y, en esta lección, se comienza a implementarla. Concretamente, se codifica:

1. La clase `Dataset` que sirve como abstracción de los conjuntos de entrenamiento y validación.

2. La clase `DataLoader` que se encarga del muestreo de un conjunto de datos usando un objeto `Sampler`.

3. La clase `DataBunch` que aúna en un mismo objeto los *dataloaders* de los datos de entrenamiento y de validación.

Finalmente, se implementa la clase `Learner` para que gestione todos los objetos implicados en el entrenamiento de un modelo.

A continuación se desarrollan una serie de *callbacks*, un conjunto de funciones que permiten modificar el comportamiento del algoritmo de entrenamiento básico a posteriori inyectando código ejecutable en partes específicas de éste, sin necesidad de modificar la implementación original. Esto va a permitir, entre otras cosas, crear fácilmente versiones mejoradas del optimizador básico o añadir técnicas de regularización. Usando *callbacks*, en esta lección se implementan: (1) un conmutador que cambia la ejecución entre el modo de entrenamiento y el de validación, (2) el cálculo de métricas y (3) un planificador de hiperparámetros.

Recomendaciones para el estudio de la lección 9 Como complemento de lo visto en la lección sobre la gestión de los datos puede ser recomendable echar un vistazo a la documentación de la *data block API*[34], para tener una idea de lo que se puede conseguir con un diseño de este tipo. Así mismo, puede ser útil buscar información sobre las clases `Dataset`[35], `Sampler`[36] y `DataLoader`[37] de PyTorch, en las que se basan las respectivas implementaciones realizadas en la lección.

Comparación con el programa de Ng et al. En el curso 2 (sección 3.3.2) se estudian (1) los mecanismos de inicialización de los parámetros; (2) el descenso del gradiente por mini-lotes, junto a otros optimizadores; (3) el bucle de entrenamiento; y (4) los problemas de clasificación multi-clase (funciones softmax y entropía cruzada). En el programa de Ng et al. no se detalla en tanta profundidad la implementación del código para la gestión de los datos ni del algoritmo de entrenamiento como en esta lección y en el resto del curso de Howard et al., en general.

[34] `https://docs.fast.ai/data_block.html`

[35] `https://pytorch.org/docs/master/data.html#torch.utils.data.Dataset`.

[36] `https://pytorch.org/docs/master/data.html?highlight=sampler#torch.utils.data.Sampler`.

[37] `https://pytorch.org/docs/master/data.html?highlight=dataloader#torch.utils.data.DataLoader`.

3.4.10 Lección 10: "Looking inside the model"

Resumen del contenido Primero se presenta una mejora del optimizador Adam denominada LAMB [You et al. 2019]. Se continúa con el estudio de los *callbacks* y de estadísticos como la varianza, la covarianza y la correlación. Así mismo, se introducen nuevos detalles sobre la función *softmax* indicando, entre otras cosas, que únicamente debe usarse cuando las muestras se etiquetan con una y solamente una de las clases existentes originalmente en el conjunto. Es decir, hay que evitar transformar ciertos problemas añadiendo una clase extra del tipo "ninguna de las otras clases" para que sea posible usar *softmax*, ya que esto fuerza a la red a aprender a reconocer elementos sin interés para dicho problema con el objetivo de descartarlos de aquellos que sí lo tienen. Posteriormente, se implementan usando *callbacks* la técnica de regularización *early stopping* y el *learning rate finder* [Smith 2015].

Se continúa con la implementación de una ConvNet, junto con el desarrollo de un nuevo *callback* para realizar el entrenamiento usando CUDA, necesario para llevar a cabo la computación en una GPU.

Posteriormente se estudian los *hooks* de PyTorch (similares a los *callbacks* de fastai) como mecanismo para inspeccionar las variables de un modelo sin tener que modificar el código de éste. En esta lección se usan para calcular ciertos estadísticos y gráficas con el objetivo de determinar la calidad del entrenamiento.

Se realiza una implementación de BatchNorm [Ioffe y Szegedy 2015] para intentar mejorar la ConvNet previamente desarrollada. También se explica que BatchNorm no es útil cuando se entrena usando muestras individuales o lotes muy pequeños. Por ello se estudian algunas normalizaciones alternativas: (1) *layer normalization* (LayerNorm) [Ba et al. 2016], (2) *instance normalization* (InstanceNorm) [Ulyanov et al. 2016] y (3) *Group normalization* (GroupNorm) [Wu y He 2018], aunque se indica que ninguna de ellas funciona tan bien como BatchNorm. Por ello, para lotes pequeños se propone una novedosa manera de usar BatchNorm, denominada *Running BatchNorm*. El algoritmo original de BatchNorm, durante el entrenamiento, calcula la media y la varianza por cada lote de activaciones de entrada y, posteriormente, los usa para normalizar el citado lote. Al mismo tiempo, va componiendo una estimación de dichos estadísticos con EWMA conforme se van introduciendo sucesivamente nuevos lotes de datos en la red. Estos valores medios se usan durante la inferencia para normalizar las activaciones de entrada de la misma forma en que se han utilizado los estadísticos originales durante el entrenamiento. El motivo de este procedimiento es que, durante la inferencia, en vez de introducir un lote en el modelo, se procesa una única muestra y, por tanto, no es posible calcular la media y la varianza de un solo elemento. La novedad que introduce *Running BatchNorm* es usar esta misma estrategia durante el entrenamiento, es decir, utiliza las estimaciones de los estadísticos calculados mediante EWMA también durante el proceso de ajuste de los parámetros del modelo, obteniendo una normalización más efectiva cuando hay un número reducido de muestras por lote.

Recomendaciones para el estudio de la lección 10 Dado que en esta lección se implementa una ConvNet por primera vez, es recomendable repasar en qué consisten este

tipo de redes y cuáles son los elementos que las forman. En [Saha 2018; Ingargiola 2019] se pueden encontrar dos guías bastante concisas pero completas sobre ConvNets.

También puede ser interesante estudiar y experimentar con el uso de *hooks*[38] en PyTorch para poder inspeccionar o modificar los parámetros y activaciones de un modelo sin cambiar la implementación de éste.

Comparación con el programa de Ng et al. En esta lección se dan indicaciones, mucho más detalladas que en el programa de Ng et al., para la implementación del algoritmo de entrenamiento y de una ConvNet. El uso de *hooks* para inspeccionar los modelos también es una estrategia que no se muestra en el programa de Coursera. En el curso 2 (sección 3.3.2) se estudian tanto *early stopping* como BatchNorm desde una aproximación más teórica que en esta lección. En el curso 4 (sección 3.3.4) se explican los elementos básicos que conforman una ConvNet de forma más exhaustiva, lo que puede servir de apoyo para la implementación de la red presentada en esta lección.

3.4.11 Lección 11: "Data Block API, and generic optimizer"

Resumen del contenido Primero se presenta un nuevo mecanismo de inicialización denominado *Layerwise Sequential Unit Variance* (LSUV) basado en [Mishkin y Matas 2016]. La idea básica consiste en recorrer cada una de las capas de la red, reescalando sus parámetros de forma que sus activaciones de salida tengan media cero y desviación típica uno, aproximadamente.

Se continua con la implementación de la *data block API*, completándolo para que pueda gestionar adecuadamente los conjuntos de datos. Se añaden funcionalidades como la carga de datos; su división en los conjuntos de entrenamiento, validación y prueba; el etiquetado; o la aplicación de transformaciones.

Posteriormente se implementa un optimizador genérico que se puede modificar para adaptar su comportamiento según se desee. Usando esta flexibilidad se implementan: (1) SGD, (2) el descenso del gradiente con momento, (3) Adam, y (4) LAMB. Adicionalmente, también se implementa el declive de los pesos.

Un punto a destacar en la lección es que explica, citando [Laarhoven 2017], que el declive de los pesos no tiene efecto de regularización alguno. Aunque aclaran que es posible que dicha conclusión sea demasiado drástica, pero aun así parece que actualmente no está del todo claro qué hace realmente esta técnica. La lección cita [Zhang et al. 2018; Hoffer et al. 2018] como diferentes intentos de resolver el problema.

Al final de la lección se explica *data augmentation* para problemas de visión artificial y se estudia implementado algunas de las transformaciones utilizadas para tal propósito.

[38] `https://pytorch.org/tutorials/beginner/former_torchies/nnft_tutorial.html#forward-and-backward-function-hooks`.

Recomendaciones para el estudio de la lección 11 Dado que el mecanismo para la gestión y preparación de los datos está basado explícitamente en la estrategia de fastai, es aconsejable estudiar la documentación oficial del *data block API*[39]. Por el mismo motivo, la documentación oficial de fastai sobre transformaciones para *data augmentation*[40] en visión artificial puede ser muy útil.

También se explican una gran cantidad de algoritmos de optimización. El artículo de [Ruder 2016] puede servir como punto de partida para resolver las posibles dudas sobre éstos.

En esta lección se pasa de usar el conjunto de datos MNIST a otro más complejo, denominado Imagenette [Howard 2019]. Este último es en realidad un subconjunto de ImageNet creado por el equipo de fastai. Puede ser interesante echar un vistazo al conjunto de datos original, ya que es uno de los más importantes en aprendizaje profundo aplicado a visión artificial y, en muchas ocasiones, se usa como *benchmark* para medir el rendimiento de nuevos algoritmos.

Comparación con el programa de Ng et al. En esta lección se estudian algunos elementos que no aparecen en el programa de Ng et al., como (1) el mecanismo de inicialización LSUV, (2) la nueva perspectiva sobre la justificación teórica del declive de los pesos, o (3) la implementación detallada de la carga y la gestión de datos. En el curso 2 (sección 3.3.2) se explican todos los algoritmos de optimización estudiados en esta lección, a excepción de LAMB, pero desde una aproximación más teórica. En el curso 4 (sección 3.3.4) se introduce *data augmentation* para problemas de visión artificial, aunque no se hace un estudio práctico tan exhaustivo como en esta lección.

3.4.12 Lección 12: "Advanced training techniques; ULMFiT from scratch"

Resumen del contenido Primero se presentan dos nuevas técnicas de regularización:

1. *Mixup* [Zhang et al. 2017]: Consiste en realizar una combinación lineal de dos muestras distintas, tanto para los valores de entrada como en las etiquetas, y usar el resultado para entrenar la red. Es decir, dadas dos muestras $(\mathbf{x}_i, \mathbf{y}_i)$ y $(\mathbf{x}_j, \mathbf{y}_j)$:

   $$\begin{aligned} \tilde{\mathbf{x}} &= \lambda \mathbf{x}_i + (1-\lambda)\mathbf{x}_j, && \text{donde } \mathbf{x}_i \text{ y } \mathbf{x}_j \text{ son vectores de entrada} \\ \tilde{\mathbf{y}} &= \lambda \mathbf{y}_i + (1-\lambda)\mathbf{y}_j, && \text{donde } \mathbf{y}_i \text{ e } \mathbf{y}_j \text{ son vectores con el etiquetado} \end{aligned}$$

   con $\lambda \in [0, 1]$. De esta forma se obtiene la nueva muestra $(\tilde{\mathbf{x}}, \tilde{\mathbf{y}})$ como combinación lineal de las dos originales.

2. *Label smoothing* [Szegedy et al. 2015]: Consiste en modificar el entrenamiento del modelo para introducir una ligera incertidumbre en sus predicciones. Generalmente, la probabilidad de la clase correcta dada una muestra es 1, mientras que es 0 para

[39] `https://docs.fast.ai/data_block.html`.
[40] `https://docs.fast.ai/vision.transform.html`.

las demás clases. En cambio, *label smoothing* asigna una probabilidad $(1 - \epsilon)$ a la clase correcta y ϵ para las demás, siendo ϵ un número positivo pequeño menor que 1.

Se indica que, para obtener buenos resultados al usar *mixup*, es recomendable utilizarlo junto con *label smoothing*.

Posteriormente se introduce el *mixed precision training*. Generalmente, una DNN se computa usando variables `float` de 32 bits. Existe la posibilidad de modificar este tipo de variables a la mitad de su capacidad, 16 bits (*half precision*), lo cual reduce el uso de memoria y acelera los cálculos. Lamentablemente, no se puede bajar la precisión de las variables en todos los estadios del entrenamiento de una red, debido a que en algunos puntos se puede producir una importante pérdida de información. La solución es precisamente *mixed precision*, que usa ambos tipos de variables, de 16 y 32 bits, dependiendo de la parte del entrenamiento que se esté ejecutando.

A continuación se realiza una implementación de una ResNet [He et al. 2015a]. Concretamente se desarrolla una evolución de esta red, denominada XResNet, que usa algunas de las mejoras propuestas en [He et al. 2018].

Después se realiza una implementación desde cero del aprendizaje por transferencia para un problema de clasificación de imágenes. Primero se entrena una red para el conjunto Imagewoof [Howard 2019] y, posteriormente, se usan los parámetros de este modelo pre-entrenado como base para ajustar otro a los datos del conjunto Pets [Parkhi et al. 2012]. La primera red contiene una base de conocimiento para reconocer razas de perros extraída del conjunto Imagewoof y, usando dicha información, se pretende acelerar el entrenamiento de la segunda red, cuyo objetivo es clasificar razas de perros y gatos.

A continuación se pasa a implementar ULMFiT [Howard y Ruder 2018], un proceso para realizar aprendizaje por transferencia aplicado a tareas de clasificación de textos. Primero se estudia el preprocesamiento de textos que, a grandes rasgos, consiste en (1) dividirlos en tokens, (2) trasformar dichos tokens en números, y (3) construir adecuadamente los lotes con los datos. A continuación se realiza una implementación de una unidad LSTM [Hochreiter y Schmidhuber 1997], para después introducir AWD-LSTM [Merity et al. 2017], una evolución del modelo previo. Por último, se procede a codificar ULMFiT usando como base un modelo entrenado con WikiText 103 [Merity et al. 2016], para luego reutilizar sus parámetros como base del entrenamiento de otra red para el conjunto IMDb [Maas et al. 2011].

Recomendaciones para el estudio de la lección 12 En [Sarofeen et al. 2018] se puede encontrar una presentación, realizada por ingenieros de NVIDIA, que explica cómo realizar *mixed precision training* en PyTorch.

Comparación con el programa de Ng et al. En esta lección se estudian algunos conceptos que no se ven en el programa de Ng et al.: (1) las técnicas de regularización *mixup* y *label smoothing*, (2) *mixed precision training*, (3) el modelo XResNet, (4) el proceso de aprendizaje por transferencia para clasificación de textos, denominado ULMFiT, y

(5) el modelo AWD-LSMT. En el curso 4 (sección 3.3.4) se puede encontrar una extensa explicación sobre el modelo ResNet, aunque no se incluyen las modificaciones que contiene XResNet. Es posible que las explicaciones sobre RNNs en esta lección de fastai y en otras anteriores resulten algo complejas, debido a que el curso de Howard et al. dedica un tiempo limitado a explicar los fundamentos de este tipo de redes. En ese caso, en el curso 5 (sección 3.3.5) se explican las RNNs partiendo de conceptos más elementales y realizando exposiciones más amplias sobre ellas que en el curso de fastai.

3.4.13 Lección 13: "Basics of Swift for Deep Learning"

Resumen del contenido A partir de esta lección se sustituye Python/PyTorch por Swift for Tensorflow, un nuevo marco de trabajo para aprendizaje profundo basado en el lenguaje de programación Swift y la librería TensorFlow. Es un nuevo punto de partida en el que se estudia cómo realizar en este nuevo entorno lo visto previamente en las lecciones 8 a 12.

Lo primero que se explica es por qué actualmente es ventajoso dejar de lado Python por un lenguaje como Swift dentro del contexto del aprendizaje profundo. Además, se da una introducción teórica sobre Swift for TensorFlow y cómo funciona internamente.

Posteriormente se estudian las nociones básicas necesarias para poder programar en Swift: declaración de variables, estructuras de control, etc. Una vez acabado este proceso, también se introduce la programación numérica en Swift y las operaciones básicas con tensores. Además, se explica cómo integrar librerías de Python en Swift.

Conociendo los elementos básicos de Swift, se pasa a implementar el operador de multiplicación de matrices, de forma similar a como se hizo en la lección 8. Después de introducir el operador de multiplicación de matrices de Swift, se explica la API de TensorFlow en este lenguaje.

Finalmente, se pasa directamente a implementar y entrenar una XResNet de forma similar a la realizada en la lección 12; mostrando que, aunque Swift for TensorFlow todavía está en fase de desarrollo, es posible construir eficazmente un modelo avanzado.

Recomendaciones para el estudio de la lección 13 Esta lección no explica nada nuevo sobre aprendizaje profundo, sino que se centra en explicar el funcionamiento de Swift for TensorFlow. Por ello, puede ser interesante dedicar un tiempo al estudio de este lenguaje para familiarizarse con él. Como ya hemos dicho, en [Apple Inc. 2019] se puede encontrar un tutorial muy completo a este respecto. Por otro lado, también sería conveniente echar un vistazo a los tutoriales oficiales y a la documentación de la API en la página de TensorFlow[41].

Comparación con el programa de Ng et al. El programa de Coursera no estudia Swift for TensorFlow.

[41] `https://www.tensorflow.org/swift`.

3.4.14 Lección 14: "C interop; Protocols; Putting all together"

Resumen del contenido Esta lección continúa con el tutorial de Swift for TensorFlow.

Se introduce XLA[42], un nuevo compilador específico para álgebra lineal que realiza optimizaciones de bajo nivel durante la construcción de los grafos de computación, aumentando la velocidad de las operaciones y reduciendo el uso de memoria respecto a ejecuciones de TensorFlow que no lo usan. Así mismo, se presenta MLIR[43], un formato de representación para TensorFlow que se sitúa entre la propia representación del modelo y los compiladores de bajo nivel encargados de generar el código específico para cada hardware. Aunando estos dos elementos, el equipo de desarrollo de TensorFlow desea presentar un lenguaje declarativo para el aprendizaje automático y el aprendizaje profundo en un futuro cercano, similar a SQL para los gestores de bases de datos, es decir, un lenguaje en que el usuario exprese qué es lo que quiere obtener pero que sea el compilador el que se encargue de encontrar la forma más eficiente de calcularlo.

Posteriormente se explica cómo integrar librerías de C en Swift y se realiza un ejemplo con la librería SoX[44] para decodificar un archivo de audio en MP3.

Finalmente, se explica el desarrollo en Swift de toda la funcionalidad intermedia entre la codificación del operador de multiplicación de matrices y el entrenamiento de la XResNet cubiertos en la lección anterior.

Recomendaciones para el estudio de la lección 14 En esta lección se cubre rápidamente todo el desarrollo de la librería de aprendizaje profundo usando el entorno Swift for TensorFlow. Dado que la teoría ya se ha estudiado en lecciones previas, sería conveniente repasar más pausadamente los *notebooks* donde está detallado todo el proceso en este nuevo lenguaje. De la misma forma, puede ser interesante comparar ambas implementaciones en Swift y Python. En GitHub[45] se pueden encontrar ambas versiones de los *notebooks*.

3.5 Comparación entre los MOOCs de Ng et al. y de Howard et al.

En las secciones precedentes ya hemos señalado algunas diferencias puntuales entre ambos MOOCs. En ésta ofrecemos una comparación global entre ellos. En cuanto a la metodología docente, los dos MOOCs estudiados tienen dos aproximaciones complementarias al estudio del aprendizaje profundo. Por un lado, el programa de Ng et al. se organiza de una forma más tradicional: se presenta primero la teoría y posteriormente se realizan actividades prácticas sobre los conceptos estudiados. En cambio, el curso de Howard et al. está centrado en aprender a través de la práctica, es decir, se experimenta con algoritmos del estado del

[42] `https://www.tensorflow.org/xla`.
[43] `https://www.tensorflow.org/mlir`.
[44] `http://sox.sourceforge.net`.
[45] `https://github.com/fastai/course-v3/tree/master/nbs`.

arte para entender luego los conceptos teóricos que los sustentan [Perkins 2009]. Además, el enfoque clásico del programa de Coursera implica el estudio secuencial y detallado de los conceptos más simples y su combinación para definir otros más complejos. El de fastai, en cambio, apuesta por ofrecer primero una panorámica global del aprendizaje profundo y posteriormente ir refinando las ideas generales para mostrar los detalles. Esta forma de organizar los cursos hace que el de Ng et al. destaque por sus explicaciones teóricas, mientras que el de Howard et al. otorgue más relevancia a la experiencia práctica.

En cuanto al contenido, en ambos cursos se presentan ideas o conceptos que no aparecen en el otro. Ng et al. explican problemas como la detección de objetos, el reconocimiento facial, la traducción automática o el reconocimiento del habla, que no se estudian en el curso de fastai; mientras que la segmentación de imágenes, la restauración de imágenes, el filtrado colaborativo o el procesamiento de datos tabulados con DNNs, únicamente se ven en el de Howard et al. También cabe decir que el MOOC de Ng et al. fue realizado dos años antes que el otro y por eso no es de extrañar que en el primero no aparezcan técnicas más recientes, como el *learning rate finder* o el planificador de hiperparámetros *one cycle*. En la tabla 3.2 se puede encontrar una comparación más detallada de los contenidos de ambos cursos.

El curso de Howard et al. no dedica mucho tiempo al estudio de los conceptos básicos sobre RNNs y datos secuenciales en favor de, por ejemplo, problemas de visión artificial, que se explican con más detalle. Aun así, en dicho curso se llega a técnicas más avanzadas, como el uso de ULMFiT para realizar aprendizaje por transferencia en este tipo de redes. Por otro lado, en el programa de Ng et al. se estudian más ampliamente los fundamentos de las RNNs y, además, se explica un rango de aplicaciones más amplio de éstas que en el curso de Howard et al.

Un punto interesante del programa de Ng et al. que no se cubre en el otro curso es la metodología de trabajo para proyectos de aprendizaje automático; se explica cómo organizar el desarrollo de un sistema de forma iterativa e incremental y, además, enseña técnicas para ayudar en la toma de decisiones y seleccionar la línea de trabajo más prometedora en cada fase del desarrollo.

En cuanto a la evaluación y la acreditación, el programa de Ng et al. da la oportunidad de obtener un certificado o título si se realizan y superan con éxito los ejercicios propuestos a lo largo de éste. En cambio, en el curso de Howard et al. no hay ejercicios en sí que corregir, pero se anima a que el estudiante lleve a cabo sus propios experimentos. Los responsables del curso invitan a que los alumnos desarrollen sus propios proyectos personales a partir de los conocimientos obtenidos en cada lección. Además, al no haber material que corregir, tampoco se proporciona ningún certificado que pruebe que se ha superado el curso con éxito.

En cuanto a la elección de un MOOC, ambos cursos son un buen punto de inicio para el estudio del aprendizaje profundo y la decisión sobre cuál de ellos realizar dependerá de las preferencias e inquietudes de la persona interesada. El de Ng et al. tiene un enfoque más teórico y posiblemente sea preferible para aquellas personas que aprendan mejor estudiando primero la teoría y, posteriormente, aplicándola en ejercicios concisos y evaluables. En cambio, el de Howard et al. puede ser más interesante para aquellas personas que aprendan mejor experimentando o que tengan un perfil más técnico. Sin embargo, ambos cursos

son complementarios y es posible (incluso recomendable) realizar ambos. Dos maneras de hacerlo pueden ser:

1. Cursar primero el de Ng et al. para alcanzar un buen nivel teórico sobre el aprendizaje profundo y, posteriormente, el de Howard et al. para ponerlo en práctica y afinar las habilidades técnicas.

2. Cursar primero la parte 1 de Howard et al. para obtener una visión global del aprendizaje profundo y adquirir las nociones básicas; realizar luego el de Ng et al. para profundizar en los conocimientos teóricos y, finalmente, estudiar la segunda parte del de fastai para aprender cómo programar algoritmos del estado del arte.

La primera aproximación está pensada para aquellas personas que prefieren estudiar primero los conceptos elementales del aprendizaje profundo, mientras que la segunda es mejor para aquellas que prefieran ver primero una panorámica de lo que se puede hacer hoy en día con el aprendizaje profundo. Estudiar el de Ng et al. antes que la segunda parte del otro curso puede aportar ciertos conocimientos teóricos que faciliten la implementación de los algoritmos que proponen Howard et al.

Contenido	Ng et al.	Howard et al.
– funciones de activación (sigmoide, tanh, ReLU, linear, softmax,...)	✓	✓
– funciones de pérdida (MSE, entropía cruzada,...)	✓	✓
– vectorización	✓	✓
– grafo de computación (pasos hacia delante y hacia atrás)	✓	✓
– derivación y *autograd*	✓	✓
– optimizadores (GD, SGD, GD por mini-lotes, *momentum*, RMSprop y Adam)	✓	✓
– LAMB (optimizador)		✓
– regresión	✓	✓
– clasificación binaria, clasificación multi-clase (softmax y entropía cruzada)	✓	✓
– clasificación multi-etiqueta		✓
– conjuntos de entrenamiento, desarrollo/validación y prueba	✓	✓
– análisis de preferencia-varianza (sub-ajuste y sobreajuste)	✓	✓
– regularización (L2, *dropout*, *early stopping*)	✓	✓
– *data augmentation*	✓	✓
– *mixup*, *label smoothing* (regularización)		✓
– problema del desvanecimiento y la explosión de los gradientes	✓	✓
– inicializaciones de Xavier y Kaiming	✓	✓
– *Layerwise Sequential Unit Variance* (LSUV)		✓
– *discriminative learning rates*		✓
– ajuste de hiperparámetros	✓	✓
– normalización por lotes (BatchNorm)	✓	✓
– LayerNorm, InstanceNorm, GroupNorm, *running* BatchNorm		✓
– *learning rate finder*		✓
– *annealing* o planificador de hiperparámetros (*one cycle*)		✓
– análisis de las activaciones		✓
– *mixed precision training*		✓
– gestión de proyectos de aprendizaje automático	✓	
– *orthogonalization*	✓	
– aprendizaje profundo extremo-a-extremo y procesamiento de múltiples etapas	✓	
– análisis manual del error	✓	✓
– aprendizaje por transferencia	✓	✓
– aprendizaje multi-tarea	✓	
– visión artificial	✓	✓
– redes neuronales convolucionales (ConvNets, CNNs)	✓	✓
– LeNet-5, AlexNet, Inception Network	✓	
– VGG-16, ResNet	✓	✓
– DenseNet, U-Net		✓
– clasificación de imágenes	✓	✓
– detección de objetos (YOLO, R-CNN)	✓	
– segmentación de imágenes		✓
– regresión de imágenes	✓	✓
– restauración de imágenes		✓
– *neural style transfer*	✓	
– reconocimiento facial (*siamese network*, *triplet loss function*)	✓	
– *generative models* (GANs, *feature losses*)		✓
– *recurrent neural networks* (RNNs)	✓	✓
– procesamiento del lenguaje natural (PLN)	✓	✓
– *one-hot encoding*, *word embeddings*	✓	✓
– *debiasing word embeddings*	✓	
– modelo de lenguaje	✓	✓
– *Gated Recurrent Unit* (GRU)	✓	
– *Long Short-Term Memory* (LSTM)	✓	✓
– AWD-LSTM		✓
– Bidirectional RNNs (BRNNs), RNN Encoder-Decoder, Attention Model	✓	
– ULMFiT		✓
– clasificación de sentimientos	✓	✓
– generación automática de pies de foto	✓	
– traducción automática (*bleu score*)	✓	
– reconocimiento del habla, detección de palabras de activación	✓	
– filtrado colaborativo		✓
– datos tabulados (estructurados) con DNNs		✓
– representación de variables categóricas con *embeddings*		✓
– ética en la IA		✓

Tabla 3.2: Contenidos de los cursos de aprendizaje profundo. Los contenidos exclusivos de Ng et al. se muestran en azul y los de Howard et al. en magenta. Los contenidos en negro se estudian en ambos cursos.

Capítulo 4

Análisis empíricos

En nuestro análisis experimental hemos estudiado el proceso de entrenamiento de redes neuronales profundas tanto en Keras[1] como en fastai [Howard y Gugger 2020]. Para ello hemos abordado un problema de clasificación multi-clase para visión artificial mediante aprendizaje por transferencia, uno de los problemas más típicos del aprendizaje profundo hoy en día.

Ambas librerías tienen objetivos y niveles de abstracción distintos. La idea original de Keras consiste en proporcionar una interfaz común a varias librerías de aprendizaje profundo, como Theano, TensorFlow o CNTK. Además, implementa cierta funcionalidad básica, como el bucle de entrenamiento, la carga de datos, *data augmentation*, etc., y define valores por defecto para algunos hiperparámetros. Sin embargo, tras la publicación de TensorFlow 2.0 en 2019, Keras ha dejado de dar soporte para otras librerías de aprendizaje profundo y ahora únicamente ofrece su interfaz de alto nivel a TensorFlow. Por otro lado, fastai se centra exclusivamente en PyTorch, proporcionando procedimientos prediseñados para resolver ciertos problemas de áreas como la visión artificial o el procesamiento de lenguaje natural. El equipo de fastai estudia estos problemas y provee de los mejores mecanismos y una configuración por defecto para abordar con éxito la mayor parte de los casos. Aun así, fastai es también totalmente configurable.

En la sección 4.1 se da un esquema general de cómo entrenar una ConvNet tanto en Keras como en fastai, y en la sección 4.2 se muestran los resultados del entrenamiento de varias ConvNets con el conjunto de datos *Skin Cancer ISIC* [Katanskiy 2019]. En los experimentos realizados tan sólo hemos usado la funcionalidad que ya viene implementada en ambas librerías.

4.1 Entrenamiento de ConvNets con Keras y fastai

A continuación se presentan los pasos básicos para entrenar un clasificador de imágenes con Keras y fastai usando aprendizaje por transferencia.

[1] `https://keras.io`.

4.1.1 Limpieza del conjunto de datos

Es posible que los datos no vengan perfectamente preparados para usarlos directamente en un entrenamiento. Hay ciertas tareas previas que se deben realizar, como eliminar imágenes duplicadas o detectar ejemplos mal etiquetados. Ni Keras ni fastai proporcionan funcionalidad específica para ello, aunque esta última sí que ofrece una API con la que explorar las imágenes del conjunto y filtrarlas.

4.1.2 Carga y preprocesado del conjunto de datos

Durante esta etapa del proceso se preparan los datos para poder entrenar con ellos una ConvNet. Suele constar de dos partes:

1. *Cargar las imágenes en lotes*: Habitualmente no es posible cargar todas las imágenes de un conjunto de datos a la vez en la memoria por falta de espacio. A causa de ello se suele crear algún tipo de objeto (en el lenguaje de programación) que agrupe aleatoriamente estas imágenes en lotes de longitud fija. Este procedimiento se realiza en cada época de entrenamiento y únicamente se cargan a la vez en memoria las imágenes que pertenecen a uno de estos lotes. De esta forma, cuando se ha procesado un lote, se elimina de la memoria y se carga el siguiente, continuando este proceso hasta que finaliza la época que se está ejecutando. Siguiendo este método se procesan tanto los datos de entrenamiento como los de validación y prueba.

2. *Preprocesar las imágenes*: No es conveniente introducir una imagen tal cual se carga en una ConvNet, sino que es recomendable ajustar previamente sus valores. Lo primero que se suele hacer es normalizar o modificar la escala de estos valores de alguna forma para que estén dentro de cierto rango de valores, como $[0, 1]$ o $[-1, 1]$. El preprocesamiento para el aprendizaje por transferencia es algo diferente. Debido a que el modelo se inicializa con pesos pre-entrenados, hay que aplicar el mismo preprocesamiento con el que se obtuvieron dichos pesos. Las librerías suelen proporcionar estos mecanismos de preprocesamiento asociados a los pesos pre-entrenados. De no usarlos, los valores de los datos no estarán dentro del rango esperado y el modelo no los procesará correctamente, dificultando el entrenamiento. Por motivos similares, también hay que usar la misma normalización para los datos de entrenamiento, validación y prueba. Por otro lado, se puede aplicar a las imágenes un conjunto de transformaciones aleatorias para llevar a cabo *data augmentation*.

Keras Las principales características de la carga y el preprocesado de datos en Keras son las siguientes:

- Es necesario realizar previamente la división en tres subconjuntos: entrenamiento, validación y prueba.
- Se basa principalmente en la clase `ImageDataGenerator`[2], cuya funcionalidad básica es la siguiente:
  - Modifica la escala de los valores que definen los píxeles de las imágenes según las indicaciones del usuario. Normalmente se multiplican sus valores por 1/255

[2] `https://keras.io/preprocessing/image`.

para dejarlos dentro del rango $[0, 1]$, suponiendo que cada píxel de la imagen está codificado en 24-bits: 3 canales de color y 8 bits por canal.

  - Se puede pasar una función de preprocesado. Es fundamental para el aprendizaje por transferencia ya que, en Keras, cada modelo preentrenado tiene asociada una función de preprocesamiento específica que hay que aplicar a todos los datos antes del entrenamiento. De esta forma, nos aseguramos que los valores de éstos estén dentro de la escala esperada por el modelo.
  - Puede aplicar transformaciones a las imágenes para *data augmentation*. Algunas de las que tiene son: *flip*, *shift*, *rotation*, *zoom*, *brightness* o *shear* [Brownlee 2019]. Estas transformaciones son más elementales que las que se pueden encontrar en fastai.
  - Crea un `iterator` que se encarga de la generación de lotes. Además, este objeto adapta las dimensiones de cada imagen a aquéllas deseadas. Éstas tienen que ser las mismas que definen la entrada del modelo que se va a usar.
  - Puede ser necesario crear un objeto de la clase `ImageDataGenerator` distinto para cada uno de los subconjuntos (entrenamiento, validación y prueba), junto con sus correspondientes `iterators`. Por ejemplo, si se usa *data augmentation*, es conveniente utilizar un `ImageDataGenerator` exclusivo para los datos de entrenamiento, ya que no hay que aplicar este tipo de regularización a los datos de validación y prueba.

- La carga y preprocesado de datos requiere de un trabajo más manual que con fastai.

fastai Esta librería tiene una API muy elaborada para la carga y el preprocesado de los datos; `data block`[3]:

- Ofrece diferentes mecanismos para (1) cargar los datos en función de la fuente de los mismos (directorio, dataframe, csv), (2) filtrar archivos no deseados, (3) dividir los datos en los subconjuntos de entrenamiento y validación, y (4) etiquetar los datos.

- Normaliza las imágenes de la misma forma (empleando la misma media y desviación típica) que durante la obtención de los pesos pre-entrenados con los que se va a inicializar el modelo en el proceso de aprendizaje por transferencia.

- Ajusta las dimensiones de las imágenes (posteriormente, también se configurará con ellas las dimensiones de la entrada del modelo a entrenar) y configura el tamaño de los lotes.

- Se puede aplicar un conjunto de transformaciones a las imágenes para *data augmentation*[4]. Ofrece una gran cantidad de funciones para este propósito como: *brightness*, *contrast*, *crop*, *flip*, *jitter*, *pad*, *perspective warp*, *rotate*, *rgb randomize*, *skew*, *squish*, *symmetric warp*, *tilt*, *zoom*, *cut out*, etc. Tiene un número más variado de transformaciones que Keras y, además, proporciona una configuración por defecto que suele funcionar bien para muchos problemas de visión artificial o, al menos, sirve como punto de partida.

[3] `https://docs.fast.ai/data_block.html`.
[4] `https://docs.fast.ai/vision.transform.html`.

- Toda la información de los datos queda encapsulada en un objeto de la clase `DataBunch`[5]. Éste se usa durante el resto del proceso de entrenamiento como interfaz del conjunto de datos.
- La tarea de carga y preprocesado de los datos es más automática que con Keras. Básicamente lo único que requiere es seleccionar las funciones adecuadas para el conjunto de datos concreto.
- Permite mostrar fácilmente un lote de imágenes mediante el método `show_batch()`.

4.1.3 Configuración del modelo

Durante este proceso se selecciona la arquitectura del modelo, se modifican su entrada y su salida (cabeza), y se inicializan los pesos.

Keras Proporciona varios modelos para aprendizaje por transferencia[6]. Las principales características de éstos y la forma de configurarlos son las siguientes:

- Arquitecturas disponibles[7]: Xception [Chollet 2016], VGG(16, 19) [Simonyan y Zisserman 2015], ResNet(50, 101, 152) [He et al. 2015a], Inception [Szegedy et al. 2014], InceptionResNet [Szegedy et al. 2016], MobileNet [Howard et al. 2017], DenseNet(121, 169, 201) [Huang et al. 2016] y NASNet(Large, Mobile) [Zoph et al. 2017].
- Los modelos se puede inicializar con pesos aleatorios o con otros preentrenados con ImageNet.
- Hay que adaptar manualmente la entrada del modelo y su salida (cabeza) para que se ajuste al problema con el que se está tratando.

fastai También ofrece modelos para aprendizaje por transferencia[8]. Sus principales características son las siguientes:

- Arquitecturas disponibles: ResNet(18, 34, 50, 101, 152), SqueezeNet(v1.0, v1.1) [Iandola et al. 2016], DenseNet(121, 161, 169, 201), VGG(16, 19)[9], AlexNet [Krizhevsky et al. 2012], DarkNet[10], U-Net [Ronneberger et al. 2015] y Wide ResNet [Zagoruyko y Komodakis 2016].
- Los modelos se inicializan con los pesos preentrenados con ImageNet o aleatoriamente.
- Utilizando el objeto `DataBunch`, fastai adapta automáticamente la entrada y la salida (cabeza) del modelo[11]. Esta cabeza creada automáticamente está formada por (1) una capa `AdaptiveConcatPool`, (2) una capa `Flatten` y (3) uno o varios bloques `[BatchNorm1d, Dropout, Linear, ReLU]`. Así mismo, si no se desea este comportamiento automático, se puede proporcionar una cabeza creada manualmente.

[5] `https://docs.fast.ai/basic_data.html#DataBunch`, `https://docs.fast.ai/vision.data.html#ImageDataBunch`.

[6] `https://keras.io/applications`.

[7] Entre paréntesis se muestra el número de capas o versiones que son posibles seleccionar.

[8] `https://docs.fast.ai/vision.models.html`.

[9] La implementación de las redes VGG de fastai incluye capas BatchNorm, al contrario que Keras.

[10] DarkNet es la base de YOLOv3 [Redmon y Farhadi 2018].

[11] `https://docs.fast.ai/vision.learner.html#cnn_learner`.

4.1.4 Entrenamiento

Ésta es la fase en la que se ajustan los pesos del modelo previamente cargado usando los datos de entrenamiento. Además, se usan los datos de validación para evaluar la calidad del modelo y ajustar los hiperparámetros.

Keras Las principales características del entrenamiento en Keras son:

- La congelación y descongelación (ver la sección 3.4.1) de las capas pertenecientes al cuerpo del modelo se realizan manualmente mediante un bucle. Esta forma de hacerlo permite al usuario elegir aquellas capas concretas que se desean congelar o descongelar.
- Se puede añadir regularización L1 y/o L2 asignándola manualmente a cada capa del modelo, aunque esto resulta algo más complicado para modelos pre-entrenados. Añadir regularización L1 y/o L2 de esta forma sólo afecta a la configuración del modelo pero no se llega a aplicar al objeto en sí y, por tanto, no se utilizará durante el entrenamiento. Para que la regularización L1 y/o L2 tenga efecto hay que: (1) obtener el JSON[12] con la configuración del modelo, (2) guardar los pesos de las capas e (3) instanciar un nuevo modelo con el JSON y los pesos previamente almacenados. Los detalles de este procedimiento se encuentran en [Silva 2019].
- Se pueden configurar diversos optimizadores (SGD, RMSprop, Adam, etc) (sección 3.3.2), funciones de pérdida (MSE, entropía cruzada, etc.)[13] y métricas (exactitud, *cosine proximity*, etc)[14], pero las métricas disponibles son más limitadas que en fastai.
- Carece de características avanzadas de entrenamiento como *learning rate finder*, *discriminative learning rates* o *one cycle*, por ejemplo. Si se quieren usar, hay que implementarlas manualmente.

fastai Las principales características de un entrenamiento en fastai son:

- Proporciona una clase `Learner`[15] que se encarga de gestionar todo el proceso de entrenamiento.
- Proporciona métodos para congelar y descongelar automáticamente las capas pertenecientes al *body* del modelo (`freeze()` y `unfreeze()`, respectivamente). También proporciona un método adicional, `freeze_to()`, para congelar un grupo de capas concreto. Aparte de estos mecanismos, siempre es posible realizar estas operaciones manualmente recorriendo las capas del modelo mediante un bucle, como en Keras, lo que permite ser más preciso a la hora de seleccionar las capas concretas que se quieren congelar o descongelar.
- Ofrece dos mecanismos de regularización, declive de los pesos y *mixup*, los cuales se configuran simplemente añadiendo un determinado parámetro o realizando una llamada a una función específica, respectivamente.

[12] JSON es un formato en texto plano para el intercambio de objetos datos compuestos por pares atributo-valor y arrays. `https://en.wikipedia.org/wiki/JSON`.

[13] `https://keras.io/losses/`.

[14] `https://keras.io/metrics/`.

[15] `https://docs.fast.ai/basic_train.html#Learner`.

- Tiene diversos optimizadores, funciones de pérdida y métricas. Usa optimizadores[16] y funciones de pérdida[17] de PyTorch. Normalmente la función de pérdida se configura de forma automática dependiendo del problema a resolver, aunque se puede seleccionar una manualmente. Tiene muchas opciones de métricas por defecto[18].
- Dispone de varios mecanismos avanzados para el entrenamiento: *learning rate finder*[19], *discriminative learning rates*[20] y *one cycle*[21].

4.1.5 Interpretación de los resultados

Una vez entrenado un modelo, en esta etapa se determina la calidad del mismo.

Keras Esta librería únicamente almacena cierta información de los entrenamientos para una posterior interpretación:

- Tras el entrenamiento se puede obtener un objeto `History`[22] que contiene, entre otra información, los valores de pérdida y métricas en cada época de entrenamiento para los conjuntos de entrenamiento y validación.
- No proporciona ninguna funcionalidad específica para interpretación de los resultados. Hay que usar otras librerías o debe ser implementada por el usuario. Por ejemplo, se tienen que crear manualmente las gráficas que muestran los valores de pérdida y las métricas durante el entrenamiento[23] o usar otra librería, como scikit-learn[24], para obtener la matriz de confusión.

fastai Proporciona más herramientas que Keras para determinar la calidad de los modelos entrenados:

- Tiene funciones para imprimir gráficas que muestran las variaciones de los valores de pérdida, métricas y *momentums* a lo largo del entrenamiento.
- Proporciona la clase `ClassificationInterpretation`[25], que permite obtener ciertas representaciones para interpretar los resultados:
  - El método `top_losses()` imprime los valores de pérdida más grandes o más pequeños.
  - `plot_top_losses()` muestra las imágenes con mayores valores de pérdida.
  - `plot_confusion_matrix()` muestra la matriz de confusión.
  - `most_confused()` imprime una lista con las clases más confusas, es decir, aquellas que tienen los valores más altos fuera de la diagonal en la matriz de confusión.

[16] `https://pytorch.org/docs/stable/optim.html`.
[17] `https://pytorch.org/docs/stable/nn.html#loss-functions`.
[18] `https://docs.fast.ai/metrics.html`.
[19] `https://docs.fast.ai/callbacks.lr_finder.html`.
[20] `https://docs.fast.ai/basic_train.html#Discriminative-layer-training`.
[21] `https://docs.fast.ai/callbacks.one_cycle.html`.
[22] `https://keras.io/callbacks/#history`.
[23] `https://keras.io/visualization`.
[24] `https://scikit-learn.org`.
[25] `https://docs.fast.ai/train.html#ClassificationInterpretation`, `https://docs.fast.ai/vision.learner.html#ClassificationInterpretation`.

4.2 Desarrollo experimental

En nuestra evaluación empírica hemos construido una serie de clasificadores multi-clase utilizando el conjunto de datos *Skin Cancer ISIC* [Katanskiy 2019], el cuál contiene 2.357 imágenes de la piel de diferentes pacientes, agrupadas en 9 clases distintas de enfermedades oncológicas benignas y malignas. El conjunto está dividido con el 95 % de los datos para entrenamiento y el 5 % para prueba. La figura 4.1a muestra la distribución de los datos entre las 9 clases.

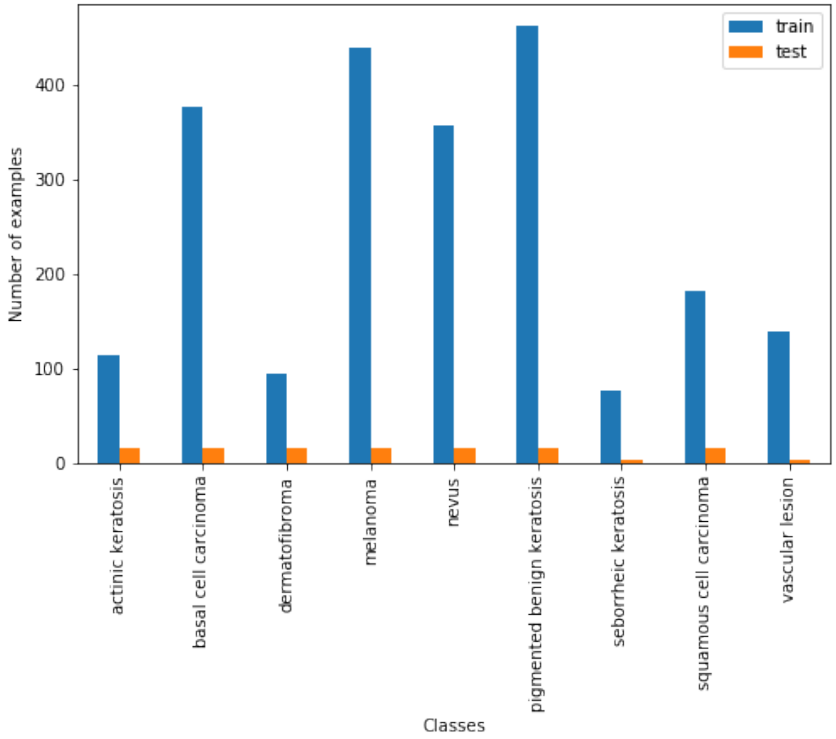
4.2.1 Análisis del conjunto de imágenes

El proceso de limpieza de este conjunto ha sido el siguiente:

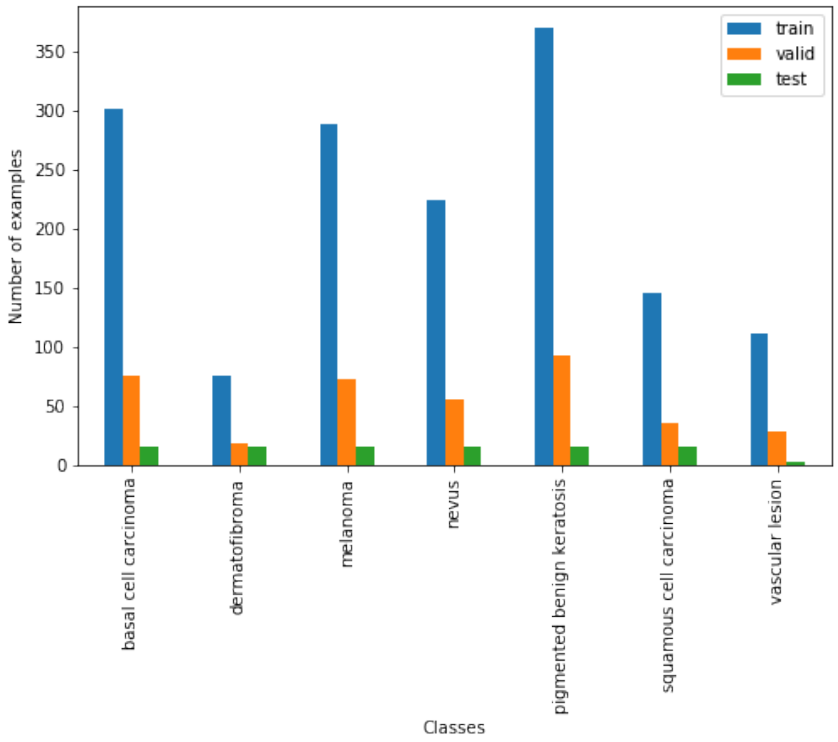
1. Hemos observado que el conjunto contiene 155 imágenes duplicadas; cada una de ellas está repetida en dos clases distintas. Ante la imposibilidad de decidir cuál era la clase correcta, hemos eliminado todas ellas, es decir, 310 imágenes, dejando así 2.047 imágenes.

2. Después de la eliminación de duplicados, las clases *actinic keratosis* y *seborrheic keratosis* han quedado con un número de ejemplos muy bajo, 53 y 2 imágenes respectivamente. Debido a esto, hemos decidido eliminarlas, dejando únicamente 7 clases en el conjunto. Esta operación ha reducido el total de imágenes a 1.992.

De forma adicional, hemos creado un subconjunto de validación tomando aleatoriamente el 20 % de los datos de entrenamiento originales. Después de este proceso, el conjunto contiene 1.515 (76.05 %) imágenes de entrenamiento, 378 (18.98 %) de validación y 99 (4.97 %) de prueba. En la figura 4.1 se puede ver la distribución del número de muestras en la versión original del conjunto de datos y después de pasar por este proceso de limpieza y división.



(a)



(b)

Figura 4.1: Número de imágenes por clase en el conjunto de datos. Arriba, el conjunto de datos original. Abajo, el número resultante tras eliminar las imágenes duplicadas y las clases poco pobladas, y extraer el 20 % de las imágenes para el conjunto de validación.

4.2.2 Diseño de los experimentos

En nuestros experimentos hemos utilizado Keras como interfaz de TensorFlow. Hemos aplicado varias transformaciones para *data augmentation*: *vertical flip*, *horizontal flip*, *rotation*, *zoom*, *brightness* y *shear*. Se han configurado de tal manera que varíen ligeramente las imágenes originales, sin distorsionarlas hasta tal punto que su contenido no sea reconocible. En cambio, en fastai se ha utilizado la configuración por defecto de las transformaciones para *data augmentation*[26], añadiendo *vertical flip*.

En Keras hemos inicializado los modelos utilizando los pesos pre-entrenados con ImageNet. Se ha creado una cabeza nueva para cada modelo con las siguientes capas: (1) `GlobalAveragePooling`, (2) `BatchNormalization`, (3) `Dropout`, (4) `Dense`, (5) `ReLU`, (6) `BatchNormalization`, (7) `Dropout` y (8) `Dense` (con activación softmax). Las capas `Dense` se han inicializado aleatoriamente utilizando del método de Kaiming uniforme[27] (sección 3.3.2). Es decir, el tensor de pesos se ha inicializado aleatoriamente usando una distribución uniforme $U(-1, 1)$ y después los valores obtenidos se han multiplicado por $\sqrt{6/n}$, donde n es el número de unidades de entrada de dicho tensor. De esta forma, los pesos tienen media 0 y desviación típica $\sqrt{6/n}$, evitando el problema del desvanecimiento y la explosión de los gradientes. Los sesgos se han inicializado a cero. La última de estas capas se ha configurado con 7 unidades, una por cada clase. En los experimentos con fastai también hemos usado los pesos de ImageNet para inicializar los modelos, pero hemos permitido que esta librería adapte automáticamente las entradas y salidas de los modelos empleados, tal como se indica en la sección 4.1.3, e inicialice la capas de la cabeza usando la configuración por defecto[28].

En Keras hemos añadido la regularización L2 a todas las capas convolucionales y a las completamente conexas. Hemos utilizado el optimizador Adam usando los valores por defecto que asigna esta librería a los *momentums* (β_1 y β_2). Cada entrenamiento ha constado de dos fases: (1) ajuste de la nueva cabeza mientras las capas del cuerpo están congeladas (excepto las capas BatchNorm, para que puedan ajustar la media y desviación típica de las activaciones de forma adecuada para el nuevo problema) y (2) un ajuste fino con todas las capas de la red descongeladas. Cada una de estas dos fases tiene una tasa de aprendizaje distinta, tomando un valor más bajo para la segunda. Hemos configurado el mismo número de épocas en todos los experimentos: 50 para la primera fase y 100 para la segunda. Son valores altos escogidos para observar el comportamiento del entrenamiento, viendo si éste converge y si se produce sobreajuste. La tabla 4.1 muestra la configuración común a todos los experimentos.

Para los experimentos en fastai también hemos utilizado el optimizador Adam, asignado los valores por defecto para los *momentums* (β_1 y β_2) que define esta librería al usar el planificador de hiperparámetros *one cycle*. En todos los entrenamientos se ha usado el mismo valor para el declive de los pesos. Al igual que en Keras, cada entrenamiento se ha dividido en dos fases: (1) ajuste en exclusiva de las capas de la cabeza (junto con las capas BatchNorm del cuerpo) y posteriormente (2) ajuste fino con todas las capas descongeladas. Para estas dos fases hemos utilizado el *learning rate finder* proporcionado

[26] `https://docs.fast.ai/vision.transform.html#get_transforms`.
[27] `https://keras.io/initializers/#he_uniform`.
[28] `https://docs.fast.ai/vision.learner.html#cnn_learner`.

para determinar la mejor tasa de aprendizaje (pero finalmente hemos usando las mismas tasas de aprendizaje en todos los experimentos para facilitar la comparación y debido a que el *learning rate finder* no estimaba valores muy distintos entre ellos) y hemos ajustado los pesos mediante *one cycle*. En la segunda fase hemos usado *discriminative learning rates* para asignar tasas de aprendizaje más bajas a las primeras capas del modelo y otras más altas a las últimas. Por motivos similares que en Keras, hemos usado el mismo número de épocas en todos los experimentos: 15 para la primera fase y 40 para la segunda. La tabla 4.1 muestra la configuración común a todos los experimentos.

configuración	**Keras**	**fastai**
imagen de entrada	256×256	256×256
tamaño del lote	32	32
data augmentation	sí	sí
regularización L2 / declive de los pesos	0.01	0.01
optimizador	Adam ($\beta_1 = 0.9$, $\beta_2 = 0.999$)	Adam ($\beta_1 = 0.95$, $\beta_2 = 0.85$)
one cycle	no	sí
discriminative learning rates	no	sí
tasa de aprendizaje 1	3×10^{-3}	3×10^{-3}
tasa de aprendizaje 2	3×10^{-5}	$[5 \times 10^{-6}, 5 \times 10^{-4}]$
épocas 1	50	15
épocas 2	100	40

Tabla 4.1: Configuración e hiperparámetros para el aprendizaje con el conjunto de datos Skin Cancer ISIC. La *tasa de aprendizaje 1* y el número de *épocas 1* indican los valores usados para estos hiperparámetros durante el ajuste de la cabeza del modelo. La *tasa de aprendizaje 2* y el número de *épocas 2* muestran los respectivos valores usados durante el ajuste fino del modelo.

Tanto en Keras como en fastai se han entrenado 6 modelos distintos: VGG16, VGG19, ResNet50, ResNet101, DenseNet121 y DenseNet169. Hemos escogido estos modelos concretos porque son de los pocos que comparten ambas librerías para clasificación de imágenes.

4.2.3 Resultados

La tabla 4.2 muestra los resultados obtenidos. Además, en el apéndice A se encuentran las gráficas con los valores de pérdida y de exactitud obtenidos durante los entrenamientos. Los experimentos se han realizado en una máquina con GNU Linux Ubuntu 18.04 LTS usando una GPU NVIDIA GeForce GTX 1070 8GB.

Tanto en Keras como en fastai hemos entrenado los modelos con muchas más épocas de las necesarias. Esto queda reflejado en las gráficas de pérdida y exactitud, donde se puede observar que los modelos se estabilizan mucho antes de que acaben los entrenamientos La figura 4.2 muestra estas gráficas para el entrenamiento de un modelo ResNet50 como ejemplo. Además, los modelos entrenados con fastai se estabilizan más rápido que en

	pérdida valid.		**exactitud valid.**	
modelo	**Keras (100 ép.)**	**fastai (40 ép.)**	**Keras (100 ép.)**	**fastai (40 ép.)**
VGG16	5.980	0.451	0.757	0.854
VGG19	6.650	0.418	0.577	0.868
ResNet50	9.474	0.469	0.757	0.873
ResNet101	8.734	0.507	0.720	0.873
DenseNet121	9.093	0.443	0.786	0.884
DenseNet169	7.637	0.404	0.783	0.881

(a) Pérdida y exactitud. Únicamente se muestran los resultados referentes al ajuste fino de los modelos. Los valores de pérdida de Keras incluyen los pesos de los modelos, debido al uso de la regularización L2. No son comparables entre modelos ni con los de fastai.

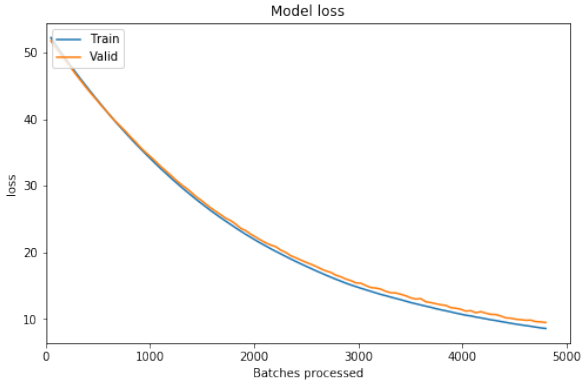
	tiempo 1		**tiempo 2**	
modelo	**Keras (50 ép.)**	**fastai (15 ép.)**	**Keras (100 ép.)**	**fastai (40 ép.)**
VGG16	00 : 33 : 15	00 : 06 : 48	01 : 07 : 37	00 : 23 : 49
VGG19	00 : 32 : 46	00 : 07 : 35	01 : 08 : 13	00 : 26 : 45
ResNet50	00 : 34 : 26	00 : 05 : 19	01 : 05 : 41	00 : 16 : 19
ResNet101	00 : 33 : 34	00 : 06 : 58	01 : 08 : 03	00 : 23 : 17
DenseNet121	00 : 33 : 33	00 : 05 : 55	01 : 07 : 28	00 : 18 : 20
DenseNet169	00 : 33 : 59	00 : 06 : 42	01 : 08 : 40	00 : 21 : 06

(b) Tiempos de ejecución en hh:mm:ss.

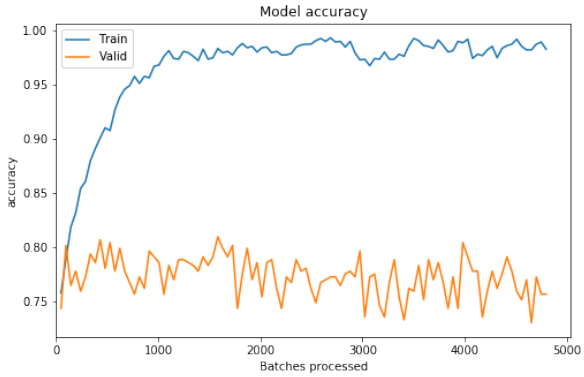
Tabla 4.2: Resultados de los entrenamientos con Skin Cancer ISIC.

Keras debido al uso de *one cycle*. Aunque los tiempos de ejecución son mucho mayores en Keras, esto se debe a que el número de épocas usado en esta librería es también considerablemente mayor, mucho más que las realmente necesarias para entrenar los modelos. En los entrenamientos no se ha observado un sobreajuste claro, pero se aprecian oscilaciones de la exactitud durante el ajuste fino de los pesos. Al no llegar a conseguirse tampoco unos valores de exactitud realmente altos, todo ello lleva a pensar que la cantidad y la variedad de las imágenes no han sido suficientes para que los modelos generalicen el problema de separar inequívocamente las 7 clases de cáncer de piel. Las redes DenseNet han conseguido una eficacia mayor que otros modelos, aunque es posible que su elevado número de capas, en comparación con el resto de redes, haya influido también en el resultado.

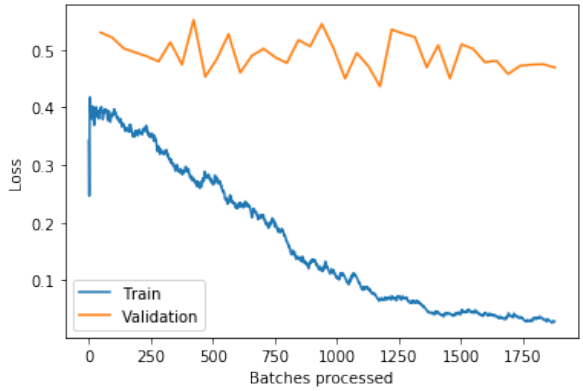
En fastai se han conseguido mejores valores de exactitud que en Keras para todos los experimentos. Esto se debe a que en fastai se han usado *one cycle* y *discriminative learning rates*, lo que ha permitido una exploración más eficiente del espacio de búsqueda y concentrar el ajuste de los modelos en las últimas capas de los mismos.



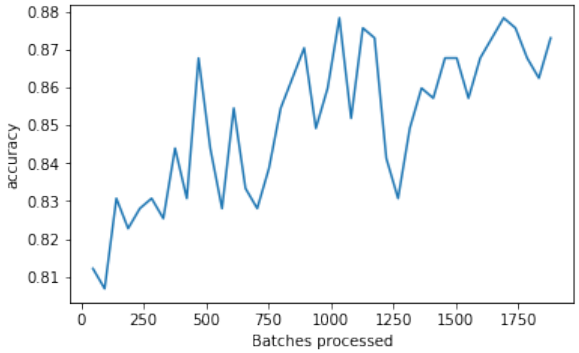
(a) Valores de pérdida en Keras.



(b) Exactitud en Keras.



(c) Valores de pérdida en fastai.



(d) Exactitud en fastai.

Figura 4.2: Gráficas de los valores de pérdida y de exactitud del entrenamiento de un modelo ResNet50 en Keras y fastai. Arriba, las gráficas obtenidas en Keras. Abajo, aquéllas obtenidas en fastai. En Keras se añaden los valores de los pesos a la función de pérdida al usar regularización L2, mientras que esto no ocurre en fastai. Por ello, el comportamiento y la magnitud de los valores de pérdida no son comparables entre ambas librerías.

4.2.4 Discusión de los resultados experimentales

En general, los resultados han sido mejores en fastai que en Keras. Esto se debe a que esta primera librería ofrece funcionalidad específica orientada a resolver problemas de clasificación multi-clase para visión artificial y, además, mecanismos de entrenamiento más avanzados que Keras. Aun así, los resultados de Keras no están tan alejados de los de fastai y es probable que implementando y utilizando dichos mecanismos avanzados se puedan mejorar sus resultados.

Hemos observado que la librería fastai es más eficiente y consigue que los entrenamientos realizados con ella converjan más rápido. La funcionalidad diseñada para la resolución de problemas específicos, además de permitir la obtención de unos resultados notablemente eficaces y eficientes, hace que requiera menos esfuerzo de programación para aquellas áreas tratadas por el equipo de desarrollo de esta librería.

Keras, en cambio, ofrece una funcionalidad más básica que fastai. Muchas de las partes de los procesos de carga de los datos, configuración del modelo, entrenamiento e interpretación de los resultados tienen que ser programadas explícitamente por el usuario. Esta codificación manual implica un mayor número de puntos posibles donde cometer errores, pero también hace que el experimentador sea más consciente de los algoritmos y técnicas que está usando. Otro efecto de la programación manual que implica el uso de Keras es que la responsabilidad de la eficacia y la eficiencia de los modelos y los entrenamientos recae en mayor medida en el propio usuario, en comparación con fastai.

La principal limitación de los experimentos que hemos realizado es que solamente hemos usado un conjunto de imágenes, por lo que es arriesgado generalizar los resultados obtenidos. Sin embargo, parece razonable suponer que, usando cualquier otro conjunto de datos, obtendríamos unos resultados similares para cualquier problema de clasificación de imágenes. En cambio, no podemos afirmar que la eficacia y eficiencia relativas entre Keras y fastai sigan los mismos patrones para otro tipo de problemas de visión artificial, como detección de objetos o segmentación de imágenes, debido a que no hemos realizado experimentos al respecto. Por el mismo motivo, tampoco podemos generalizar los resultados para problemas con datos secuenciales o de PLN.

No parece probable que la elección de hardware o sistema operativo afecte a los valores de pérdida y la exactitud (mostrados en la tabla 4.2a y en el apéndice A), es decir, deberían mostrar unos valores similares dentro de las fluctuaciones estadísticas. Sin embargo, el cambio de hardware o sistema operativo sí que podría implicar un impacto significativo en los tiempos de ejecución de los entrenamientos (mostrados en la tabla 4.2b), pero es de esperar que la proporción entre los tiempos de Keras y fastai sea similar.

Capítulo 5

Conclusiones y trabajo futuro

5.1 Principales aportaciones y conclusiones

Una de las aportaciones de este trabajo es el análisis de los 7 MOOCs de aprendizaje profundo (sección 3.1 y tabla 3.1). Estos cursos suelen requerir una serie de conocimientos previos, generalmente, ciertas habilidades de programación y nociones sobre álgebra lineal básica. Aunque varíe la metodología docente que utilicen, suelen impartir algunas materias comunes, como los fundamentos de las RNAs, ConvNets y RNNs aplicadas a problemas de PLN. Pero también es cierto que hay MOOCs que imparten algunas materias menos comunes dentro de este tipo de cursos, como la gestión de un proyecto de aprendizaje profundo. Respecto a la parte práctica, se suelen usar alternativamente las librerías TensorFlow o PyTorch y, adicionalmente, alguna otra de alto nivel, generalmente Keras. La duración de los cursos es muy variable, estando habitualmente dentro de un rango de 4 semanas y 4 meses, aunque la duración real dependerá de los conocimientos previos del alumno y del tiempo de estudio que dedique cada día. Muchos de los MOOCs ofrecen algún tipo de certificación, aunque ello suele requerir el pago de alguna determinada tarifa.

La principal aportación de este trabajo es el análisis detallado de dos de los cursos que, en nuestra opinión, resultan más relevantes hoy en día, por las razones expuestas en la sección 3.2: el *Programa Especializado Aprendizaje Profundo* de Coursera porque es uno de los más populares y recomendados, y *fastai course v3* debido a que la librería desarrollada por el equipo docente es conocida por su eficacia y por mantenerse al día con los algoritmos más actuales. Nuestro objetivo ha sido analizar en profundidad la metodología, los contenidos, las prácticas y la evaluación de estos dos cursos. Hemos observado que difieren en múltiples aspectos, pero que se complementan entre sí. Aunque sus metodologías docentes son radicalmente opuestas, ambos cursos ofrecen aproximaciones que cubren los puntos más débiles del otro, permitiendo que el alumno que realice los dos MOOCs tenga una buena base, tanto teórica como práctica, del aprendizaje profundo. Los dos cursos comparten parte de los contenidos, especialmente aquellos relacionados con los fundamentos del aprendizaje profundo, pero cada uno de ellos se especializa en explicar un conjunto de algoritmos, técnicas y problemas exclusivos. El *Programa Especializado Aprendizaje Profundo* es el único de estos dos MOOCs que proporciona un procedimiento de evaluación y acreditación, mientras que *fastai course v3* no tiene ejercicios evaluables, sino que se centra en fomentar la experimentación y la colaboración entre los estudiantes. La elección de cursar un MOOC u otro dependerá de las preferencias e inquietudes de la persona interesada, pero quien opte por realizar los dos, tendrá un formación más amplia

debido a las diferencias complementarias de ambos cursos. Aparte del análisis de estos dos cursos, hemos ofrecido una serie de recomendaciones para ayudar a la persona interesada a elegir uno de los cursos (o ambos), algunas orientaciones para el estudio de las diferentes lecciones y bibliografía adicional.

Por último, hemos comparado empíricamente las librerías de aprendizaje profundo Keras y fastai, estudiadas en cada uno de los cursos, mediante un problema de clasificación de imágenes. Hemos observado que, en general, fastai ha tenido mejores resultados que Keras (tabla 4.2), tanto en cuestión de eficacia (precisión de los resultados) como de eficiencia (tiempo de convergencia). Fastai cuenta con dos principales ventajas: tiene implementados algunos mecanismos de entrenamiento avanzados y ofrece un conjunto de funcionalidad para resolver ciertos problemas concretos, entre los que se encuentra la clasificación de imágenes. Atribuimos a estos dos factores el éxito de fastai sobre Keras en las pruebas realizadas. Por otro lado, Keras requiere mucho más trabajo de programación, aunque es presumible que implementado los mismos mecanismos que tiene fastai, se puedan conseguir unos resultados similares. La principal limitación de los experimentos es que han sido realizados con un único conjunto de imágenes, por lo que es arriesgado generalizar los resultados obtenidos. Sin embargo, es razonable suponer que obtendríamos unos valores similares de eficacia y eficiencia relativos entre ambas librerías para cualquier otro conjunto de datos para clasificación de imágenes y para un hardware diferente. En cambio, no podemos realizar la misma afirmación para otro tipo de problemas de visión artificial, como la detección de objetos y la segmentación de imágenes, ni para otras áreas de aplicación, como el PLN, porque no hemos hecho pruebas para respaldarla.

Esperamos que el esfuerzo invertido en este trabajo pueda servir como guía para aquellas personas que en un futuro deseen introducirse en el campo del aprendizaje profundo.

5.2 Trabajo futuro

El alcance de este trabajo se ha limitado a analizar 7 MOOCs de aprendizaje profundo, estudiar en profundidad dos de ellos y comparar empíricamente dos librerías estudiadas en los cursos anteriores, por lo que una línea de trabajo futuro podría consistir en ampliar el estudio incluido en esta memoria, tanto sobre los MOOCs como sobre las librerías.

Respecto a los MOOCs de aprendizaje profundo, se podrían actualizar los contenidos de los dos cursos analizados en detalle en función de los cambios que puedan sufrir con el paso del tiempo. En este trabajo se han indicado 5 MOOCs adicionales, pero existe una oferta mucho más amplia. Se podrían cursar, detallar los contenidos y comparar estos 5 cursos y, también, realizar el mismo trabajo para otros MOOCs no contemplados en esta memoria o que puedan aparecer en un futuro.

Por otro lado, el análisis experimental llevado a cabo ha tenido un alcance limitado y es posible ampliarlo de diferentes formas. Sería conveniente realizar más experimentos entre las librerías Keras y fastai usando más conjuntos de imágenes, con el objetivo de ratificar o rectificar las conclusiones presentadas en este trabajo. Además, se podría

estudiar otro tipo de arquitecturas y problemas de visión artificial, como la detección de objetos o la segmentación de imágenes. También sería conveniente extender la comparación a otras áreas, como el PLN o los datos tabulados. Los experimentos realizados están centrados únicamente en dos librerías. Podría ser interesante ampliar el estudio a otras librerías, marcos de trabajo y lenguajes de programación para tener una visión más general de la oferta de funcionalidad y de rendimiento existente. Sin embargo, dado que actualmente Keras únicamente da soporte a TensorFlow, no parece conveniente realizar en un futuro experimentos con los otros *backends* con los que ha sido históricamente compatible (Theano, CNTK, MXNet, etc.).

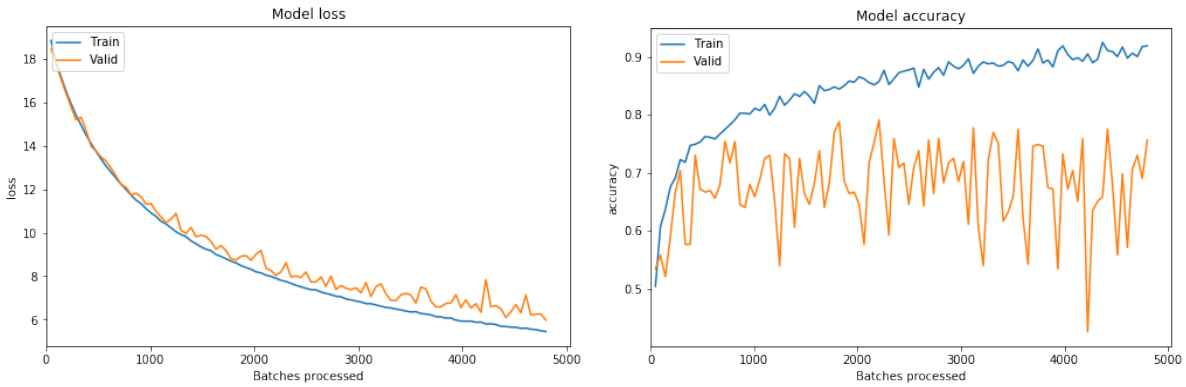
Apéndices

Apéndice A

Gráficas de entrenamiento con Skin Cancer ISIC

En este apéndice se muestran las gráficas de pérdida y de exactitud obtenidas durante el ajuste fino de los modelos (fase 2). Las métricas en fastai únicamente se aplican al conjunto de validación. Por ello, en las gráficas de exactitud, sólo se muestran los valores referentes a dicho conjunto. En Keras, al contrario que en fastai, se añaden los valores de los pesos a la función de pérdida al usar la regularización L2. Por esta causa, el comportamiento y la magnitud de los valores de pérdida no son comparables entre ambas librerías.

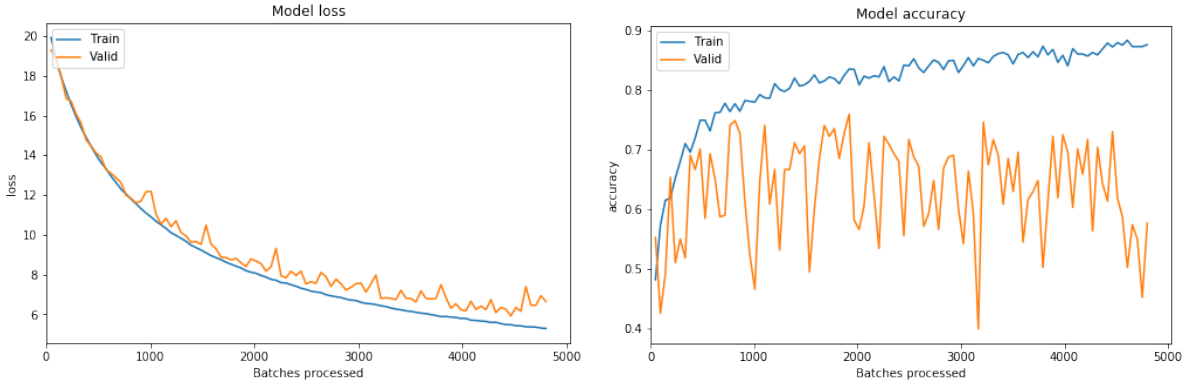
A.1 Keras



(a) Valores de pérdida.

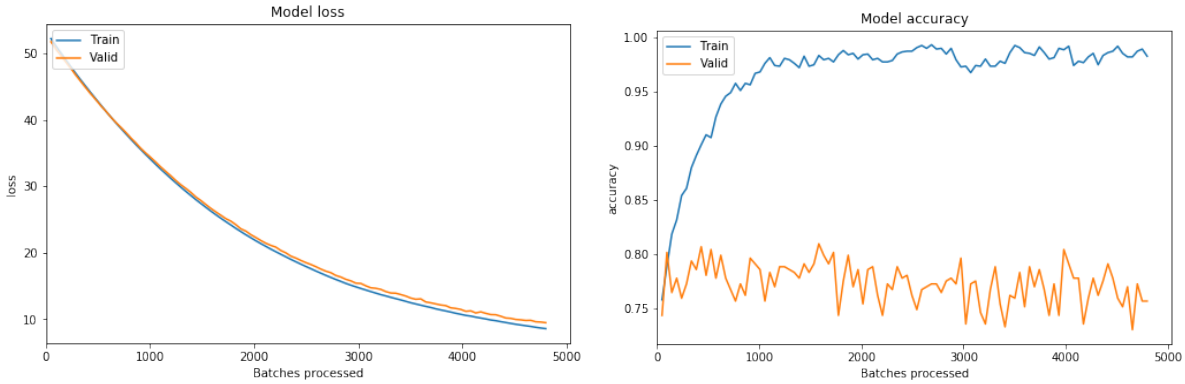
(b) Exactitud.

Figura A.1: Valores de pérdida y exactitud para la red VGG16 en Keras.



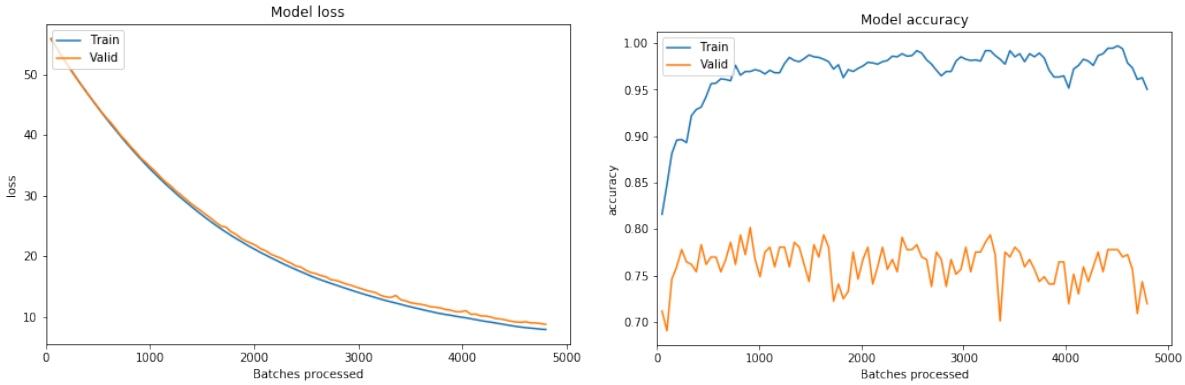
(a) Valores de pérdida. (b) Exactitud.

Figura A.2: Valores de pérdida y exactitud para la red VGG19 en Keras.



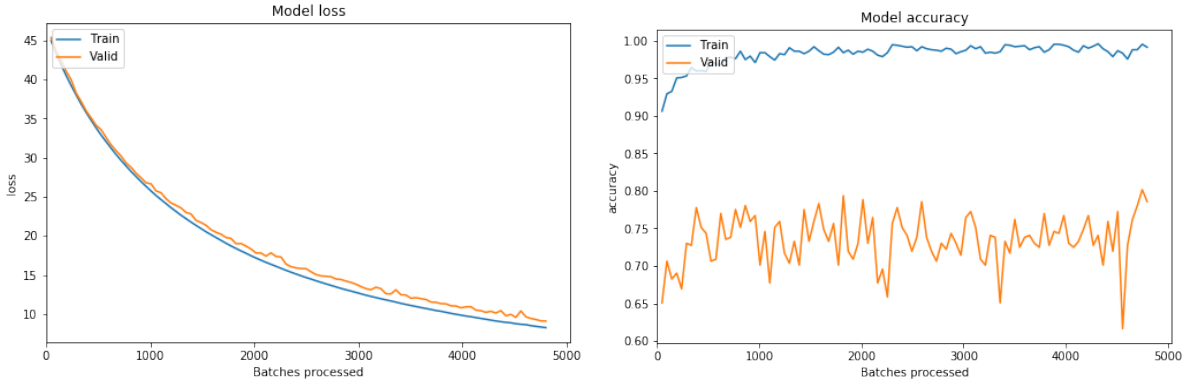
(a) Valores de pérdida. (b) Exactitud.

Figura A.3: Valores de pérdida y exactitud para la red ResNet50 en Keras.



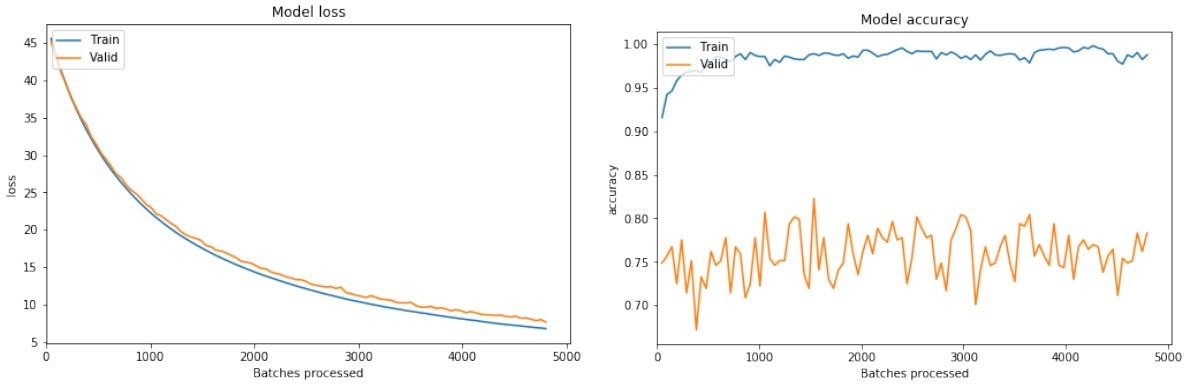
(a) Valores de pérdida. (b) Exactitud.

Figura A.4: Valores de pérdida y exactitud para la red ResNet101 en Keras.



(a) Valores de pérdida. (b) Exactitud.

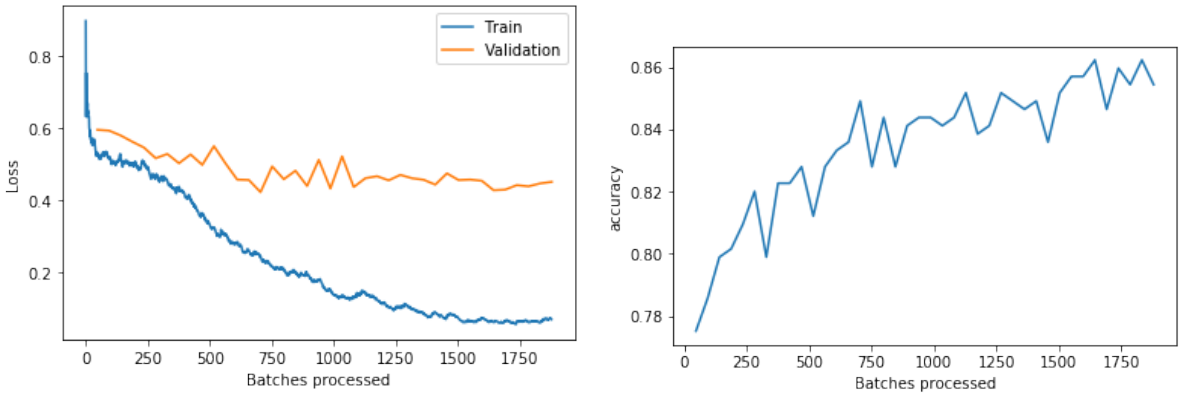
Figura A.5: Valores de pérdida y exactitud para la red DenseNet121 en Keras.



(a) Valores de pérdida. (b) Exactitud.

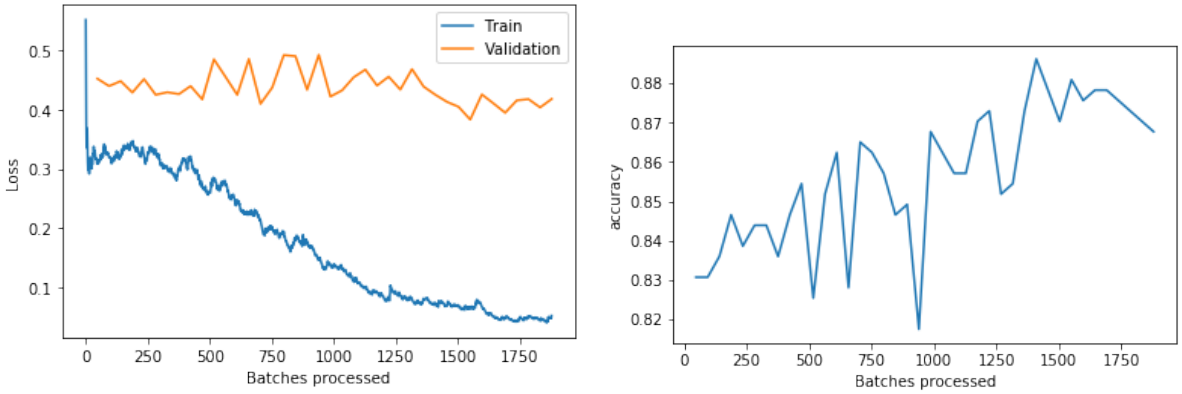
Figura A.6: Valores de pérdida y exactitud para la red DenseNet169 en Keras.

A.2 fastai



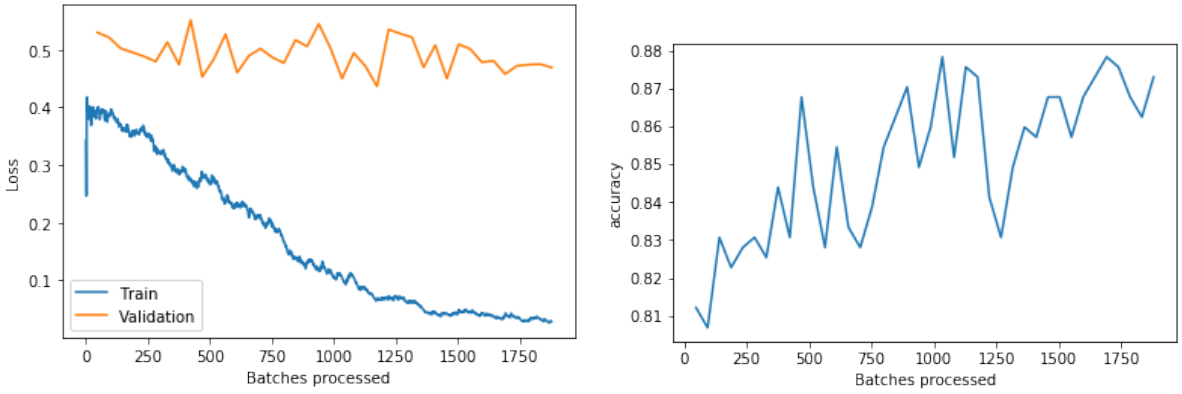
(a) Valores de pérdida. (b) Exactitud.

Figura A.7: Valores de pérdida y exactitud para la red VGG16 en fastai.



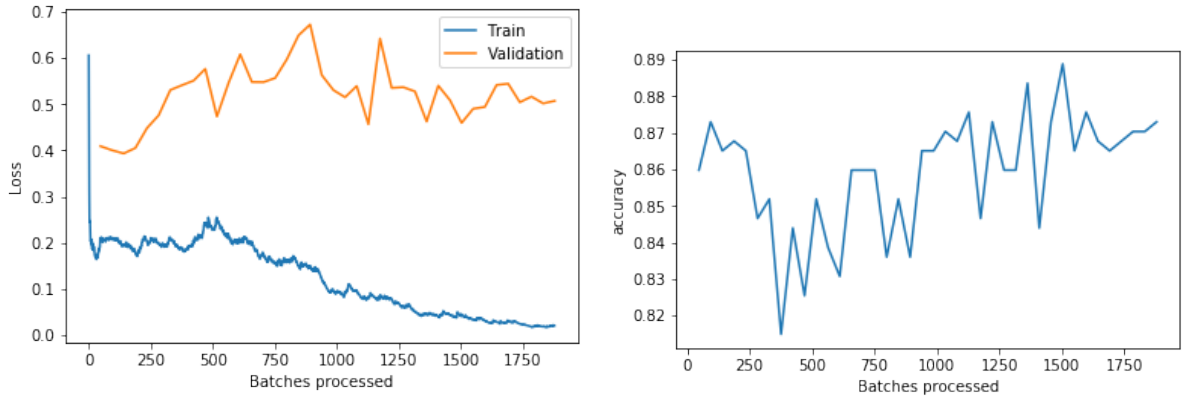
(a) Valores de pérdida. (b) Exactitud.

Figura A.8: Valores de pérdida y exactitud para la red VGG19 en fastai.



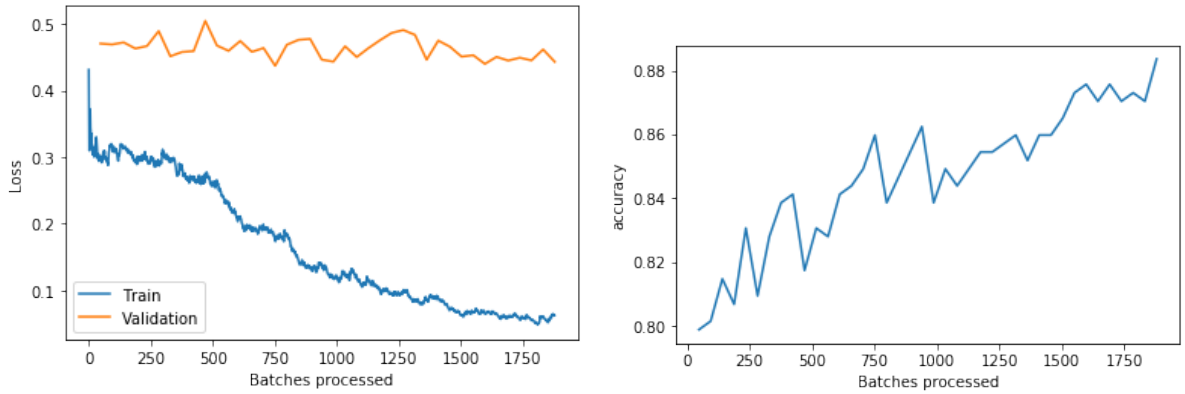
(a) Valores de pérdida. (b) Exactitud.

Figura A.9: Valores de pérdida y exactitud para la red ResNet50 en fastai.



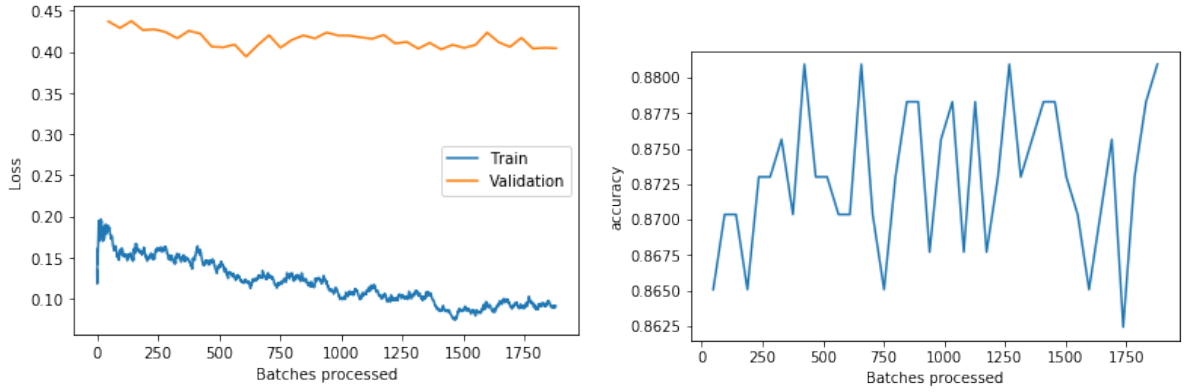
(a) Valores de pérdida. (b) Exactitud.

Figura A.10: Valores de pérdida y exactitud para la red ResNet101 en fastai.



(a) Valores de pérdida. (b) Exactitud.

Figura A.11: Valores de pérdida y exactitud para la red DenseNet121 en fastai.



(a) Valores de pérdida. (b) Exactitud.

Figura A.12: Valores de pérdida y exactitud para la red DenseNet169 en fastai.

Apéndice B

Traducción de la terminología

A

ajuste fino fine tuning.
análisis de preferencia-varianza bias-variance analysis.
aprendizaje automático machine learning.
aprendizaje hebbiano Hebbian learning.
aprendizaje multi-tarea multi-task learning.
aprendizaje por transferencia transfer learning.
aprendizaje profundo deep learning.
aprendizaje profundo extremo-a-extremo end-to-end deep learning.
autenticación facial face verification.

C

cabeza head.
capa layer.
capa completamente conexa fully connected layer, dense layer.
capa convolucional convolutional layer.
capa de entrada input layer.
capa de salida output layer.
capa oculta hidden layer.
ciencia de datos data science.
clasificación con localización classification with localization.
clasificación conexionista temporal Connectionist Temporal Classification, CTC.
clasificación de imágenes image classification.
clasificación de sentimientos sentiment classification.
clasificación multi-clase multi-class classification.
clasificación multi-etiqueta multi-label classification.
codificador encoder.
compromiso entre preferencia y varianza bias-variance tradeoff.
congelar freeze.
conjunto de datos dataset.
conjunto de desarrollo development set, dev set.
conjunto de entrenamiento training set, train set.
conjunto de prueba test set.
conjunto de validación validation set, valid set.
correlación cruzada cross-correlation.
cuerpo body.

D

datos estructurados structured data.
datos no estructurados unstructured data.
datos tabulados tabular data.
declive de los pesos weight decay.
decodificador decoder.
dependecias a largo plazo long-term dependencies.
desarrollo iterativo e incremental iterative and incremental development.
desarrollo ágil agile development, agile software development.
descenso del gradiente gradient descent, GD.
descenso del gradiente con momento gradient descent with momentum.
descenso del gradiente estocástico stochastic gradient descent, SGD.
descenso del gradiente por mini-lotes mini-batch gradient descent.
descongelar unfreeze.
desplazamiento de las covariables covariate-shift.
detección de objetos object detection.
detección de palabras de activación trigger word detection.
diferenciación automática automatic differenciation.
dilatación dilation.

E

eliminación de enlaces dropout.
entropía cruzada cross entropy.
entropía cruzada binaria binary cross entropy.
época epoch.

error de entrenamiento training loss, training error.
error de validación validation loss, validation error.
exactitud accuracy.
exhaustividad recall.

F

filtrado colaborativo collaborative filtering, CF.
filtro filter, kernel.
función de activación activation function.
función de coste cost function.
función de pérdida loss function.
función lineal linear function.

G

generación automática de pies de foto image captioning.
grafo de computación computation graph.

I

inicialización de Kaiming Kaiming (He) initialization.
inicialización de Xavier Xavier initialization.

L

lote batch.

M

mapa autoorganizado self-organizing map.
mapa de características feature map.
modelado del lenguaje language modeling.
modelo de lenguaje language model.
modelo de secuencia-a-secuencia sequence-to-sequence model, sequence-to-sequence architecture.
modelo generativo generative model.
muestra example, sample.
máquinas de vectores soporte support vector machines.
métrica F1 F1 score.

N

normalización por lotes batch normalization, BatchNorm.

O

optimizador optimizer.
ortogonalización orthogonalization.

P

parámetros parameters.
paso hacia atrás backward pass.
paso hacia delante forward pass.
perceptrón multicapa multilayer perceptron.
planificador de hiperparámetros hyperparameter scheduler, annealing.
precisión precision.
preferencia elevada high bias.
preprocesamiento preprocessing.
problema de preferencia-varianza bias-variance problem.
problema del desvanecimiento y la explosión de los gradientes vanishing/exploding gradient problem.
procesamiento de múltiples etapas multiple stages processing.
procesamiento del lenguaje natural natural language processing, NLP.
promedio móvil ponderado exponencialmente exponentially weighted moving average, EWMA.

R

razonamiento analógico analogical reasoning.
raíz stem.
reconocimiento del habla speech recognition.
reconocimiento facial face recognition.
red generativa antagónica generative adversarial network, GAN.
red neuronal convolucional convolutional neural network, ConvNet, CNN.
red neuronal profunda deep neural network.
red neuronal recurrente recurrent neural network, RNN.
regresión de imágenes image regression.
regresión lineal linear regression.
regresión logística logistic regression.
relleno padding.
rendimiento a nivel humano human-level performance.
restauración de imágenes image restoration.
retropropagación del error backpropagation.

S

segmentación de imágenes image segmentation.
sesgo bias.
sistema de recomendación recommender system.
sobreajuste overfitting.
sub-ajuste underfitting.

T

tasa de aprendizaje learning rate.
teorema de aproximación universal universal approximation theorem.
traducción automática machine translation.

U

unidad lineal rectificada Rectified Linear Unit, ReLU.

V

variable categórica categorical variable.
varianza elevada high variance.
verosimilitud logarítmica negativa negative log likelihood.
visión artificial computer vision.

Z

zancada stride.

Bibliografía

Abadi, M. et al. (2016). "TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems". En: *arXiv.org* abs/1603.04467. https://arxiv.org/abs/1603.04467.

Ananthram, A. (2018). "Deep Learning For Beginners Using Transfer Learning in Keras". En: *Towards Data Science*. https://towardsdatascience.com/keras-transfer-learning-for-beginners-6c9b8b7143e.

Apple Inc. (2019). *The Swift Programming Language*. Apple Inc. Online: https://docs.swift.org/swift-book/.

Ba, J. L., Kiros, J. R. e Hinton, G. E. (2016). "Layer Normalization". En: *arXiv.org* abs/1607.06450. https://arxiv.org/abs/1607.06450.

Bahdanau, D., Cho, K. y Bengio, Y. (2014). "Neural Machine Translation by Jointly Learning to Align and Translate". En: *arXiv.org* abs/1409.0473. https://arxiv.org/abs/1409.0473.

Barnes, R. J. (2006). *Matrix Differentiation (and some other stuff)*. Inf. téc. Minneapolis, Minnesota, USA: Department of Civil Engineering, Univiersity of Minnesota. https://atmos.washington.edu/~dennis/MatrixCalculus.pdf.

Bengio, Y., Ducharme, R., Vincent, P. y Jauvin, C. (2003). "A Neural Probabilistic Language Model". En: *Journal of Machine Learning Research* 3, págs. 1137-1155. http://www.jmlr.org/papers/volume3/bengio03a/bengio03a.pdf.

Bolukbasi, T., Chang, K., Zou, J., Saligrama, V. y Kalai, A. (2016). "Man is to Computer Programmer as Woman is to Homemaker? Debiasing Word Embeddings". En: *arXiv.org* abs/1607.06520. https://arxiv.org/abs/1607.06520.

Brostow, G. J., Shotton, J., Fauqueur, J. y Cipolla, R. (2008). "Segmentation and Recognition Using Structure from Motion Point Clouds". En: *ECCV (1)*, págs. 44-57. https://www.semanticscholar.org/paper/Segmentation-and-Recognition-Using-Structure-from-Brostow-Shotton/08f624f7ee5c3b05b1b604357fb1532241e208db#paper-header - CamVid: http://mi.eng.cam.ac.uk/research/projects/VideoRec/CamVid/.

Brownlee, J. (2019). "How to Configure Image Data Augmentation in Keras". En: *Machine Learning Mastery*. https://machinelearningmastery.com/how-to-configure-image-data-augmentation-when-training-deep-learning-neural-networks/.

Cave, S. y Dihal, K. (2018). "Ancient dreams of intelligent machines: 3,000 years of robots". En: *Nature* 559.7715. https://www.nature.com/articles/d41586-018-05773-y.

Chellapilla, K., Puri, S. y Simard, P. (2006). "High Performance Convolutional Neural Networks for Document Processing". En: *Tenth International Workshop on Frontiers in Handwriting Recognition*. Université de Rennes 1, La Baule (France). https://hal.inria.fr/inria-00112631/document.

Chen, G. (2016). "A Gentle Tutorial of Recurrent Neural Network with Error Backpropagation". En: *arXiv.org* abs/1610.02583. https://arxiv.org/abs/1610.02583.

Cho, K. et al. (2014a). "Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation". En: *arXiv.org* abs/1406.1078. https://arxiv.org/abs/1406.1078.

Cho, K., van Merrienboer, B., Bahdanau, D. y Bengio, Y. (2014b). "On the Properties of Neural Machine Translation: Encoder-Decoder Approaches". En: *arXiv.org* abs/1409.1259. https://arxiv.org/abs/1409.1259.

Chollet, F. (2016). "Xception: Deep Learning with Depthwise Separable Convolutions". En: *arXiv.org* abs/1610.02357. https://arxiv.org/abs/1610.02357.

— (2018). *Deep Learning with Python*. Manning.

Chung, J., Gulcehre, C., Cho, K. y Bengio, Y. (2014). "Empirical Evaluation of Gated Recurrent Neural Networks on Sequence Modeling". En: *arXiv.org* abs/1412.3555. https://arxiv.org/abs/1412.3555.

Cireşan, D. C., Meier, U., Masci, J., Gambardella, L. M. y Schmidhuber, J. (2011a). "Flexible, High Performance Convolutional Neural Networks for Image Classification". En: *Proceedings of the Twenty-Second International Joint Conference on Artificial Intelligence (IJCAI'11)*. Vol. 2, págs. 1237-1242. http://people.idsia.ch/~juergen/ijcai2011.pdf.

Cireşan, D. C., Meier, U., Masci, J. y Schmidhuber, J. (2011b). "A Committee of Neural Networks for Traffic Sign Classification". En: *Proceedings of the Twenty-Second International Joint Conference on Artificial Intelligence (IJCAI'11)*. http://people.idsia.ch/~ciresan/data/ijcnn2011.pdf.

Collobert, R., Bengio, S. y Mariéthoz, J. (2002). "Torch: a modular machine learning software library". En: https://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.8.9850, http://torch.ch.

Cortes, C. y Vapnik, V. (1995). "Support-vector networks". En: *Machine Learning* 20.3, págs. 273-297.

Deng, J., Dong, W., Socher, R., Li, L.-J., Li, K. y Fei-Fei, L. (2009). "ImageNet: A Large-Scale Hierarchical Image Database". En: *2009 IEEE Conference on Computer Vision and Pattern Recognition (CVPR09)*. http://www.image-net.org/papers/imagenet_cvpr09.pdf - ImageNet: http://www.image-net.org/.

Downey, A. B. (2015). *Think Python: How to Think Like a Computer Scientist*. 2.ª ed. O'Reilly Media. https://greenteapress.com/wp/think-python-2e/.

Fanelli, G., Dantone, M., Gall, J., Fossati, A. y Van Gool, L. (2013). "Random Forests for Real Time 3D Face Analysis". En: *Int. J. Comput. Vision* 101.3, págs. 437-458. https://data.vision.ee.ethz.ch/cvl/gfanelli/pubs/ijcv.pdf - Biwi Kinect Head Pose Database: https://data.vision.ee.ethz.ch/cvl/gfanelli/head_pose/head_forest.html#db.

Fortmann-Roe, S. (2012). "Understanding the Bias-Variance Tradeoff". En: *scott.fortmann-roe.com*. http://scott.fortmann-roe.com/docs/BiasVariance.html.

Fukushima, K. (1979). "Neural network model for a mechanism of pattern recognition unaffected by shift in position - Neocognitron". En: *IECE* J62-A.10, págs. 658-665.

— (1980). "Neocognitron: A self-organizing neural network for a mechanism of pattern recognition unaffected by shift in position". En: *Biological Cybernetics* 36.4, págs. 193-202. https://www.rctn.org/bruno/public/papers/Fukushima1980.pdf.

Fung, V. (2017). "An Overview of ResNet and its Variants". En: *Towards Data Science*. https://towardsdatascience.com/an-overview-of-resnet-and-its-variants-5281e2f56035.

Gatys, L. A., Ecker, A. S. y Bethge, M. (2015). "A Neural Algorithm of Artistic Style". En: *arXiv.org* abs/1508.06576. https://arxiv.org/abs/1508.06576.

Girshick, R. (2015). "Fast R-CNN". En: *arXiv.org* abs/1504.08083. https://arxiv.org/abs/1504.08083.

Girshick, R., Donahue, J., Darrell, T. y Malik, J. (2013). "Rich feature hierarchies for accurate object detection and semantic segmentation". En: *arXiv.org* abs/1311.2524. https://arxiv.org/abs/1311.2524.

Glorot, X. y Bengio, Y. (2010). "Understanding the difficulty of training deep feedforward neural networks". En: *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*. Ed. por Y. W. Teh y M. Titterington. Vol. 9. Proceedings of Machine Learning Research. Chia Laguna Resort, Sardinia, Italy: PMLR, págs. 249-256. http://proceedings.mlr.press/v9/glorot10a.html.

Graves, A., Fernández, S., Gomez, F. y Schmidhuber, J. (2006). "Connectionist Temporal Classification: Labelling Unsegmented Sequence Data with Recurrent Neural Networks". En: *ICML '06 Proceedings of the 23rd International Conference on Machine Learning*, págs. 369-376. https://www.cs.toronto.edu/~graves/icml_2006.pdf.

GroupLens (2019). "MovieLens". En: *grouplens.org*. https://grouplens.org/datasets/movielens/.

Guo, C. y Berkhahn, F. (2016). "Entity Embeddings of Categorical Variables". En: *arXiv.org* abs/1604.06737. https://arxiv.org/abs/1604.06737.

He, K., Zhang, X., Ren, S. y Sun, J. (2015a). "Deep Residual Learning for Image Recognition". En: *arXiv.org* abs/1512.03385. https://arxiv.org/abs/1512.03385.

— (2015b). "Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification". En: *arXiv.org* abs/1502.01852. https://arxiv.org/abs/1502.01852.

He, T., Zhang, Z., Zhang, H., Zhang, Z., Xie, J. y Li, M. (2018). "Bag of Tricks for Image Classification with Convolutional Neural Networks". En: *arXiv.org* abs/1812.01187. http://arxiv.org/abs/1812.01187.

He, X., Liao, L., Zhang, H., Nie, K., Hu, X. y Chua, T. (2017). "Neural Collaborative Filtering". En: *arXiv.org* abs/1708.05031. https://arxiv.org/abs/1708.05031.

Hebb, D. O. (1949). *The Organization of Behavior: A Neuropsycological Theory*. New York: John Wiley & Sons, Inc.

Ho, T. K. (1995). "Random decision forest". En: *Proceedings of 3rd International Conference on Document Analysis and Recognition*, págs. 278-282.

Hochreiter, S. (1991). "Untersuchungen zu dynamischen neuronalen Netzen". Diploma thesis. Institut für Informatik, Lehrstuhl Prof. Brauer, Technische Universität München. Advisor: J. Schmidhuber.

Hochreiter, S. y Schmidhuber, J. (1997). "Long Short-Term Memory". En: *Neural Computation* 9, págs. 1735-1780. http://www.bioinf.jku.at/publications/older/2604.pdf.

Hoffer, E., Banner, R., Golan, I. y Soudry, D. (2018). "Norm matters: efficient and accurate normalization schemes in deep networks". En: *arXiv.org* abs/1803.01814. http://arxiv.org/abs/1803.01814.

Hopfield, J. J. (1982). "Neural networks and physical systems with emergent collective compuational abilities". En: *Proceedings of the National Academy of Sciences*. Vol. 79, págs. 2554-2558. https://doi.org/10.1073/pnas.79.8.2554.

Howard, A. G. et al. (2017). "MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications". En: *arXiv.org* abs/1704.04861. https://arxiv.org/abs/1704.04861.

Howard, J. y Gugger, S. (2020). "fastai: A Layered API for Deep Learning". En: *arXiv.org* abs/2002.04688. https://arxiv.org/abs/2002.04688, https://www.fast.ai/, https://docs.fast.ai/.

Howard, J. y Ruder, S. (2018). "Universal Language Model Fine-tuning for Text Classification". En: *arXiv.org* abs/1801.06146. https://arxiv.org/abs/1801.06146.

Howard, J. et al. (2019). "Imagenette". En: *GitHub*. https://github.com/fastai/imagenette.

Huang, G., Liu, Z., van der Maaten, L. y Weinberger, K. Q. (2016). "Densely Connected Convolutional Networks". En: *arXiv.org* abs/1608.06993. https://arxiv.org/abs/1608.06993.

Iandola, F. N., Han, S., Moskewicz, M. W., Ashraf, K., Dally, W. J. y Keutzer, K. (2016). "SqueezeNet: AlexNet-level accuracy with 50x fewer parameters and <0.5MB model size". En: *arXiv.org* abs/1602.07360. https://arxiv.org/abs/1602.07360.

Ingargiola, A. (2019). "Deep-dive into Convolutional Networks". En: *Towards Data Science*. https://towardsdatascience.com/deep-dive-into-convolutional-networks-48db75969fdf.

Ioffe, S. y Szegedy, C. (2015). "Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift". En: *arXiv.org* abs/1502.03167. https://arxiv.org/abs/1502.03167.

Ivakhnenko, A. G. (1971). "Polynomial Theory of Complex Systems". En: *IEEE Transactions on Systems, Man and Cybernetics* SMC-1.4, págs. 364-378.

Ivakhnenko, A. G. y Lapa, V. G. (1965). *Cybernetics Predicting Devices*. CCM Information Corporation.

Ivakhnenko, A. G., Lapa, V. G. y McDonough, R. N. (1967). *Cybernetics and forecasting techniques*. American Elsevier.

Izadi, S. (2017). "What Is Covariate Shift?" En: *Medium*. https://medium.com/@izadi/what-is-covariate-shift-d7a7af541e6.

Jain, S. (2018). "An Overview of Regularization Techniques in Deep Learning (with Python code)". En: *Analytics Vidhya*. https://www.analyticsvidhya.com/blog/2018/04/fundamentals-deep-learning-regularization-techniques/.

Jia, Y. et al. (2014). "Caffe: Convolutional Architecture for Fast Feature Embedding". En: *arXiv.org* abs/1408.5093. https://arxiv.org/abs/1408.5093, https://caffe.berkeleyvision.org.

Johnson, J. (2015). "neural-style". En: *GitHub*. https://github.com/jcjohnson/neural-style.

Johnson, J., Alahi, A. y Li, F. (2016). "Perceptual Losses for Real-Time Style Transfer and Super-Resolution". En: *arXiv.org* abs/1603.08155. https://arxiv.org/abs/1603.08155 - código: https://github.com/jcjohnson/fast-neural-style.

Karpathy, A. (2015). "The Unreasonable Effectiveness of Recurrent Neural Networks". En: *GitHub*. http://karpathy.github.io/2015/05/21/rnn-effectiveness/.

Karpathy, A. y Fei-Fei, L. (2014). "Deep Visual-Semantic Alignments for Generating Image Descriptions". En: *arXiv.org* abs/1412.2306. https://arxiv.org/abs/1412.2306.

Karras, T., Laine, S. y Aila, T. (2018). "A Style-Based Generator Architecture for Generative Adversarial Networks". En: *arXiv.org* abs/1812.04948. https://arxiv.org/abs/1812.04948 - código: https://github.com/NVlabs/stylegan.

Katanskiy, A. (2019). "Skin Cancer ISIC". En: *Kaggle*. https://www.kaggle.com/nodoubttome/skin-cancer9-classesisic.

Khan Academy (2017). *Derivative Rules*. Khan Academy. https://www.khanacademy.org/math/old-ap-calculus-ab/ab-derivative-rules.

Kingma, D. P. y Ba, J. (2014). "Adam: A Method for Stochastic Optimization". En: *arXiv.org* abs/1412.6980. http://arxiv.org/abs/1412.6980.

Kohonen, T. (1982). "Self-organized formation of topologically correct feature maps". En: *Biological Cybernetics* 43, págs. 59-69. http://www.cnbc.cmu.edu/~tai/nc19journalclubs/Kohonen1982_Article_Self-organizedFormationOfTopol.pdf.

Kopczyk, D. (2019). "Covariate Shift in Machine Learning". En: *dkopczyk.quantee.co.uk*. https://dkopczyk.quantee.co.uk/covariate_shift/.

Krizhevsky, A., Sutskever, I. e Hinton, G. E. (2012). "ImageNet classification with deep convolutional neural networks". En: *Advances in Neural Information Processing Systems*. http://www.cs.toronto.edu/~hinton/absps/imagenet.pdf.

Laarhoven, T. van (2017). "L2 Regularization versus Batch and Weight Normalization". En: *arXiv.org* abs/1706.05350. http://arxiv.org/abs/1706.05350.

LeCun, Y. et al. (1989). "Back-propagation applied to handwritten zip code recognition". En: *Neural Computation* 1.4, págs. 541-551. http://yann.lecun.com/exdb/publis/pdf/lecun-89e.pdf.

LeCun, Y., Bottou, L., Bengio, Y. y Haffner, P. (1998). "Gradient-based learning applied to document recognition". En: *Proceedings of the IEEE*, págs. 2278-2324. http://vision.stanford.edu/cs598_spring07/papers/Lecun98.pdf.

Lin, M., Chen, Q. y Yan, S. (2013). "Network In Network". En: *arXiv.org* abs/1312.4400. https://arxiv.org/abs/1312.4400.

Lin, T. et al. (2014). "Microsoft COCO: Common Objects in Context". En: *arXiv.org* abs/1405.0312. https://arxiv.org/abs/1405.0312 - COCO Dataset: http://cocodataset.org.

Linnainmaa, S. (1970). "The representation of the cumulative rounding error of an algorithm as a Taylor expansion of the local rounding errors". Tesis de mtría. University of Helsinki.

Little, W. A. (1974). "The existence of persistent states in the brain". En: *Mathematical Biosciences* 19, págs. 101-120.

Maas, A. L., Daly, R. E., Pham, P. T., Huang, D., Ng, A. Y. y Potts, C. (2011). "Learning Word Vectors for Sentiment Analysis". En: *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies*. Portland, Oregon, USA: Association for Computational Linguistics, págs. 142-150. http://ai.stanford.edu/~amaas/papers/wvSent_acl2011.pdf, IMDb Large Movie Review Dataset: http://ai.stanford.edu/~amaas/data/sentiment/.

Mahapatra, A. (2018). "[Beginner] ML Basics: Exponentially weighted moving average". En: *Medium*. https://medium.com/@abhinav.mahapatra10/beginners-ml-basics-exponentially-weighted-moving-average-8ce3e75768f6.

Mallya, A. (2019). "An Illustrated Explanation of the LSTM Forward-Backward Pass". En: *GitHub*. http://arunmallya.github.io/writeups/nn/lstm/index.html#/.

Mao, J., Xu, W., Yang, Y., Wang, J., Huang, Z. y Yuille, A. (2014). "Deep Captioning with Multimodal Recurrent Neural Network (m-RNN)". En: *arXiv.org* abs/1412.6632. https://arxiv.org/abs/1412.6632.

Marcelino, P. (2018). "Trasfer learning from pre-trained models". En: *Towards Data Science*. https://towardsdatascience.com/transfer-learning-from-pre-trained-models-f2393f124751.

McCulloc, W. S. y Pitts, W. H. (1943). "A Logical Calculus of the Ideas Immanent in Nervous Activity". En: *Bulletin of Mathematical Biophysics* 5.4, págs. 115-133. Reedición: https://www.cs.cmu.edu/~./epxing/Class/10715/reading/McCulloch.and.Pitts.pdf.

McDonald, C. (2017). "Machine learning fundamentals (I): Cost functions and gradient descent". En: *Towards Data Science*. https://towardsdatascience.com/machine-learning-fundamentals-via-linear-regression-41a5d11f5220.

Menon, A. (2018). "Linear Regression using Gradient Descent". En: *Towards Data Science*. https://towardsdatascience.com/linear-regression-using-gradient-descent-97a6c8700931.

Merity, S., Xiong, C., Bradbury, J. y Socher, R. (2016). "Pointer Sentinel Mixture Models". En: *arXiv.org* abs/1609.07843. http://arxiv.org/abs/1609.07843 - WikiText Dataset: https://blog.einstein.ai/the-wikitext-long-term-dependency-language-modeling-dataset/.

Merity, S., Keskar, N. S. y Socher, R. (2017). "Regularizing and Optimizing LSTM Language Models". En: *arXiv.org* abs/1708.02182. http://arxiv.org/abs/1708.02182.

Mikolov, T., Chen, K., Corrado, G. y Dean, J. (2013a). "Efficient Estimation of Word Representations in Vector Space". En: *arXiv.org*. https://arxiv.org/abs/1301.3781.

Mikolov, T., Yih, W. y Zweig, G. (2013b). "Linguistic Regularities in Continuous Space Word Representations". En: *Proceedings of the 2013 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, págs. 746-751. https://www.aclweb.org/anthology/N13-1090.

Mikolov, T., Sutskever, I., Chen, K., Corrado, G. y Dean, J. (2013c). "Distributed Representations of Words and Phrases and their Compositionality". En: *arXiv.org* abs/1310.4546. https://arxiv.org/abs/1310.4546.

Mishkin, D. y Matas, J. (2016). "All you need is a good init". En: *4th International Conference on Learning Representations, ICLR 2016, San Juan, Puerto Rico, May 2-4, 2016, Conference Track Proceedings*. https://arxiv.org/abs/1511.06422.

Mishra, D. (2019). "Categorical Embedding and Transfer Learning". En: *Towards Data Science*. https://towardsdatascience.com/categorical-embedding-and-transfer-learning-dd3c4af6345d.

Moran, M. E. (2006). "The da Vinci Robot". En: *Journal of Endourology* 20.12. https://doi.org/10.1089/end.2006.20.986.

Nagy, G. (1991). "Neural Networks – Then and Now". En: *IEEE Transactions on Neural Networks* 2.2, págs. 316-318. https://www.ecse.rpi.edu/~nagy/PDF_chrono/1991_Nagy_NN1991-ThenAndNow.pdf.

Nielsen, M. (2019). *Neural Networks and Deep Learning*. http://neuralnetworksanddeeplearning.com/.

Nilsson, N. J. (2010). *The Quest for Artificial Intelligence: A History of Ideas and Achievements*. Cambridge University Press. versión web: http://ai.stanford.edu/~nilsson/QAI/qai.pdf.

Olah, C. (2015). "Understanding LSTM Networks". En: *GitHub*. https://colah.github.io/posts/2015-08-Understanding-LSTMs/.

Olazaran, M. (1996). "A Sociological Study of the Official History of the Perceptrons Controversy". En: *Social Studies of Science* 26.3, págs. 611-659. https://pdfs.semanticscholar.org/f3b6/e5ef511b471ff508959f660c94036b434277.pdf.

Oliphant, T. E. (2006). *A guide to NumPy*. USA: Trelgol Publishing. https://web.mit.edu/dvp/Public/numpybook.pdf.

Pandey, P. (2019). "Understanding the Mathematics behind Gradient Descent". En: *Towards Data Science*. https://towardsdatascience.com/understanding-the-mathematics-behind-gradient-descent-dde5dc9be06e.

Papineni, K., Roukos, S., Ward, T. y Zhu, W. (2002). "Bleu: A Method for Automatic Evaluation of Machine Translation". En: *Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics (ACL)*, págs. 311-318. https://www.aclweb.org/anthology/P02-1040/.

Parkhi, O. M., Vedaldi, A., Zisserman, A. y Jawahar, C. V. (2012). "Cats and Dogs". En: *IEEE Conference on Computer Vision and Pattern Recognition*. The Oxford-IIIT Pet Dataset: http://www.robots.ox.ac.uk/~vgg/data/pets/.

Parr, T. y Howard, J. (2018). "The Matrix Calculus You Need For Deep Learning". En: *arXiv.org* abs/1802.01528. https://arxiv.org/abs/1802.01528.

Paszke, A. et al. (2019). "PyTorch: An Imperative Style, High-Performance Deep Learning Library". En: *Advances in Neural Information Processing Systems 32*. Ed. por H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox y R. Garnett. Curran Associates, Inc., págs. 8024-8035. http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf, https://pytorch.org/.

Pennington, J., Socher, R. y Manning, C. D. (2014). "GloVe: Global Vectors for Word Representation". En: *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, págs. 1532-1543. https://www.aclweb.org/anthology/D14-1162 - código: https://nlp.stanford.edu/projects/glove/.

Perkins, D. N. (2009). *Making Learning Whole: How Seven Principles of Teaching Can Transform Education*. Jossey-Bass. Resumen realizado por Ruth Walker: https://www.gse.harvard.edu/news/uk/09/01/education-bat-seven-principles-educators.

Ping Shung, K. (2018). "Accuracy, Precision, Recall or F1?" En: *Towards Data Science*. https://towardsdatascience.com/accuracy-precision-recall-or-f1-331fb37c5cb9.

Python Software Foundation (2020a). *The Python Standard Library*. Python Software Foundation. https://docs.python.org/3/library/index.html.

— (2020b). *The Python Tutorial*. Python Software Foundation. https://docs.python.org/3/tutorial/index.html.

Redmon, J. y Farhadi, A. (2016). "YOLO9000: Better, Faster, Stronger". En: *arXiv.org* abs/1612.08242. https://arxiv.org/abs/1612.08242.

— (2018). "YOLOv3: An Incremental Improvement". En: *arXiv.org* abs/1804.02767. https://arxiv.org/abs/1804.02767.

Redmon, J., Divvala, S., Girshick, R. y Farhadi, A. (2015). "You Only Look Once: Unified, Real-Time Object Detection". En: *arXiv.org* abs/1506.02640. https://arxiv.org/abs/1506.02640 - Sitio web: https://pjreddie.com/darknet/yolo/ - Repositorio oficial: https://github.com/pjreddie/darknet.

Ren, S., He, K., Girshick, R. y Sun, J. (2016). "Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks". En: *arXiv.org* abs/1506.01497. https://arxiv.org/abs/1506.01497.

Ronaghan, S. (2018). "Deep Learning: Which Loss and Activation Functions should I use?" En: *Towards Data Science*. https://towardsdatascience.com/deep-learning-which-loss-and-activation-functions-should-i-use-ac02f1c56aa8.

Ronneberger, O., Fischer, P. y Brox, T. (2015). "U-Net: Convolutional Networks for Biomedical Image Segmentation". En: *Medical Image Computing and Computer-Assisted Intervention (MICCAI)*. Vol. 9351. LNCS. Springer, págs. 234-241. https://arxiv.org/abs/1505.04597.

Rosenblatt, F. (1958). "The perceptron: A probabilistic model for information storage and organization in the brain". En: *Psychological review* 65.6. http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.335.3398&rep=rep1&type=pdf.

— (1962). *Principles of Neurodynamics: Perceptrons and the Theory of Brain Mechanisms*. Spartan Books.

Rosenbrock, A. (2019). "Keras vs. tf.keras: What's the difference in TensorFlow 2.0". En: *pyimagesearch*. https://www.pyimagesearch.com/2019/10/21/keras-vs-tf-keras-whats-the-difference-in-tensorflow-2-0/.

Ruder, S. (2016). "An overview of gradient descent optimization algorithms". En: *arXiv.org* abs/1609.04747. https://arxiv.org/abs/1609.04747 - versión web: https://ruder.io/optimizing-gradient-descent/.

Rumelhart, D. E., Hinton, G. E. y Williams, R. J. (1986). "Learning internal representations by error propagation". En: *Parallel Distributed Processing*. Ed. por D. E. Rumelhart y J. L. McClelland. Vol. 1. MIT Press, págs. 318-362.

Russakovsky, O. et al. (2015). "ImageNet Large Scale Visual Recognition Challenge". En: *International Journal of Computer Vision (IJCV)* 115.3, págs. 211-252. https://arxiv.org/abs/1409.0575, http://www.image-net.org/challenges/LSVRC/.

Russell, S. y Norvig, P. (2009). *Artificial Intelligence: A Modern Approach*. 3rd. USA: Prentice Hall Press.

Saha, S. (2018). "A Comprehensive Guide to Convolutional Neural Networks - the ELI5 way". En: *Towards Data Science*. https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53.

Samuel, A. L. (1959). "Some Studies in Machile Learning Using the Game of Checkers". En: *IBM Journal* 3.3. Reedición: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.368.2254&rep=rep1&type=pdf.

Sankesara, H. (2019). "UNet: Introducing Symmetry in Segmentation". En: *Towards Data Science*. https://towardsdatascience.com/u-net-b229b32b4a71.

Santurkar, S., Tsipras, D., Ilyas, A. y Madry, A. (2018). "How Does Batch Normalization Help Optimization?" En: *arXiv.org* abs/1805.11604. https://arxiv.org/abs/1805.11604.

Sarkar, D. (2018). "A Comprehensive Hands-on Guide to Transfer Learning with Real-World Applications in Deep Learning". En: *Towards Data Science*. https://towardsdatascience.com/a-comprehensive-hands-on-guide-to-transfer-learning-with-real-world-applications-in-deep-learning-212bf3b2f27a.

Sarofeen, C., Barsdell, B., Carilli, M. y Ruberry, M. (2018). "Training Neural Networks with Mixed Precision: Real Examples". En: *NVIDIA GPU Technology Conference 2018 - San Jose, California*. http://on-demand.gputechconf.com/gtc/2018/video/S81012/.

Schmidhuber, J. (2014). *Deep Learning in Neural Networks: An Overview*. Inf. téc. Galleria 2, 6928 Manno-Lugan. Switzerland: The Swiss AI Lab IDSIA (Instituto Dalle Molle di Studi sull'Intelligenza Artificiale) University of Lugano & SUPSI. https://arxiv.org/abs/1404.7828.

Schroff, F., Kalenichenko, D. y Philbin, J. (2015). "FaceNet: A Unified Embedding for Face Recognition and Clustering". En: *arXiv.org* abs/1503.03832. https://arxiv.org/abs/1503.03832 - FaceNet: https://github.com/davidsandberg/facenet.

Schuster, M. (1999). "On supervised learning from sequential data with applications for speech recognition". Tesis de mtría. Nara Institute of Science y Technology, Kyoto, Japan. https://pdfs.semanticscholar.org/...

Schuster, M. y Paliwal, K. K. (1997). "Bidirectional recurrent neural networks". En: *IEEE Transactions on Signal Processing*. Vol. 45, págs. 2673-2681. https://pdfs.semanticscholar.org/4b80/89bc9b49f84de43acc2eb8900035f7d492b2.pdf?_ga=2.213141277.800372750.1581414946-638859210.1581414946.

SciPy community (2019a). *Broadcasting*. The SciPy community. https://docs.scipy.org/doc/numpy/user/basics.broadcasting.html.

— (2019b). *Indexing*. The SciPy community. https://docs.scipy.org/doc/numpy/reference/arrays.indexing.html.

Sermanet, P., Eigen, D., Zhang, X., Mathieu, M., Fergus, R. y LeCun, Y. (2014). "OverFeat: Integrated Recognition, Localization and Detection using Convolutional Networks". En: *arXiv.org* abs/1312.6229. https://arxiv.org/abs/1312.6229.

Sharma, H. (2019). "Activation Functions: Sigmoid, ReLU, Leaky ReLU and Softmax for Neural Networks and Deep Learning". En: *Medium*. https://medium.com/@himanshuxd/activation-functions-sigmoid-relu-leaky-relu-and-softmax-basics-for-neural-networks-and-deep-8d9c70eed91e.

Sharma, S. (2017). "Activation Functions in Neural Networks: Sigmoid, tanh, Softmax, ReLU, Leaky ReLU EXPLAINED!!!" En: *Towards Data Science*. https://towardsdatascience.com/activation-functions-neural-networks-1cbd9f8d91d6.

Silva, T. (2019). "How to Add Regularization to Keras Pre-trained Models the Right Way". En: *GitHub*. https://sthalles.github.io/keras-regularizer/.

Simonyan, K. y Zisserman, A. (2015). "Very Deep Convolutional Networks for Large-Scale Image Recognition". En: *arXiv.org* abs/1409.1556. https://arxiv.org/abs/1409.1556.

Singh, S. (2018). "Understanding the Bias-Variance Tradeoff". En: *Towards Data Science*. https://towardsdatascience.com/understanding-the-bias-variance-tradeoff-165e6942b229.

Skalski, P. (2019). "Gentle Dive into Math Behind Convolutional Neural Networks". En: *Towards Data Science*. https://towardsdatascience.com/gentle-dive-into-math-behind-convolutional-neural-networks-79a07dd44cf9.

Smith, L. N. (2015). "Cyclical Learning Rates for Training Neural Networks". En: *arXiv.org* abs/1506.01186. https://arxiv.org/abs/1506.01186.

— (2018). "A disciplined approach to neural network hyper-parameters: Part 1 - learning rate, batch size, momentum, and weight decay". En: *arXiv.org*. https://arxiv.org/abs/1803.09820.

Solai, P. (2018). "Convolutions and Backpropagations". En: *Medium*. https://medium.com/@pavisj/convolutions-and-backpropagations-46026a8f5d2c.

Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I. y Salakhutdinov, R. (2014). "Dropout: A Simple Way to Prevent Neural Networks from Overfitting". En: *Journal of Machine Learning Research* 15, págs. 1929-1958. http://jmlr.org/papers/v15/srivastava14a.html.

Steinkraus, D., Simard, P. Y. y Buck, I. (2005). "Using GPUs for machine learning algorithms". En: *Eighth International Conference on Document Analysis and Recognition (ICDAR'05)*.

Sutskever, I., Vinyals, O. y Le, Q. V. (2014). "Sequence to Sequence Learning with Neural Networks". En: *arXiv.org* abs/1409.3215. https://arxiv.org/abs/1409.3215.

Szegedy, C. et al. (2014). "Going Deeper with Convolutions". En: *arXiv.org* abs/1409.4842. http://arxiv.org/abs/1409.4842.

Szegedy, C., Vanhoucke, V., Ioffe, S., Shlens, J. y Wojna, Z. (2015). "Rethinking the Inception Architecture for Computer Vision". En: *arXiv.org* abs/1512.00567. http://arxiv.org/abs/1512.00567.

Szegedy, C., Ioffe, S., Vanhoucke, V. y Alemi, A. (2016). "Inception-v4, Inception-ResNet and the Impact of Residual Connections on Learning". En: *arXiv.org* abs/1602.07261. https://arxiv.org/abs/1602.07261.

Taigman, Y., Yang, M., Ranzato, M. y Wolf, L. (2014). "DeepFace: Closing the Gap to Human-Level Performance in Face Verification". En: *2014 IEEE Conference on Computer Vision and Pattern Recognition*. IEEE. https://www.cs.toronto.edu/~ranzato/publications/taigman_cvpr14.pdf.

Thomas, C. (2019a). "Tabular data analysis with deep neural nets". En: *Towards Data Science*. https://towardsdatascience.com/tabular-data-analysis-with-deep-neural-nets-d39e10efb6e0.

Thomas, R. (2017). "How (and why) to create a good validation set". En: *fast.ai*. https://www.fast.ai/2017/11/13/validation-sets/.

— (2018a). "Artificial Intelligence needs all of us". En: *TEDx Talks*. https://youtu.be/LqjP7O9SxOM.
— (2018b). "Keyaddendum - Some Healthy Principles About Ethics & Bias in AI". En: *PyBay2018*. https://youtu.be/WC1kPtG8Iz8.
— (2019b). "Five Things That Scare Me About AI". En: *fast.ai*. https://www.fast.ai/2019/01/29/five-scary-things/.

Turing, A. M. (1950). "Computing Machinery and Intelligence". En: *Mind* 49. Reedición: https://www.csee.umbc.edu/courses/471/papers/turing.pdf.

Ulyanov, D., Vedaldi, A. y Lempitsky, V. S. (2016). "Instance Normalization: The Missing Ingredient for Fast Stylization". En: *arXiv.org* abs/1607.08022. http://arxiv.org/abs/1607.08022.

van der Maaten, L. e Hinton, G. (2008). "Visualizing Data using t-SNE". En: *Journal of Machine Learning Research* 9, págs. 2579-2605. http://www.jmlr.org/papers/volume9/vandermaaten08a/vandermaaten08a.pdf.

Varoquaux, G. et al. (2017). *SciPy Lecture Notes*. Creative Commons. http://www.scipy-lectures.org/.

Vinyals, O., Toshev, A., Bengio, S. y Erhan, D. (2014). "Show and Tell: A Neural Image Caption Generator". En: *arXiv.org* abs/1411.4555. https://arxiv.org/abs/1411.4555.

Weng, J., Ahuja, N. y Huang, T. S. (1992). "Cresceptron: A Self-Organizing Neural Network Which Grows Adaptively". En: *International Joint Conference on Neural Networks (IJCNN)*. Vol. 1, págs. 576-581. http://vision.ai.illinois.edu/publications/cresceptron_1992.pdf.

Werbos, P. J. (1974). "Beyond Regression: New Tools for Prediction and Analysis in the Behavioral Sciences". Tesis doct. Harvard University.

Wu, Y. y He, K. (2018). "Group Normalization". En: *arXiv.org* abs/1803.08494. http://arxiv.org/abs/1803.08494.

Xu, K. et al. (2015). "Show, Attend and Tell: Neural Image Caption Generation with Visual Attention". En: *arXiv.org* abs/1502.03044. https://arxiv.org/abs/1502.03044.

Yadav, D. (2019). "Categorical encoding using Label-Encoding and One-Hot-Encoder". En: *Towards Data Science*. https://towardsdatascience.com/categorical-encoding-using-label-encoding-and-one-hot-encoder-911ef77fb5bd.

You, Y. et al. (2019). "Large Batch Optimization for Deep Learning: Training BERT in 76 minutes". En: *arXiv.org* abs/1904.00962. http://arxiv.org/abs/1904.00962.

Zagoruyko, S. y Komodakis, N. (2016). "Wide Residual Networks". En: *arXiv.org* abs/1605.07146. https://arxiv.org/abs/1605.07146.

Zeiler, M. D. y Fergus, R. (2013). "Visualizing and Understanding Convolutional Networks". En: *arXiv.org* abs/1311.2901. http://arxiv.org/abs/1311.2901.

Zhang, G., Wang, C., Xu, B. y Grosse, R. B. (2018). "Three Mechanisms of Weight Decay Regularization". En: *arXiv.org* abs/1810.12281. http://arxiv.org/abs/1810.12281.

Zhang, H., Cissé, M., Dauphin, Y. N. y Lopez-Paz, D. (2017). "mixup: Beyond Empirical Risk Minimization". En: *arXiv.org* abs/1710.09412. http://arxiv.org/abs/1710.09412.

Zoph, B., Vasudevan, V., Shiens, J. y Le, Q. V. (2017). "Learning Transferable Architectures for Scalable Image Recognition". En: *arXiv.org* abs/1707.07012. https://arxiv.org/abs/1707.07012.