TRABAJO FIN DE GRADO

Doble Grado Ingeniería Informática y Matemáticas

Exploración de técnicas de entrenamiento de redes neuronales profundas

Enfoques clásicos y técnicas metaheurísticas

Autor
Eduardo Morales Muñoz

Directores
Pablo Mesejo Santiago
Javier Merí de la Maza

Facultad de Ciencias
Escuela Técnica Superior de Ingenierías Informática y de Telecomunicación

—

Granada, mes de 201

Índice

Parte I
Parte informática: enfoque clásico vs técnicas metaheurísticas

1. Introducción

Las redes neuronales profundas han revolucionado el campo de la inteligencia artificial, permitiendo avances significativos en varios campos como el procesamiento del lenguaje natural, reconocimiento de voz o visión por computador. Su capacidad para extraer patrones y representaciones complejas de grandes datasets y a un coste computacional muy eficiente en comparación con otras técnicas las han convertido en piedra angular de los sistemas modernos de aprendizaje automático. En el campo de la visión por computador, las Convolutional Neural Networks (ConvNets) se han erigido como la familia de modelos que consigue un rendimiento del estado del arte [GBC16].

Las ConvNets son un tipo de red neuronal para procesar datos en forma de grid. Se caracterizan porque tienen al menos una capa donde usan la operación de convolución en lugar de una matriz general de multiplicación. La convolución es un tipo de operación lineal que permite capturar representaciones espaciales aplicando un filtro a la entrada, detectando primero características de bajo nivel como bordes y texturas y aumentando el nivel de complejidad de la representación en las capas sucesivas [GBC16].

Las ResNets son una subfamilia de ConvNets que atajan el problema del desvanecimiento y explosión de gradiente. Cuantas más capas tiene una red neuronal profunda más probable es que sufra este problema, ya que se arrastran más operaciones. Las ResNet crean bloques residuales donde se crea un atajo entre el inicio y el final del bloque en el que se suma la identidad al final del bloque, provocando que el gradiente pueda fluir de manera más efectiva durante el proceso de BP [He+15].

El gradiente descendente es un algoritmo de aprendizaje que nos permite entrenar este tipo de modelos de forma eficiente, robusta, y con mucho rigor teórico. Sin embargo como ya se ha comentado en la parte anterior tiene algunas limitaciones, y estas se ven incrementadas cuantas más capas y más parámetros tiene el modelo que entrenamos. A parte de desarrollar mejoras en él con los optimizadores, se buscan nuevos algoritmos de aprendizaje que permitan evitar los problemas que presenta el gradiente descendente.

Una de estas aproximaciones son las técnicas metaheuristicas: estrategias de optimización basadas normalmente en componentes bio-inspirados y que son flexibles y adaptables a gran variedad de problemas. Ofrecen una solución cercana a la óptima en un tiempo razonable en muchos problemas cuya solución óptima es computacionalmente inalcanzable, como en problemas NP-Hard. Son técnicas iterativas que no ofrecen una garantía teórica de hallar una buena solución, pero a través de restricciones en el algoritmo se espera que lo haga [Mar+20].

Los más conocidos son los algoritmos evolutivos inspirados en la evolución genética. En ellos se genera una población aleatoria e iterativamente se realizan los procesos de: seleccionar los mejores individuos, recombinarlos en-

tre ellos, mutarlos para obtener más diversidad genética, y reemplazamiento de los nuevos indiviudos en la población. Se pueden introducir modificaciones como criterios elitistas, en los que por ejemplo reemplazaríamos la población antigua sólo si fuera peor que la nueva. Los algoritmos evolutivos basados en Differential Evolution (DE) se especializan en optimización con parámetros reales y enfatizan la mutación, utilizando el operador de cruce a posteriori de ella. Alcanzan el rendimiento de estado del arte en optimización continua [TBS23].

Los algoritmos meméticos son una hibridación de las técnicas metaheurísticas con algoritmos de búsqueda local, que añaden el uso de información específica del problema. Combinan así la capacidad exploradora del espacio de soluciones que tienen los algoritmos evolutivos con la capacidad explotadora de la búsqueda local. El optimizador local se considera una etapa más dentro del proceso evolutivo y debe incluirse en él.

1.1. Motivación

El ajuste de pesos de un modelo es una de las partes más importantes en su desarrollo y por eso necesitamos de técnicas que nos ofrezcan cada vez mejores resultados a la vez que mayor eficiencia. No se trata de un problema sencillo ya que el número de parámetros de los modelos, es decir la dimensión del problema de optimización, tiene una tendencia que va rápidamente en aumento. Aunque el gradiente descente sea una estrategia muy buena hemos visto sus limitaciones, que nos incitan a intentar encontrar otras estrategias de aprendizaje. Las metaheurísticas toman cada vez un papel más protagonista en la optimización de problemas complejos y de grandes dimensiones a un bajo coste, lo que las sitúa como un posible candidato a sustituirlo.

Para la realización del presente TFG nos basaremos en el reciente paper de Daniel Molina y Francisco Herrera [Mar+20] donde se analiza el papel que juegan actualmente las metaheurísticas tanto en el en entrenamiento de los modelos, como en la selección de los hiperparámetros y de la topología de la red. Nos centraremos únicamente en el primer caso. En él se realiza también un experimento práctico comparativo entre Adam, un optimizador basado en el gradiente descendente, y diferentes versiones de SHADE-ILS, una técnica metaheurística basada en DE que hace uso de búsqueda local (técnica memética) que ofrece los mejores resultados actualmente en el entrenamiento de modelos.

En dicha publicación se realiza una revisión de la literatura en lo referente a las técnicas metaheurísticas para el entrenamiento de modelos, analizando los resultados de los paper más recientes y criticando de manera general la falta de rigor metodológico en la mayoría de ellos, ya que no resulta fácil realizar una comparación totalmente objetiva entre dos ténicas tan distintas como el gradiente descendente y los algoritmos bio-inspirados. Algunas de las principales carencias en la literatura que se intentarán abordar en este

TFG son las siguientes:

- De manera general no se suelen comparar las técnicas metaheurísticas con los métodos clásicos del gradiente descendente, sino que se comparan entre sí.
- Falta de homogeneidad en los datasets usados, lo que no permite una comparación objetiva entre papers
- Modelos no realistas que usan unos pocos de miles de parámetros.
- La gran mayoría de hibridaciones entre técnicas metaheurísticas y gradiente descendente son probadas con ConvNets

Por ello, aunque la literatura sea extensa y de manera general se evidencie la superioridad del gradiente descendente, se hace necesario tanto la realización de experimentos con las mismas condiciones que en otros papers como la repetición de los mismos para obtener esa independencia, objetividad y rigor metodológico que son necesarias en el campo. Vamos a reproducir el experimento en modelos MLP y ConvNets de diferente número de parámetros (desde 2 mil hasta 1 millón), usando técnicas clásicas y metaheurísticas reconocidas por su buen funcionamiento y además probando hibridaciones entre ellas. Se intentará reproducir en la medida de lo posible las condiciones de experimentación del paper de referencia haciendo uso de algunos datasets que se proponen en la publicación y usando las mismas técnicas de entrenamiento, llegando a añadir alguna más.

1.2. Objetivos

El objetivo principal de este TFG es realizar una comparación experimental de las técnicas de entrenamiento de modelos clásicas basadas en gradiente descendente y las nuevas basadas en metaheurísticas, proponiendo hibridaciones para las más consolidadas. Para ello dividimos en los siguientes objetivos secundarios:

1. Reproducción de la experimentación del paper de referencia en el ámbito del entrenamiento de ConvNets.
2. Realización de pruebas experimentales análogas a las anteriores para el entrenamiento de MLP
3. Hibridación de técnicas metaheurísticas usadas con gradiente descendente, con sus pruebas correspondientes.
4. Análisis de resultados de las técnicas clásicas con las metaheurísticas, en comparación con los resultados del paper cuando corresponda, e intentando responder a las cuestiones siguiente:

Familia	MLP				ConvNets		
Modelo	1,2,5 y 11				LeNet5, ResNet-15 y ResNet57		
Datasets	BHP	BCW	WQ		MNIST	F-MNIST	CIFAR10-G
Tarea	R	C	R	C	Clasificación de imágenes		

Cuadro 1: Resumen de la experimentación. BHP: Boston Housing Price, BCW: Breast Cancer Winsconsin, WQ: Wine Quality. R: regresión, C: clasificación. En el caso de MLP, en la fila modelo se indica el número de capas ocultas.

- ¿Qué técnica generaliza mejor?
- ¿Qué técnica tarda menos?
- ¿Cuál tiene mejores propiedades (estabilidad, rapidez...) en su convergencia?

Se responderán a ests cuestiones observando si se encuentran diferencias en base a: tipo de tarea (clasificación/regresión), tamaño y complejidad del dataset, número de parámetros del modelo, familia de modelos.

2. Experimentación

Se ha desarrollado una batería de pruebas amplia y diversa que permita una correcta comparación entre el gradiente descendente y las técnicas metaheurísticas atendiendo a los objetivos señalados anteriormente. En la tabla 1 se ofrece un resumen esquemático de las pruebas que se van a realizar. Todas ellas se realizan con los siguientes optimizadores de gradiente descendente: Nesterov Accelerated Gradient (NAG), RMSProp y Adam. Para las metaheurísticas se ha elegido los ya conocidos en la literatura SHADE y SHADE-ILS, y añadiendo una versión híbrida de cada uno con el gradiente descendente. Todas estas elecciones se detallarán a lo largo de esta sección. Para el desarrollo del código se usa el lenguaje Python con las librerías PyTorch y FastAI, entre otras; implementado y ejecutado en Google Colab. El código puede encontrarse en: https://github.com/eedduu/TFG.

2.1. Modelos

Usaremos dos familias de modelos: MLP y ConvNets. Con los primeros usaremos datasets tabulares para clasificación y regresión, y con los segundos datasets de imágenes para la tarea de clasificación. En la siguiente sección se detallan los datasets con sus características.

La implementación de los MLP se ha realizado a través de la librería FastAI por simplicidad ya que ofrece lo necesario para usarlos directamente. La implementación de las ConvNets se ha realizado desde cero, observando la

Capas ocultas	**Neuronas por capa**	**Parámetros**
1	64	2238
2	64, 64	6462
5	64, 128, 256, 128, 64	85k
11	32, 64, 128, 256, 512, 1024, 512, 256, 128, 64 y 32	1.4M

Cuadro 2: Detalles de los modelos MLP

topología de LeNet5 y las ResNets en sus papers originales, ya que en ellas sí que se han introducido ciertos cambios que se comentan más adelante. Estas modificaciones tienen como objetivo una batería experimental más amplia y acorde a las condiciones que buscamos. Todos los modelos han sido entrenados desde cero.

Para los MLP usaremos 5 modelos, con 1,2,5 y 11 capas ocultas cada uno. El número de neuronas por capa es una potencia de 2 y con estructura piramidal incremental, es decir primero aumentando el número de neuronas por capa y luego disminuyéndolo. Estas son elecciones comunes en la literatura ya que facilitan las operaciones por su estructura (la primera) y el tratamiento de los datos (la segunda). El objetivo es conseguir una variedad experimental que permita medir los efectos de la complejidad del modelo sobre la tarea y el overfitting además de adecuarse a las condiciones del paper de referencia, con modelos desde aproximadamente mil parámetros hasta casi 1.5M como vemos en 2, acercándose al modelo más grande presentado en dicho paper.

Antes de cada capa linal hay una de BatchNorm1D, ya que es la implementación por defecto de FastAI y mejora el rendimiento en el entrenamiento. Los parámetros asociados a este tipo de capa y a los de la capa de salida van incluidos en el cómputo anterior. Se incluye al final del modelo una capa de SoftMax en caso de que la tarea sea clasificación.

Para los modelos basados en convoluciones usamos LeNet5 y dos ResNets, con 15 y 57 capas. El objetivo es de nuevo ofrecer una variedad experimental, con el primer modelo teniendo unos 60k parámetros mientras que el tercero tiene 1.3M, imitando de nuevo las condiciones del paper de referencia. Hay que resaltar que la intención es comparar varios modelos con diferente número de parámetros para ver el efecto sobre éstos, y no hablamos de modelos más o menos potentes, ya que no siempre un incremento en el número de parámetros se traduce en un mejor rendimiento.

LeNet5 es un modelo de sobra conocido presentado por Yann LeCun en [Lec+98], al que sustituimos las funciones de activación por ReLU, ya que en la literatura posterior a la presentación del modelo se han demostrado superiores a las sigmoides y la tangente hiperbólica. También se han sustituido las capas de AveragePool por MaxPool y añadido capas de BatchNorm por

Capa	Dimensión	Kernel	Canales
Convolución	28x28	5x5	6
BatchNorm2D	28x28	-	-
ReLU	28x28	-	-
Max Pool	14x14	2x2, stride 2	-
Convolución	10x10	5x5	16
BatchNorm2D	10x10	-	-
ReLU	10x10	-	-
Max Pooling	5x5	2x2	-
Lineal	120	-	-
BatchNorm1D	120	-	-
ReLU	120	-	-
Lineal	84	-	-
BatchNorm1D	84	-	-
ReLU	84	-	-
Lineal	num_classes	-	-

Cuadro 3: Topología de LeNet5 para imágenes 32x32 con un canal de entrada. Las columnas dimensión y canales hacen referencia a la salida de la capa.

los mismos motivos. En la tabla 3 se muestra la topología de este modelo, obviando las capas de Flatten y de SoftMax. Tiene un total de 62 mil parámetros.

Se han diseñado dos modelos de ResNet, uno con 15 capas y otro con 57. A priori puede parecer excesivo 57 capas, pero con la implementación a través de BottleNeckBlocks se aumenta el número de capas considerablemente con un leve incremento del número de parámetros. Sigue además la tendencia en la literatura de aumentar el número de capas antes que el número de filtros. Se han elegido 57 capas con la intención de aproximarse al número de parámetros del modelo más grande del paper de referencia y aprovechar la estructura de esta familia de modelos. Las ResNets se caracterizan por usar bloques convolucionales que agrupan varias capas de convolución donde al final de cada bloque se suma la entrada del mismo, con el objetivo de evitar el problema del desvanecimiento de gradiente (ver sección **??**). En un modelo con pocas capas no se aprecia tan bien este efecto. Su topología se detalla en la tabla 4.

El modelo intermedio, Resnet15, sigue la misma estructura que ResNet57 pero usando menos bloques convolucionales. El objetivo es tener un modelo intermedio, con unos 500 mil parámetros y que nos permita comparar con los otros dos, en términos de entrenamiento y de overfitting, algo muy usual en las redes neuronales con gran número de parámetros. Su topología se

Capa	Dimensión	Kernel/Stride	Canales
Convolución	26x26	7x7	64
BatchNorm2d	26x26	-	-
ReLU	26x26	-	-
MaxPool2d	13x13	2x2, stride 2, padding 1	-
BottleneckBlock x3	13x13	1x1, 3x3, 1x1	64
BottleneckBlock x4	7x7	1x1, 3x3, 1x1, stride 2	128
BottleneckBlock x4	4x4	1x1, 3x3, 1x1, stride 2	256
BottleneckBlock x3	2x2	1x1, 3x3, 1x1, stride 2	512
AdaptiveAvgPool2d	512	-	-
BatchNorm1d	512	-	-
Dropout	512	-	-
Lineal	num_classes	-	-

Cuadro 4: Topología de ResNet57 para imágenes 32x32 con un canal de entrada. Las columnas dimensión y canales hacen referencia a la salida de la capa.

detalla en la tabla 5.

Los bloques convolucionales agrupan 3 capas de convolución con sus respectivas capas BatchNorm, y se usan convoluciones 1x1 para hacer cuello de botella, reduciendo así el número de parámetros sin perder expresividad de la red [LCY14; Sze+14]. Se sigue el diseño usual de esta familia de modelos, por ejemplo agrupando más bloques convolucionales en mitad de la red, con una convolución preia a los bloques convolucionales y usando solo una capa lineal. El modelo ResNet57 que se implementa tiene un total de 1.3M de parámetros.

2.2. Datasets

TODO: añadir información sobre los datasets: número de características, target, tamaño, qué contienen, imagenes.

2.2.1. Tabulares

BCW y WQ para clasificación, BHP y WQ para regresión. El objetivo es comparar también el rendimiento de las metaheurísticas en estas tareas, ya que la gran mayoría de la literatura sobre MH se realiza con ConvNets y por tanto con tareas de clasificación. BCW y BHP son dos dataset pequeños (alrededor de 500 instancias) y más fáciles que WQ, que tiene una cantidad de unas 6000 instancias. Por ello hay una tarea sencilla y una difícil para regresión y clasificación.

Capa	Dimensión	Kernel/Stride	Canales
Convolución	26x26	7x7	64
BatchNorm2d	26x26	-	-
ReLU	26x26	-	-
MaxPool2d	13x13	2x2, stride 2, padding 1	-
BottleneckBlock x1	13x13	1x1, 3x3, 1x1	64
BottleneckBlock x1	7x7	1x1, 3x3, 1x1, stride 2	128
BottleneckBlock x1	4x4	1x1, 3x3, 1x1, stride 2	256
BottleneckBlock x1	2x2	1x1, 3x3, 1x1, stride 2	512
AdaptiveAvgPool2d	512	-	-
BatchNorm1d	512	-	-
Dropout	512	-	-
Lineal	num_classes	-	-

Cuadro 5: Topología de ResNet15 para imágenes 32x32 con un canal de entrada. Las columnas dimensión y canales hacen referencia a la salida de la capa.

Para estos datasets se ha realizado un preprocesado de los datos básico y con decisiones comunes basadas en la literatura. Se han eliminado las variables que tienen menos de un 5 o 10 % (dependiendo de la cantidad de variables del dataset) de correlación con el target. Con las parejas de variables que tienen más de un 90 % de correlación entre sí se elimina una de las dos. Se han eliminado outliers con el método zscore usando un threshold de 3 y se han normalizado los datos de entrada.

El tamaño del batch se ha elegido mediante pruebas experimentales entre los valores 32, 64 y 128. Se divide el conjunto de datos en entrenamiento-validación-test, con un porcentaje 70-10-20.

2.2.2. Imágenes

Para los datasets de imágenes se han elegido tres de los usados en el paper de referencia. Se han elegido MNIST y FMNIST ya que son los más usados para la comparativa en dicho paper, y CIFAR10 ya que a priori es más difícil que los otros, siendo MNIST el más fácil de los tres. Esto proporciona unas pruebas equilibradas, en el mismo marco que el paper y permite una fácil comparación. Al igual que allí, se ha reducido el tamaño de los datasets a 10 mil imágenes para el entrenamiento y 5 mil para el test. El conjunto de test se divide 50-50 para validación y test.

Se usan las imágenes con una resolución de 32x32 y un solo canal, adaptando las imágenes a estas dimensiones cuando sea necesario. No se usa preprocesamiento de datos ya que se entiende que la propia red a través de las convoluciones los procesa.

2.3. Otras decisiones

Para la reproducibilidad de la experimentación, se fija la semilla 42 en todos las librerías necesarias. No se usa cross validation debido a que las técnicas metaheuristicas requieren de mucho más poder de cómputo que el disponible para poder ejecutarlas varias veces en un tiempo razonable.

Se ha usado la inicialización de pesos Glorot como en el paper a comparar. Para los optimizadores basados en gradiente descendente se han usado los mismos pesos iniciales, mientras que en las técnicas metaheurísticas se ha usado la misma población inicial, para asegurar la igualdad de condiciones en el entrenamiento debido a la gran sensibilidad de éste a los parámetros iniciales.

Se fija 20 como número de épocas en todos los entrenamientos ya que es suficiente para la convergencia de los métodos de gradiente descendente. Siguiendo el criterio del paper de referencia, en las técnicas metaheurísticas una época es algo distinta, siendo en el paper equivalente a $N_eval * N_epochs * N_capas$. En nuestro caso no se ha usado el factor número de capas por diversas razones: no se realiza entrenamiento individualizado por capas, no se dispone de tanto poder de cómputo, usamos modelos con muchas capas. Aunque usemos menos evaluaciones del conjunto de entrenamiento por época, la comparación sigue en la misma dinámica y las metaheurísticas recibirán mucho más poder de procesamiento que las técnicas clásicas.

Para las tareas de regresión se ha usado el error cuadrático medio como función de coste. Es ampliamente usada en la literatura y aunque es sensible a los outliers, como tenemos preprocesamiento de datos no nos afecta. Se ha usado también la métrica R2 cuadrado para medir la explicación de la varianza con respecto a la media como predicción, para tener un criterio objetivo de comparación ya que en las dos tareas de regresión la escala del objetivo es distinta. Para las tareas de clasificación se ha usado CrossEntropyLoss como función de error y accuracy como métrica, opciones ampliamente usadas en la literatura. En los datasets tabulares se ha usado BalancedAccuracy en lugar de Accuracy, ya que las clases no están balanceadas, y se conoce que en estos casos accuracy no es una métrica representativa, de hecho se hace especial mención a esto en el paper. En los datasets de imágenes las clases están perfectamente balanceadas por lo que se usa accuracy, aunque en este caso su versión balanceada coincidiría con la normal.

2.4. Optimizadores

Existen tres familias de optimizadores del gradiente descendente principales: las que usan el momento, las que autoajustan el learning rate y las que combinan estas dos estrategias. Para la experimentación se elige un optmizador de cada familia con la intención de ver si existe alguna diferencia en el entrenamiento del gradiente descendente debido a éstos. El criterio para

la elección del optimizador de cada familia ha sido el número de citas en el paper de presentación y su extensión de uso, analizando cómo de frecuente es su uso en la literatura y su implementación por defecto en las principales librerías de aprendizaje automático. Respectivamente se han elegido: NAG, RMSProp y Adam.

Se ha usado la implementación de estos optimizadores de PyTorch, usandose en el entrenamiento a través de la librería FastAI. Se usa en el entrenamiento la política de ciclos de Leslie, que proporciona una velocidad de convergencia mucho mayor variando el learning rate de forma cíclica. Para elegir el valor máximo del learning rate se usa un buscador de learning rate, que aporta unos resultados muy buenos a un coste bajo. Cabe destacar que la importancia del learning rate es sensiblemente menor en los optimizadores de RMSProp y Adam, ya que de manera automática ajustan estos valores a lo largo del entrenamiento.

Para los hiperparámetros de dichos optimizadores se han usado los valores por defecto de la librería PyTorch. En primer lugar, la intención con la experimentación es establecer una comparativa en igualdad de condiciones sobre el entrenamiento de modelos, y no obtener el máximo rendimiento de cada uno. En segundo lugar estos valores por defecto están basados en los propuestos en los respectivos papers originales y ajustados a través de numerosas pruebas experimentales, de manera que proporcionen los mejores resultados posibles de manera general basandose en la propia implementación de PyTorch.

2.5. Metaheurísticas

Para una correcta comparación con el paper de referencia, que usa el algoritmo SHADE-ILS, se eligen 4 algoritmos metaheurísticos entorno a éste. En primer lugar usamos SHADE-ILS, en su versión completa (entrenando todos los parámetros a la vez). Usamos también el algoritmo de SHADE, sin búsqueda local, para usar un algoritmo puramente metaheurístico que no sea memético. Luego estas dos versiones anteriores las hibridamos con el gradiente descendente, para tener algoritmos meméticos que usen información específica del problema.

El algoritmo de SHADE se implementa a través de la librería pyade-master (LINK), y se le han realizado modificaciones para mantener los parámetros adaptativos del algoritmo SHADE entre ejecuciones distintas y adaptar las estructuras de datos a las del resto del código. A partir de dicho algoritmo se implementan manualmente el resto. Para el algoritmo SHADE-ILS, que combina dos tipos de búsqueda local en su paper original, con una aplicación general, se ha decidido mantener sólo la búsqueda local a través de L-BFGS, ya que al hacer uso del gradiente se maneja información más específica del problema. Esto se realiza a través de la librería scipy.

Se mantiene la misma división de los datos que en el uso de optimi-

zadores. Se ejecutan los algoritmos evaluando el error sobre el conjunto de entrenamiento y guardando en cada epoch el mejor individuo de la población junto a su error en el entrenamiento, para luego evaluarlos sobre el conjunto de validación y observar qué modelo generaliza mejor a priori, igual que hacemos en el entrenamiento de modelos con gradiente descendente. Luego evaluamos el modelo sobre el conjunto de test, calculando las métricas asociadas a cada tarea.

Como en el paper de referencia no aparecen los hiperparámetros asociados a estos algoritmos, se decide usar el mismo criterio que con los optimizadores y usar los hiperparámetros por defecto de la implementación de la librería pyade-master, que además son valores comúnes en la literatura.

Referencias

[GBC16] Ian J. Goodfellow, Yoshua Bengio y Aaron Courville. *Deep Learning*. http://www.deeplearningbook.org. Cambridge, MA, USA: MIT Press, 2016.

[He+15] Kaiming He et al. *Deep Residual Learning for Image Recognition*. 2015. arXiv: 1512.03385 [cs.CV]. URL: https://arxiv.org/abs/1512.03385.

[LCY14] Min Lin, Qiang Chen y Shuicheng Yan. *Network In Network*. 2014. arXiv: 1312.4400 [cs.NE]. URL: https://arxiv.org/abs/1312.4400.

[Lec+98] Y. Lecun et al. "Gradient-based learning applied to document recognition". En: *Proceedings of the IEEE* 86.11 (1998), págs. 2278-2324. DOI: 10.1109/5.726791.

[Mar+20] Aritz D. Martinez et al. *Lights and Shadows in Evolutionary Deep Learning: Taxonomy, Critical Methodological Analysis, Cases of Study, Learned Lessons, Recommendations and Challenges*. 2020. arXiv: 2008.03620 [cs.NE].

[Sze+14] Christian Szegedy et al. *Going Deeper with Convolutions*. 2014. arXiv: 1409.4842 [cs.CV]. URL: https://arxiv.org/abs/1409.4842.

[TBS23] Vinita Tomar, Mamta Bansal y Pooja Singh. "Metaheuristic Algorithms for Optimization: A Brief Review". En: *Engineering Proceedings* 59.1 (2023). ISSN: 2673-4591. DOI: 10.3390/engproc2023059238. URL: https://www.mdpi.com/2673-4591/59/1/238.