Aprendizaje profundo

Marta Rodríguez Sampayo
Minería de datos
Universidad Nacional de Educación a Distancia
mrodrigue8255@alumno.uned.es
16 de agosto de 2020

1. Introducción

El término aprendizaje profundo, más conocido como *deep learning*, hace referencia a una rama del aprendizaje automático y, por tanto, de la Inteligencia Artificial, basada en el uso de algoritmos y estructuras lógicas que pretenden imitar la organización del sistema nervioso de los mamíferos, creando representaciones de alto nivel de conjuntos de datos y permitiendo el procesado de estos a través de diferentes niveles jerárquicos, empleando transformaciones no lineales. Dentro de este conjunto de métodos han ganado popularidad las llamadas redes neuronales, tema del que trata esta memoria.

Una red neuronal es un modelo estadístico no lineal, empleado para clasificación o regresión. Estas redes se dividen en capas, que a su vez están formadas por un número limitado de unidades denominadas neuronas. El proceso de aprendizaje consiste en derivar ciertas características a partir de combinaciones lineales de las entradas y modelar las salidas como combinaciones lineales de estas características [1]. Cada modelo posee un conjunto de parámetros o pesos cuyos valores se calculan durante el entrenamiento, minimizando una función de error dada, y permiten que la red se ajuste a los datos de entrada empleados. Un tipo particular de red neuronal son las redes convolucionales o *Convolutional Neural Networks* (CNNs), que emplean la operación de convolución y se especializan en datos que siguen una estructura de cuadrícula, como imágenes o series temporales [2].

El objetivo de esta práctica es evaluar el desempeño de diferentes modelos de redes neuronales al llevar a cabo una tarea de clasificación sobre un conjunto de imágenes de granos de polen. La estructura de este documento se describe a continuación. En la Sección 2 se presenta el conjunto de datos empleado en los experimentos. Seguidamente, los modelos de redes empleados se describen en detalle en la Sección 3, junto a la justificación de las decisiones de diseño tomadas. Los resultados experimentales se muestran en la Sección 4 y se analizan en la Sección 5. Finalmente, en la Sección 6 se recogen las conclusiones obtenidas.

2. Conjunto de datos

La tarea a realizar consiste en la identificación de dos especies neozelandesas diferentes de granos de polen. Ambas especies resultan muy similares, prácticamente indistinguibles por expertos humanos, dificultando la tarea de clasificación. El conjunto de datos proporcionado consta de 2400 imágenes, 1200 de cada uno de los tipos, de 100 píxeles de alto y ancho. Para llevar a cabo la evaluación de los distintos modelos de redes neuronales, se realiza la división de estas imágenes en dos subconjuntos: entrenamiento y pruebas.

Los datos de entrenamiento son aquellos que emplea la red para actualizar los pesos, constan de 1920 imágenes. Durante este entrenamiento se emplea el conjunto de pruebas de 480 imágenes para evaluar el desempeño del modelo sobre imágenes que no han sido vistas, permitiendo evitar un sobreajuste y monitorizar el proceso de aprendizaje. Estos subconjuntos se generan de forma balanceada, manteniendo la proporción entre las muestras de cada clase.

A mayores, se han realizado experimentos con generación de nuevos datos artificiales o *data augmentation*, de forma que aplicando pequeñas transformaciones a las imágenes (rotaciones, variaciones en la localización de los píxeles y giros) se consiguen nuevas muestras a utilizar durante el entrenamiento de la red. Este procedimiento suele emplearse en caso de disponer de conjuntos de datos de entrada limitados para aumentar su tamaño y trabajar con un número mayor de muestras. El éxito de un modelo de red neuronal no está directamente relacionado con el tamaño del conjunto de entrenamiento, otros factores tienen una influencia mayor tanto a nivel de eficiencia como eficacia, por ejemplo, la calidad de los datos o las características representadas en las imágenes, por lo que la generación artificial de nuevas imágenes no tiene por qué tener un efecto positivo en el rendimiento final del sistema. La utilidad o no de esta técnica en los experimentos llevados a cabo se detalla más adelante, junto a los resultados obtenidos.

3. Descripción de los modelos

Se han empleado cinco modelos de redes neuronales para llevar a cabo los experimentos: (1) una red neuronal sin capa oculta; (2) una red neuronal con una capa oculta; (3) una red neuronal convolucional con una única capa convolucional; (4) una red neuronal convolucional con una topología más compleja y (5) una red neuronal convolucional preentrenada. Aunque varía la topología de cada red, ciertas decisiones de diseño son comunes en todos los modelos, puesto que son relativas al problema a tratar en concreto o con el objetivo de realizar una comparación uniforme.

Función de pérdidas Esta función es utilizada por la red para optimizar el valor de los parámetros o pesos durante el entrenamiento [3]. Puesto que se trata de un problema de clasificación con varias categorías, se emplea para este cálculo la Entropía Categórica

Cruzada o *Categorical Crossentropy (CCE)*, definida según la siguiente fórmula:

$$CCE = -\sum_{i=1}^{N} y_i \cdot \log \hat{y}_i$$

donde para un número N de observaciones, i representa cada una de ellas, $\hat{y}_i$ es el valor predicho por el modelo correspondiente al valor objetivo y_i para dicha observación.

Esta función es la más utilizada para este tipo de problemas, ya que permite medir el error entre la distribución de probabilidad inferida y la real. Existen diferentes variantes según el tipo de clasificación a realizar (binaria, con un número elevado de clases, etc.), pero el cálculo es esencialmente el mismo.

Métrica El objetivo de esta función es evaluar la calidad de las predicciones realizadas durante el entrenamiento. En este caso, se utiliza la precisión de clasificación, definida como el número correcto de predicciones entre el total. Ya que el tamaño de muestras de cada clase es balanceado, esta métrica proporciona un buen indicador del rendimiento del modelo.

Algoritmo de optimización En este contexto, los optimizadores son utilizados para encontrar un equilibrio entre tiempo de ejecución y calidad de los resultados obtenidos [4]. Para todos los casos se emplea AdaM [5], ampliamente utilizado en este tipo de sistemas, ya que combina eficiencia y eficacia de forma conveniente para aportar soluciones sencillas de implementar y apropiadas para una gran variedad de problemas [6].

Tasa de aprendizaje (*learning rate*) Este parámetro controla cuánto varía la salida del modelo en respuesta al error estimado cada vez que se actualizan los pesos. Es decir, con una tasa de aprendizaje pequeña, los cambios efectuados en los pesos son menores, aumentando el tiempo de ejecución del entrenamiento y pudiendo conllevar un estancamiento en el aprendizaje; por el contrario, un valor demasiado grande puede resultar en un aprendizaje subóptimo, de forma que se convergerá a una mayor velocidad a un conjunto de pesos que puede no resultar adecuado. Este factor se encuentra íntimamente ligado al optimizador elegido y se ha tomado como referencia la sugerencia de los autores en [5] de utilizar un valor inicial de 0.001 para obtener buenos resultados empleando AdaM.

Funciones de activación Las diferentes funciones que se pueden emplear para la activación de las neuronas de cada capa de la red se presentan de forma resumida en [7]. En base a este recurso, se emplea ReLU, siglas procedentes del inglés *Rectified Linear Units*, en las capas ocultas de las redes, debido a su popularidad y sus ventajas sobre otras funciones (mayor velocidad o disminución de errores de propagación, entre otras). Sin embargo, en la última capa de los modelos, la función de activación a utilizar es *Softmax*, puesto que permite obtener una distribución de probabilidad como salida. Es decir, el número de neuronas de la capa final es igual al número de clases del problema y cada entrada del vector generado se corresponde con la probabilidad de que cierta muestra de entrada pertenezca a cierta clase.

Épocas (*epochs)* El número de épocas se corresponde con el número de iteraciones sobre el conjunto completo de datos durante el entrenamiento, por lo que es un uno de los factores que define la duración del mismo. Este valor se ha configurado de forma predeterminada como 60, para realizar una comparación de forma visual en las gráficas de todos los modelos, pero no es el óptimo para cada caso. Se ha escogido el mayor valor entre los óptimos para cada modelo.

Sobreajuste (*overfitting*) Este problema es muy común en este tipo de modelos y consiste en la sobreadaptación o ajuste del modelo al conjunto de datos empleados para el entrenamiento, alcanzando el mínimo global para este subconjunto y, posiblemente, impidiendo realizar una generalización a otras muestras. Se han empleado varias estrategias para tratar de prevenir este inconveniente:

- *Dropout*: esta técnica consiste en descartar algunas neuronas aleatoriamente durante el entrenamiento para evitar que se sobreadapten a los datos [8]. Se han añadido bloques de *dropout* a la salida de las capas ocultas de la red convolucional para comprobar su efecto. Este método introduce un nuevo hiperparámetro que regula la probabilidad de descarte de neuronas, tras realizar varios experimentos se emplea el valor de 0.2.

- Monitorización del entrenamiento: tras finalizar cada iteración sobre el conjunto de datos de entrenamiento y actualizar los parámetros del modelo, se evalúa este contra el conjunto de datos de validación y se calculan tanto las pérdidas como la precisión. Esta monitorización permite establecer una regla de parada anticipada del entrenamiento, de forma que si durante un número preestablecido de épocas (*patience*) la precisión sobre el conjunto de validación no mejora, se detiene el entrenamiento. De esta forma, aunque la precisión de entrenamiento continúe aumentando, se evita alcanzar el óptimo global, evitando el *overfitting*.

- Reducción de la tasa de aprendizaje: otra esquema de funcionamiento que puede resultar efectivo es el denominado *reduce on plateau*, consistente en reducir la tasa de aprendizaje de forma automática cuando se detecta que la precisión se mantiene constante durante varias épocas consecutivas (*patience*). Esta reducción disminuye la velocidad de aprendizaje cuando el modelo se encuentra cerca del mínimo. En la Figura 1 se puede ver el esquema de funcionamiento de este método.

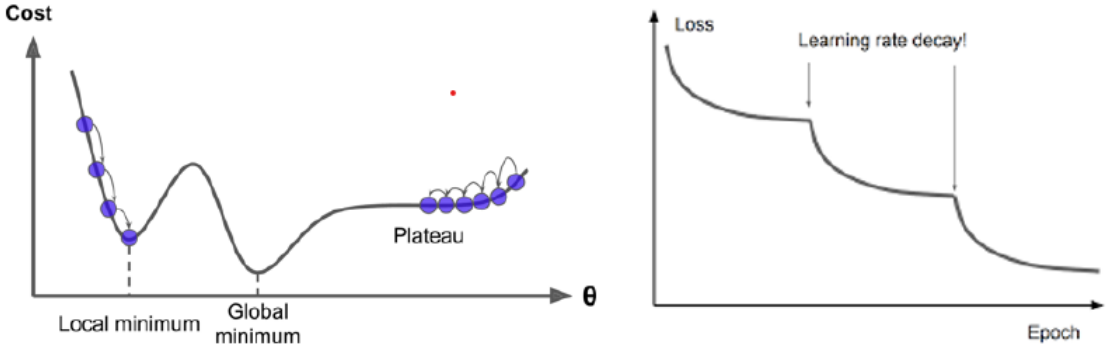
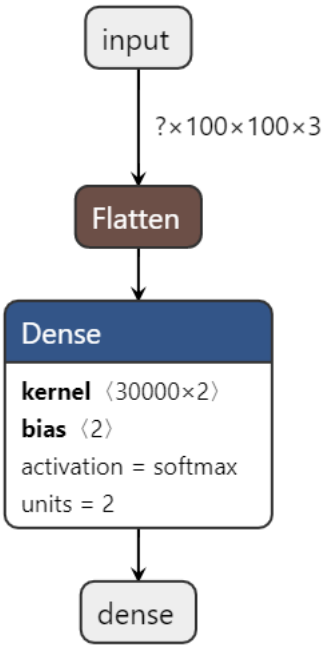


Figura 1: Reducción automática de la tasa de aprendizaje.

Los parámetros de configuración de estas estrategias se han fijado tras realizar diferentes experimentos en base a los mejores resultados obtenidos. De esta forma, el parámetro *patience* de parada anticipada es de 25 *epochs*, mientras que el de reducción de la tasa de aprendizaje es de 15 *epochs*, además, en esta última configuración la reducción se hace con un factor de 0.2 y se establece un valor mínimo de $1e-7$.

3.1. Red neuronal sin capa oculta



(a) Diagrama

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
flatten (Flatten)            (None, 30000)             0
_________________________________________________________________
dense (Dense)                (None, 2)                 60002
=================================================================
Total params: 60,002
Trainable params: 60,002
Non-trainable params: 0
_________________________________________________________________
```

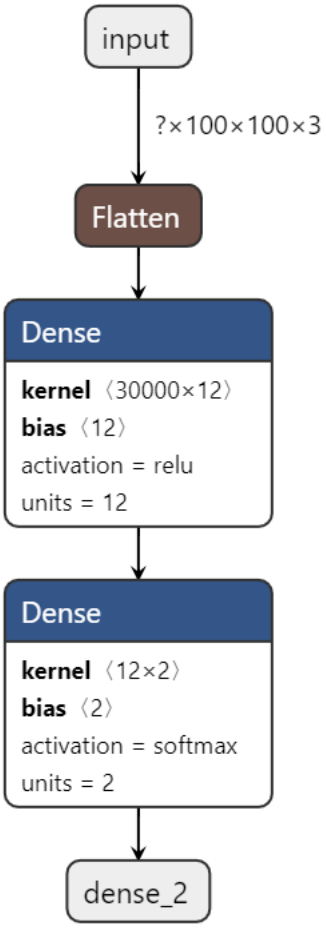
(b) Parámetros

Figura 2: Topología del modelo 1.

Esta primera red neuronal es la más sencilla, ya que consta únicamente de una capa de entrada y una de salida. Como se puede observar en la Figura 2a, la entrada a la

red está compuesta por imágenes de 100x100 píxeles y 3 canales[1]. La primera capa de tipo *Flatten* simplemente convierte la imagen a un vector de tamaño 30000 que es el número de unidades de entrada de la red. La segunda capa de tipo *Dense* es una capa totalmente conectada, con función de activación *softmax* y dos neuronas de salida, cada una correspondiente a una de las clases de granos de polen. Por cada unidad de salida se emplea una unidad *bias* (o sesgo) que almacena el valor constante 1, contribuyendo a la salida de la red [9]. Por tanto, como se muestra en al Figura 2b, la red está formada por 60002 parámetros o pesos (30000 neuronas de entrada conectadas a 2 de salida más 2 neuronas de sesgo).

3.2. Red neuronal con una capa oculta



(a) Diagrama

Layer (type)	Output Shape	Param #
flatten_1 (Flatten)	(None, 30000)	0
dense_1 (Dense)	(None, 12)	360012
dense_2 (Dense)	(None, 2)	26

Total params: 360,038
Trainable params: 360,038
Non-trainable params: 0

(b) Parámetros

Figura 3: Topología del modelo 2.

[1]Cada canal se corresponde con un color del formato RGB, al tratarse de imágenes en escala de grises solo sería necesario un canal, pero se emplean los 3 para homogeneizar la entrada de los diferentes modelos ya que la red preentrenada necesita los 3 canales.

De la misma forma que el modelo anterior, esta red está basada en el Ejemplo 11.7 de [1]. En este caso, se añade una capa oculta formada por 12 neuronas totalmente conectadas. De esta forma, la entrada a la primera capa sigue siendo un vector de tamaño 30000 creado a partir de cada imagen, pero esta vez la función de activación de esta capa es *ReLU* y se añaden 12 unidades de *bias*, una por cada neurona oculta. La capa de salida es similar a la del modelo anterior, salvo que está conectada a las 12 neuronas de la capa oculta. En la Figura 3b se representa el número de parámetros de este modelo, correspondientes a 30000 unidades de entrada, conectadas a 12 ocultas, más 12 de sesgo y estas conectadas a 2 de salida, más 2 de sesgo correspondientes, haciendo un total de 360038 parámetros.

3.3. Red neuronal convolucional

Se han diseñado dos arquitecturas de redes neuronales convolucionales. La primera, más simple, consta de la red neuronal descrita en el apartado anterior precedida de una capa de convolución y una de *pooling* para comprobar el efecto de la capa de convolución añadida. La segunda, de mayor complejidad, sigue la topología mostrada en el diagrama de la Figura 4.

Los modelos de redes convolucionales están formados por bloques de tres etapas (a veces consideradas capas del modelo):

1. Etapa o capa de convolución: se realiza la operación matemática de convolución entre la entrada y el filtro o *kernel*, esta operación puede verse como una multiplicación matricial. Puesto que se trabaja con imágenes en dos dimensiones, el *kernel* elegido también tiene esta dimensionalidad y los valores que lo conforman son aprendidos durante el entrenamiento. Además, el *kernel* suele ser de menor tamaño que la entrada y se aplica a bloques de la imagen, reduciendo el número de parámetros a almacenar por la red y mejorando su eficiencia. Al realizar una serie de convoluciones en paralelo sobre la imagen se producen un conjunto de activaciones lineales en esta etapa.

2. Etapa o capa de detección: tras realizar la convolución, se aplica una función de activación no lineal. En este caso, *ReLU*.

3. Etapa o capa de *pooling*: para finalizar, se produce una reducción de muestreo, consistente en reemplazar la salida del modelo con un resumen estadístico de salidas cercanas. Esta operación convierte la representación de salida en invariante a pequeñas traslaciones de la entrada. Se realizan experimentos con dos funciones, por un lado *max pooling*, que selecciona el máximo valor de una subregión rectangular de la entrada. Por otro lado, *average pooling*, que emplea como valor el promedio de la región rectangular. Se presentan los resultados obtenidos con ambos tipos de *pooling*.

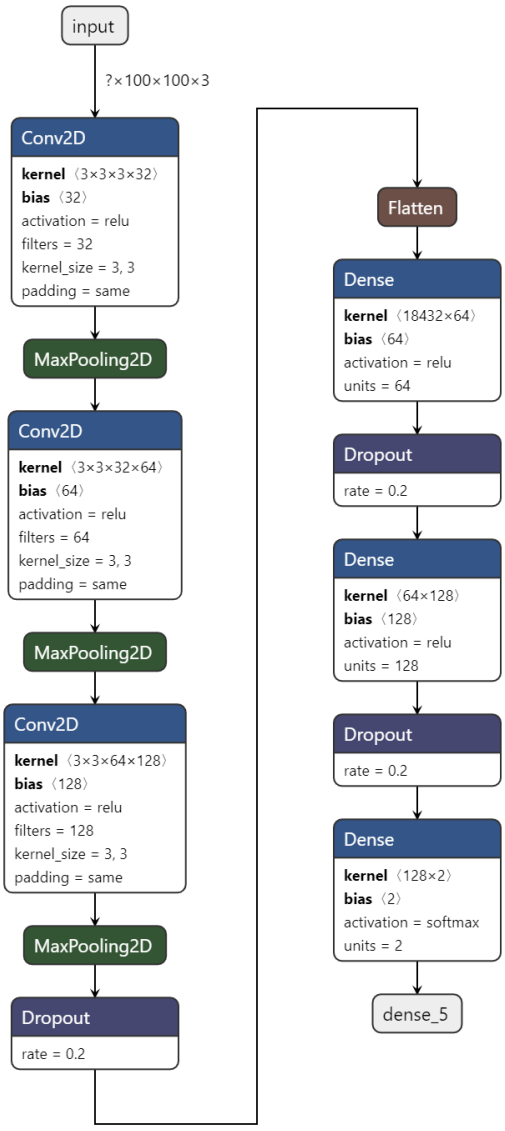


Figura 4: Diagrama de red del modelo 3.

Otro concepto relevante de las CNNs es el de paso o *stride*, ya que influye tanto en la etapa de convolución como en la reducción de resolución. El valor de este parámetro es la cantidad en la que se desplaza el *kernel* sobre la imagen de entrada. Aumentando el valor de paso se disminuye la superposición de los campos receptivos y se disminuyen las dimensiones espaciales. También resulta posible añadir relleno o *padding* a los datos de entrada (habitualmente, ceros), de forma que el volumen no disminuya entre capas convolucionales y sea posible extraer un mayor número de características de bajo nivel, preservando una mayor cantidad de información.

En lo referente al modelo diseñado para los experimentos (Figura 4), se emplea *padding* en todas las capas de convolución para mantener el tamaño de entrada y reducirlo en las capas de *pooling* mediante la estrategia *max pooling*. El *kernel* utilizado para *pooling* es de 2x2, igual que el paso; mientras que para la convolución se emplea un *kernel* de tamaño 3x3 en todas las capas. Se emplean filtros de este tamaño para recoger características

locales de pequeños conjuntos de píxeles de la imagen, ya que las diferencias entre las dos clases de granos de polen son casi imperceptibles (con filtros de mayor tamaño se identificarían características más globales o generales). El número de filtros empleado en cada capa convolucional se dobla, de forma que al principio se detecten características de bajo nivel y se combinen para aumentar su complejidad representando información de mayor nivel en las capas posteriores[2].

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 100, 100, 32)	896
max_pooling2d (MaxPooling2D)	(None, 50, 50, 32)	0
conv2d_1 (Conv2D)	(None, 50, 50, 64)	18496
max_pooling2d_1 (MaxPooling2	(None, 25, 25, 64)	0
conv2d_2 (Conv2D)	(None, 25, 25, 128)	73856
max_pooling2d_2 (MaxPooling2	(None, 12, 12, 128)	0
dropout (Dropout)	(None, 12, 12, 128)	0
flatten_2 (Flatten)	(None, 18432)	0
dense_3 (Dense)	(None, 64)	1179712
dropout_1 (Dropout)	(None, 64)	0
dense_4 (Dense)	(None, 128)	8320
dropout_2 (Dropout)	(None, 128)	0
dense_5 (Dense)	(None, 2)	258

Total params: 1,281,538
Trainable params: 1,281,538
Non-trainable params: 0

Figura 5: Parámetros del modelo 3.

Al finalizar la llamada base convolucional del modelo, es decir, las capas de convolución y *pooling*, se incluye una capa de *Dropout* con una probabilidad de 0.2. Tras esta capa, para realizar la clasificación, se debe incluir un bloque de capas totalmente conectadas como las que conforman los modelos 1 y 2 anteriormente explicados. A mayores se incluye otra capa de *Dropout* con la misma tasa para prevenir el *overfitting*.

El número de pesos de cada capa convolucional, mostrado en la Figura 5, depende de la siguiente fórmula:

$$P = K^2 \cdot C \cdot N + B$$

[2]Estas decisiones de diseño se han tomado en base a información encontrada en la web (por ejemplo, el recurso [10]) y tras realizar varios experimentos con distintos valores y comparar los resultados.

donde P es el número efectivo de parámetros, K el tamaño del filtro, C el número de canales de la imagen de entrada (que aumenta al avanzar en la red) y N el número de filtros. El resto de parámetros se calcula como en los dos modelos anteriores.

3.4. Red convolucional preentrenada

La técnica consistente en emplear una red neuronal preentrenada se denomina *Transfer Learning* [11]. Este método permite cargar un modelo que ya ha sido entrenado sobre un gran conjunto de datos y usar sus parámetros como punto de partida para continuar entrenando el modelo sobre el conjunto de datos del problema en cuestión. Es posible llevar a cabo esta técnica aunque los conjuntos de datos difieran considerablemente gracias al sistema de aprendizaje de las redes neuronales convolucionales, según el cual las primeras capas de la red extraen características más genéricas aplicables a otros problemas.

Existen varios modelos disponibles *online* para su uso en problemas de clasificación de imágenes, se ha decidido emplear uno de gran popularidad y estructura secuencial de 16 capas llamado VGG16 [12], entrenado en el conjunto de imágenes Imagenet (`http://www.image-net.org/`). Este modelo consta de 5 bloques convolucionales (capas convolucionales y de *pooling*) y un clasificador final (Figura 6).

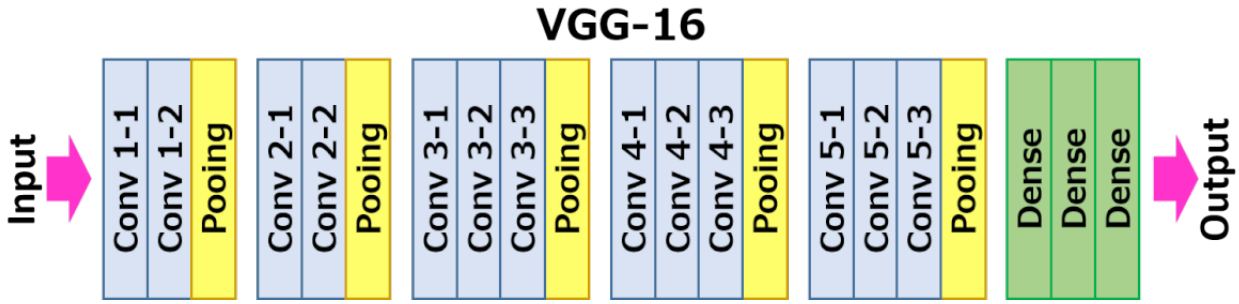


Figura 6: Arquitectura del modelo VGG16.

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
vgg16 (Model)                (None, 3, 3, 512)         14714688
_________________________________________________________________
flatten_3 (Flatten)          (None, 4608)              0
_________________________________________________________________
dense_6 (Dense)              (None, 256)               1179904
_________________________________________________________________
dense_7 (Dense)              (None, 2)                 514
=================================================================
Total params: 15,895,106
Trainable params: 1,180,418
Non-trainable params: 14,714,688
```

Figura 7: Últimas capas del modelo de CNN preentrenada.

El procedimiento consiste en congelar las capas de los bloques convolucionales del modelo VGG16 (no necesariamente todas[3]), de forma que no se entrenen, y añadir un nuevo clasificador (capas totalmente conectadas) al final de la red a entrenar con el conjunto de datos de granos de polen. En la Figura 7 se puede comprobar que los 14714688 parámetros de la red VGG16 no se modifican durante el proceso de entrenamiento, mientras que los parámetros pertenecientes a las dos últimas capas $((4608 \cdot 256+256)+(256 \cdot 2+2) = 1180418)$ sí, para adaptarse al nuevo *dataset*.

De nuevo, la entrada al modelo VGG16 es un tensor de tamaño 100x100x3 y su salida de la forma 3x3x512 se convierte en vector de una dimensión de longitud 4608 a través de una capa *Flatten*. Las capas de convolución pertenecientes a un mismo bloque tienen la misma profundidad (número de filtros), en orden ascendente se corresponden con 64, 128, 256 y 512 (los dos últimos bloques). Cada capa de *max-pooling* reduce a la mitad el número de unidades de entrada, pasando del tamaño de entrada original a las dimensiones 3x3 en el último bloque. Además, los *kernel* empleados en la convolución son de tamaño 3x3 y se emplea *padding* en estas capas para que no se reduzca el tamaño de la entrada tras realizar la convolución. Por último, añadir que la función de activación de las capas intermedias es *ReLU* y de la capa de salida *softmax*, igual que en las configuraciones anteriores.

4. Resultados

Experimentos realizados:

- Monitorización del entrenamiento de los modelos, obteniendo las gráficas de precisión y pérdidas de cada modelo.
- Variación en el tipo de *pooling* empleado en la red neuronal convolucional compleja
- Variación en el número de capas entrenadas del modelo VGG16.
- Comparación de la precisión de validación de los modelos sin y con *data augmentation*.

Tal y como se ha explicado en secciones anteriores, también se han realizado experimentos para decidir los valores de los diferentes hiperparámetros empleados, sin embargo, no se han incluido estos resultados para aliviar la densidad de la memoria y presentar la información esencial que es objetivo de esta práctica.

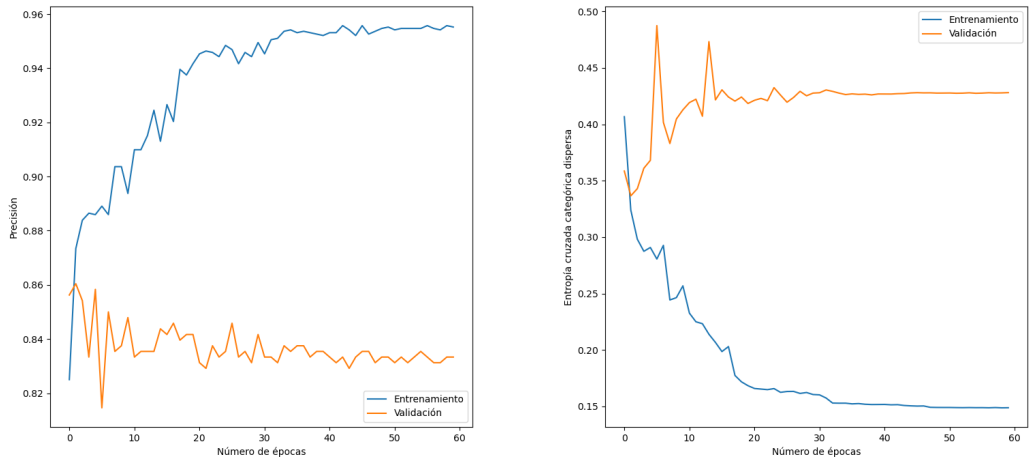
Nomenclatura: únicamente se emplean 5 modelos de redes neuronales, dos de ellos con modificaciones no referentes a la topología, pero por simplicidad se enumeran del 1 al 7 como sigue:

- Modelo 1: Red neuronal sin capa oculta.
- Modelo 2: Red neuronal con una capa oculta.
- Modelo 3: Adición de una capa convolucional al Modelo 2.

[3]Se han realizado experimentos congelando todos los bloques convolucionales (Modelo 5) y empleando los parámetros del último bloque durante el entrenamiento (Modelo 6).

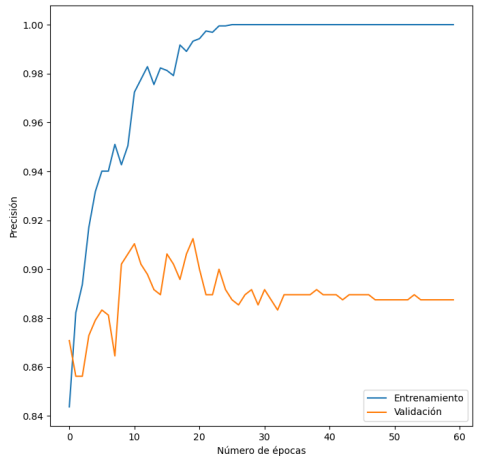
- Modelos de CNN con más de un bloque convolucional:
  - Modelo 4: Versión con *max pooling*
  - Modelo 5: Versión con *average pooling*
- Modelo VGG16:
  - Modelo 6: Versión con todos los pesos conseguidos con Imagenet.
  - Modelo 7: Versión entrenando el último bloque convolucional.

4.1. Rendimiento individual

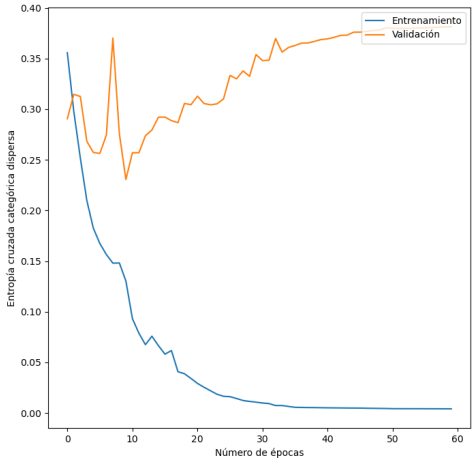


(a) Precisión durante el entrenamiento. (b) Pérdidas durante el entrenamiento.

Figura 8: Modelo 1. Precisión y pérdidas durante el entrenamiento (datos de entrenamiento y validación).

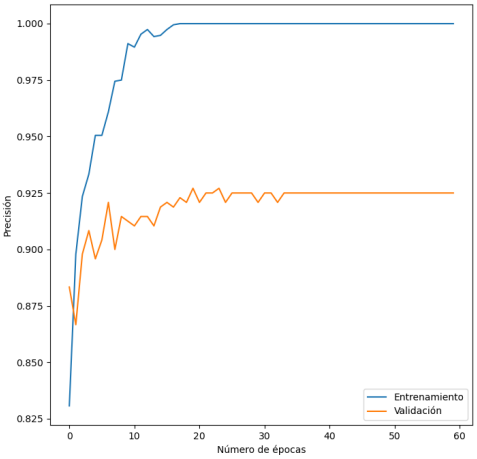


(a) Precisión durante el entrenamiento.

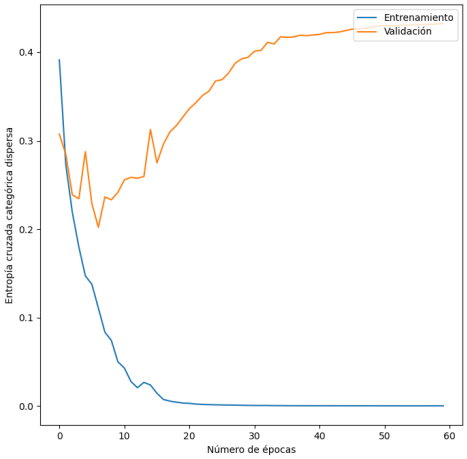


(b) Pérdidas durante el entrenamiento.

Figura 9: Modelo 2. Precisión y pérdidas durante el entrenamiento (datos de entrenamiento y validación).

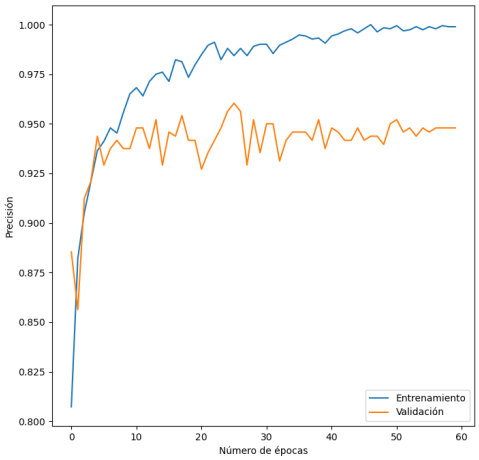


(a) Modelo 3. Precisión categórica.

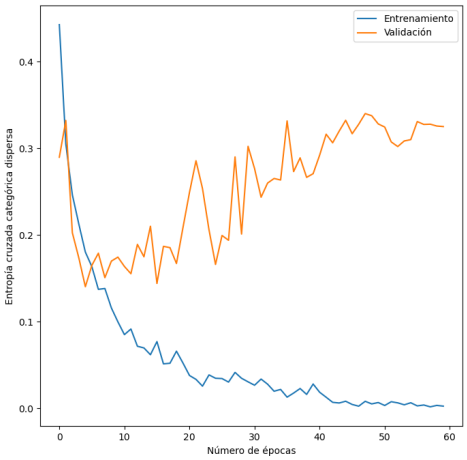


(b) Modelo 3. Entropía cruzada categórica.

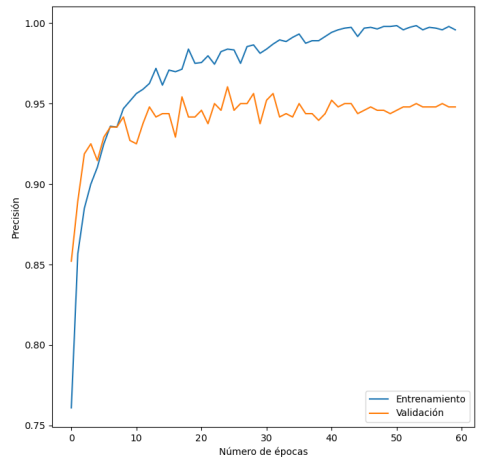
Figura 10: CNN básica: Precisión y pérdidas durante el entrenamiento (datos de entrenamiento y validación).



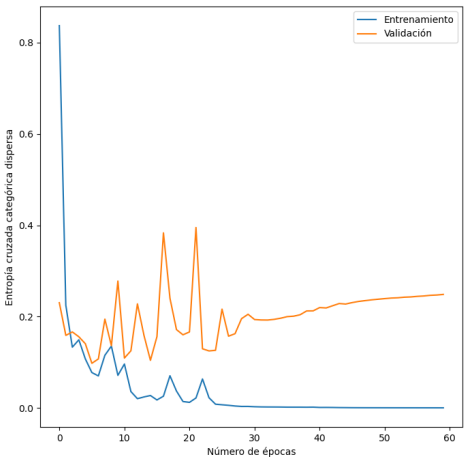
(a) Modelo 4. Precisión categórica.



(b) Modelo 4. Entropía cruzada categórica.

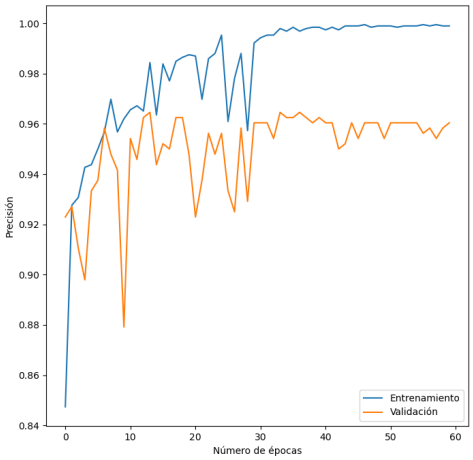


(c) Modelo 5. Precisión categórica.

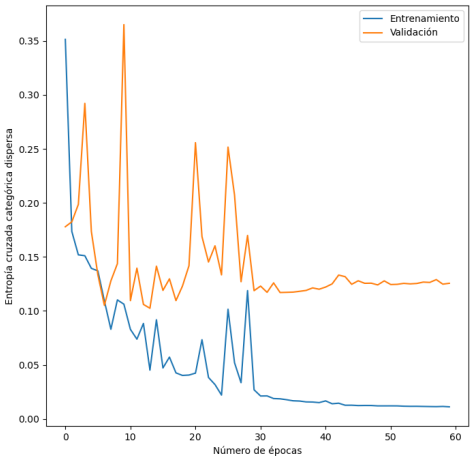


(d) Modelo 5. Entropía cruzada categórica.

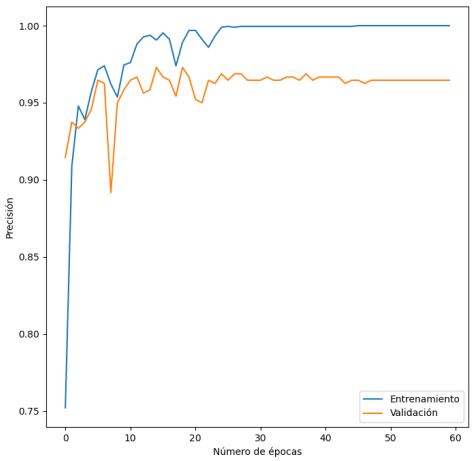
Figura 11: CNNs complejas: Precisión y pérdidas durante el entrenamiento (datos de entrenamiento y validación).



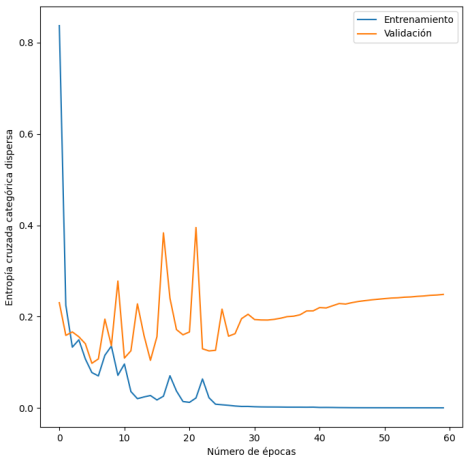
(a) Modelo 6. Precisión categórica.



(b) Modelo 6. Entropía cruzada categórica.



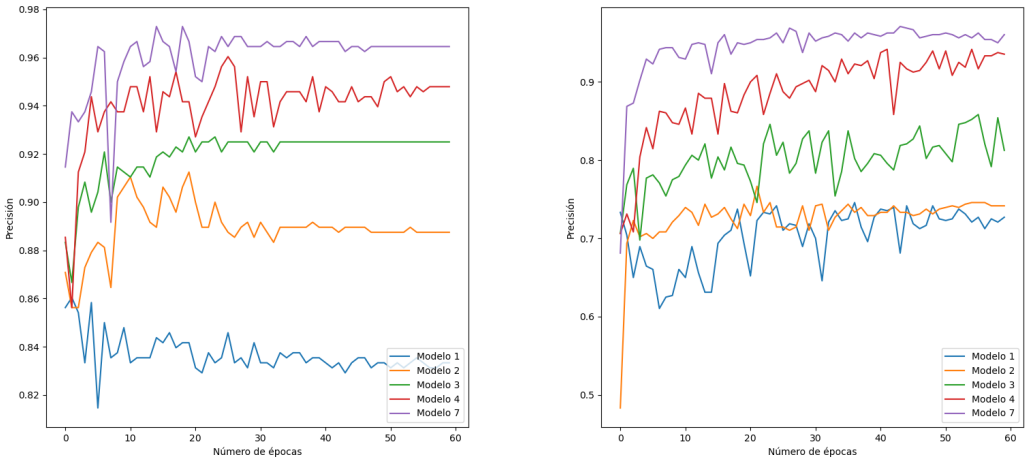
(c) Modelo 7. Precisión categórica.



(d) Modelo 7. Entropía cruzada categórica.

Figura 12: VGG16: Precisión y pérdidas durante el entrenamiento (datos de entrenamiento y validación).

4.2. Comparativa



(a) Sin aumento de datos.

(b) Con aumento de datos.

Figura 13: Precisión sobre el conjunto de validación de todos los modelos.

Red	Número de parámetros	Última precisión	Mejor Precisión
Modelo 1	60002	83.33 %	86.04 %
Modelo 2	360038	88.75 %	91.25 %
Modelo 3	960934	92.5 %	92.71 %
Modelo 4	1281538	94.79 %	96.04 %
Modelo 5	1281538	94.79 %	96.04 %
Modelo 6	15895106	96.04 %	96.46 %
Modelo 7	15895106	96.46 %	97.29 %

Tabla 1: Rendimiento de los modelos.

5. Discusión

A la vista de las Figuras 8 y 9, se puede afirmar que las redes neuronales simples de los dos modelos fallan al generalizar sobre imágenes no empleadas durante el entrenamiento, es decir, se produce una sobre adaptación al conjunto de entrenamiento durante las primeras épocas. Mientras que la precisión de entrenamiento aumenta hasta alcanzar prácticamente el 96 o 100 %, respectivamente, la de validación no supera el 90 % en ambos casos. Este fenómeno también se observa en la red neuronal convolucional del Modelo 3, 10, aunque los resultados sobre las imágenes de prueba son mejores que en las anteriores, aproximadamente del 92 %, demostrando la efectividad de la operación de convolución en problemas de clasificación de imágenes. Como se podía esperar, el rendimiento de la CNN con una topología de red menos básica, alcanza una tasa de aciertos superior a la red convolucional

simple y, como se observa en la Figura 11, las capas de *Dropout* cumplen su función reduciendo el *overfitting*. El desempeño de esta topología con *average* (Figura 11c) o *max pooling* (Figura 11a) es muy similar, sin cambios significativos en la precisión de la clasificación. La red preentrenada consigue una mejor generalización en el conjunto de validación, aunque no ideal, la precisión alcanzada clasificando estas imágenes oscila entre el 96 y 97 %.

A la hora de mostrar los resultados conjuntos en las gráficas de la Figura 13, para mayor claridad, se muestran los diferentes modelos y no las variantes con las que se ha experimentado. Por ello, se muestra la precisión del Modelo 4 con *max pooling* y de la red VGG16 con el último bloque no congelado durante el entrenamiento. La primera diferencia destacable entre las Figuras 13a y 13b, es que claramente la técnica de *data augmentation* no favorece la tarea de clasificación en ninguno de los casos estudiados. Probablemente, esto sea debido a las mínimas diferencias existentes entre ambas especies de granos de polen, de forma que las nuevas imágenes generadas a partir de pequeñas variaciones en las originales introducen ruido en el sistema, impidiendo que se extraigan correctamente las características relevantes para la distinción entre categorías. En esta gráfica también se aprecia como, con generación de nuevos datos, la precisión parte de valores mucho menores y se va estabilizando entorno a tasas de acierto más elevadas. Aún con estas pequeñas diferencias, el comportamiento de los modelos es similar en ambos casos.

En la Tabla 1, además del número de parámetros de cada modelo, se presenta tanto la precisión obtenida realizando la clasificación de las imágenes del conjunto de prueba con el modelo tras completar las 60 épocas de entrenamiento ('Última precisión') como el mejor valor de precisión obtenido sobre este mismo conjunto durante el entrenamiento ('Mejor precisión'). La diferencia entre estos dos valores demuestra que un entrenamiento más largo no garantiza una eficiencia mayor a la hora de aplicar el modelo entrenado a un nuevo conjunto de datos.

El número de parámetros de cada modelo aumenta considerablemente debido a las capas densas o totalmente conectadas, puesto que todas las neuronas de esas capas deben estar conectadas con las de la anterior y la siguiente, convirtiendo el tensor de entrada de 3 dimensiones en uno de una única dimensión de salida. Mientras que las capas convolucionales tienen conexiones locales donde el número de pesos se corresponde con el tamaño de los filtros empleados, tal y como se explica en la Sección 3. Es decir, estas capas están conectadas localmente y comparten parámetros, ya que el mismo filtro es aplicado a diferentes localizaciones de la entrada, produciendo el mapa de características empleado por el modelo para realizar la clasificación.

Cabe mencionar, aunque no se haya incluido entre los resultados presentados, que el tiempo de ejecución del entrenamiento de cada uno de los modelos es mayor cuando el número de parámetros a aprender por la red aumenta. Por tanto, la red VGG16, aunque alcanza la mayor tasa de aciertos en todos los escenarios, también consume más recursos computacionales, incluyendo un aumento en el tiempo de ejecución del proceso de aprendizaje.

6. Conclusiones

Dentro del aprendizaje automático, el aprendizaje profundo permite ampliar el conjunto de soluciones a diferentes problemas que incluyen conjuntos de datos con relaciones complejas, extrayendo características de forma automática de los mismos y empleándolas para realizar, por ejemplo, una clasificación con múltiples categorías, de forma mucho más eficiente. Dentro de este paradigma, las redes neuronales gozan de gran popularidad, gracias a su versatilidad y potencia. En el problema de clasificación estudiado, se han podido comprobar las diferencias entre las redes neuronales básicas y un tipo particular de las mismas conocidas como redes neuronales convolucionales. En base a los resultados de los experimentos realizados, se puede concluir que las redes convolucionales poseen múltiples ventajas a la hora de trabajar con imágenes. El uso de filtros permite aliviar el número de pesos de la red, lo que reduce el efecto de *overfitting* y la complejidad de la red sin perder calidad en los resultados; además de capturar características espaciales de las imágenes de entrada ya que se trabaja en varias dimensiones, teniendo en cuenta la información compartida en pequeños vecindarios de píxeles.

La complejidad de la topología de red no es necesariamente un factor decisivo a la hora de obtener buenos resultados en la clasificación, aunque en este caso de estudio se correspondan las redes más profundas con mejores resultados, también se han añadido características a los modelos más complejos que mejoran su rendimiento (capas de *Dropout*, *learning rate* variable o bloques convolucionales), es decir, añadir más capas ocultas o unidades por capa no implica necesariamente obtener resultados óptimos Las CNNs, aunque más adecuadas que las redes neuronales para imágenes como entrada, no son tampoco la panacea, estando limitadas, por ejemplo, a la detección de objetos sin codificar su posición u orientación dentro de la imagen.

Otro enfoque es el uso del *transfer learning* que aporta numerosas ventajas, utilizando redes ya entrenadas para resolver un problema en concreto. En este caso, los mejores resultados se han obtenido con esta técnica, ya que el modelo VGG16 ha obtenido aproximadamente un 97 % de precisión en el *dataset* empleado. El tiempo de ejecución es menor que el que se emplearía en entrenar todo el modelo de red exclusivamente con las imágenes de los granos de polen y su implementación es sencilla. Además es una muestra de que es posible aplicar el aprendizaje conseguido sobre un dominio específico a otro diferente, ya que las características extraídas en las primeras capas son suficientemente generales. Aunque en este caso ha resultado beneficioso, es posible que la transferencia de conocimiento sea negativa si no existe suficiente relación entre los conjuntos de datos y no existe un método general definitivo de decisión de cuántas capas se deben volver a entrenar para mantener el equilibrio entre buenos resultados y rendimiento computacional no excesivo.

En resumen, el campo del aprendizaje profundo sigue presentando varios desafíos a los investigadores que tratan de resolver los inconvenientes y dificultades de forma que el comportamiento de las redes neuronales se asemeje cada vez más al del cerebro humano, ya que la utilidad de estas técnicas de aprendizaje automático está más que demostrada.

Referencias

[1] Trevor Hastie, Robert Tibshirani, and Jerome Friedman. *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*, volume 27. Math. Intell., 01 2017.

[2] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. `http://www.deeplearningbook.org`.

[3] Shiva Verma. Understanding different Loss Functions for Neural Networks, 2019.

[4] Rochak Agrawal. Optimization Algorithms for Deep Learning, 2019.

[5] Diederik P. Kingma and Jimmy Lei Ba. Adam: A Method for Stochastic Optimization. 2018.

[6] Jason Brownlee. Gentle Introduction to the Adam Optimization Algorithm for Deep Learning, 2019.

[7] Himanshu Sharma. Activation Functions : Sigmoid, ReLU, Leaky ReLU and Softmax basics for Neural Networks and Deep Learning, 2019.

[8] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15(56):1929–1958, 2014.

[9] Rohan Kapur. Rohan #5: What are bias units?, 2016.

[10] Shashank Ramesh. A guide to an efficient way to build neural network architectures-Part II: Hyper-parameter selection and tuning for Convolutional Neural Networks using Hyperas on Fashion-MNIST, 2018.

[11] Jason Brownlee. A Gentle Introduction to Transfer Learning for Deep Learning, 2017.

[12] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv 1409.1556*, 09 2014.