CARRERA DE ESPECIALIZACIÓN EN INTELEGENCIA ARTIFICIAL

MEMORIA DEL TRABAJO FINAL

Exploración de hiperparámetros en modelos para clasificación de imágenes con redes neuronales convolucionales

Autor:
Ing. Nicolás E. Cecchi

Director:
MSc. Javier Kreiner (UBA, Almafintech)

Jurados:
Alejandro Celery (UTN-FRBA)
Rodrigo Cárdenas (ZoomAgri)
Marcos Maillot (UTN - FRGP)

Este trabajo fue realizado en la Ciudad de La Plata,
entre marzo de 2021 y diciembre de 2021.

Resumen

En este trabajo se valida y extiende una heurística para la selección de hiperparámetros en modelos de redes neuronales convolucionales para la clasificación de imágenes. El desarrollo permite a los profesionales ahorrar tiempo y dinero ya que evalúa el potencial desempeño de los modelos en las etapas tempranas del entrenamiento.

Para la implementación se utilizaron conocimientos relacionados al aprendizaje automático para la clasificación de imágenes, principalmente redes neuronales convolucionales. También, resultaron relevantes conocimientos de estadística.

Índice general

Índice de figuras

Índice de tablas

Capítulo 1

Introducción general

En este capítulo se discuten la motivación, alcance y objetivos del presente trabajo. También se realiza una descripción del aprendizaje automático y del aprendizaje profundo; se presentan conceptos claves de estos campos y su relevancia para la sociedad.

1.1. Motivación

El interés en modelos de redes neuronales profundas (*Deep Neural Networks* o DNN) experimenta un rápido crecimiento [1][2] ya que se obtuvieron resultados muy satisfactorios en diferentes ámbitos de aplicación como medicina [3][4][5], vehículos autónomos [6][7], manufactura industrial [8][9], entre otros. Ninguna organización, sea privada o pública, quiere quedarse atrás, lo que genera una gran demanda de profesionales en el sector [10]. Esta dinámica hace que muchos jóvenes elijan seguir carreras relacionadas, que profesores e investigadores dejen la academia y que trabajadores busquen reinventarse profesionalmente con capacitaciones específicas [11]. Por lo que se identifican dos clases de profesionales que ocupan esos cargos: personas con poca o nula experiencia y personas altamente especializadas [12].
A pesar de este éxito, el uso de DNN sigue requiriendo años de experiencia para tomar decisiones de diseño correctas y se basa mucho en el "prueba y error" [13]. Existen técnicas que ayudan a optimizar el diseño, pero son computacionalmente muy costosas [14] o requieren conocimientos avanzados en el tema [15][16].
Por lo descripto anteriormente, este trabajo hace un aporte, a través de evidencia empírica, que facilita la tarea de evaluar diseños de modelos de redes neuronales convolucionales para la clasificación de imágenes con una metodología sencilla y eficiente; siendo así particularmente atractiva para personas que se inician en el campo.

1.2. Objetivos y alcance

El objetivo de este trabajo es colaborar con la comunidad de la inteligencia artificial al aportar conocimientos novedosos para la toma de decisiones en el diseño de modelos para clasificación de imágenes basados en redes neuronales convolucionales.
El trabajo se circunscribe únicamente al caso de redes neuronales convolucionales para clasificación, con énfasis en el learning rate y la arquitectura, que se evalúan utilizando diferentes algoritmos de optimización y diferentes conjuntos de datos.

1.3. Aprendizaje automático

El aprendizaje automático, aprendizaje de máquina o *machine learning* es la disciplina que se encarga de la investigación y el diseño de algoritmos que, de manera automática, buscan ajustar los parámetros internos θ de un modelo para aproximar una función $f^*(x)$, al minimizar una función de costo $J(\theta)$ obedeciendo a un algoritmo de optimización. En la programación clásica, la computadora recibía como entrada los datos y las reglas (un programa) con las cuales procesar esos datos para dar como salida las respuestas. Con el machine learning, las entradas serían los datos y las respuestas esperadas, y el resultado serían las reglas. Estas pueden luego ser aplicadas a nuevos datos para producir respuestas originales [17]. En la figura 1.1 puede verse a modo esquemático la diferencias entre estos dos paradigmas.

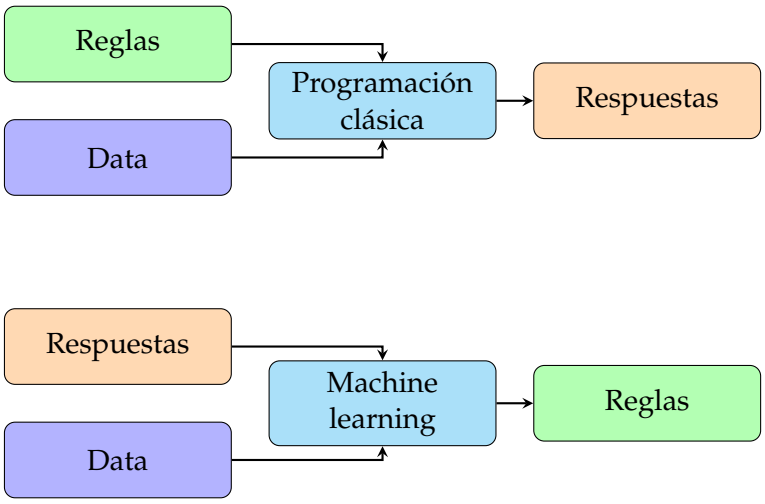


FIGURA 1.1. Comparación entre la programación clásica y el machine learning[1].

De esta manera, el machine learning cambia el paradigma de la computación clásica. Un sistema de aprendizaje automático, en vez de ser programado de manera explícita, es "entrenado". Se le presentan ejemplos relevantes para una tarea y encuentra en ellos una estructura estadística subyacente que le permite automatizar la tarea.

Un problema de machine learning puede definirse de manera precisa con alguna medida de performance P al realizar una tarea T, con algún tipo de experiencia E. Una vez que las tres componentes (T, P, E) están bien especificadas, el problema de aprendizaje se encuentra bien definido [18].

1.3.1. La tarea (T)

La tarea puede definirse como aquella acción que busca automarizarse, delegándola al sistema. Esto, computacionalmente, se reduce al procesamiento que una computadora debe hacer de un vector de datos, o *input*, $x_m \in \mathbb{R}^n$, donde cada $x_{m,i}$ es un valor que describe numéricamente un atributo del ejemplo x_m. Una gran variedad de problemas pueden resolverse con machine learning, algunos de

[1] Imagen reproducida de Deep learning with Python [17].

los más comunes son [19]:

- Clasificación: el sistema debe especificar a cuál de k categorías pertenece el input. Para resolverlo, usualmente se trabaja con algoritmos que producen una función $f : \mathbb{R}^n \to \{1, \ldots, k\}$. Existen otras variantes, como clasificadores que predicen una densidad de probabilidad sobre las clases.
- Regresión: en este tipo de tareas se busca predecir un valor numérico. La función representada por el algoritmo luego del entrenamiento es entonces $f : \mathbb{R}^n \to \mathbb{R}$.
- Transcripción: en tareas de transcripción el sistema recibe como entrada representaciones relativamente desestructuradas de datos y transcribe la información en forma de texto.
- *Machine translation*: el input consiste en una secuencia de símbolos en algún idioma, y el programa debe convertirlo en una secuencia de símbolos en otro idioma.
- Salidas estructuradas: la salida del sistema es multidimensional, donde existe una relación profunda entre sus componentes.
- Detección de anomalías: es un caso particular de clasificación binaria con un gran desbalance de casos positivos, descriptos como anómalos.

En la tabla 1.1 pueden observarse ejemplos de aplicación en cada una de estas tareas.

TABLA 1.1. Ejemplos de tareas que pueden realizar sistemas de aprendizaje automático. [19]

Tarea	**Ejemplos**
Clasificación	Reconocimiento de objetos.
	Análisis de sentimientos.
Regresión	Predicción de temperatura.
	Predicción de precio de un activo financiero.
Transcripción	OCR (*Optical Character Recognition)* .
Machine translation	Traducción entre idiomas.
Salidas estructurales	Segmentación de imágenes a nivel de pixel.
	Descripción de imágenes.
Detección de anomalías	Fraudes financieros.

1.3.2. La experiencia (E)

Los algoritmos de machine learning pueden clasificarse en "no supervisados" y "supervisados", según el tipo de experiencia que se les permite tener en el proceso de aprendizaje [19].
En el aprendizaje supervisado, el objetivo es aprender un mapeo de los datos de entrada o *inputs* x a las salidas o *outputs* y, dado un conjunto etiquetado de pares $\mathcal{D} = \{(x_i, y_i)\}_{i=1}^{N}$. $\mathcal{D}$ se conoce como conjunto de entrenamiento y N es la cantidad de ejemplos que contiene.
En el caso más sencillo, cada input x es un vector D-dimensional, donde cada componente representa un rasgo, atributo o covariable. Sin embargo, x puede ser

un objeto con una estructura compleja como una imagen, el cuerpo de un email, una serie de tiempo, un grafo, una estructura molecular, etc. La variable de salida puede también ser de cualquier tipo: una clase, un número, una imagen, una o varias palabras, etc. La mayoría de los métodos asumen que y_i es categórica, definiendo un problema de clasificación, o numérica, definiendo uno de regresión.
En el aprendizaje no supervisado se cuenta únicamente con inputs $\mathcal{D}\{(x_i)\}_{i=1}^{N}$ y el objetivo es encontrar "patrones interesantes". Este tipo de problemas están definidos de manera menos clara, ya que no está claro desde el inicio qué patrones buscar y qué medida de error utilizar [20].

Algoritmos de optimización y aprendizaje basados en gradientes

Esta experiencia por sí sola no sirve de mucho y debe estar acompañada por un método de aprendizaje. Dado que los modelos de machine learning se implementan en computadoras, este aprendizaje se ejecuta como métodos de optimización numérica [21].
Los problemas de optimización son aquellos en los que se busca hallar un conjunto de parámetros x^* que minimizan o maximizan una función $f(x)$, como se define en la ecuación 1.1. Esta función se llama "función objetivo" o "criterio". Cuando se busca minimizarla, también se habla de "función de costo", "función de pérdida" o "función de error" [19], términos que se utilizan de manera indistinta.

$$x^* = \begin{cases} \arg\max f(x) & \text{para maximización.} \\ \arg\min f(x) & \text{para minimización.} \end{cases} \tag{1.1}$$

Estos valores de x pueden clasificarse según las siguientes definiciones [22]:

Sea $\mathcal{D}$ es dominio de f :
$f(c)$ es un mínimo local de f si $f(c) \leq f(x)$ cuando x está cerca de c.
$f(c)$ es un máximo local de f si $f(c) \geq f(x)$ cuando x está cerca de c.
$f(c)$ es un mínimo global de f si $f(c) \leq f(x), \forall x \in \mathcal{D}$.
$f(c)$ es un máximo global de f si $f(c) \geq f(x), \forall x \in \mathcal{D}$.

Por definición, todo máximo/mínimo global, lo es también local.

En la figura 1.2 puede verse una ilustación que representa cada uno de los casos anteriores en una función de una variable.

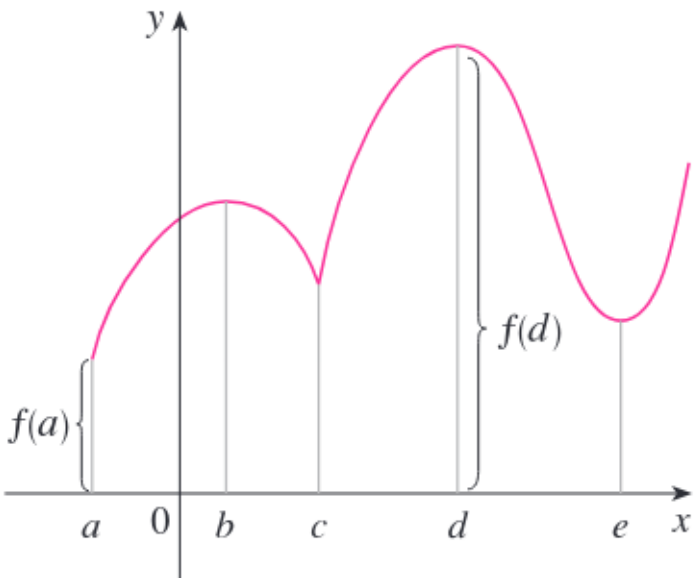


FIGURA 1.2. Gráfica de una función $f(x)$ definida en un intervalo de x que presenta mínimo global en a, máximo global en d, mínimos locales en a, c, e y máximos locales en b, d [2].

La existencia de los valores extremos no siempre está garantizada. El teorema del valor extremo [22] garantiza las condiciones bajo las cuales sí puede afirmarse la existencia.

Recordando que la derivada de una función f en un punto a indica la pendiente de la recta tangente a f en a, la aproximación lineal de f en un entorno pequeño cerca de a se escribe: $f(a+\epsilon) \approx f(a) + \gamma \cdot f'(a)$, cuando $\gamma \to 0$.
La derivada puede utilizarse para optimizar una función, ya que nos indica en qué dirección crece o decrece. En este hecho se basa la técnica de descenso por el gradiente: si se sabe que $f(x - \gamma \cdot \text{sign}(f'(x))) < f(x)$ para ϵ suficientemente pequeño, se puede reducir $f(x)$ al mover x lentamente y de manera iterativa en la dirección opuesta de la derivada. En funciones de varias variables se escribe como la ecuación 1.2 [23].

$$x_{n+1} = x_n - \gamma \cdot \nabla_x f(x) \tag{1.2}$$

[2] Imagen tomada de [22].

El valor γ se conoce como *learning rate* y será tratado en mayor detalle en el capítulo 2. Un ejemplo de optimización aplicando el método del descenso por gradiente puede observarse en la figura 1.3.

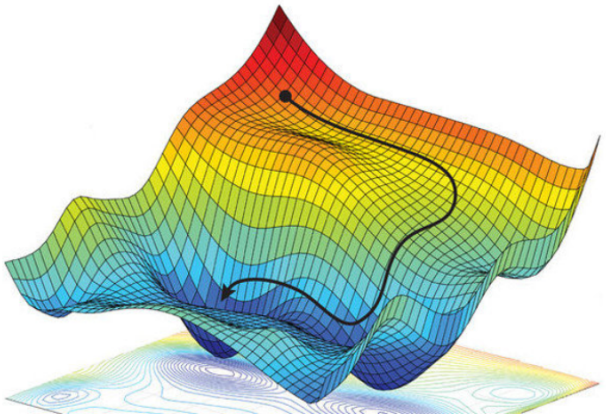


FIGURA 1.3. En la figura se observa la superficie descripta por una función de dos variables y la trayectoria (en negro) indicada por el algoritmo.[3]

Cuando se utilizan métodos basados en gradientes se corre el riesgo de encontrar gradientes nulos o *vanishing gradients*: configuraciones de parámetros donde el gradiente es cero o muy cercano a cero y, por lo tanto, el aprendizaje queda estancado, como se observa en la figura 1.4. Desde la matemática esto se explica por el teorema de Fermat [22] que relaciona los valores extremos con el valor de la derivada (o el gradiente, en funciones de varias variables).

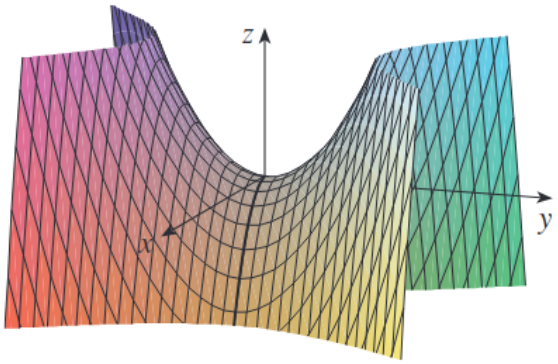


FIGURA 1.4. Gráfica de la función $z = y^2 - x^2$. El gradiente de la función se anula en el origen, en ese punto y sus cercanías el entrenamiento del modelo puede quedar estancado [4].

El algoritmo más utilizado para el cálculo numérico de los gradientes es *backpropagation* o *backprop* [25]. Este algoritmo calcula el gradiente de la función de costo respecto a cada parámetro por la regla de la cadena, iterando hacia atrás para evitar calculos redundantes. Backprop se puede utilizar para cualquier función,

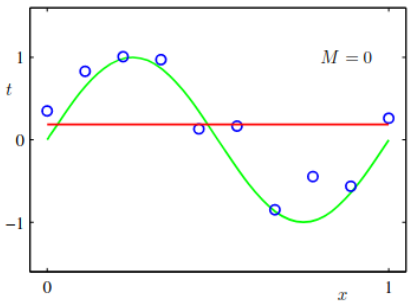
[3] Imagen tomada de [24]

[4] Imagen tomada de [22].

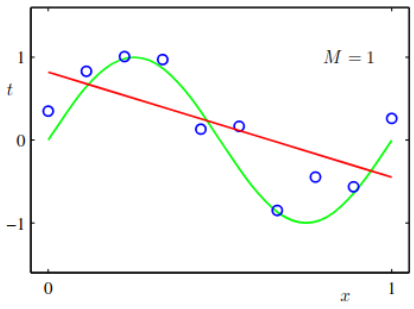
aunque resulta de particular interés en el contexto de las redes neuronales, por lo que se desarrolla el detalle de ese caso particular en la subsección 1.4.3.

1.3.3. La medida de performance (P)

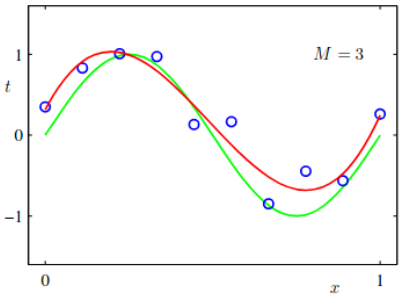
Para evaluar qué tan bien funciona el algoritmo se debe diseñar una medida cuantitativa de su performance. Usualmente, esta medida P es específica de la tarea [19]. Es decir, una medida que se usa en un problema de regresión no puede usarse en una de clasificación ya que la naturaleza de los problemas es diferente.
Al mismo tiempo, dos tareas que son desde un punto de vista computacional lo mismo pueden no serlo considerando el contexto por lo que requieren medidas diferentes. Por ejemplo: en tareas de clasificación, según la cantidad de clases y el balance de ejemplos entre las clases, se prefieren unas u otras métricas.
Dos de los problemas más importantes que ocurren con la medida de performance son el *overfitting* y el *underfitting*. Se conoce como *overfitting* al fenómeno en el cual un modelo ajusta sus parámetros excesivamente a los datos de entrenamiento, perdiendo poder de generalización. El overfitting se puede producir por utilizar un modelo demasiado complejo respecto al proceso subyacente, un excesivo tiempo de entrenamiento, entre otras causas. El *underfitting* es el fenómeno opuesto: el modelo entrenado no logra captar la estructura de los datos, por lo que sus predicciones no son buenas. La figura 1.5 muestra un ejemplo donde se busca ajustar un polinomio a los datos observados. Los polinomios más sencillos no logran modelar correctamente los datos, mientras que aquel excesivamente complejo interpola los datos sin capacidad de generalización. Entre esos dos extremos se produce un buen ajuste.



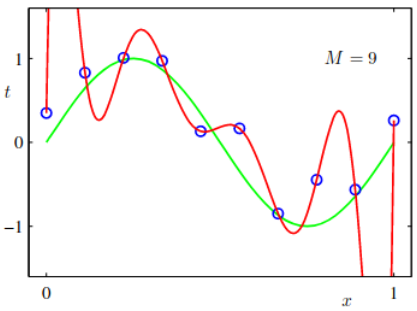
(A) *Underfitting* con polinomio constante.



(B) Caso de *underfitting* con polinomio lineal.



(C) Buen ajuste a los datos con un polinomio de grado 3.



(D) Un polinomio de grado 9 se ajusta a los datos con *overfitting*.

FIGURA 1.5. Ejemplos de ajustes polinómicos (curvas rojas) sobre las muestras (puntos azules) tomadas del proceso subyacente (curva verde, $f(x) = \sin(2\pi x)$) [5].

Sus principales causas son opuestas al caso anterior: modelo demasiado sencillo, poco entrenamiento, entre otras.
Es posible realizar un monitoreo durante el entrenamiento para evitar estos inconvenientes. El seguimiento de la función de pérdida evaluada sobre los conjuntos de entrenamiento y de validación permite diagnosticarlos.
Como se observa en la figura 1.6, ambas curvas suelen comenzar decreciendo rápidamente, indicador de una región de underfitting. Con las iteraciones, el modelo mejora y esas pendientes disminuyen: mejorar el desempeño se vuelve computacionalmente más costoso. Si el entrenamiento continúa lo suficiente, la pérdida de validación tendrá un punto de inflexión a partir del cual comienza a aumentar, mientras que la de entrenamiento sigue decreciendo lentamente. Desde ese punto en adelante el modelo entra en régimen de overfitting: los parámetros están tan adaptados a los datos de entrenamiento que pierden poder de generalización sobre los datos de validación.

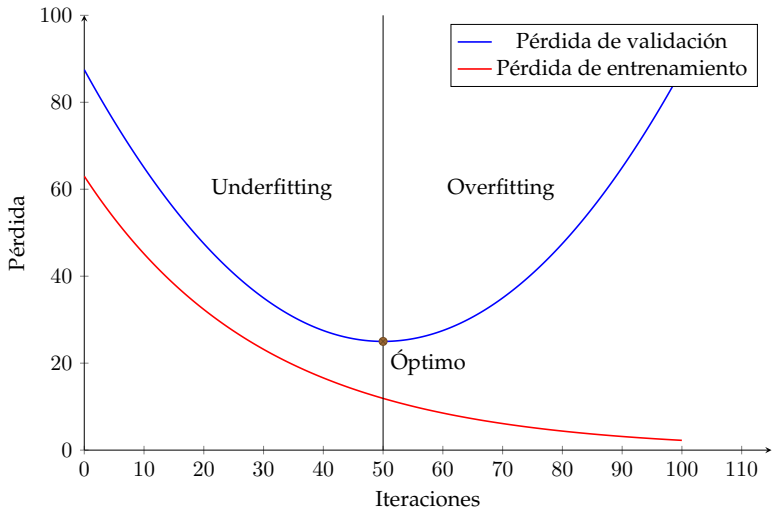


FIGURA 1.6. Gráficas de pérdida de entrenamiento y validación con el avance de las iteraciones de entrenamiento. Se identifican zonas de underfitting, overfitting y el punto óptimo teórico.

Otra forma de evitarlos es separar el conjunto de datos en tres partes: conjunto de entrenamiento, de validación y de prueba o *testing*. El conjunto de entrenamiento es con el cual se entrena el modelo para ajustar sus parámetros. Finalizado el entrenamiento, se evalúa la métrica sobre el conjunto de validación, para luego ajustar hiperparámetros del modelo de manera iterativa, con el objetivo de obtener mejores resultados. Finalizado el ciclo de optimización de hiperparámetros se procede a la evaluación sobre el conjunto de testing, al que nunca estuvo expuesto el modelo, para tener una referencia del desempeño que tendría en producción.

[5]Imagen tomada de [26].

1.4. Redes neuronales profundas

Las redes neuronales artificiales (ANN, por sus siglas en inglés) son una familia de modelos de aprendizaje automático. Las más sencillas son llamadas *feedforward* porque la información fluye únicamente desde la entrada hacia la salida sin ningún lazo de retroalimentación.
El nombre de "red" surge por el hecho de que se las puede representar a través de un grafo acíclico dirigido, como puede observarse en la figura 1.7. Cada "columna" de nodos se conoce como una "capa" de la red, cuando se tienen múltiples capas se habla de una red "profunda" o "*Deep Neural Network*". La capa de entrada se conoce como *input layer* y la de salida como *output layer*, las capas intermedias se llaman "capas ocultas" o "*hidden layers*".

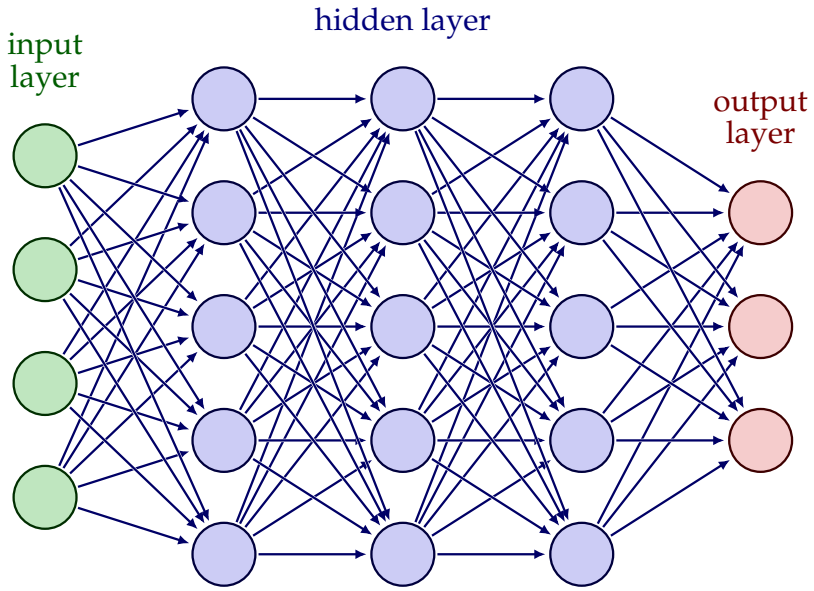


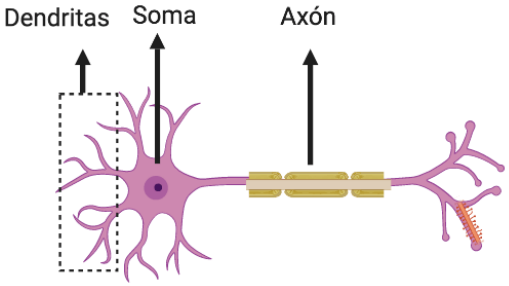
FIGURA 1.7. Esquema general de una red neuronal *fully connected* (FC), donde cada nodo se conecta con todos los nodos de la capa anterior.

El término "neuronal" se emplea por el hecho de que algunas de sus propiedades están directamente inspiradas en el funcionamiento de los cerebros animales. Sin embargo, se recomienda pensar a las redes neuronales como máquinas aproximadoras de funciones diseñadas para lograr generalizaciones estadísticas, tomando ocasionalmente ideas del funcionamiento cerebral, que como modelos del cerebro [19] [27] [28] [29].

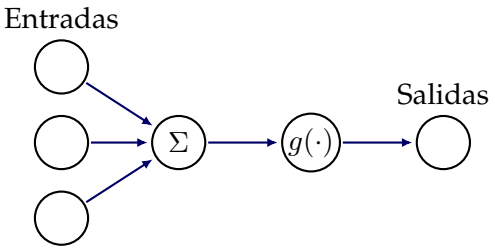
1.4.1. Perceptrón

El bloque fundamental que compone una red neuronal es el perceptrón, también llamado "unidad", "neurona" o "nodo". El perceptrón es un modelo matemático, propuesto por F. Rosenblatt [27], inspirado en el funcionamiento de una neurona biológica. Mientras que en estas últimas las dendritas reciben señales eléctricas de los axones de otras neuronas, el perceptrón es "estimulado" por las salidas de las capas anteriores. Al producirse la sinapsis, las señales son moduladas, concepto que aparece en el perceptrón al aplicar una transformación lineal a su entrada. Finalmente, una neurona biológica emite una señal de salida solamente cuando el estímulo de entrada supera un cierto umbral, mientras que el perceptrón tiene

una "función de activación" que regula la intensidad de la salida (posiblemente nula).
En la figura 1.8a puede verse la representación gráfica de una neurona biológica, y en la figura 1.8b un perceptrón con tres entradas.



(A) Representación de una neurona biológica [6].



(B) Representación de un perceptrón. Σ representa la combinación lineal que se hace de las entradas y $g(\cdot)$ la transformación no lineal.

FIGURA 1.8. Comparación entre los esquemas de una neurona biológica y un perceptrón.

Matemáticamente, el perceptrón toma un vector de entrada al que le aplica una transformación lineal seguida de una no lineal, como se ve en las ecuaciones 1.3 y 1.4, respectivamente.

$$z = w^T x + b \tag{1.3}$$
$$h = g(z) \tag{1.4}$$

Los parámetros w (pesos) y b (términos independientes o *biases*) de la transformación lineal son los que se ajustan durante el entrenamiento. La función de activación g es una decisión de diseño.

1.4.2. Forward pass

Se conoce como *forward pass* al proceso predictivo de la red, cuando la información fluye desde la entrada hacia la salida. La forma general de la salida de la n-ésima capa obedece a la ecuación 1.5, donde $h^{(n)}$ es la salida, $g^{(n)}$ su función de activación, $W^{(n)}$ la matriz con los vectores w aprendidos por los perceptrones de la capa y $b^{(n)}$ el vector con sus biases. Se considera $h^{(0)} = x$, el vector de entrada.

$$h^{(n)} = g^{(n)}(W^{(n)T} h^{(n-1)} + b^{(n)}) \tag{1.5}$$

Interpretando la sucesión de capas como una composición de funciones, la salida de una red de N capas es entonces una composición $h^{(N)}(h^{(N-1)}(\cdots(h^{(1)}(h^{(0)})))\cdots))$. El algoritmo 1 describe el proceso.

[6] Imagen tomadas de https://cs.stanford.edu/people/eroberts/courses/soco/projects/neural-networks/Neuron/index.html

Algoritmo 1 Algoritmo de propagación de la información en una DNN con l capas. Luego de calcular la predicción $\hat{y}$, durante la fase de entrenamiento, calcula el costo total J como la suma de la pérdida de predicción y un término de regularización.

Require: Profundidad, l
Require: $W^{(i)}, i \in 1, \cdots, l$, matrices de pesos del modelo.
Require: $b^{(i)}, i \in 1, \cdots, l$, matrices de bias del modelo.
Require: x, la entrada.
Require: y, target.
$h^{(0)} \leftarrow x$
for $k \leftarrow 1, \cdots, l$ **do**
$a^{(k)} \leftarrow b^{(k)} + W^{(k)} h^{(k-1)}$
$h^{(k)} \leftarrow g(a^{(k)})$
end for
$\hat{y} \leftarrow h^{(l)}$
$J \leftarrow L(\hat{y}, y) + \lambda \Omega(w)$ // Si es fase de entrenamiento.

1.4.3. Backpropagation

El algoritmo de backpropagation calcula el gradiente de la función de costo respecto a los parámetros entrenables, de modo que esté disponible para los cálculos de optimización (Ver sección 1.3.2).
Durante la fase de entrenamiento, luego de cada forwardpass se tiene el vector de pérdida J calculado sobre un subconjunto de los datos de entrenamiento (potencialmente todos). El algoritmo 2 muestra la propagación hacia atrás de la información para el cálculo de los gradientes.

Algoritmo 2 Algoritmo de backpropagation que calcula los gradientes por regla de la cadena.

$g \leftarrow \nabla_{\hat{y}} J = \nabla_{\hat{y}} L(\hat{y}, y)$ // Cálculo del gradiente en la capa de salida.
for $k = l, l-1, \cdots, 1$ **do**
$g \leftarrow \nabla_{a^{(k)}} J = g \odot f'(a^{(k)})$ // Actualiza gradiente g.
$\nabla_{b^{(k)}} J = g + \lambda \nabla_{\boldsymbol{b}^{(k)}} \Omega(w)$ // Cálculo de gradiente respecto a biases b.
$\nabla_{W^{(k)}} J = g h^{(k-1)} + \lambda \nabla_{W^{(k)}} \Omega(w)$ // Cálculo de gradiente respecto a pesos w.
$g \leftarrow \nabla_{h^{(k-1)}} J = W^{(k)T} g$ // Propagación de gradiente.
end for

1.4.4. Hiperparámetros y diseño

Una de las principales diferencias con el aprendizaje automático tradicional es la cantidad de parámetros entrenables. Un modelo de deep learning puede tener millones o miles de millones de parámetros. Esto genera un alto costo computacional al momento de entrenarlo y genera dependencias complejas de la función de costo, con muchos mínimos locales y puntos sillas que dificultan el entrenamiento.
Otra diferencia importante es la cantidad de hiperparámetros con los que se cuenta para ajustar el modelo. Un hiperparámetro es una variable ajustable al momento de diseñar un modelo, pero que es fija a lo largo de todo el entrenamiento. En este trabajo se habla del "diseño de una red" para referirse al proceso de selección

de estos hiperparámetros.
En la tabla 1.2 se resumen algunos de los hiperparámetros que se utilizan con mayor frecuencia. Existen otros, que se llamarán "de segundo orden" ya que dependen de las decisiones adoptadas en estos primeros, por ejemplo, la parametrización de un algoritmo de optimización depende del algoritmo seleccionado.

TABLA 1.2. Ejemplos de hiperparámetros ajustables en un modelo de deep learning.

Hiperparámetro	**Descripción**
Arquitectura	Tipo y cantidad de capas. Cantidad de neuronas por capa.
Función de activación	No linealidad aplicada en los nodos.
Optimizador	Algoritmo de optimización. Parámetros del algoritmo (ver sección 2.2).
Learning rate	Tipo de política. Parámetros de la política.
Regularización de pérdida	Regularización de la función de pérdida.
Función de pérdida	Es la función que se busca minimizar.
Inicializador de parámetros	Método para inicializar parámetros entrenables.

1.5. Estado del arte

La optimización de hiperparámetros o *hyperparameter tuning* es el proceso de búsqueda de configuraciones de hiperparámetros que permitan al modelo entrenado alcanzar un desempeño superior a un valor base inicial. A grandes rasgos, estas técnicas pueden dividirse en manuales y automáticas, siendo estas últimas de "fuerza bruta" o basadas en métodos más sofisticados. Si bien existen trabajos que se enfocan exclusivamente en estos temas, es también habitual que los avances se den por innovaciones en trabajos que se enfocaban en resolver alguna tarea. A continuación se presenta el estado del arte para arquitectura y learning rate, este último incluye los avances relacionados a los algoritmos de optimización.

1.5.1. Arquitectura

La optimización manual de arquitecturas de redes neuronales en un caso de aplicación concreto no tiene hoy tanta relevancia si se lo compara con los métodos automáticos. Los diseños manuales suelen estar relacionados a proyectos que buscan innovar con nuevas estructuras superiores a las conocidas actualmente. Esto se debe, en parte, a la gran cantidad de literatura validando ciertos paradigmas, como puede ser la utilización de redes convolucionales para trabajos con imágenes. Smith y Topin [30] realizaron un relevamiento de publicaciones e identificaron 14 patrones de diseño en redes neuronales convolucionales.

En cuanto a la búsqueda automática, Franklin y Carbin [31] presentan la "hipótesis del ticket de lotería" donde describen un proceso iterativo de *pruning* ("poda" de una red neuronal, para reducir su tamaño) donde logran alcanzar una

performance muy similar a la red original, con un costo computacional significativamente menor. Debe destacarse que en este caso sigue siendo necesario que la primera iteración haya sido diseñada por una persona, y el método encuentra "subredes" con un poder expresivo equivalente. Elksen, Metzen y Hutter [32] investigan el campo del NAS (*Neural Architecture Search*), el subcampo de AutoML enfocado en la arquitectura de redes. Aún así, sigue teniendo un grado importante de sesgo humano ya que deben definirse el espacio de modelos en que buscar, la estrategia de búsqueda y la estrategia de estimación de desempeño.
Finalmente, Baldominos, Saez e Isasi [33] presentan de manera exhaustiva el diseño evolutivo, con énfasis en la neuroevolución de redes convolucionales. En este campo se aplican conceptos de computación genética y evolutiva al diseño de una red. Se presentan casos de aplicación en evolución de número de capas, tipos de capas (convolucionales, fully connected y recurrentes), funciones de activación, entre otros.

1.5.2. Learning rate y optimización

Para la selección del learning rate existen muchos esquemas. De "*Neural networks: tricks of the trade*" [34] se concluye que un learning rate dinámico es beneficioso. Se describen esquemas en que el learning rate se achica cuando el vector de pesos oscila y se agranda cuando sigue una dirección estable. Otra posibilidad es que cada parámetro tenga su propio learning rate y utilizar segundas derivadas.
Y. Bengio publicó recomendaciones para optimización basada en gradientes [35]: dadas entradas normalizadas, valores iniciales entre 0 y 10^{-6}, aunque destaca que dependerá altamente de la parametrización del modelo. Resalta, asímismo, la importancia de una estrategia de adaptación (*learning rate schedule*).
Otra estrategia que cobró notoriedad es el uso de learning rates adaptativos, como en AdaDelta [36] o Adam [37]. Yedida, Saha y Prashanth [38] muestran que la inversa de la constante de Lipschitz de la función de pérdida como learning rate logra superar los anteriores métodos.
Por su parte, Smith [39] [13] [40] discute de manera extensa la aplicación de learning rate cíclicos con diferentes variantes. Wen, Li y Gao [41] proponen un learning rate scheduler basado en un agente de *reinforcement learning*.

Capítulo 2

Introducción específica

En este capítulo se presentan las herramientas y los métodos utilizados durante la realización del trabajo, justificando su elección. Se explican en profundidad las redes neuronales convolucionales, algoritmos de optimización y se presenta el *learning rate range test*. Se introducen también las funciones de inicialización de parámetros, de activación y de pérdida. Finalmente, se describen los conjuntos de datos y la plataforma de trabajo.

2.1. Redes neuronales convolucionales

Las redes neuronales convolucionales (*Convolutional Neural Networks*, CNN) son una clase de redes neuronales que funcionan excepcionalmente bien procesando imágenes [42]. Aprovechando ciertas propiedades de la distribución de la información en imágenes las CNN requieren menos conexiones y parámetros, por lo que son más fáciles de entrenar que redes *fully connected* (FC) de profundidades similares [43]. Las CNN organizan sus nodos en tres dimensiones: largo, alto y profundidad, esta última refiriendo a los canales de la imagen y no de la red.
Los tres tipos de capas que se utilizan principalmente en CNN son: convolucional, *pooling* y fully connected.
Las capas convolucionales tienen filtros o *kernels* con los cuales se computa la operación de convolución. Cumplen el mismo rol que los filtros en visión por computadora tradicional con la diferencia de que, en lugar de ser programados, son aprendidos durante el entrenamiento. El kernel tiene una geometría tridimensional con dos grados de libertad que son el alto F_h y el ancho F_w, a definir por el programador, mientras que su profundidad queda determinada por la cantidad de canales que tiene su entrada. Lo más habitual es que alto y ancho coincidan en un valor F impar, ya que permite identificar un centro. El filtro se desplaza sobre toda la entrada a razón de (s_h, s_v) donde s_h es el paso o *stride* horizontal y s_v el vertical, la práctica habitual es que tomen el mismo valor s.
La operación de convolución consiste en el desplazamiento del filtro sobre la entrada computando el producto elemento a elemento (conocido como "de Hadamard") y sumando todos esos valores. De manera más sencilla, el filtro "selecciona" un volumen de la entrada y calcula la suma ponderada con los valores aprendidos. Un ejemplo de convolución se representa en la figura 2.1.

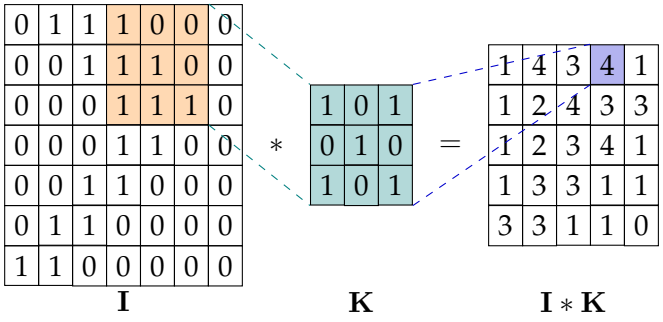


FIGURA 2.1. Convolución de un kernel **K** de 3x3 sobre una matriz **I** de 7x7.

Como se observa en la figura anterior, la salida resultante tiene sus filas y columnas reducidas respecto de la original. Este efecto se debe a que el kernel nunca tiene su centro en los pixeles límite de la imagen (ya que no tendría valores sobre los que computar el producto) y puede verse magnificado en el caso de que el paso definido sea diferente de $(1, 1)$. Este último efecto no puede contrarrestarse, pero para solucionar el primero se desarrolló el *padding*, que consiste en agregar $2P$ filas y $2P$ columnas de valores auxiliares para que salida preserve las dimensiones originales.
En resumen, una capa convolucional recibe un volumen de entrada $W_i \times H_i \times D_i$ y produce uno de salida $H_o \times W_o \times D_o$. Los hiperparámetros que afectan las dimensiones son: cantidad de filtros K, dimensiones del campo de recepción (F_h, F_w), paso s y padding P. Las dimensiones de salida se calculan según las siguientes expresiones:

- Alto de salida: $H_o = \frac{H_i - F_h + 2P}{s} + 1$
- Ancho de salida: $W_o = \frac{W_i - F_w + 2P}{s} + 1$
- Canales de salida: $D_o = K$

La figura 2.2 muestra un ejemplo de convolución con dos filtros de $(3, 3)$, paso $(1, 1)$ y sin padding, aplicado sobre una imagen RGB.

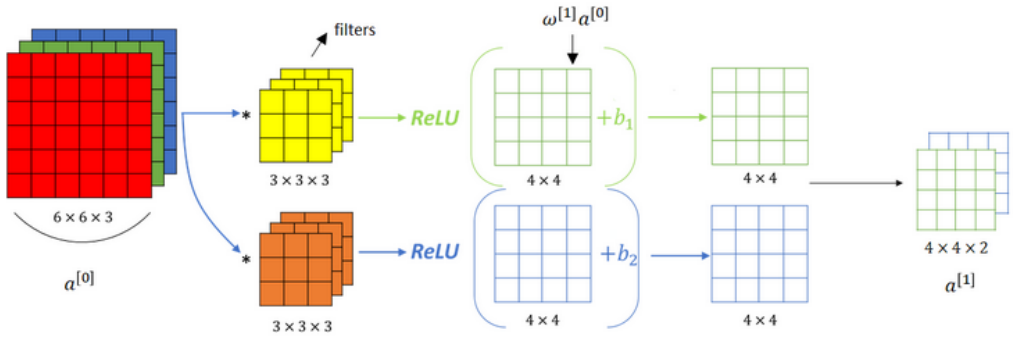


FIGURA 2.2. Convolución con dos filtros sobre una imagen de tres canales. El volumen de entrada $6 \times 6 \times 3$ se reduce a $4 \times 4 \times 2$ [1].

[1]Imagen tomada de https://datahacker.rs/.

Es habitual que las capas de convolución sean seguidas por capas de pooling. Su función es reducir progresivamente las dimensiones para reducir la cantidad de parámetros y cálculos, lo que ayuda a controlar el overfitting. La capa de pooling opera independientemente en cada canal de profundidad aplicando una función de agregación como máximo, media, etc. sobre el volumen de la imagen de entrada que selecciona su filtro. Lo más común es utilizar filtros de (2,2) y $s = 2$, lo que reduce filas y columnas a la mitad, descartando el 75 % de las activaciones.

Finalmente, las capas fully connected no tienen diferencia respecto a lo presentado en el capítulo 1. Estas capas se utilizan como capas terminales de la red y son las que terminan realizando la predicción. Suelen implementarse en grupo y su entrada es el vector que resulta de la reconfiguración de la salida de la última capa de pooling con un volumen $m \times n \times d$ en $\mathbb{R}^{(m \cdot n \cdot d) \times 1}$.

2.2. Algoritmos de optimización

En machine learning se busca minimizar la función de pérdida $J(\theta, x)$, donde θ son los parámetros y x los datos. Sea $\mathcal{D} = \{x_1, x_2, \cdots, x_n\}$ el conjunto de datos, $J(\theta, \mathcal{D})$ denota la función de pérdida J con los parámetros θ evaluada sobre el conjunto de datos $\mathcal{D}$ y $\nabla_\theta J(\theta, \mathcal{D})$ su gradiente respecto a θ calculado sobre el conjunto $\mathcal{D}$. Queda de este modo definida la fórmula 2.1 mediante la cual son actualizados los parámetros de la función en el proceso de aprendizaje. Esta modalidad que utiliza en cada iteración de actualización de parámetros la totalidad de los datos disponibles se conoce como *batch gradient descent* [44] y opera según el algoritmo 3.

$$\theta_{t+1} = \theta_t - \gamma \cdot \nabla_\theta J(\theta, \mathcal{D}) \tag{2.1}$$

Algoritmo 3 Algoritmo batch gradient descent con learning rate constante.

Require: γ : Stepsize/Learning rate.
Require: $J(\theta, x)$: Función objetivo.
Require: θ_0 : Vector de parámetros inicial.
  $t \leftarrow 0$ Inicializa paso temporal.
  while θ not converged **do**
    $t \leftarrow t + 1$
    $g_t \leftarrow \nabla_\theta J_t(\theta_{(t-1)}$ (Calcula gradientes respecto al objetivo en el momento t.)
    $\theta_t \leftarrow \theta_{t-1} - \gamma \cdot g_t$ (Actualizar parámetros.)
  end while
  return θ_t (Parámetros optimizados.)

2.2.1. Stochastic y minibatch stochastic gradient descent

La primera variación sobre batch gradient descent es *stochastic gradient descent* (SGD) [45]. En cada iteración, en lugar de calcular el gradiente sobre todo el conjunto de datos, toma aleatoriamente un único elemento $x_i \in \mathcal{D}$, como en la ecuación 2.2.

$$\theta_{t+1} = \theta_t - \gamma \cdot \nabla_\theta J(\theta, x_i) \quad x_i \in \mathcal{D} \tag{2.2}$$

El principal beneficio es la gran reducción en el uso de memoria que se requiere para calcular el gradiente en cada iteración. Como contrapartida, el gradiente calculado es inexacto, ya que se utiliza un único ejemplo en la operación [46]. Para mitigar este efecto indeseado se diseñó *minibatch stochastic gradient descent*, donde se seleccionan aleatoriamente un conjunto $B \subset \mathcal{D}$ de elementos, como lo define la ecuación 2.3. De esta manera, se observan todos los datos cada $\frac{|\mathcal{D}|}{|B|}$ iteraciones.

$$\theta_{t+1} = \theta_t - \gamma \cdot \nabla_\theta J(\theta, B) \quad B \subset \mathcal{D} \tag{2.3}$$

2.2.2. Adam

Adam es un algoritmo para optimización basada en gradientes de primer orden, que se construye con estimaciones adaptativas de momentos de bajo orden. Su implementación es sencilla, computacionalmente eficiente, tiene pocos requerimientos de memoria y es apropiado para problemas con grandes volúmenes de datos o parámetros a optimizar. También se adapta a problemas con objetivos no estacionarios y problemas con gradientes muy ruidosos o poco densos. [37]
Formalmente, se define $f(\theta)$ como una función diferenciable respecto a los parámetros θ. El objetivo es minimizar la esperanza $\mathbb{E}[f(\theta)]$ respecto a sus parámetros. Las funciones $f_1(\theta), \ldots, f_T(\theta)$ denotan realizaciones de f al tiempo t , y $g_t = \nabla_\theta f_t(\theta)$ su gradiente, es decir, el vector de derivadas partciales de f_t respecto a θ en el instante t.
El algoritmo actualiza el gradiente (m_t) y el cuadrado del gradiente (v_t) mediante medias móviles exponenciales, los parámetros $\beta_1, \beta_2 \in [0, 1)$ controlan los ratios de decaimiento exponencial. Estas medias estiman la media y la varianza no centrada del gradiente. Dado que ambos vectores se inicializan con todos sus valores en 0, ambos estimadores están sesgados hacia este valor, principalmente al comienzo del entrenamiento o en casos de parámetros β muy cercanos a 1. Los autores lo tuvieron en cuenta y aportan las correcciones necesarias.
Este proceso queda resumido en el algoritmo 4; en la tabla 2.1 pueden consultarse los parámetros por defecto que trae su implementación en Keras [47].

Algoritmo 4 Algoritmo Adam. g_t^2 indica el cuadrado elemento a elemento $g_t \odot g_t$.

Require: α : $Stepsize$.
Require: $\beta_1, \beta_2 \in [0, 1)$: Ratios de decaimiento exponencial.
Require: $f(\theta)$: Función objetivo con parámetros θ.
Require: θ_0 : Vector de parámetros inicial.
$m_0 \leftarrow 0$ Inicializa vector de primer momento.
$v_0 \leftarrow 0$ Inicializa vector de segundo momento.
$t \leftarrow 0$ Inicializa paso temporal.
while θ not converged **do**
$t \leftarrow t + 1$
$g_t \leftarrow \nabla_\theta f_t(\theta_{(t-1)})$ (Calcula gradientes respecto al objetivo en el momento t.)
$m_t \leftarrow \beta_1 \cdot m_{t-1} + (1 - \beta_1) \cdot g_t$ (Actualiza estimador sesgado de m_t.)
$v_t \leftarrow \beta_2 \cdot v_{t-1} + (1 - \beta_2) \cdot g_t^2$ (Actualiza estimador sesgado de v_t.)
$\hat{m}_t \leftarrow m_t / (1 - \beta_1^2)$ (Calcula estimación del primer momento insesgado.)
$\hat{v}_t \leftarrow v_t / (1 - \beta_2^2)$ (Calcula estimación del segundo momento insesgado.)
$\theta_t \leftarrow \theta_{t-1} - \alpha \cdot \hat{m}_t / (\sqrt{\hat{v}_2} + \epsilon)$ (Actualiza parámetros.)
end while
return θ_t (Parámetros resultantes.)

TABLA 2.1. Parámetros por defecto en la implementación de Adam que se encuentra en la librería Keras. Los parámetros son nombrados de acuerdo a esa implementación.

Argumento	Valor por defecto	Notas
learning_rate	0.001	Puede tomar un valor fijo o un objeto que contenga su política.
beta_1	0.9	Corresponde al parámetro β_1 en el algoritmo 4
beta_2	0.999	Corresponde al parámetro β_2 en el algoritmo 4
epsilon	1e-07	Se trata realmente de $\hat{\epsilon}$ del paper original.
amsgrad	False	Variable booleana para activar la variante AMSGrad del algoritmo.
name	Adam	Nombre del objeto.

2.3. Política de learning rate

La política de cyclical learning rate hace variar al learning rate entre un valor máximo y uno mínimo que pueden ser constantes o modificarse al correr las iteraciones. La variación de estos valores puede ser lineal, sinusoidal, parabólica, entre otras [39]. Un manera intuitiva de entender el por qué de la efectividad de este método es tener presente la topología de la función de pérdida: el efecto de los puntos silla supone ser superior al de los mínimos locales [48]. Por esto, incrementar el learning rate en la zona de un punto silla permitiría escapar de la región y continuar la optimización de los parámetros.
Una recomendación es utilizar límites constantes con variación lineal entre ellos [39], debido a que se trata de la configuración más sencilla y en los ensayos no se encontraron diferencias significativas con otros métodos más complejos. Otra política recomendada en la misma fuente es la "triangular2", que mantiene el límite

inferior constante, pero divide a la mitad el superior en cada ciclo, como se observa en la figura 2.3. Para encontrar valores "razonables" de los límites del learning rate propone el *learning rate range test* (LRRT), que consiste en un entrenamiento de unas pocas epochs (seis a ocho) dejando que el learning rate crezca linealmente entre un mínimo y un máximo. Luego, al graficar la métrica en función del valor del learning rate se puede observar un valor a partir del cual comienza a mejorar y otro donde empieza a desacelerarse. Esos dos valores son buenas elecciones para los límites. Otra alternativa sugerida es considerar que el learning rate óptimo está dentro de un factor de 2 del valor más grande alcanzado manteniendo régimen de convergencia y utilizar $lr_{min} = 1/3 \cdot lr_{max}$ o $lr_{min} = 1/4 \cdot lr_{max}$ [39]. Valores recomendados para la longitud del ciclo o *stepsize* son entre dos y diez veces el número de iteraciones que entran en una epoch, donde una iteración se define como una actualización de los parámetros.

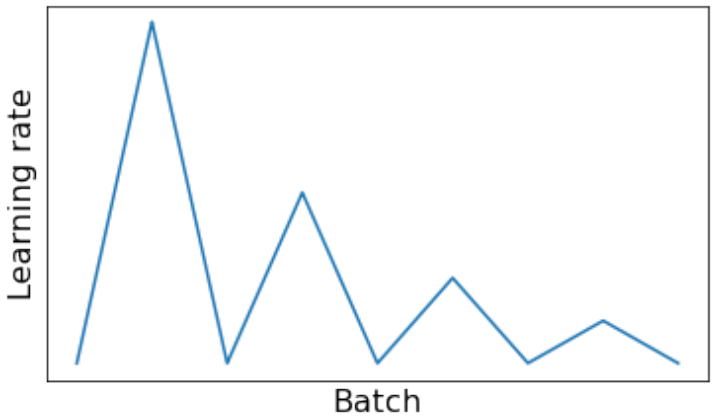


FIGURA 2.3. Evolución del learning rate con una política "triangular2".

2.4. Función inicializadora de parámetros

El desempeño de las DNN fue incrementando a la par de su profundidad. Estos resultados experimentales se obtuvieron con nuevos métodos de inicialización de parámetros o mecanismos de entrenamiento. Sin embargo, la utilización de una inicialización aleatoria generaba malos resultados al combinarla con gradient descent, ya que se generaban problemas en la propagación de los gradientes a través de las capas. En la misma publicación donde se discutieron estos problemas también se propuso como solución un nuevo método de inicialización de parámetros con una distribución uniforme cuyo intervalo varía con la profundidad de la capa [49]. El método se conoce como *Glorot Uniform* y la distribución se define como en la ecuación 2.4.

$$W \sim U\Big[-\frac{\sqrt{6}}{\sqrt{n_j + n_{j+1}}}, \frac{\sqrt{6}}{\sqrt{n_j + n_{j+1}}}\Big] \tag{2.4}$$

2.5. Regularización

La regularización $L2$ o *weight decay* es una técnica de regularización que penaliza vectores w con norma grande. Se modifica la función de pérdida original al agregarle el término de regularización $\frac{\lambda}{2}w^T w$, quedando la nueva función de pérdida y su gradiente respecto a los parámetros como las ecuaciones 2.5 y 2.6, respectivamente.

$$\tilde{J}(w; X, y) = \frac{\lambda}{2}w^T w + J(w; X, y) \tag{2.5}$$

$$\nabla_w \tilde{J}(w; X, y) = \lambda w + \nabla_w J(w; X, y) \tag{2.6}$$

Donde λ determina la "fuerza" de la penalización: valores más grandes de λ favorecen pesos cercanos al origen, mientras que valores más pequeños ofrecen mayor libertad al crecimiento de w.
Dado que este hiperparámetro excede el alcance de este trabajo y el alto grado de adopción de esta técnica de regularización en particular, se consideró adecuado su utilización con su parámetro por defecto de la librería Keras ($\lambda = 0{,}01$) [50].

2.6. Funciones de activación

2.6.1. Hidden layers

Las capas ocultas se implementan con la función de activación ReLU (*Rectified Linear Unit*), una función de activación no lineal representada en la figura 2.4 y definida según la ecuación 2.7 [51].
Los beneficios de usar ReLU sobre otras funciones de activación son que acelera los cálculos ya que su derivada es siempre 1 para inputs positivos y que el entrenamiento no sufre de vanishing gradients, por no tener saturación [52].

$$f(x) = \begin{cases} 0 \text{ si } x < 0 \\ x \text{ si } x \geq 0 \end{cases} \tag{2.7}$$

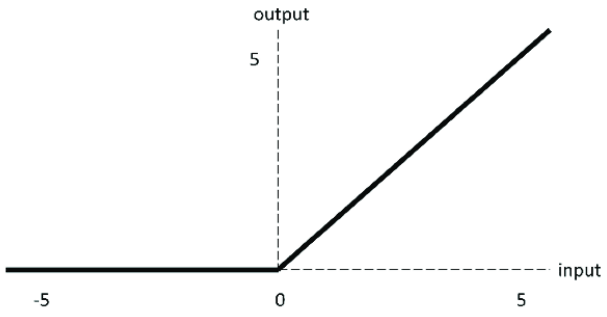


FIGURA 2.4. Gráfica de la función de activación ReLU [2].

[2]Imagen tomada de [53].

2.6.2. Output layer

Para la capa de salida se utiliza la función *Softmax* que calcula como salida un vector real $\hat{y} \in \mathbb{R}^C$ cuyas componentes están definidas en la ecuación 2.8, donde los z_j son los elementos de entrada de la capa. Este vector se interpreta como una distribución de probabilidad sobre las C clases [54].

$$\hat{y}_i = \frac{\exp(z_i)}{\sum_{j=0}^{C-1} \exp(z_j)} \tag{2.8}$$

2.7. Medición de performance

2.7.1. Función de pérdida

La función de pérdida utilizada fue *categorical crossentropy*, que pertenece al conjunto de funciones de pérdida probabilística. Su utilización es adecuada para tareas de clasificación y se computa como la suma ponderada sobre las C clases, como se observa en la ecuación 2.9. $y \in \mathbb{R}^C$ es el vector solución con un uno en la posición correspondiente a la clase y cero en el resto. $\hat{y}$ es el vector de predicción del modelo.

$$CCE = -\sum_{i=0}^{C-1} y_i log(\hat{y}_i) \tag{2.9}$$

2.7.2. Medida de la performance

Para medir la performance se utilizó la *categorical accuracy* (CA) que calcula la frecuencia con la que la predicción coincide con la etiqueta correcta. Se calcula según la ecuación 2.10, donde $M = \sum_{i=1}^{|D|} \mathcal{I}_i$, siendo $\mathcal{I}_i$ la función indicadora de si la predicción sobre el ejemplo i es correcta, y $|\mathcal{D}|$ la cardinalidad del conjunto de datos sobre el que se está evaluando.

$$CA = \frac{M}{|\mathcal{D}|} \tag{2.10}$$

2.8. Conjuntos de datos

En esta sección se describen el proceso de selección de los conjuntos de datos (*datasets*), sus características y el preprocesamiento.

2.8.1. Criterios de selección

A la hora de seleccionar con qué conjuntos de datos trabajar se buscó un balance entre la posibilidad de comparar los resultados obtenidos con los de trabajos previos, la extensión de esos resultados y el aporte innovador. Al mismo tiempo, se tuvieron en cuenta las limitaciones del hardware y el tiempo de uso disponible.
La posibilidad de construir un conjunto de datos propios fue descartada por la cantidad de horas-hombre necesarias para alcanzar un tamaño razonable, siendo que existe una gran variedad disponible de manera gratuita en línea. Además,

utilizar un conjunto de datos nuevo, sin un objetivo específico asociado al mismo, representa un esfuerzo innecesario y complejiza la reproducibilidad y extensión de los resultados.
Por lo anterior, las características deseables son: buen balance entre tamaño de imagen y cantidad de ejemplos, para que el procesamiento sea posible con el hardware disponible; fácil acceso en línea, para facilitar posibles trabajos futuros; popularidad en el ámbito de la investigación, para que los resultados obtenidos se construyan sobre un cuerpo sólido de conocimientos; novedoso, para que los resultados se extiendan más allá de los datos utilizados habitualmente.

2.8.2. Datasets seleccionados

CIFAR10

CIFAR10 [55] es un dataset construído por el Instituto Canadiense para Investigaciones Avanzadas (CIFAR, por sus siglas en inglés) con el objetivo de tener una referencia para el entrenamiento de algoritmos de machine learning y visión por computadora. Es un subconjunto etiquetado del *"80 million tiny images dataset"*, un dataset de 79302017 imágenes reducidas, obtenidas por búsquedas automáticas de sustantivos de *WordNet*. Está constituído por 60000 imágenes RGB de 32x32 con 10 clases, está perfectamente balanceado, con 6000 imágenes por clase. Los datos se dividen en 50000 imágenes de entrenamiento y 10000 de test. En la figura 2.5 pueden observarse imágenes del conjunto, con sus etiquetas.

FIGURA 2.5. Extracto de CIFAR10 con etiquetas [3].

CIFAR100

CIFAR100 [55] es muy similar a CIFAR10, con la excepción de tener 100 clases y 600 ejemplos de cada una (500 de entrenamiento y 100 de test). Las 100 clases se agrupan en 20 "supraclases", cada imagen trae la información de ambas etiquetas, como se especifica a continuación en la tabla 2.2.

TABLA 2.2. Supraclases y clases del dataset CIFAR 100 [4].

Supraclases	**Clases**
aquatic mammals	beaver, dolphin, otter, seal, whale
fish	aquarium fish, flatfish, ray, shark, trout
flowers	orchids, poppies, roses, sunflowers, tulips
food containers	bottles, bowls, cans, cups, plates
fruit and vegetables	apples, mushrooms, oranges, pears, sweet peppers
household electrical devices	clock, computer keyboard, lamp, telephone, television
household furniture	bed, chair, couch, table, wardrobe
insects	bee, beetle, butterfly, caterpillar, cockroach
large carnivores	bear, leopard, lion, tiger, wolf
large man-made outdoor things	bridge, castle, house, road, skyscraper
large natural outdoor scenes	cloud, forest, mountain, plain, sea
large omnivores and herbivores	camel, cattle, chimpanzee, elephant, kangaroo
medium-sized mammals	fox, porcupine, possum, raccoon, skunk
non-insect invertebrates	crab, lobster, snail, spider, worm
people	baby, boy, girl, man, woman
reptiles	crocodile, dinosaur, lizard, snake, turtle
small mammals	hamster, mouse, rabbit, shrew, squirrel
trees	maple, oak, palm, pine, willow
vehicles 1	bicycle, bus, motorcycle, pickup truck, train
vehicles 2	lawn-mower, rocket, streetcar, tank, tractor

[3] Imagen tomada de https://www.cs.toronto.edu/~kriz/cifar.html.

[4] Tabla tomada de https://www.cs.toronto.edu/~kriz/cifar.html.

EUROSAT

EUROSAT [56] es un dataset con imágenes del satélite Sentinel-2 con dos presentaciones: una versión de tres canales (RGB) y una de 13 canales que incluye bandas del espectro infrarojo. Contiene 27000 imágenes de 64x64 y 10 clases.
Para este trabajo se ha utilizado la versión RGB por ser representativa del caso de clasificación de imágenes. En la figura 2.6 se puede ver un extracto de los datos.

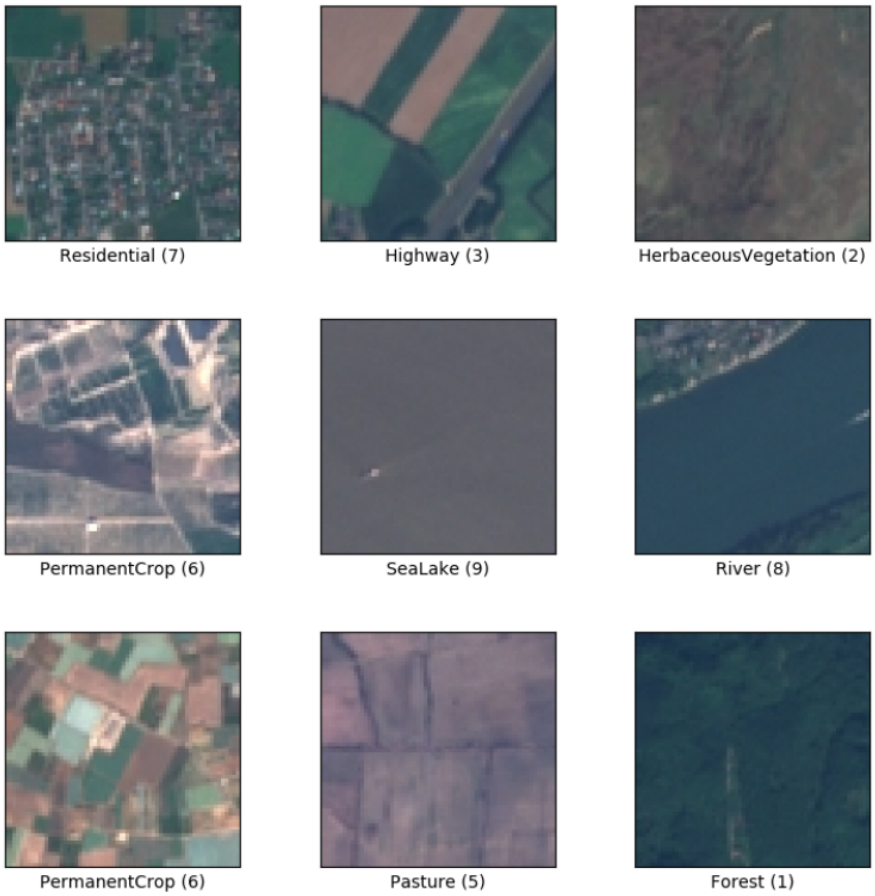


FIGURA 2.6. Muestra del dataset EUROSAT RGB [5].

[5]Imagen tomada de https://www.tensorflow.org/datasets/catalog/eurosat

2.9. Plataforma de trabajo

El trabajo fue desarrollado enteramente sobre la plataforma *Colab* de Google en su versión gratuita.
Colab es un servicio de Google que permite trabajar en la nube con Jupyter-Notebooks alojadas en los servidores de la empresa y que pone a disposición de los usuarios máquinas virtuales con diferentes asignaciones de hardware, priorizando la potencia del mismo en función del servicio contratado. Por no contar con una suscripción paga, la prioridad asignada a nuestra VM (*virtual machine*) fue menor, recibiendo consistentemente hardware del más básico disponible. A pesar de esta limitación, el hardware disponible en la versión gratuita fue considerado suficiente para la elaboración del trabajo y se lo tuvo en cuenta a la hora de seleccionar los datasets y definir el tamaño de los modelos a utilizar.
Las dos características principales que se consideraron para seleccionar esta plataforma de trabajo son:

- Posibilidad de compartir notebook de trabajo: esto permitió al director tener acceso directo al código para poder inspeccionarlo, realizar comentarios e identificar oportunidades de mejora en las implementaciones.
- Especificaciones de las placas de video (GPU) disponibles: siendo que el hardware específico para trabajar con imágenes era crítico para este proyecto y que las especificaciones técnicas de las GPU disponibles en Colab son superiores al hardware disponible localmente, los beneficios de ejecutar los experimentos en la nube fueron sustanciales.

En cuanto a los posibles inconvenientes de trabajar en Colab, se identificaron los siguientes:

- Requiere conexión estable a internet: por ser un servicio alojado en la nube requiere una conexión estable para que la sesión no caduque y se pierda el trabajo realizado. No se identificaron acciones preventivas o correctivas frente a este tipo de incidente.
- Tiempo de ejecución con GPU limitada: por utilizar la versión gratuita del servicio, el tiempo de uso de GPU es limitado. Para evitar la interrupción del entrenamiento, se decidió entrenar grupos de modelos en diferentes días.
- RAM y vRAM limitadas: tanto la memoria RAM como la vRAM están limitadas por el el hardware asignado, no pudiendo acceder a ampliaciones sin una suscripción paga. Se implementan funciones para gestionar la memoria, consultar sección 3.5.

2.9.1. Hardware

Memoria RAM

La memoria RAM disponible es de 12 GB. No ha sido posible conseguir acceso a mayores especificaciones de la misma.

Placa de video

En la tabla 2.3 pueden verse las especificaciones técnicas de la placa de video NVIDIA T4 asignada a nuestra VM durante la ejecución de los experimentos.

TABLA 2.3. Especificaciones técnicas de GPU NVIDIA Tesla T4 [6].

Performance	Núcleos tensoriales Turing	320
	Núcleos NVIDIA CUDA	2560
	Performance precisión simple (FP32)	8.1 TFlops
	Precisión mixta (FP16/FP32)	65 FP16 TFlops
	Precisión INT8	130 INT8 TOPS
	Precisión INT4	260 INT4 TOPS
Conectividad	GEN3 x16 PCIe	
Memoria	Capacidad	16GB GDDR6
	Ancho de banda	320+ GB/s
Potencia	70 watts	

Procesador (CPU)

Se cuenta con un CPU con las especificaciones técnicas indicadas en la tabla 2.4.

TABLA 2.4. Especificaciones técnicas del CPU disponible en la máquina virtual de Google Colab [7].

Característica	**Valor**
Architecture	x86_64
CPU op-mode(s)	32-bit, 64-bit
Byte Order	Little Endian
CPU(s)	2
On-line CPU(s) list	0,1
Thread(s) per core	2
Core(s) per socket	1
Socket(s)	1
NUMA node(s)	1
Vendor ID	GenuineIntel
CPU family	6
Model	79
Model name	Intel(R) Xeon(R) CPU @ 2.20GHz
Stepping	0
CPU MHz	2199.998
BogoMIPS	4399.99
Hypervisor vendor	KVM
Virtualization type	full
L1d cache	32K
L1i cache	32K
L2 cache	256K
L3 cache	56320K

[6] Tabla tomada de https://www.nvidia.com/es-la/data-center/tesla-t4/

[7] Tabla tomada del entorno de ejecución de Colab.

Capítulo 3

Diseño e implementación

En este capítulo se presentan las decisiones de diseño para la implementación de los experimentos. Primero, se detalla la obtención de los conjuntos de datos y su tratamiento. Luego, se presentan cada uno de los desarrollos que permiten realizar los experimentos. Finalmente, se presenta la función que integra estos componentes y habilita la posibilidad de realizar múltiples ejecuciones consecutivas.

3.1. Conjuntos de datos

3.1.1. Obtención y preprocesamiento

Los tres datasets utilizados se descargan desde los repositorios incluídos en las librerías Keras (CIFAR10 y CIFAR100) y TensorFlow (EUROSAT).
Se implementa la función "preprocess_dataset" que procesa tanto la información de las imágenes como sus etiquetas de clase.
Las imágenes son normalizadas con la función de la ecuación 3.1. El dividir por 255 lleva los valores al intervalo $[0, 1]$ y se centran en 0 al restar 0.5, esto se realiza en los tres canales de color. Se selecciona esta transformación por ser habitual tanto en el dominio de las imágenes como otro tipo de datos, generalmente con buenos resultados.

$$f : \{0, 1, ..., 255\} \rightarrow [-0/5, 0{,}5] \qquad f(p) = \frac{p}{255} - 0{,}5 \tag{3.1}$$

En la imagen 3.1 puede verse un ejemplo de la transformación de un canal en una región de 3x3 pixeles.

Los valores de las etiquetas se convierten a la codificación *one hot encoding*, donde cada uno de los C valores que codifica una clase se convierte en un vector en $\{0, 1\}^{|C|}$, donde todas las posiciones contienen un '0', con excepción de la posición que corresponde a la clase codificada, que tiene un '1'.

200	195	192
190	192	190
185	182	180

(A) Valores originales de los píxeles.

0.284	0.265	0.253
0.245	0.253	0.245
0.225	0.214	0.206

(B) Valores normalizado con la función f.

FIGURA 3.1. Valores de la imagen antes y después del procesamiento.

$$OHE : \{0, 1, ..., C-1\} \rightarrow \{0, 1\}^{|C|} \tag{3.2}$$

$$OHE(n) = [x_0, x_1, ..., x_{C-1}]$$

$$x_i = \begin{cases} 1 & \text{si } i = n \\ 0 & \text{si } i \neq n \end{cases} \tag{3.3}$$

A continuación se muestra un ejemplo. Si se tienen 10 clases codificadas en 0, 1, 2, 3, 4, 5, 6, 7, 8, 9 y se procesa un ejemplo que pertenece a la clase '4', resulta un vector de 10 posiciones con un '1' en la quinta posición.

$$OHE(4) = [0, 0, 0, 0, 1, 0, 0, 0, 0, 0]$$

Otro factor importante en el procesamiento de los datos es la separación en conjuntos de entrenamiento y validación. Como se describió en la sección 2.8, los datos de CIFAR ya están separados, por lo que se respeta la división de diseño de sus creadores. En cuanto a EUROSAT, se utilizan el 80 % de las imágenes para entrenamiento y el 20 % restante para validación. Si bien no existen proporciones óptimas para esta división, la regla de 80/20 es muy recomendada en la comunidad del aprendizaje automático [57][58][59], lo que motiva la decisión.
Asimismo, se determinó no ser necesaria la utilización de un conjunto de *testing* ya que el foco de este trabajo está puesto en el entrenamiento de las redes y de esta manera se aprovechan más las imágenes disponibles.

3.1.2. Problemas encontrados

Durante la fase de ejecución de learning rate range test con el dataset CIFAR100 ninguna de las variaciones evaluadas lograba mejorar su performance respecto a la inicialización de pesos aleatorios. Por este motivo, se decidió no continuar los ensayos con dicho dataset, entendiendo que la compejidad de los modelos ensayados no era suficiente para captar la estructura de las 100 clases teniendo tan poca información disponible (pocos ejemplos por clase y ejemplos de tamaño pequeño). En la bibliografía consultada otros autores también observan una importante caída en la performance de sus modelos al entrenar sobre CIFAR100 [13] [33].

3.2. Automatización de la construcción de arquitecturas

Para este trabajo se desarrolló una función constructora de arquitecturas descripta en el algoritmo 5. La función hace uso del concepto de "bloque" que se ha definido como un conjunto de tres capas: dos convolucionales y una de max pooling. A la salida del último bloque se la transforma en un vector de $m \times 1$, donde m es la cantidad total de elementos que la componen. Luego de esta operación, siguen dos capas fully connected, con m unidades cada una. Este patrón está inspirado en la arquitectura utilizada por K. Simonyan y A. Zisserman en sus redes VGG [60]. La última capa es también una capa FC, pero con tantas unidades como clases tenga el conjunto de datos.

Algoritmo 5 Algoritmo para construcción de arquitecturas

```
function CONSTRUCTOR(bloques, init_convlayers, optimizer, input_shape, seed)
    modelo ← Input layer                          // Capa de entrada
    nbloque ← 0
    for i in range(0, bloques) do
        modelo ← bloque(i)                        // Agrega nuevo bloque
    end for
    modelo ← Flatten layer                        // Transforma output del bloque
    for i do in range(2)
        modelo ← fc-layer                         // Agrega capas FC
    end for
    modelo ← output layer
    return modelo
end function
```

Las capas convolucionales de un mismo bloque tienen siempre la misma configuración, que depende del número de bloque que se trate, especificada en la tabla 3.1. En los ensayos también se han hecho variaciones en los filtros de la primera capa, lo que modifica la complejidad de la red final. Inicialmente se buscaba utilizar $F_0 \in [4, 8, 16, 32]$, pero al momento de trabajar con EUROSAT se volvió necesario reducirlo por el tamaño más grande de las imágenes, por lo que en ese caso se utilizó $F_0 \in [2, 4, 8, 16]$.
Los filtros de $(3, 3)$ se eligen por ser los más pequeños que permiten tener noción direccional sobre la imagen, sumado a que las imágenes con que se trabaja no son de alta definición. Esto último también motiva la elección del padding "same" y el stride $(1, 1)$ para preservar lo más posible la información a lo largo de la profundidad de la red. La activación ReLU, el inicializador Glorot Uniform y la regularización L2 se seleccionan por ser todos métodos validados por la comunidad y que suelen ser recomendados en libros [19][26] y tutoriales [61][62].

Las capas de pooling son invariantes a lo largo de todos los bloques de la red, como lo indica la tabla 3.2. La dimensión del filtro de pooling se define de (2,2) ya que es la mínima necesaria para poder realizar la operación, mientras que el stride se seleccionó de manera que las dimensiones se vean reducidas a la mitad tras cada capa. Dado que en cada capa de pooling se pierde el 75 % de la información, se ha decidido tomar el valor máximo, que indicaría la mayor activación de un determinado filtro en las convoluciones anteriores.

Considerando la complejidad de los conjuntos de datos utilizados y los recursos computacionales disponibles, se definió utilizar arquitecturas con dos, tres y

TABLA 3.1. Configuración de las capas convolucionales. La cantidad de filtros depende del número de bloque $nbloque$ y los filtros iniciales F_0.

Parámetro	Valor
Filtros	$F_0 \cdot 2^{\text{nbloque}}$
Activación	ReLU
Tamaño de kernel	$(3, 3)$
Stride	$(1, 1)$
Padding	same
Inicializador de kernel	Glorot Uniform
Regularizador de kernel	L2

TABLA 3.2. Configuración de las capas de pooling.

Parámetro	Valor
Pool size	$(2, 2)$
Stride	$(2, 2)$
Operación	Max

cuatro bloques para los ensayos.

3.3. Callbacks

Se desarrolló el callback "LRRangeTest" con el que se ejecuta el learning rate range test y guarda la información del entrenamiento. Al inicializarse la instancia computa la cantidad de batches por epoch como la división entera , luego corrige de ser necesario, en caso que el tamaño del batch no sea divisor exacto del conjunto de entrenamiento. Con los valores mínimo y máximo del learning rate clacula la pendiente necesaria para el ciclo. Durante las diferentes instancias del entrenamiento realiza las siguientes acciones:

- Comienzo de entrenamiento
  - Definir learning rate inicial
- Al finalizar el entrenamiento
  - Almacenar medias móviles de la pérdida y performance
  - Historial de learning rates
  - Índice de batch
- Al finalizar un batch
  - Almacena learning rate del batch
  - Almacena pérdida y accuracy del batch
  - Agrega número de batch al vector de índices
  - Actualiza learning rate
- Al comenzar una epoch

    - Registra el número de epoch
- Al finalizar una epoch
    - Almacena pérdida y accuracy de la epoch
    - Ejecutar rutina de liberación de memoria

Una vez que se corre el test y se determinan los valores mínimos y máximos para el entrenamiento final de la arquitectura se vuelve a ejecutar la misma configuración, modificando el callback utilizado por uno que implementa la política "triangular2" [39], descripta en la sección 2.3.

3.4. Learning rate

La política de learning rate adoptada es la variación "triangular2" de cyclical learning rate propuesta por Smith [39], determinando los valores extremos a partir del learning rate range test [39]. El test fue implementado dentro del callback descripto en la sección 3.3, tanto la política de learning rate como el test se encuentran resumidos en la sección 2.3.
El test se corrió durante ocho epochs, ya que es un valor dentro de lo recomendado por el autor y que no implicaba un gran compromiso de recursos computacionales. En cuanto al rango de variación, se realizaron dos pruebas diferentes. La primera en el rango $[0{,}0001; 0{,}01]$ y, la segunda en $[0{,}01; 0{,}1]$, rangos también recomendados por Smith.
Durante la ejecución de los primeros tests comenzaron a surgir problemas de utilización de memoria por las imágenes cargadas en RAM, llegando a detenerse el kernel de la notebook de la máquina virtual. Estas problemáticas en las etapas tempranas del trabajo motivaron el desarrollo de las funciones de gestión de memoria.

3.5. Gestión de la memoria

Para hacer un seguimiento y gestión del uso de la memoria se utilizaron las siguientes librerías:

- gputil: permite acceder a un resumen del estado de GPUs NVIDIA.
- psutil: es una librería para monitorear los procesos activos y la utilización del sistema (CPU, memoria, discos, redes, sensores).
- humanize: transforma unidades para mostrar en pantalla de manera amigable para seres humanos.

Se desarrolló una función de reporte del uso de memoria en tiempo de ejecución que muestra en pantalla la memoria RAM libre y, para cada GPU disponible, su id, vRAM libre, vRAM total y porcentaje de utilización.
Disponer de esta información permitió mejorar la gestión de la memoria e implementar las siguientes soluciones para evitar interrupciones en la ejecución:

- Al finalizar cada *epoch* de entrenamiento se utilizó el *garbage collector* de Python "gc.collect", realizando una limpieza completa de la memoria con archivos innecesarios.

- Se construye la función "reset_tensorflow_keras_backend" para ejecutar luego de cada ejecución completa. Esta función ejecuta primero el reporte de memoria, libera la memoria que ocupa Keras por ejecutar múltiples modelos en una misma sesión y limpia el *stack* del grafo de ejecución de TensorFlow.

3.6. Almacenamiento de resultados

Para el almacenamiento físico de los resultados obtenidos se desarrollaron dos funciones. La primera, "registrar_resultados", guarda en formato pickle la metadata que permite identificar el modelo y las métricas registradas por el callback en las cinco corridas realizadas. La información se organiza en formato de diccionario de Python, pudiendo accederse a cada una de las corridas o su mediana.
La función "plot_results" muestra en pantalla la métrica y la pérdida en eje vertical y el horizontal puede configurarse para que sea el learning rate o el número de batch. También posee un parámetro que permite guardar las imágenes en disco.

3.7. Función de ejecución de experimentos

El código 3.1 muestra la función "run_experiment" que consolida todo lo anterior. Sus parámetros de entrada se describen en la tabla 3.3, algunos son parámetros de funciones descriptas previamente que son llamadas dentro de esta rutina.
En la línea 14 se define el diccionario "run" que almacenará la información de todas las ejecuciones de la configuración parametrizada. En el ciclo "for" se construye un modelo con la función constructora, variando la semilla de aleatoriedad, "seed". Luego se define el callback "LRFinder" para el LRRT.
En la línea 29 se evalúa si el parámetro "val_split" existe. En caso afirmativo, se entrena el modelo con esa proporción de separación indicada. De lo contrario, se utilizan los conjuntos de test especificados. Esta separación permite utilizar la misma función tanto para EUROSAT (primer caso) como para CIFAR10 y CIFAR100 (segundo caso).
Luego del entrenamiento, se actualiza "run" agregando la información de esa iteración. Cuando concluyen las ejecuciones, la función "registrar_resultados" guarda en disco el diccionario con todas la información.

```
1  def run_experiment(bloques,
2                     input_shape,
3                     init_conv_filters,
4                     batch_size,
5                     epochs,
6                     init_lr,
7                     end_lr,
8                     nruns,
9                     optimizer,
10                    sma_periods,
11                    nclasses,
12                    metadic,
13                    val_split=None):
14    run = {}
15
16    for i in range(nruns):
17      modelo = model_constructor(bloques = bloques,
18                                 nclasses = nclasses,
19                                 init_conv_filters = init_conv_filters,
```

```
                                optimizer = optimizer,
                                input_shape = input_shape,
                                seed = i)
    LRFinder = LRRangeTest(init_lr = init_lr,
                           end_lr = end_lr,
                           train_size = len(xtrain),
                           batch_size = batch_size,
                           epochs = epochs,
                           sma_periods = sma_periods)
    if val_split:
          modelo.fit(
              x = xtrain,
              y = ytrain,
              validation_split = val_split,
              batch_size = batch_size,
              epochs = epochs,
              callbacks = [LRFinder]
              )
    else:
      modelo.fit(
                x = xtrain,
                y = ytrain,
                validation_data = (xtest, ytest),
                batch_size = batch_size,
                epochs = epochs,
                callbacks = [LRFinder]
                )
    run.update({'run_'+str(i):LRFinder.results})
    reset_tensorflow_keras_backend()
registrar_resultados(metadic,run)
return run

```

CÓDIGO 3.1. Función que consolida todo el desarrollo para ejecutar experimentos completos.

TABLA 3.3. Descripción de los parámetros de la función que consolida el experimento.

Parámetro	**Descripción**
bloques	Cantidad de bloques de la red.
input_shape	Dimensiones de la entrada.
batch_size	Tamaño de los batches de entrenamiento.
epochs	Cantidad de epochs de entrenamiento.
init_lr	Learning rate inicial.
end_lr	Learning rate final.
nruns	Cantidad de ejecuciones con la configuración.
optimizer	Algoritmo de optimización.
sma_periods	Períodos para calcular la media móvil.
nclasses	Clases de clasificar.
metadic	Diccionario con metadata del modelo.
val_split	Porcentaje para split de test (EUROSAT).

Capítulo 4

Ensayos y resultados

En este capítulo se presenta la metodología utilizada para realizar los ensayos, consolidando en un único *pipeline* de trabajo todo lo desarrollado en los capítulos anteriores, y se presentan los resultados obtenidos. Se expone un caso de aplicación completo y, finalmente, se comparan los resultados con diferentes fuentes.

4.1. Metodología aplicada

Para los ensayos, se selecciona un elemento del conjunto de configuraciones posibles $cfg = \{\mathcal{D}, rango, optimizador, arquitectura\}$, que describen las variaciones descriptas en capítulos anteriores, donde "$\mathcal{D}$" son los conjuntos de datos seleccionados, "rango" los rangos para el LRRT, "optimizador" el algoritmo de optimización a utilizar y "arquitectura" será un par (nbloques, filtros iniciales).
Los LRRT se ejecutan fijando pares $\{\mathcal{D}_i, rango_j\}$ y explorando las diferentes variaciones de $\{optimizador, arquitectura\}$, de este modo se tiene una idea general de qué esperar en estas configuraciones en particular.
Con este esquema se realiza el LRRT y se analizan sus resultados: si la configuración no converge o muestra un desempeño muy por debajo del resto de ese conjunto, finaliza el proceso y se descarta la configuración en cuestión. En caso contrario, para cada par $\{\mathcal{D}_i, rango_j\}$ que no haya sido descartado se comparan las variaciones de $\{optimizador, arquitectura\}$ y se selecciona el que muestra mejores resultados. Con esta última configuración se realiza un entrenamiento extendido de 50 epochs, utlizando la política de learning rate "triangular2" con los límites que se hayan identificado en el LRRT. Este proceso queda resumido en la figura 4.1.
Para la documentación de los ensayos se utiliza la siguiente codificación B(Cantidad de bloques)FL(Filtros primera capa)(Optimizador). Así, por ejemplo, un modelo entrenado con dos bloques, 16 filtros iniciales y SGD como algoritmo de optimización quedará etiquetado como B2FL16SGD.

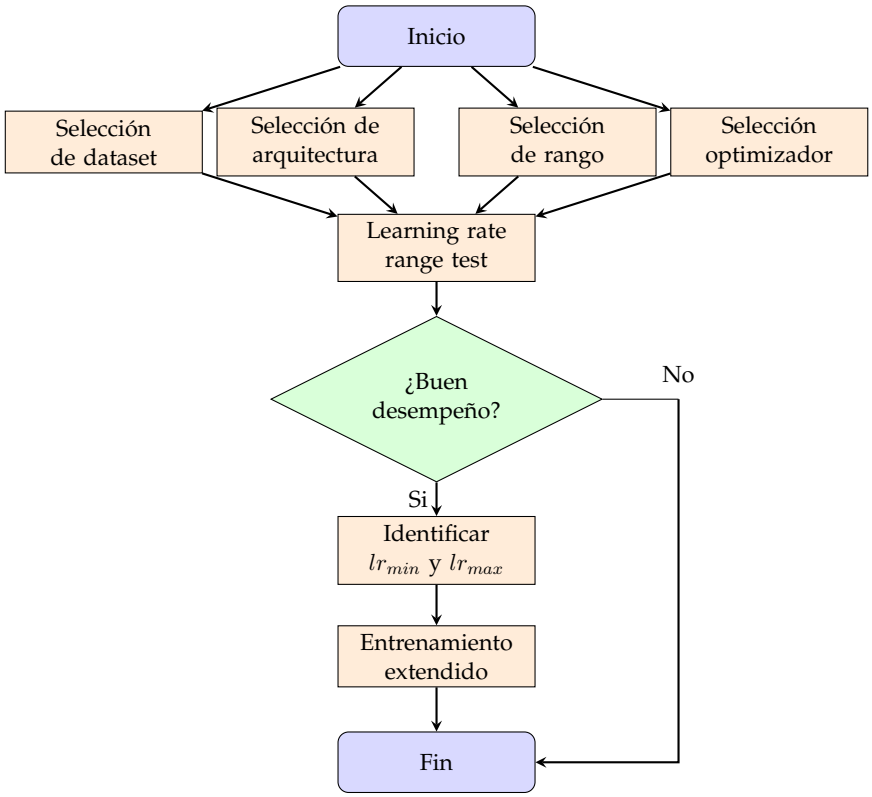


FIGURA 4.1. Diagrama de flujo de un experimento completo.

4.2. Aplicación a CIFAR10

La tabla 4.1 registra los resultados obtenidos del ensayo de las configuraciones con rango de LRRT (0.0001, 0.01) sobre CIFAR10.
En las arquitecturas con dos bloques puede verse que se obtuvieron mejores resultados con el optimizador Adam que con SGD. La diferencia por cantidad de filtros se nota más en el segundo caso que en el primero.
Con tres bloques las combinaciones de Adam con 8 y 16 filtros en la primera capa tuvieron una performance de nivel "aleatorio", con una distribución uniforme sobre las clases. En los casos de SGD esto no ocurre, sin embargo, tampoco logran superar la performance de aquellas redes con Adam que sí convergen.
Finalmente, las redes entrenadas con cuatro bloques sobre CIFAR10 no logran entrenarse con efectividad en ninguna de las configuraciones probadas, razón por la cual no se realizaron entrenamientos extendidos con ellas.
En la figura 4.2 se muestran las curvas de LRRT de la configuración de dos bloques con Adam y 32 filtros iniciales. Por su parte, la figura 4.3 las muestra en la configuración de tres bloques con Adam y 32 filtros iniciales.
En ambos casos se observa un comportamiento similar de las curvas. La curva de accuracy comienza con un crecimiento algo marcado, pero rápidamente pierde momento. En cuanto a la pérdida, ya en la segunda epoch deja de tener una mejoría notable, incluso empeora un poco, dado que no es algo muy significativo, el comportamiento podría atribuirse a la naturaleza de la función de costo utilizada y no a una merma en la performance.

TABLA 4.1. Resultados de ensayos sobre el dataset CIFAR10 con LR en (0.0001,0.01). Se documentan los máximos valores de accuracy de batch y epoch (conjunto train) y de validación.

Bloques	Filtros	Optimizador	Batch	Epoch	Validación
2	4	Adam	0.450402	0.44142	0.4541
2	8	Adam	0.502481	0.48580	0.4948
2	16	Adam	0.506809	0.50150	0.5008
2	32	Adam	0.525596	0.50372	0.4958
2	4	SGD	0.364949	0.36436	0.3709
2	8	SGD	0.381419	0.38134	0.3853
2	16	SGD	0.409110	0.39724	0.4145
2	32	SGD	0.429967	0.42976	0.4453
3	4	Adam	0.403353	0.39876	0.4083
3	8	Adam	0.146533	0.10216	0.1000
3	16	Adam	0.118881	0.10196	0.1000
3	32	Adam	0.423833	0.40746	0.4136
3	4	SGD	0.280569	0.27814	0.2824
3	8	SGD	0.214873	0.19508	0.2144
3	16	SGD	0.254956	0.25506	0.2817
3	32	SGD	0.321933	0.31178	0.3385
4	4	Adam	0.135533	0.10038	0.1000
4	8	Adam	0.126800	0.10044	0.1000
4	16	Adam	0.111700	0.10084	0.1000
4	32	Adam	0.108088	0.09950	0.1000
4	4	SGD	0.181954	0.16930	0.1906
4	8	SGD	0.156489	0.13392	0.1602
4	16	SGD	0.177600	0.14434	0.1727
4	32	SGD	0.230833	0.17938	0.2265

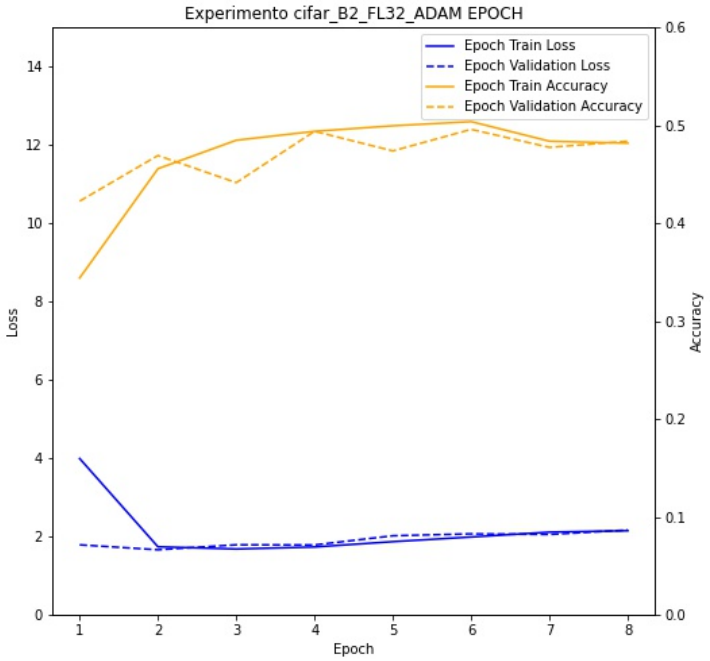


FIGURA 4.2. Curvas de entrenamiento para configuración B2FL32Adam

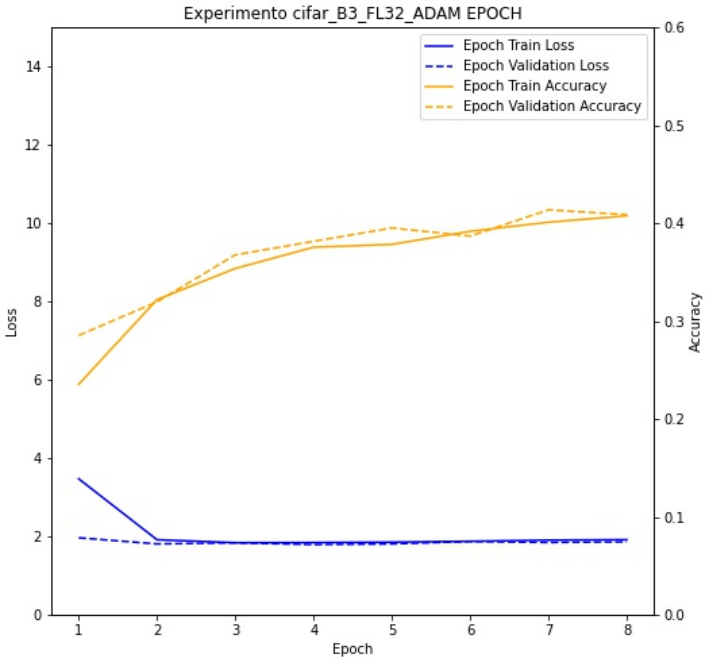


FIGURA 4.3. Curvas de entrenamiento para configuración B3FL32Adam.

En la tabla 4.2 se muestran los resultados registrados al ensayar las configuraciones de arquitectura sobre CIFAR10, con la variante de learning rate mayor. Puede observarse que en estos rangos de learning rate el optimizador Adam no logra mejorar la performance de modelo y queda atascado en lo que serían valores de performance aleatorios.
Por el lado de SGD, únicamente encuentra dificultades en el caso de cuatro bloques, mientras que los otros dos logra comenzar la convergencia, lo que es un buen indicador para los entrenamientos extendidos.
De esta manera, todos los modelos que quedan estancados se descartan y se seleccionan aquel con 2 bloques, SGD y 32 filtros iniciales, y con 3 bloques, SGD 32 capas iniciales, cuyas gráficas de learning rate range test se presentan en las figuras 4.4 y 4.5 respectivamente.
Analizando las imágenes puede observarse un muy elevado grado de convergencia por una pendiente negativa importante de la función de costo en las primeras epochs, inclusive las primeras tres quedan fuera de escala. El ratio de crecimiento de la categorical accuracy también es bastante importante en estas dos configuraciones. El caso dos bloques logra, en el LRRT, alcanza una performance algo superior al de tres bloques.

TABLA 4.2. Resultados de ensayos sobre el dataset CIFAR10 con LR en (0.01, 0.1). Se documentan los máximos valores de accuracy de batch y epoch (conjunto train) y de validación.

Bloques	Filtros	Optimizador	Batch	Epoch	Validación
2	4	Adam	0.103465	0.10028	0.1000
2	8	Adam	0.114119	0.09972	0.1000
2	16	Adam	0.106307	0.10022	0.1000
2	32	Adam	0.109405	0.10152	0.1000
2	4	SGD	0.446741	0.42104	0.4427
2	8	SGD	0.485700	0.45636	0.4671
2	16	SGD	0.446871	0.44708	0.4417
2	32	SGD	0.490343	0.48178	0.5036
3	4	Adam	0.107933	0.10064	0.1000
3	8	Adam	0.116740	0.10032	0.1000
3	16	Adam	0.113340	0.10062	0.1000
3	32	Adam	0.109400	0.10050	0.1000
3	4	SGD	0.384037	0.38350	0.3754
3	8	SGD	0.397567	0.38026	0.4012
3	16	SGD	0.405621	0.39906	0.4042
3	32	SGD	0.421307	0.41876	0.4297
4	4	Adam	0.111167	0.10100	0.1000
4	8	Adam	0.111267	0.10006	0.1000
4	16	Adam	0.118733	0.09992	0.1000
4	32	Adam	0.124300	0.10028	0.1000
4	4	SGD	0.132133	0.11506	0.1068
4	8	SGD	0.120226	0.11016	0.1006
4	16	SGD	0.118054	0.10988	0.1000
4	32	SGD	0.148423	0.12986	0.1480

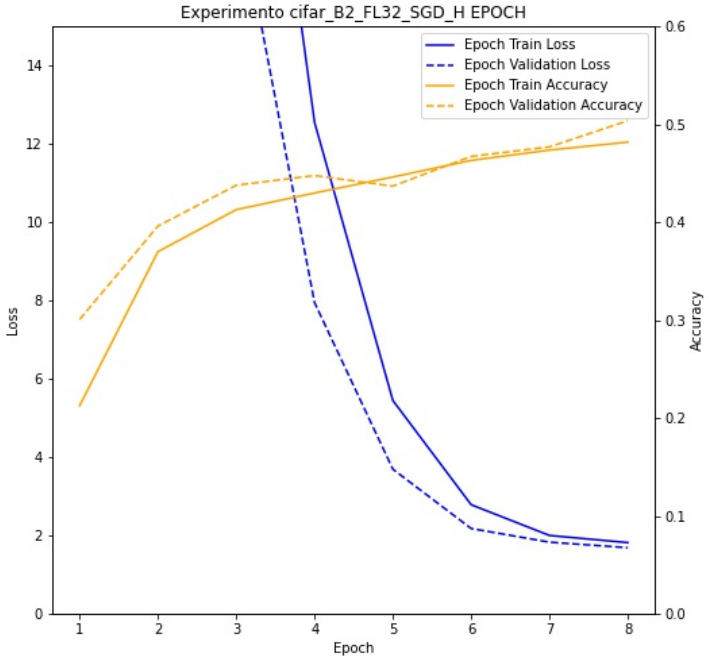


FIGURA 4.4. Curvas de entrenamiento para configuración B2FL32SGD.

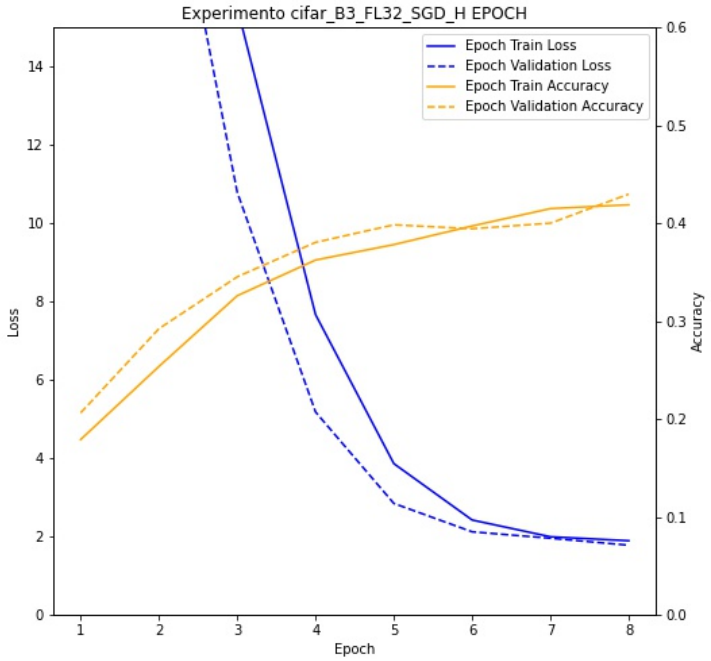


FIGURA 4.5. Curvas de entrenamiento para configuración B3FL32SGD.

4.3. Aplicación a EUROSAT

Para los tres casos de bloques estudiados el optimizador Adam resulta superior a SGD, alcanzando resultados incluso superiores a los de CIFAR10 en un dataset que es algo más complejo.
En este caso no aparecen diferencias significativas de rendimiento al comparar los modelos de 8 y 16 filtros iniciales, inclusive, en el caso de dos bloques, el más pequeño tiene mejor performance, como puede observarse en la tabla 4.3.
Las figuras 4.6 y 4.7 muestran los ensayos de las configuraciones con mejor performance. Los casos son parecidos, la función de pérdida alcanza un valor mínimo que se mantiene estable relativamente temprano en el entrenamiento. En cuanto a la categorical accuracy indica también que el modelo se encuentra en un régimen favorable para el entrenamiento con pendientes ascendentes pronunciadas.

TABLA 4.3. Resultados de ensayos sobre el dataset EUROSAT con LR en (0.0001,0.01). Se documentan los máximos valores de accuracy de batch y epoch (conjunto train) y de validación.

Bloques	Filtros	Optimizador	Batch	Epoch	Validación
2	2	Adam	0.590033	0.529167	0.545556
2	4	Adam	0.651768	0.621435	0.600370
2	8	Adam	0.690117	0.659259	0.665556
2	16	Adam	0.649849	0.617778	0.632963
2	2	SGD	0.309027	0.309120	0.306111
2	4	SGD	0.350253	0.350185	0.332778
2	8	SGD	0.388458	0.384630	0.373889
2	16	SGD	0.459553	0.444861	0.446852
3	2	Adam	0.456433	0.437269	0.480556
3	4	Adam	0.580774	0.581204	0.575370
3	8	Adam	0.641276	0.612870	0.620000
3	16	Adam	0.620048	0.613426	0.607593
3	2	SGD	0.227411	0.227454	0.246481
3	4	SGD	0.312748	0.291944	0.277037
3	8	SGD	0.328376	0.327685	0.318889
3	16	SGD	0.337676	0.336435	0.337593
4	2	Adam	0.462149	0.461574	0.474815
4	4	Adam	0.501481	0.481574	0.500741
4	8	Adam	0.498202	0.480185	0.517037
4	16	Adam	0.506933	0.488935	0.519259
4	2	SGD	0.217763	0.205556	0.210926
4	4	SGD	0.231720	0.220694	0.251111
4	8	SGD	0.250282	0.246713	0.236852
4	16	SGD	0.259781	0.241065	0.246296

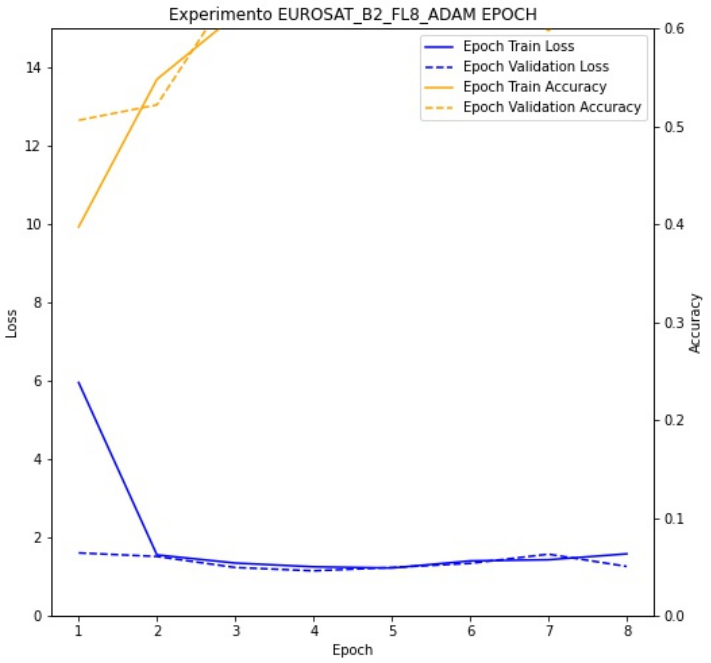


FIGURA 4.6. Curvas de entrenamiento para configuración B2FL8Adam.

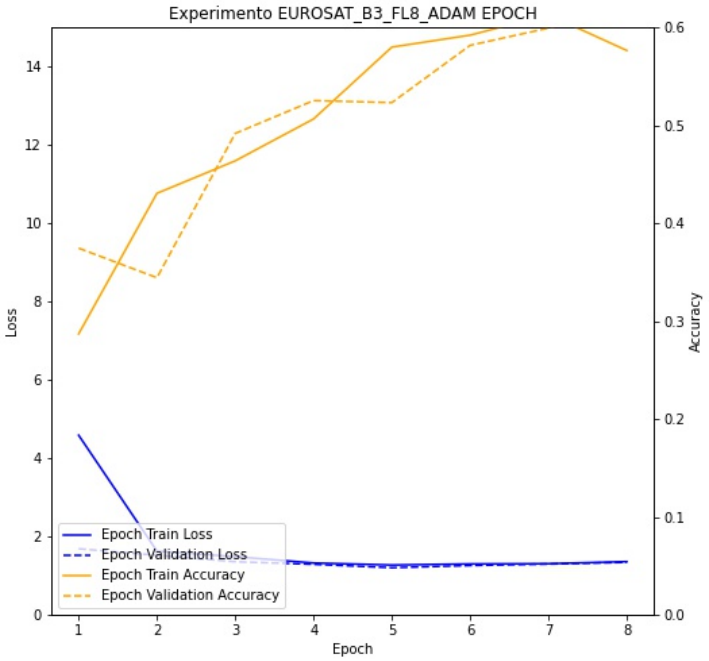


FIGURA 4.7. Curvas de entrenamiento para configuración B3FL8Adam.

Finalmente, la tabla 4.4 resume los resultados de EUROSAT en el rango de mayores learning rates para el test. Al igual que en el caso de CIFAR10, el optimizador Adam no logra buenos resultados en este rango de learning rates. Las tres configuraciones de bloques permanecen sin signos de aprendizaje a lo largo del test.
En cuanto a las variantes con SGD, sí logran obtener buenos ratios de convergencia. Curiosamente, la performance de estos modelos se deteriora al incorporar más bloques a la red.
Las figuras 4.8 y 4.9 muestran los resultados del LRRT. Ambos casos presentan comportamiento muy similares, la categorical accuracy muestra estar en una zona de crecimiento sostenido, mientras que la pérdida viene de estar muy fuera de escala con una pendiente que nuevamente valida que el modelo se encuentra en un buen régimen de optimización.

TABLA 4.4. Resultados de ensayos sobre el dataset EUROSAT con LR en (0.01; 0.1). Se documentan los máximos valores de accuracy de batch y epoch (conjunto train) y de validación.

Bloques	Filtros	Optimizador	Batch	Epoch	Validación
2	2	Adam	0.133967	0.112824	0.114259
2	4	Adam	0.124747	0.110231	0.117593
2	8	Adam	0.122000	0.110000	0.117593
3	16	Adam	0.120906	0.109630	0.114259
2	2	SGD	0.455700	0.429444	0.439630
2	4	SGD	0.514794	0.491759	0.465000
2	8	SGD	0.560619	0.511991	0.577407
2	16	SGD	0.560433	0.510833	0.558333
3	2	Adam	0.117400	0.111111	0.113704
3	4	Adam	0.126067	0.110648	0.113704
3	8	Adam	0.117700	0.111296	0.113704
4	16	Adam	0.118309	0.110972	0.114259
3	2	SGD	0.448481	0.419722	0.442593
3	4	SGD	0.417642	0.415648	0.408333
3	8	SGD	0.447021	0.446111	0.450000
3	16	SGD	0.481600	0.441806	0.490556
4	2	Adam	0.117327	0.109676	0.110185
4	4	Adam	0.116762	0.109769	0.114259
4	8	Adam	0.115497	0.110972	0.114259
4	16	Adam	0.118309	0.110972	0.114259
4	2	SGD	0.359533	0.333750	0.372407
4	4	SGD	0.365414	0.359583	0.373704
4	8	SGD	0.377067	0.368009	0.400556
4	16	SGD	0.390443	0.372361	0.391481

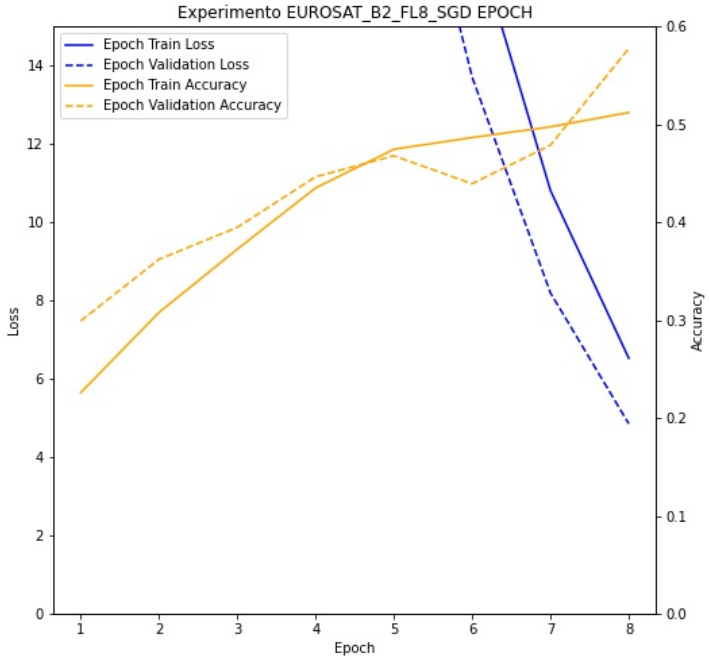


FIGURA 4.8. Curvas de entrenamiento para configuración B2FL8SGD.

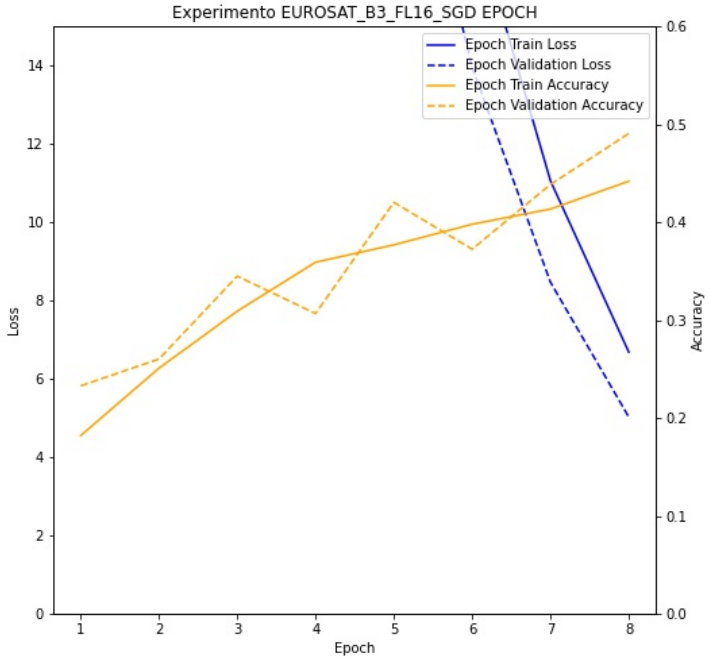


FIGURA 4.9. Curvas de entrenamiento para configuración B3FL16SGD.

4.4. Demostración de caso

Se presenta aquí un ejemplo completo sobre CIFAR10 con dos bloques. Se corren las ocho epochs con los dos optimizadores y las cuatro configuraciones de filtros de primera capa consideradas. Las figuras pueden consultarse en el anexo A.
Del análisis de esas figuras, se determina que la configuración con mejor performance es con 32 filtros en la primera capa (FL=32) y optimizador Adam. Para el entrenamiento extendido se selecciona como límite inferior el mínimo del test (0.0001) y como límite superior el valor al finalizar la primera epoch (0.0013375),y el learning rate seguirá una política triangular2, como se muestra en la figura 4.10.

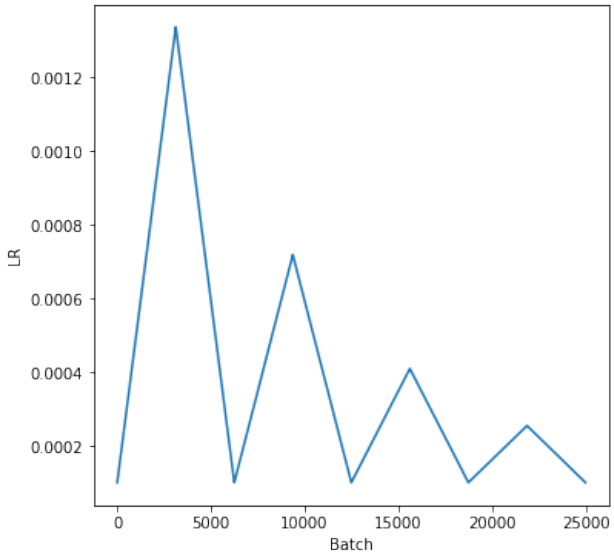


FIGURA 4.10. Evolución del learning rate siguiendo una política triangular en un caso de aplicación de la metodología propuesta.

En el entrenamiento extendido de 50 epochs con esta política de learning rate se ha alcanzando una accuracy en validación de 74 %, como puede observarse en la figura 4.11.

De este modo se muestra como la evaluación durante las etapas tempranas del entrenamiento permitieron encontrar un conjunto de hiperparámetros que efectivamente permiten la convergencia del modelo, que es lo que se estaba buscando. Para alcanzar niveles de performance superiores podrían hacerse entrenamientos extendidos de mayor longitud o incorporar pruebas con diferentes variantes de los hiperparámetros.

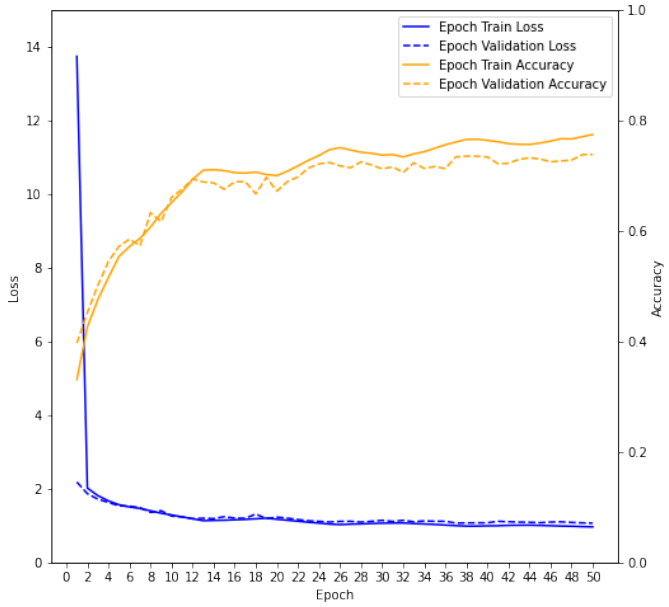


FIGURA 4.11. Métrica y pérdida durante el entrenamiento extendido.

4.5. Resultados

En la tabla 4.5 se presentan los resultados obtenidos para la mejor configuración en cada uno de los casos evaluados según lo descripto previamente. No se documentan aquí resultados relacionados al dataset CIFAR100 ya que en ninguno de los ensayos realizados los modelos mostraban indicios de convergencia.

TABLA 4.5. Evaluación de la performance de los diferentes modelos entrenados en tandas largas tras realizar la prueba del learning rate range test y utilizar política triangular2. N/A: No aplica porque ninguno de los modelos entrenados durante el test converge.

Dataset	Bloques	Filtros	LR range	Optimizador	CA (%)
EUROSAT	2	8	[0.0001; 0.0038125]	Adam	79.74
EUROSAT	2	8	[0.01;0.04]	SGD	78.07
EUROSAT	3	8	[0.0001; 0.0045]	Adam	83.59
EUROSAT	3	8	[0.01; 0.1]	SGD	71.66
EUROSAT	4	16	[0.0001; 0.0015]	Adam	77.68
EUROSAT	4	8	[0.01;0.07]	SGD	79.24
CIFAR10	2	32	[0.0001; 0.0013375]	Adam	73.82
CIFAR10	2	32	[0.01;0.047125]	SGD	67.32
CIFAR10	3	32	[0.0001; 0.0013375]	Adam	74.57
CIFAR10	3	32	[0.01; 0.047125]	SGD	66.06
CIFAR10	4	N/A	N/A	N/A	N/A
CIFAR10	4	N/A	N/A	N/A	N/A

En la tabla 4.6 se comparan los resultados obtenidos con el dataset CIFAR100 con datos disponibles de la bibliografía consultada.

Puede observarse que el mejor modelo final entrenado con el procedimiento propuesto no logra los mismos niveles de accuracy en test que los trabajos citados.

TABLA 4.6. Comparación de resultados de trabajos citados en el estado del arte y el trabajo realizado. CA: categorical accuracy en test. CLR: cyclical learning rate. CM: cyclical momentum. WD: weight decay.

Trabajo	Arquitectura	Metodologías	Epochs	CA (%)
Este trabajo	3-bloques	CLR	50	74.6
Smith, 2018 [13]	wide resnet	WD	800	90.3
	wide resnet	CLR+CM+WD	100	91.9
	densenet	WD	400	92.7
	densnet	CLR+CM+WD	150	92.8
	3-layer	CLR+CM+WD	25	81.3
	resnet-56	CLR+CM+WD	95	92.0
Franklin et al., 2018 [31]	VGG-19	Prunning	160	91.0
	Resnet-18	Prunning	240	87.0

Sin embargo, hay que destacar las diferencias en las metodologías utilizadas que justifican estos resultados. La metodología aquí propuesta es la que menos cantidad de ”innovaciones” incorpora, lo que fue una decisión de diseño para que sea utilizable por personas con poca experiencia en el campo. Al mismo tiempo, es el segundo entrenamiento más corto de todos, teniendo casi la mitad de epochs que el que le sigue.

Capítulo 5

Conclusiones

En este capítulo se presentan las conclusiones y los principales aportes del trabajo realizado. A su vez, se presentan posibles lineas de trabajo futuro.

5.1. Conclusiones generales

En este trabajo se realizó un estudio de los efectos de la variación de ciertos hiperparámetros para entrenar modelos de clasificación de imágenes con redes neuronales convolucionales. A continuación se detallan los logros más importantes:

- Descripción de una heurística para la evaluación del diseño de una CNN para una tarea específica.
- Desarrollo de una función constructora de redes convolucionales implementando un patrón de diseño predeterminado.
- Validación y extensión del uso de ciclos triangulares de *learning rate*.
- Validación de la metodología *learning rate range test* para calibrar los rangos de la anterior política.
- Alto grado de reproducibilidad de las experiencias realizadas.

Por lo cual se entiende que el aporte es significativo para personas que se inician en este campo.
En cuanto a los riesgos identificados, la limitación de memoria disponible impuesta por el proveedor del servicio fue el único que se manifestó y afectó al proyecto en sus comienzos. Para mitigarlo se desarrollaron funciones adicionales que optimizaran el uso de la RAM y vRAM disponibles.
Lo estudiado en diferentes materias de la Especialización en Inteligencia Artificial permitió concebir y desarrollar este trabajo. A continuación se detallan aquellas materias que tuvieron mayor influencia:

- Aprendizaje de máquina I y II: Por su introducción a los conceptos fundamentales del aprendizaje automático, los principales algoritmos, las métricas y la historia de su evolución, poniendo en contexto la carrera.
- Inteligencia artificial: Por su presentación de la inteligencia artificial moderna y cómo interactúan los diferentes componentes de un modelo. Cuáles fueron los hitos de la disciplina y cuáles son los desafíos actuales. La explicabilidad y diseño de redes neuronales motivaron este trabajo.

- Aprendizaje profundo: Por el estudio en profundidad del funcionamiento de las redes neuronales profundas y sus componentes, en particular las funciones de activación, algoritmos de optimización y funciones de costo.
- Visión por computadora I y II: Por presentar un área de estudio con larga trayectoria que se vió revolucionada por la aparición de las redes convolucionales. La gran cantidad de campos de aplicación y su impacto positivo en la sociedad motivaron la selección de la visión por computadora como eje de este trabajo.
- Gestión de proyectos: Por permitir transformar una idea en algo concreto y brindar una estructura de trabajo eficaz y eficiente.

5.2. Próximos pasos

Para continuar el desarrollo de esta línea de trabajo se identifican las siguientes oportunidades:

- Utilización de imágenes en alta definición.
- Extensión a problemas de clasificación con otro tipo de datos, como texto, sonido o video.
- Extensión a problemas de regresión.
- Implementación con otros tipos de redes profundas, como RNN o transformers.
- Realizar mediciones sobre la eficiencia de los diferentes modelos.

Apéndice A

Figuras del ejemplo de aplicación

Se presentan aquí las figuras correspondientes al ejemplo de aplicación presentado en el capítulo 4.

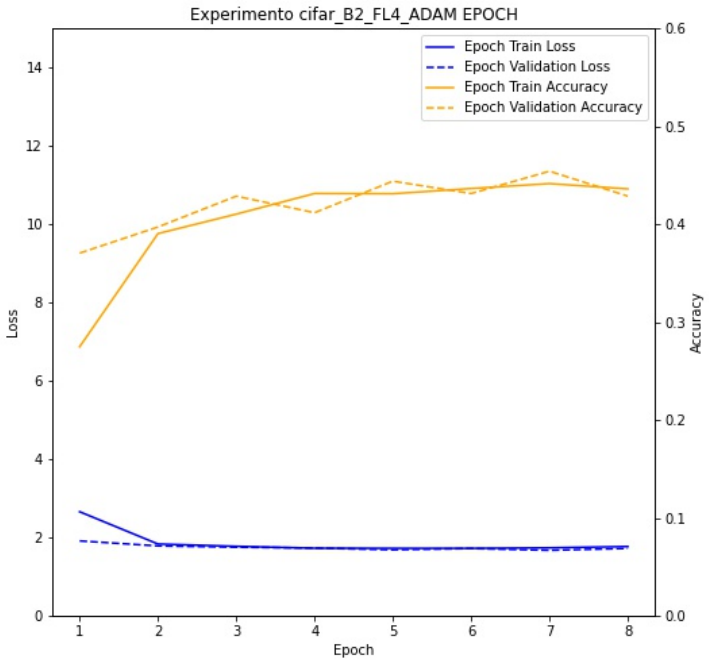


FIGURA A.1. Caso de aplicación en CIFAR10 - FL=4, Optimizador=ADAM.

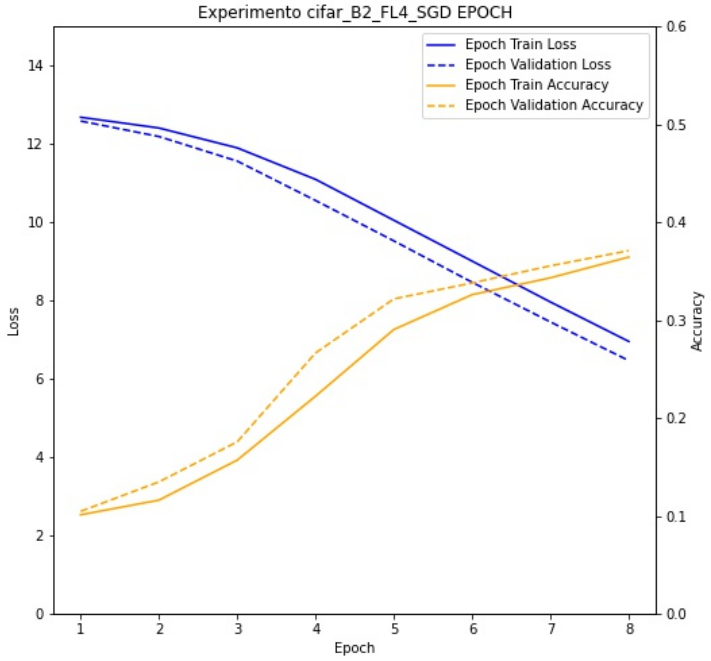


FIGURA A.2. Caso de aplicación en CIFAR10 - FL=4, Optimizador=SGD.

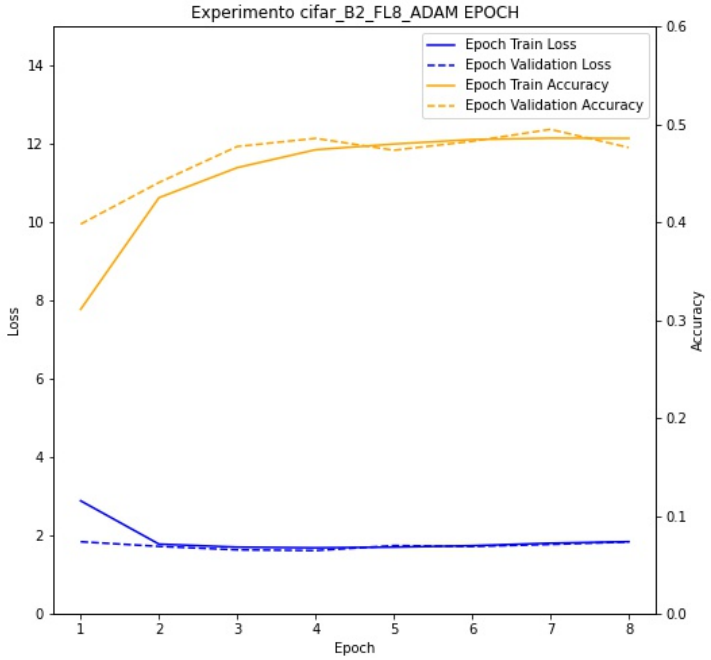


FIGURA A.3. Caso de aplicación en CIFAR10 - FL=8, Optimizador=ADAM.

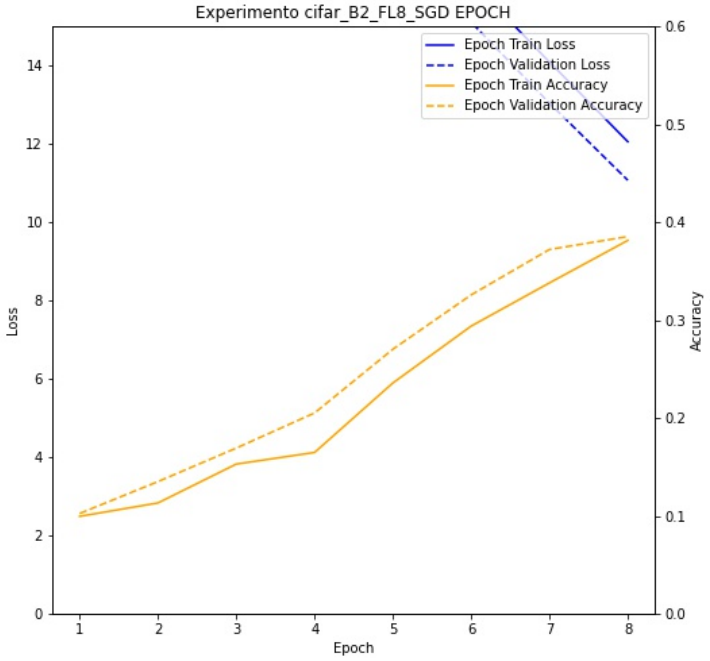


FIGURA A.4. Caso de aplicación en CIFAR10 - FL=8, Optimizador=SGD.

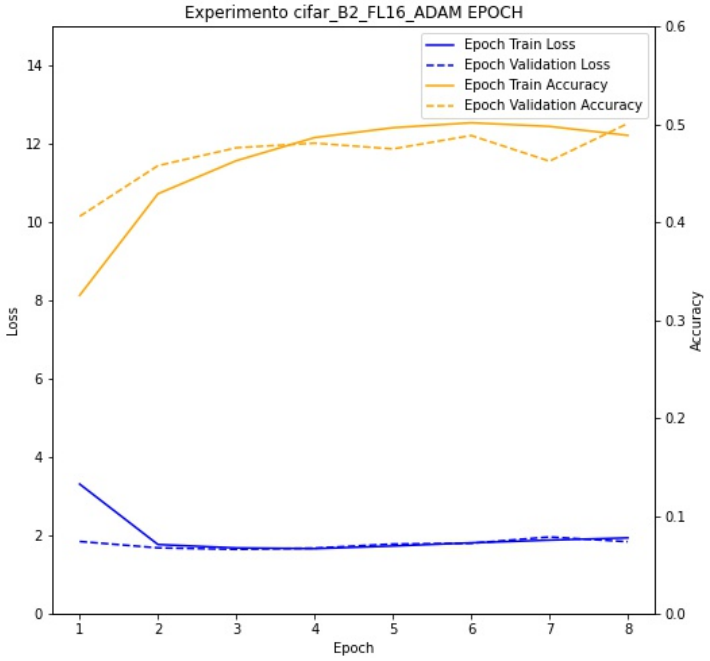


FIGURA A.5. Caso de aplicación en CIFAR10 - FL=16, Optimizador=ADAM.

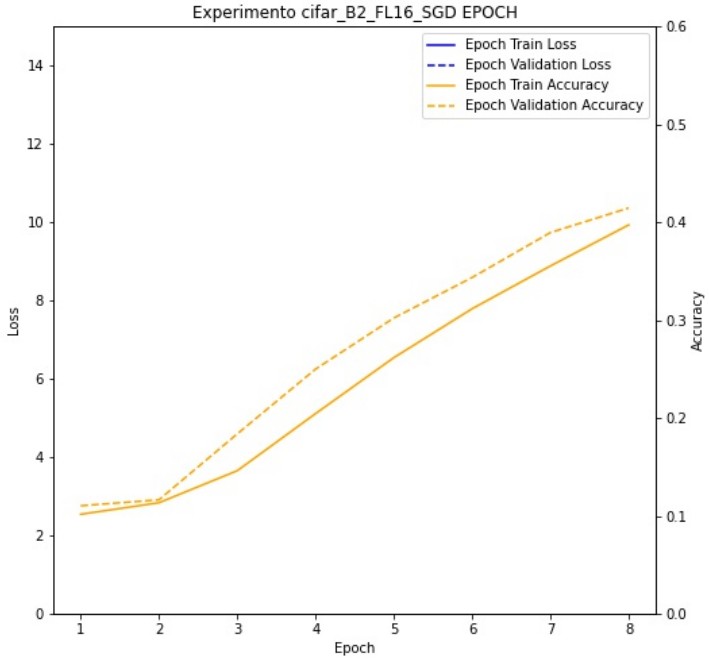


FIGURA A.6. Caso de aplicación en CIFAR10 - FL=16, Optimizador=SGD.

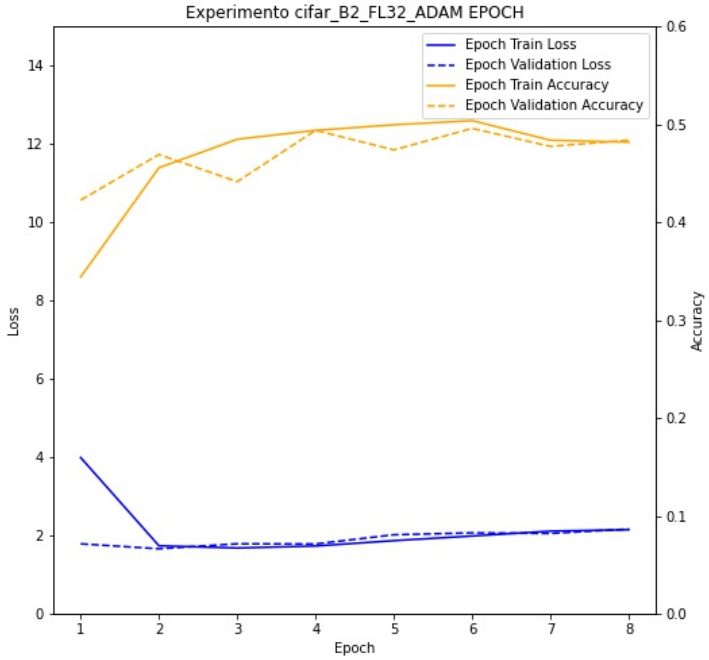


FIGURA A.7. Caso de aplicación en CIFAR10 - FL=32, Optimizador=ADAM.

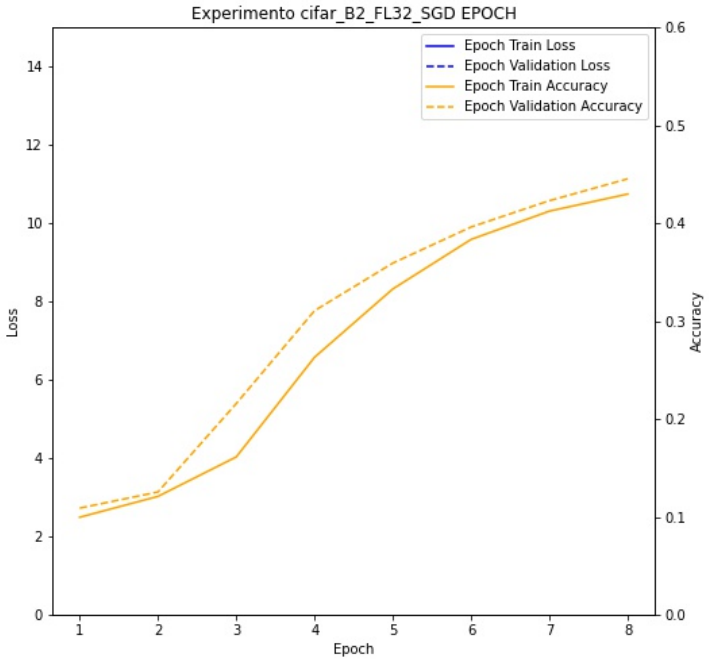


FIGURA A.8. Caso de aplicación en CIFAR10 - ssFL=32, Optimizador=SGD.

Bibliografía

[1] Statista. *Forecast growth of the artificial intelligence (AI) software market worldwide from 2019 to 2025*. https://www.statista.com/statistics/607960/worldwide-artificial-intelligence-market-growth/. (Visitado 17-03-2022).

[2] *Artificial Intelligence Chipsets Market Size And Forecast*.

[3] V. Gulshan et al. «Development and validation of a deep learning algorithm for detection of diabetic retinopathy in retinal fundus photographs.» En: *JAMA* (2016).

[4] A. Esteva et al. «Demartologist-level classification of skin cancer with deep neural networks.» En: *Nature VOL 542-546* (2017).

[5] L. P. Casalino F. Wang y D. Khullar. «Deep learning in medicine - promise, progress and challenges.» En: *JAMA* (2018).

[6] Joe D'Allegro. *How Google's Self-Driving Car Will Change Everything*. https://www.investopedia.com/articles/investing/052014/how-googles-selfdriving-car-will-change-everything.asp. (Visitado 17-03-2022).

[7] G. Biggi y J. Stilgoe. «Artificial Intelligence in Self-Driving Cars Research and Innovation: A Scientometric and Bibliometric Analysis.» En: (2021).

[8] B. Hrnjica y S. Softic. «Explainable AI in Manufacturing: A Predictive Maintenance Case Study.» En: (2020).

[9] Chandan K. Sahu, Crystal Young y Rahul Rai. «Artificial intelligence (AI) in augmented reality (AR)-assisted manufacturing applications: a review». En: *International Journal of Production Research* 59.16 (2021), págs. 4903-4959. DOI: 10.1080/00207543.2020.1859636. URL: \url{https://doi.org/10.1080/00207543.2020.1859636}.

[10] Liudmila Alekseeva y col. «The demand for AI skills in the labor market». En: *Labour Economics* 71 (2021), pág. 102002. ISSN: 0927-5371. DOI: \url{https://doi.org/10.1016/j.labeco.2021.102002}. URL: \url{https://www.sciencedirect.com/science/article/pii/S0927537121000373}.

[11] Stanford Institute for Human-Centered Artificial Intelligence. *Artificial Intelligence Index Report 2022*. 2022.

[12] *State of Machine Learning and Data Science 2021*. 2021. URL: \url{https://storage.googleapis.com/kaggle-media/surveys/Kaggle's%20State%20of%20Machine%20Learning%20and%20Data%20Science%202021.pdf}.

[13] Leslie N. Smith. «A disciplined approach to neural network hyper-parameters: Part 1 - learning rate, batch size, momentum, and weight decay». En: *CoRR* abs/1803.09820 (2018). arXiv: 1803.09820. URL: http://arxiv.org/abs/1803.09820.

[14] James Bergstra y Yoshua Bengio. «Random Search for Hyper-Parameter Optimization». En: *J. Mach. Learn. Res.* 13.1 (feb. de 2012), 281–305. ISSN: 1532-4435.

[15] Martin Wistuba, Arlind Kadra y Josif Grabocka. *Dynamic and Efficient Gray-Box Hyperparameter Optimization for Deep Learning*. 2022. arXiv: 2202.09774 [cs.LG].

[16] Greg Yang y col. *Tensor Programs V: Tuning Large Neural Networks via Zero-Shot Hyperparameter Transfer*. 2022. arXiv: 2203.03466 [cs.LG].
[17] François Chollet. *Deep Learning with Python*. Manning Publications, 2017.
[18] Tom Mitchell. *Machine Learning*. McGraw Hill, 1997.
[19] Ian Goodfellow, Yoshua Bengio y Aaron Courville. *Deep Learning*. http://www.deeplearningbook.org. MIT Press, 2016.
[20] Kevin P. Murphy. *Machine Learning: A probabilistic perspective*. MIT Press, 2012.
[21] A. A. Faisal y C. S. Ong M. P. Deisenroth. *Mathematics for machine learning*. https://mml-book.com/. Cambridge University Press, 2019.
[22] J. Stewart. *Calculus.* octava. Cengage learning, 2016.
[23] A. Cauchy. «Método general para la resolución de sistemas de ecuaciones simultáneas.» En: *Rendición de cuentas a la academia de ciencias* (1847).
[24] Alexander Amini y col. «Spatial Uncertainty Sampling for End-to-End Control». En: *CoRR* abs/1805.04829 (2018). arXiv: 1805.04829. URL: http://arxiv.org/abs/1805.04829.
[25] G. E. Hinton y R. J. Williams D. E. Rumelhart. «Learning representations by back-propagating errors.» En: *Nature VOL. 323* (1986).
[26] Christopher Bishop. *Pattern Recognition and Machine Learning*. Springer, ene. de 2006. URL: https://www.microsoft.com/en-us/research/publication/pattern-recognition-machine-learning/.
[27] F. Rosenblatt. «The perceptron: A probabilistic model for information storage and organization in the brain.» En: *Psycological Review* (1958).
[28] Warren S. McCulloch y Walter Pitts. «A logical calculus of the ideas immanent in nervous activity.» En: *Bulletin of Mathematical Biology* (1990).
[29] Bernard Widrow y Marcian E. Hoff. «Adaptive switching circuits.» En: (1960).
[30] Leslie N. Smith y Nicholay Topin. «Deep Convolutional Neural Network Design Patterns». En: *CoRR* abs/1611.00847 (2016). arXiv: 1611.00847. URL: http://arxiv.org/abs/1611.00847.
[31] Jonathan Frankle y Michael Carbin. «The Lottery Ticket Hypothesis: Training Pruned Neural Networks». En: *CoRR* abs/1803.03635 (2018). arXiv: 1803.03635. URL: http://arxiv.org/abs/1803.03635.
[32] Thomas Elsken, Jan Hendrik Metzen y Frank Hutter. «Neural Architecture Search: A Survey». En: (2018). DOI: 10.48550/ARXIV.1808.05377. URL: https://arxiv.org/abs/1808.05377.
[33] Alejandro Baldominos, Yago Saez y Pedro Isasi. «On the Automated, Evolutionary Design of Neural Networks: Past, Present, and Future». En: *Neural Comput. Appl.* 32.2 (ene. de 2020), 519–545. ISSN: 0941-0643. DOI: 10.1007/s00521-019-04160-6. URL: https://doi.org/10.1007/s00521-019-04160-6.
[34] *Neural Networks: Tricks of the Trade*. Vol. 7700. Springer, 2012. DOI: https://doi.org/10.1007/978-3-642-35289-8.
[35] Yoshua Bengio. «Practical recommendations for gradient-based training of deep architectures». En: *CoRR* abs/1206.5533 (2012). arXiv: 1206.5533. URL: http://arxiv.org/abs/1206.5533.
[36] Matthew D. Zeiler. «ADADELTA: An Adaptive Learning Rate Method». En: *CoRR* abs/1212.5701 (2012). arXiv: 1212.5701. URL: http://arxiv.org/abs/1212.5701.
[37] Diederik P. Kingma y Jimmy Ba. «Adam: A Method for Stochastic Optimization». En: *3rd International Conference on Learning Representations,*

ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings. Ed. por Yoshua Bengio y Yann LeCun. 2015. URL: http://arxiv.org/abs/1412.6980.

[38] Rahul Yedida y Snehanshu Saha. «A novel adaptive learning rate scheduler for deep neural networks». En: *CoRR* abs/1902.07399 (2019). arXiv: 1902.07399. URL: http://arxiv.org/abs/1902.07399.

[39] Leslie N. Smith. «Cyclical Learning Rates for Training Neural Networks». En: *2017 IEEE Winter Conference on Applications of Computer Vision, WACV 2017, Santa Rosa, CA, USA, March 24-31, 2017*. IEEE Computer Society, 2017, págs. 464-472. DOI: 10.1109/WACV.2017.58. URL: https://doi.org/10.1109/WACV.2017.58.

[40] Leslie N. Smith. «No More Pesky Learning Rate Guessing Games». En: *CoRR* abs/1506.01186 (2015). arXiv: 1506.01186. URL: http://arxiv.org/abs/1506.01186.

[41] Long Wen, Xinyu Li y Liang Gao. «A New Reinforcement Learning Based Learning Rate Scheduler for Convolutional Neural Network in Fault Classification». En: *IEEE Transactions on Industrial Electronics* 68.12 (2021), págs. 12890-12900. DOI: 10.1109/TIE.2020.3044808.

[42] Saptarshi Sengupta y col. «A Review of Deep Learning with Special Emphasis on Architectures, Applications and Recent Trends». En: *CoRR* abs/1905.13294 (2019). arXiv: 1905.13294. URL: http://arxiv.org/abs/1905.13294.

[43] Alex Krizhevsky, Ilya Sutskever y Geoffrey E Hinton. «ImageNet Classification with Deep Convolutional Neural Networks». En: *Advances in Neural Information Processing Systems*. Ed. por F. Pereira y col. Vol. 25. Curran Associates, Inc., 2012. URL: https://proceedings.neurips.cc/paper/2012/file/c399862d3b9d6b76c8436e924a68c45b-Paper.pdf.

[44] L. Bottou. «Online learning and stochastic approximations. (Versión revisada)». En: (2018).

[45] L. Bottou y O. Bousquet. «The tradeoffs of large scale learning». En: (2007).

[46] Nikhil Ketkar. «Stochastic Gradient Descent». En: oct. de 2017, págs. 111-130. ISBN: 978-1-4842-2765-7. DOI: 10.1007/978-1-4842-2766-4_8.

[47] F. Chollet. *Adam*. https://keras.io/api/optimizers/adam/. (Visitado 08-03-2022).

[48] Yann N. Dauphin y col. «RMSProp and equilibrated adaptive learning rates for non-convex optimization». En: *CoRR* abs/1502.04390 (2015). arXiv: 1502.04390. URL: http://arxiv.org/abs/1502.04390.

[49] Xavier Glorot y Yoshua Bengio. «Understanding the difficulty of training deep feedforward neural networks». En: *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*. Ed. por Yee Whye Teh y Mike Titterington. Vol. 9. Proceedings of Machine Learning Research. Chia Laguna Resort, Sardinia, Italy: PMLR, mayo de 2010, págs. 249-256. URL: https://proceedings.mlr.press/v9/glorot10a.html.

[50] F. Chollet. *Layer weight regularizers*. https://keras.io/api/layers/regularizers/. (Visitado 27-03-2022).

[51] K. Fukushima. «Neocognition: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift position». En: *Biological cybernetics* 36 (1980), págs. 193-202.

[52] *What is ReLu?* https://deepai.org/machine-learning-glossary-and-terms/relu. (Visitado 21-03-2022).

[53] Hossam H. Sultan, Nancy Salem y Walid Al-Atabany. «Multi-Classification of Brain Tumor Images Using Deep Neural Network». En: *IEEE Access* PP (mayo de 2019), págs. 1-1. DOI: 10.1109/ACCESS.2019.2919122.

[54] Chigozie Nwankpa y col. «Activation Functions: Comparison of trends in Practice and Research for Deep Learning». En: *CoRR* abs/1811.03378 (2018). arXiv: 1811.03378. URL: http://arxiv.org/abs/1811.03378.

[55] Alex Krizhevsky. *Learning multiple layers of features from tiny images*. Inf. téc. 2009.

[56] Patrick Helber y col. *EuroSAT: A Novel Dataset and Deep Learning Benchmark for Land Use and Land Cover Classification*. 2017. arXiv: 1709.00029 [cs.CV].

[57] Google. *Training and test sets*. https://developers.google.com/machine-learning/crash-course/training-and-test-sets/video-lecture?hl=es_419. (Visitado 26-02-2022).

[58] Ben Allison. *Respuesta a "Is there a rule-of-thumb for how to divide a dataset into training and validation sets?"* https://stackoverflow.com/questions/13610074/is-there-a-rule-of-thumb-for-how-to-divide-a-dataset-into-training-and-validation. (Visitado 26-02-2022).

[59] Ingolifs. *Respuesta a "How do I know what proportion of data to take for testing in deep learning?"* https://stats.stackexchange.com/questions/352041/how-do-i-know-what-proportion-of-data-to-take-for-testing-in-deep-learning. (Visitado 26-02-2022).

[60] Karen Simonyan y Andrew Zisserman. «Very deep convolutional networks for large-scale image recognition.» En: *ICLR* (2015).

[61] F. Chollet. *Image classification from scratch*. https://keras.io/examples/vision/image_classification_from_scratch/. (Visitado 26-04-2022).

[62] Martin Görner. *Convolutional neural networks, with Keras and TPUs*. https://codelabs.developers.google.com/codelabs/keras-flowers-convnets/. (Visitado 26-04-2022).