UNIVERSIDAD NACIONAL DE CUYO

Aprendizaje y Análisis de Redes Neuronales Artificiales Profundas

por

Agustina Dinamarca

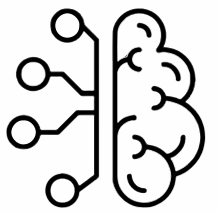
Director: Dr. David A. Monge

Codirector: Dr. Pablo F. Kaluza

Tesis de Seminario de Investigación presentada como requerimiento para la obtención del grado de
Licenciada en Ciencias Básicas con Orientación en Física

en la
Facultad de Ciencias Exactas y Naturales

Junio 2018



"What we want is a machine that can learn from experience. "

Alan Turing

Resumen

Esta tesis trata sobre las redes neuronales profundas (RNPs), modelos computacionales de aprendizaje autónomo, inspirados en el funcionamiento del sistema nervioso de los seres vivos. Actualmente, las RNPs han logrado un desempeño muy notable en tareas de Inteligencia Artificial. Sin embargo, es bien sabido que el entrenamiento de estos modelos viene acompañado de un alto costo y complejidad computacional. Por otro lado, se diseñan redes con gran variedad de formas y tamaños dependiendo de su aplicación. Por lo tanto, muchos modelos de RNPs han sido desarrollados, e incluso mejorados, para lograr cada vez mayor eficacia y eficiencia en las tareas para las cuales fueron diseñados. En particular, la tesis se centra en dos tipos de redes muy populares en el área de Visión Computacional: las redes densas (RDs) y las redes convolucionales (RCs).

Los objetivos principales de esta investigación fueron medir cuán eficaces y eficientes son distintas configuraciones de RDs frente a RCs en una tarea de clasificación multiclase. Para cumplir con los objetivos fue necesario: aprender RDs y RCs que clasifiquen imágenes; evaluar el desempeño de cada red en términos de exactitud de clasificación y tiempo de aprendizaje; y comparar aquellas cantidades entre ambos tipos de modelo.

Los resultados obtenidos fueron parcialmente consistentes con las hipótesis propuestas. Los más relevantes fueron que el 91 % de las RCs aprendidas fueron mínimamente un 9.11 % más eficaces que las RDs. Esto indica que las primeras presentan mejor capacidad de aprender patrones complejos que las segundas. Tal capacidad se debe a que las RCs poseen mayor cantidad de unidades ocultas que las RDs, sumado al hecho de que cada unidad de una capa convolucional tiene conexiones locales con regiones de la capa anterior, y comparte parámetros con el resto de las unidades de la misma capa. Por otro lado, el 82 % de las RDs aprendidas fueron como mínimo 6.4 min más eficientes que las RCs. Estos resultados se atribuyen fundamentalmente a la cantidad y complejidad de operaciones que las redes deben efectuar y, en menor medida, a la cantidad de parámetros que las mismas deben aprender.

Por último, los resultados obtenidos en este trabajo sirven para entender el impacto de las variaciones estructurales de las RNPs en sus desempeños. Esta clase de estudios, junto a otros, permite incorporar una correcta y adecuada flexibilidad a cualquier dispositivo de RNPs para que funcione con mejor eficacia y eficiencia.

Palabras Clave: aprendizaje profundo, clasificación multiclase, redes convolucionales, redes densas, eficacia, eficiencia, parámetros, patrones, unidades ocultas.

Agradecimientos

Quiero expresar mi más sincero agradecimiento a todas aquellas personas que han colaborado en la realización del presente trabajo. En especial al Dr. David A. Monge, director de esta tesis, por la orientación, el seguimiento y la supervisión continua de la misma, pero por sobre todo, la motivación y el apoyo recibido en este tiempo. Asimismo, al Dr. Pablo F. Kaluza, codirector, por sus aportes y por su buena disposición siempre.

También deseo mostrar mi agradecimiento a dos instituciones: el Instituto para las Tecnologías de la Información y las Comunicaciones (ITIC) y la Facultad de Ciencias Exactas y Naturales (FCEN) de la Universidad Nacional de Cuyo (UNCuyo), por haber puesto sus recursos a mi disposición cuando los solicité.

Además, agradezco las correcciones y sugerencias de mi director y codirector sobre el póster basado en resultados preliminares de la tesis, que expuse en la 102° Reunión de la Asociación Física Argentina (102° RAFA), realizada en septiembre de 2017 en la Ciudad de La Plata, Buenos Aires, Argentina. También, agradezco enormemente la ayuda económica otorgada por la FCEN y por la Asociación Física Argentina (AFA) para que pueda asistir al evento.

Por último, agradezco a mi familia y amigos, especialmente a María Cristina Rodríguez, por su cariño y apoyo en todo momento en estos últimos años.

Mendoza, Argentina — Agustina Dinamarca
Junio 2018

Índice General

Índice de Figuras

Índice de Tablas

Acrónimos y Siglas

Adam	Adaptative Moment Estimation
gap	Global Average-Pooling
ap	Average-Pooling
c	Capa Convolucional
CIFAR	**C**anadian **I**nstitute **F**or **A**dvanced **R**esearch
CPU	**C**entral **P**rocessing **U**nit
CV	k-fold **C**ross **V**alidation
CGN	**C**ontraste **G**lobal **N**ormalizado
d	Capa Densa
GPU	**G**raphics **P**rocessing **U**nit
IA	**I**nteligencia **A**rtificial
lc	Capa Localmente Conectada
mp	Max-Pooling
PC	**P**ersonal **C**omputer
PReLU	**P**arametric **R**ectified **L**inear **U**nit
RC	**R**ed **C**onvolucional
RD	**R**ed **D**ensa
RF	**R**ed **F**eedforward
RNP	**R**ed **N**euronal **P**rofunda
ReLU	**R**ectified **L**inear **U**nit
RGB	**R**ed, **G**reen, **B**lue
SGD	**S**tochastic **G**radient **D**escent

Notación

Números y Arreglos

a	Un escalar (entero o real)
$\boldsymbol{a}$	Un vector
$\boldsymbol{A}$	Una matriz
$\mathbf{A}$	Un tensor

Conjuntos

$\mathbb{A}$	Un conjunto
$\mathbb{R}$	El conjunto de los números reales
$\mathbb{R}^n$	El conjunto de las *n-tuplas* de números reales
$\{0, 1, ..., n\}$	El conjunto de todos los números enteros entre 0 y n
$[a, b]$	El intervalo real incluyendo a a y a b
$(a, b]$	El intervalo real excluyendo a a pero incluyendo a b

Índices

a_i	Elemento i del vector $\boldsymbol{a}$, con índices comenzando en 1
$A_{i,j}$	Elemento i,j de la matriz $\boldsymbol{A}$

Álgebra Lineal

$\boldsymbol{A}^\top$	Traspuesta de la matriz $\boldsymbol{A}$
$\boldsymbol{A} \odot \boldsymbol{B}$	Producto de Hadamard de $\boldsymbol{A}$ y $\boldsymbol{B}$

Cálculo

$\frac{dy}{dx}$	Derivada de y con respecto a x
$\frac{\partial y}{\partial x}$	Derivada parcial de y con respecto a x

$\nabla_{\boldsymbol{x}}\ y$	Gradiente de y con respecto a $\boldsymbol{x}$

Probabilidad

$P(\mathrm{a})$	Una distribución de probabilidad sobre una variable discreta
$\mathrm{a} \sim P$	Una variable aleatoria tiene una distribución P

Funciones

$f : \mathbb{A} \to \mathbb{B}$	La función f con dominio $\mathbb{A}$ e imagen $\mathbb{B}$
$f(\boldsymbol{x}; \boldsymbol{\theta})$	Una función de $\boldsymbol{x}$ parametrizada por $\boldsymbol{\theta}$
$f * g$	Convolución de f y g
$\log x$	El logaritmo natural de x
$\sigma(x)$	Función Logística Sigmoide, $1/(1 + e^{-x})$
$\delta_{i,j}$	Delta de Kronecker, 1 si $i = j$, 0 en otro caso

Conjunto de Datos

$\mathbb{X}$	Un conjunto de ejemplos de entrenamiento
$\boldsymbol{x}^{(i)}$	El *i-ésimo* ejemplo de un conjunto de datos
$y^{(i)}$ o $\boldsymbol{y}^{(i)}$	La etiqueta asociada a $\boldsymbol{x}^{(i)}$ en aprendizaje supervisado

A quienes el destino otorgó un nuevo y promisorio rumbo a partir del cambio más impensado...

Capítulo 1

Introducción

"If you just have a single problem to solve, then fine, go ahead and use a neural network. But if you want to do science and understand how to choose architectures, or how to go to a new problem, you have to understand what different architectures can and cannot do."

Marvin Minsky

En este capítulo se presenta una introducción a las redes neuronales artificiales y al aprendizaje profundo. Para ello, se exponen los antecedentes, la contextualización temática y se construye el problema de investigación. En relación a esto se exponen los correspondientes objetivos e hipótesis del estudio. Finalmente, se describe el tipo de tesis, se mencionan los aportes, la relevancia y la justificación del trabajo.

1.1. Antecedentes y Contextualización Temática

Los sistemas nerviosos de los seres vivos se organizan en forma de redes complejas con nodos (neuronas) y conexiones (sinapsis). Dichas redes son capaces de realizar tareas de percepción y detección de patrones de manera sumamente efectiva y eficiente (Rojas, 1996; London & Häusser, 2005). Por ello, han sido objeto de estudio en numerosas investigaciones y han servido de inspiración para el desarrollo de algoritmos y modelos computacionales de aprendizaje. Ejemplo de ello son las diversas aplicaciones en Visión Computacional, tal como detección de cáncer (Esteva *et* al., 2017; Mohsen *et* al., 2017) y manejo de vehículos autónomos (Bechtel *et* al., 2018); y en reconocimiento del habla, tal como reconocimiento de emociones (Chernykh *et* al., 2017); por solo mencionar algunas.

En primer lugar, un algoritmo de aprendizaje es aquel que es capaz de aprender y mejorar automáticamente a partir de la experiencia, es decir, a través de datos empíricos. Para

lograr esto, es necesario crear programas computacionales que sean capaces de aprender cómo hacer ciertas tareas o actividades, que las personas suelen hacer de forma intuitiva y a veces hasta automáticamente, pero sin ser explícitamente programadas.

Dentro de los varios modelos de aprendizaje, uno muy popular, y en sus comienzos inspirado en la estructura y funcionamiento de las redes neuronales biológicas, son las redes neuronales artificiales.

Es bien sabido que la unidad básica estructural y funcional del cerebro es la neurona. Según el modelo biológico de una neurona genérica (correspondiente a una neurona motora típica), la misma está formada principalmente por tres partes: dendritas, cuerpo celular o soma y axón (Rojas, 1996). Como ilustra la Figura 1.1, una neurona recibe señales de entrada por sus dendritas (señales producidas por otras neuronas) y genera una señal de salida a través de su único axón. Este último eventualmente se ramifica en las denominadas terminaciones sinápticas, y se conecta a las dendritas de otras neuronas para transmitir la señal de salida. Precisamente, la región de conexión y comunicación entre neuronas se la conoce como sinapsis. Para tener una idea, en el sistema nervioso humano hay aproximadamente 86 mil millones de neuronas que están conectadas unas con otras haciendo un total de entre 10^{14} a 10^{15} sinapsis. Asimismo, cabe mencionar que aquel modelo de neurona biológica es bastante simple, pues existen muchos tipos de neuronas, cada una con diferentes características (Azevedo *et al.*, 2009; Karpathy, 2017).

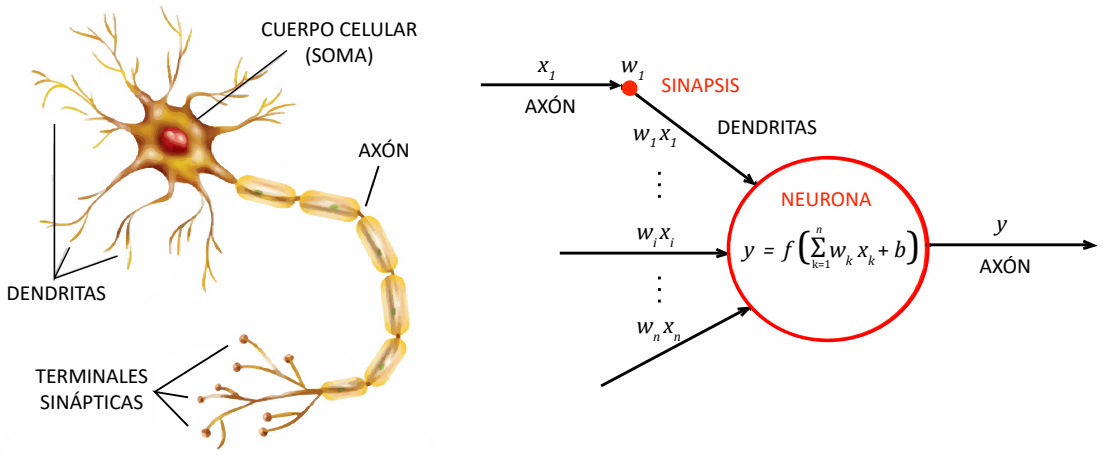


FIGURA 1.1: Modelo biológico de una neurona genérica, comúnmente una neurona motora típica (izquierda) y el respectivo modelo matemático (derecha).

Por otro lado, está el modelo matemático de una neurona, también conocido como neurona artificial. Como se muestra en la Figura 1.1, ésta recibe n entradas (x_1,..., x_n) representadas como un vector $\boldsymbol{x} \in \mathbb{R}^n$. Aquellas entradas no son más que las señales de salida de otras neuronas. Asimismo, cada entrada x_1,..., x_n, tiene asignado un peso o ponderación, $w_1, ..., w_n$ respectivamente. Las entradas ponderadas son combinadas en

una suma y luego transformadas a través de una función de activación (o de transferencia) f no lineal, produciendo de esta manera una salida y. En otras palabras, la neurona lleva a cabo un producto punto entre las entradas y sus respectivos pesos, suma un sesgo (constante real) y aplica luego una función no lineal que puede seleccionarse de forma arbitraria. Un ejemplo típico de dicha función es la logística sigmoide σ, donde $\sigma(x) = 1/(1 + e^{-x})$. La misma recibe un número real como entrada (el valor de la señal luego de la suma) y retorna un número real entre 0 y 1.

En consecuencia, se puede construir una red neuronal artificial mediante un conjunto de neuronas artificiales, es decir, mediante un conjunto de funciones $f^{(1)}, ..., f^{(k)}$, y conectando comúnmente la salida de cada una a las entradas de otras diferentes. De esta manera, las redes neuronales artificiales no son más que redes de funciones, típicamente representadas mediante la composición de varias funciones $f(\boldsymbol{x}) = f^{(k)}(\cdots(f^{(1)}(\boldsymbol{x}))$. Esto hace que los diferentes modelos de redes neuronales difieran principalmente en las funciones de activación utilizadas, el patrón de interconexión, e inclusive el tiempo de trasmisión de la información.

Es importante señalar que la característica clave de la sinapsis, el escalar las señales de entrada por factores (pesos), es la manera en la que se cree que el cerebro aprende. Por lo tanto, distintos pesos dan como resultado diferentes respuestas a una entrada. De esta manera, se puede decir que el aprendizaje es el ajuste de los pesos en respuesta a un estímulo, mientras que la estructura del cerebro no cambia. Este hecho hace que el cerebro sea una excelente inspiración para un algoritmo de aprendizaje.

Cabe mencionar que si bien las redes neuronales artificiales se originaron desde una perspectiva neurocientífica, como modelos computacionales del aprendizaje biológico, la investigación moderna presenta un perfil ingenieril. Por lo tanto, hoy en día las redes neuronales artificiales son más precisamente modelos de aproximación de funciones diseñados para lograr generalización estadística.

Dada una red neuronal artificial, su entrada, representada típicamente mediante un vector $\boldsymbol{x}$, es el conjunto de valores que constituyen la información a ser analizada por la misma. Por ejemplo, aquellos valores pueden ser las intensidades de píxel de una imagen, las amplitudes muestreadas de una señal de audio, o inclusive la representación numérica de un estado correspondiente a un sistema. Una vez que ésta ingresa a la red, sufre transformaciones a través de múltiples operaciones llevadas a cabo por cada neurona, de manera tal que el conjunto final de aquellas operaciones genera la salida de la red, un valor real y o un vector $\boldsymbol{y}$. Por ejemplo, la probabilidad de que una imagen contenga un objeto particular, la probabilidad de que una señal de audio contenga una determinada palabra, un cuadro alrededor de un objeto indicando su ubicación en una imagen o video, entre otras.

Si bien existen diferentes estructuras o arquitecturas de redes neuronales artificiales, un modelo típico son las denominadas redes hacia adelante (*feedforward neural networks*, en inglés). En este trabajo se referirá a ellas como redes *feedforward* (RFs). Estos modelos se los llama de aquella manera debido a que la información fluye a través de la red hacia adelante, es decir, desde la función evaluada en $\boldsymbol{x}$ (entrada) a través de las operaciones intermedias usadas para definir f, hasta el cálculo de la salida y o $\boldsymbol{y}$. Como ilustra la Figura 1.3, estos son modelos compuestos por unidades y conexiones (representando neuronas y sinapsis respectivamente) organizadas en diferentes capas y en donde las conexiones no forman ciclos o bucles. Las unidades en una misma capa funcionan en paralelo y cada una de ellas puede interpretarse como una función no lineal que mapea un vector a un escalar. De esta manera, una unidad toma varias entradas de la capa anterior para computar su propio valor de activación o salida.

El objetivo de una RF es aproximar alguna función f^*. La red en sí define un mapa $\boldsymbol{y} = f(\boldsymbol{x}; \boldsymbol{\theta})$ en el que hay que aprender el valor de los parámetros (pesos y sesgos) $\boldsymbol{\theta}$ que resultan en la mejor función f de aproximación. Uno de los hechos más sorprendentes sobre las redes neuronales artificiales es que pueden representar cualquier función, es decir, están dotadas de cierta universalidad. Precisamente, el Teorema de Aproximación Universal establece que no importa que función se quiera calcular, se sabe que existe una red neuronal artificial que puede hacerlo (Cybenko, 1989; Hornik *et al.*, 1989; Nielsen, 2015).

Por su parte, el Aprendizaje Profundo (*Deep Learning*, en inglés) es una subárea del Aprendizaje Automático (*Machine Learning*, en inglés) que estudia el aprendizaje de redes neuronales profundas (RNPs), es decir, redes compuestas por múltiples capas. El apilado de múltiples capas permite a los modelos capturar y aprender patrones, representaciones de los datos, con mayor complejidad y con múltiples niveles de abstracción que redes de una sola capa o pocas. La cantidad de capas existente, determina la profundidad de estos modelos y, por ende, su capacidad para capturar patrones más complejos. En consecuencia, esta jerarquía profunda de características permite a las redes lograr un mejor desempeño en muchas tareas de Inteligencia Artificial (IA). Sin embargo, esto incrementa considerablemente la cantidad de parámetros a ajustar (pesos y sesgos) haciendo más difícil el proceso de aprendizaje (Rojas, 1996; Glorot & Bengio, 2010).

Ahora bien, considérese una RNP, específicamente una RF que lleva a cabo una tarea de clasificación multiclase de imágenes. Se le proporciona a la misma una imagen de entrada $\boldsymbol{x}$ y, como resultado la salida de la red retorna un vector $\boldsymbol{y}$ con los valores de probabilidades correspondientes a cada clase o categoría (previamente definida) a la cual podría pertenecer. Como muestra la Figura 1.2, la clase con la probabilidad más alta indica la categoría más probable a la cual pertenece la imagen. Para lograr aquello

es necesario primero entrenar o aprender la RF. El entrenamiento de la red f permite que se aproxime a la función solución del problema f^* para cada entrada o ejemplo de entrenamiento $\boldsymbol{x}$.

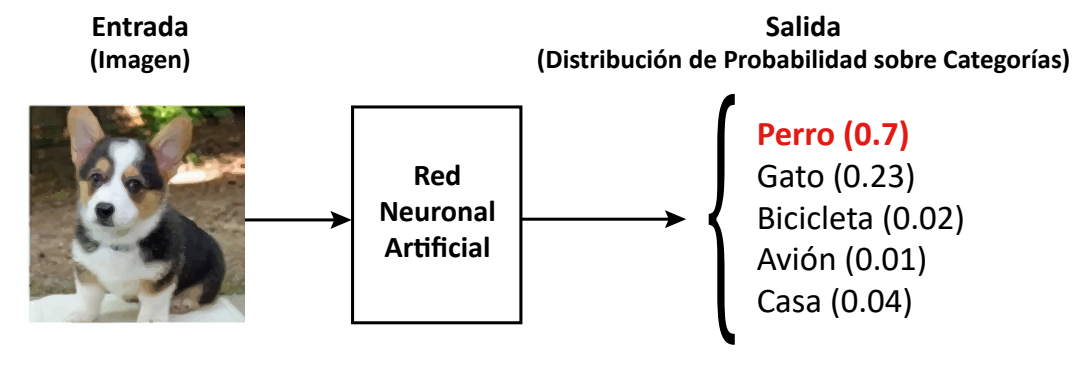


FIGURA 1.2: Ilustración de la tarea de clasificación multiclase. La red neuronal recibe como entrada una imagen y retorna como salida una distribución de probabilidad sobre categorías o clases previamente definidas. La categoría con la probabilidad más alta indica la clase más probable a la cual pertenece la imagen.

El entrenamiento de una RNP consiste principalmente en determinar los valores de los pesos y sesgos que maximizan la probabilidad de la clase correcta y que minimizan las probabilidades de las clases incorrectas. Hay varias formas de aprender aquellos parámetros, el enfoque más común se denomina aprendizaje supervisado, donde todas las muestras de entrenamiento están etiquetadas. Por ejemplo, en clasificación multiclase de imágenes, cada imagen está acompañada de la categoría correcta a la cual pertenece.

La manera de evaluar la calidad de un conjunto particular de parámetros de una red está basada en que tan bien la clase predicha por la misma se aproxima a la clase verdadera para cada ejemplo de entrenamiento. Aquel grado de aproximación se cuantifica mediante una función de costo o error. Por lo tanto, el objetivo de entrenar RNPs es encontrar un conjunto de parámetros que minimicen el error o costo promedio global en un conjunto de entrenamiento.

Cabe mencionar que en el aprendizaje supervisado los ejemplos de entrenamiento especifican directamente lo que la capa de salida de la red debe hacer en cada punto $\boldsymbol{x}$, es decir, debe producir una valor que sea cercano a $\boldsymbol{y}$. Sin embargo, el comportamiento de las demás capas de la red no está especificado de forma directa por los datos de entrenamiento. El algoritmo de aprendizaje debe decidir como usar aquellas capas para producir la salida deseada, es decir, implementar la mejor aproximación a f^*. Debido que los datos de entrenamiento no muestran la salida deseada para las capas intermedias, se las denomina a aquellas capas ocultas.

Al entrenar una RNP, los valores de los parámetros (pesos y sesgos) usualmente se actualizan usando un proceso de optimización llamado gradiente descendente (*gradient*

descent, en inglés). Para actualizar dichos valores se usa un múltiplo del gradiente de la función de costo relativo a cada parámetro, que es la derivada parcial de la función de error con respecto al peso o sesgo. Este gradiente indica cómo deben cambiar los valores de los pesos y sesgos para reducir el error. El proceso se repite de manera iterativa para reducir el costo promedio global. Asimismo, una forma eficiente de calcular las derivadas parciales del gradiente es a través de un algoritmo llamado retropropagación (*backpropagation*, en inglés). A grandes rasgos, este es una operación derivada de la regla de la cadena del Cálculo, que opera al pasar valores hacia atrás a través de la red para calcular cómo el error se ve afectado por cada parámetro.

Particularmente, interesa qué tan bien se desempeña la red neuronal frente a nuevos datos, nunca antes vistos por la misma, pues esto determina la calidad de su trabajo en la vida real. Por ello, se evalúa el desempeño del modelo usando datos de prueba, que es un conjunto separado del conjunto de entrenamiento, usado para entrenar la red.

Las RNPs presentan gran variedad estructural en cuanto a sus formas y tamaños dependiendo de su aplicación. Sus arquitecturas varían en cuanto a número y tipo de capas, así como también, cantidad de conexiones entre capas. Hay numerosos estudios en los que se proponen arquitecturas de redes, o mejoras de otras ya estudiadas, para lograr mejores exactitudes de desempeño y eficiencia en las tareas de IA para las cuales fueron diseñadas.

Existen distintos tipos de RFs. Típicamente, las conexiones existentes entre capas son densas, es decir, todas las unidades de una capa se conectan con las de la siguiente. Estos modelos con capas completamente conectadas se los conoce como perceptrones multicapa (*multilayer perceptrons*, en inglés), o bien redes densas (RDs), como se adoptará en el presente trabajo. Las capas completamente conectadas requieren gran cantidad de almacenamiento y cómputo. Sin embargo, en muchas aplicaciones se han removido conexiones entre capas haciendo los pesos cero sin afectar el desempeño. Esto resulta en capas parcialmente conectadas como ilustra la Figura 1.3. Un tipo de red muy conocida con capas con esta característica es la red convolucional (RC). La Figura 1.4 muestra un esquema de este tipo de red neuronal.

Las RCs tienen una estructura más fuertemente inspirada en la estructura y funcionamiento de las redes de neuronas de la corteza visual. A diferencia de las RDs, las RCs tienen estructura de conexiones menos densa, es decir, no todas las unidades entre capas se encuentran conectadas, si no que estas se encuentran relacionadas mediante operadores matemáticos denominados convoluciones. Un hecho destacable es que las convoluciones introducen tres conceptos que ayudan a obtener mejores modelos: (1) interacciones ralas, (2) compartición de parámetros entre distintas unidades, y (3) representaciones equivariantes. Las implicancias de dichos conceptos pueden resumirse en que la cantidad

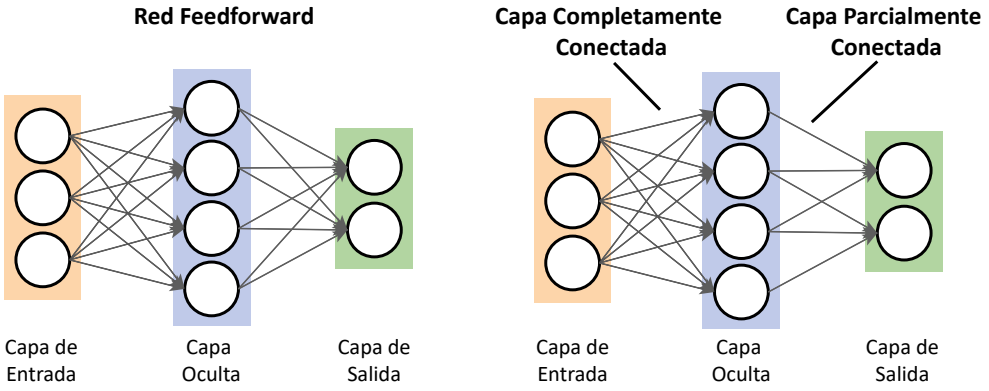


FIGURA 1.3: Esquema de una red *feedforward* con una capa de entrada, una oculta y otra de salida (izquierda); y diferencia entre capa completamente conectada o densa y parcialmente conectada en una red *feedforward* (derecha).

de parámetros a aprender se ve dramáticamente reducida (menores requerimientos de memoria y de tiempo de aprendizaje) pero más importante aún es que permiten aprender características (patrones) que no son afectados por desplazamientos en el tiempo o el espacio.

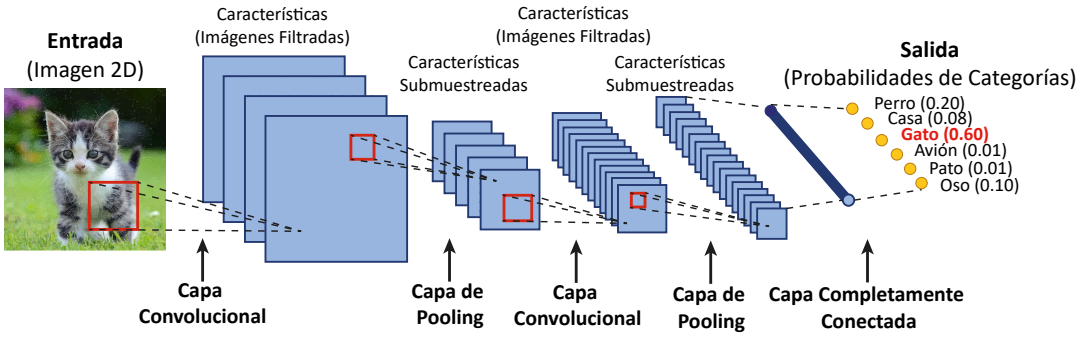


FIGURA 1.4: Esquema de una red convolucional.

Finalmente, esta tesis busca estudiar la eficacia y la eficiencia de RNPs, precisamente RDs frente a RCs, en una tarea popular: la clasificación multiclase de imágenes. Para ello, se deben aprender diferentes arquitecturas de redes correspondientes a esos modelos y medir sus desempeños en términos de exactitud de clasificación y tiempo de aprendizaje. Asimismo, se buscan medir correlaciones entre eficacia y eficiencia; y de exactitud y tiempo de aprendizaje frente a características estructurales, como lo son el número de unidades ocultas (neuronas) y número de parámetros a aprender (pesos y sesgos). Por último, se buscan identificar patrones aprendidos por las convoluciones en las RCs.

1.2. Situación Problemática

Las RNPs han logrado desempeños muy notables en tareas de IA, principalmente en problemas de clasificación y de detección de objetos en imágenes. Sin embargo, es bien sabido que el entrenamiento de estos modelos viene acompañada de un alto costo y complejidad computacional (Livni *et al.*, 2014; Liu *et al.*, 2015; Sze *et al.*, 2017).

Las RNPs vienen de gran variedad de formas y tamaños dependiendo de su aplicación: cada uno tiene una arquitectura diferente en cuanto a número, tipo y forma de capas, y conexiones entre capas. Por lo tanto, muchos modelos de RNPs han sido desarrollados en el último tiempo, e incluso mejorados, para lograr mayor eficacia y eficiencia en las tareas para las cuales fueron diseñados.

En particular, se han estudiado diversas arquitecturas de modelos de RNPs populares en los tradicionales problemas de clasificación de imágenes: las tradicionales RDs y las destacables RCs.

En consecuencia, entender aquellas variaciones estructurales en los modelos de RNPs y tendencias en sus desempeños es importante para incorporar una correcta flexibilidad a cualquier dispositivo de RNPs eficiente y eficaz (Sze *et al.*, 2017).

1.3. Formulación del Problema de Investigación

Dado que para cualquier tarea de IA, en particular la tarea de clasificación de imágenes, que haga uso de RNPs, se buscan modelos con arquitecturas eficientes y eficaces es necesario preguntarse:

a. ¿En qué medida es posible aprender RCs que sean más eficaces que RDs?

b. ¿En qué medida es posible aprender RCs que sean más eficientes que RDs?

Por último, para entender las tendencias en los resultados de eficacia y eficiencia a partir de las características estructurales de los modelos de RNPs es necesario preguntarse:

c. ¿Cuál es la correlación entre la eficacia y la eficiencia de las redes?

d. ¿Cuál es la correlación entre la eficacia de las redes y el número de unidades ocultas y de parámetros a aprender?

e. ¿Cuál es la correlación entre la eficiencia de las redes y el número de unidades ocultas y de parámetros a aprender?

f. Y en el único caso posible en cuanto a interpretabilidad, ¿qué tipo de patrones capturan las convoluciones en RCs?

1.4. Objetivos

Esta investigación tiene los siguientes objetivos principales:

- Medir que tan eficaces son distintas configuraciones de RDs al clasificar imágenes frente a RCs.
- Medir que tan eficientes son las distintas configuraciones de RDs al aprender a clasificar imágenes frente a RCs.

Para ello es necesario:

1. Aprender RDs y RCs que clasifiquen objetos en imágenes.
2. Evaluar los desempeños de cada red en términos de exactitud con la que clasifican correctamente imágenes y tiempo de aprendizaje.
3. Comparar los desempeños de las RDs frente a los alcanzados por las RCs.

Además, si se determinan los niveles de eficacia y eficiencia de las RNPs, los objetivos secundarios son los siguientes:

- Identificar la correlación entre la eficacia y la eficiencia de las redes.
- Identificar la correlación entre la eficacia de las redes y características estructurales de las mismas, tales como, el número de unidades ocultas y el número de parámetros a aprender.
- Identificar la correlación entre la eficiencia de las redes y características estructurales de las mismas, tales como, el número de unidades ocultas y el número de parámetros a aprender.
- Identificar los patrones capturados por las convoluciones en las RCs.

1.5. Hipótesis de Investigación

Hipótesis Primaria

Las RCs son siempre más eficaces y eficientes que las RDs en la tarea de clasificación multiclase de imágenes.

Las particulares características de las arquitecturas convolucionales: interacciones ralas, compartición de parámetros entre distintas unidades y representaciones equivariantes, tienen una fuerte implicancia en el proceso y resultado del aprendizaje. Precisamente, la cantidad de parámetros a aprender se ve drásticamente reducida y permiten capturar patrones que no son afectados por desplazamientos en el tiempo o el espacio. En consecuencia, es esperable que este tipo de redes tenga una mayor capacidad para aprender patrones de alto nivel y con mayor facilidad que las RDs. Esto se vería reflejado en exactitudes de clasificación mayores y tiempos de aprendizaje siempre menores a aquellas otras.

Hipótesis Secundarias

1. El número de unidades ocultas de una red tiene una correlación positiva frente a la exactitud de clasificación alcanzada por la red y frente al tiempo de aprendizaje. Esto es así porque un mayor número de unidades ocultas incrementa la capacidad del modelo para aprender patrones complejos.

2. El número de parámetros a aprender en una red tiene una correlación negativa frente a la exactitud de clasificación alcanzada por la red y frente al tiempo de aprendizaje. Esto se debe a que un mayor número de parámetros a determinar incrementa el fenómeno de sobreajuste (*overfitting*, en inglés), lo cual lleva a una pérdida de la capacidad de generalización. Esta pérdida puede interpretarse como que el modelo tiene a "memorizar" los datos presentados en lugar de capturar los patrones que subyacen en ellos. Asimismo, aumenta el tiempo que le lleva al algoritmo aprender el valor de cada uno de los parámetros.

3. Los patrones capturados por las convoluciones en las RCs son cada vez más abstractos y no interpretables en capas más profundas respecto a las primeras capas. Esto se debe a que la estructura de la RC permite generar una jerarquía de abstracciones a partir de la entrada.

1.6. Tipo de Tesis

La investigación que se propone es aplicada, dado que se busca obtener conocimiento a través de la construcción computacional de distintas arquitecturas de RNPs aplicadas a una tarea específica. Esto permitirá la medición de variables asociadas a su aprendizaje y estructura. Por ello, en esta tesis se emplea una metodología de carácter experimental: se realiza a partir del entrenamiento de RNPs, del cual se extraerán datos cuantitativos y cualitativos vinculados a sus aprendizajes.

Asimismo, es correlacional, pues se buscan medir variables, asociadas al aprendizaje y a la estructura de RNPs, dado que se pretende ver si están relacionadas entre sí.

Finamente, en menor medida, es descriptiva, dado que se pretende identificar y caracterizar un concepto (el de patrones capturados o características aprendidas) a partir de los resultados correspondientes a una clase de RNP.

1.7. Justificación

Las RNPs están siendo centrales en diversas áreas de Visión Computacional y diferentes arquitecturas se han propuesto para resolver problemas específicos, dependiendo de su aplicación. Inclusive las formas y tamaños más populares también han evolucionado rápidamente para mejorar la eficacia y la eficiencia en el desempeño de tareas modernas de IA, tal como la típica clasificación de objetos en imágenes. Cada uno de aquellos modelos tiene una arquitectura diferente en cuanto a número, tipo y forma de capas, y cantidad de conexiones entre capas. Entender el impacto de aquellas variaciones y tendencias en los resultados que se obtienen es de suma importancia para lograr incorporar una correcta y adecuada flexibilidad en cualquier dispositivo de RNPs eficiente y eficaz.

Es por ello que para lograr esas características en dispositivos que hacen uso de RNPs es necesario un estudio de eficiencia y eficacia de las redes en una tarea determinada. En particular, con dicho estudio en este trabajo se logrará explorar las eficiencia y eficacia de dos modelos populares de RNPs, denso y convolucional, para clasificar objetos en imágenes.

1.8. Aportes y Relevancia del Tema

El principal aporte de esta tesis es obtener conocimiento mediante el diseño y aprendizaje computacional de modelos de RNPs: RDs y RCs. Esto permitirá la medición de

variables asociadas a la estructura de las redes y al aprendizaje de las mismas en una tarea en particular: la clasificación multiclase de imágenes. El aporte consiste en nuevas arquitecturas de RDs y RCs que se desempeñen en aquella actividad, distintas a otras redes ya estudiadas en la literatura y con distintos grados de eficacia y eficiencia. Ese conjunto de RNPs mostrará en qué medida se pueden aprenden RDs más eficaces y eficientes que RCs, así como también, identificará correlaciones entre variables vinculadas a sus arquitecturas y aprendizajes.

El estudio es relevante en cuanto a que las RNPs se han convertido en herramientas centrales en diversas áreas de IA, entres ellas, Visión Computacional y Reconocimiento del Habla. Esto ha llevado a la propuesta y al desarrollo de diferentes arquitecturas de RNPs para resolver problemas específicos. Por consiguiente, se continúan realizando estudios para entender el impacto de las arquitecturas de RNPs en su capacidad para llevar a cabo tareas inteligentes. Esto es de suma importancia para lograr incorporar una correcta y adecuada flexibilidad en cualquier dispositivo de RNPs eficiente y eficaz.

Capítulo 2

Marco Teórico

Este marco teórico busca ofrecer luz acerca de los conceptos asociados a los modelos de RNPs densos y convolucionales, así como también, al proceso de aprendizaje de tipo supervisado para la tarea de clasificación multiclase de imágenes.

2.1. Algoritmo de Aprendizaje y Clasificación Multiclase

Mitchell (1997) y autores actuales como Goodfellow *et al.* (2016), sostienen la siguiente definición de algoritmo de aprendizaje:

"Un programa computacional se dice que aprende a partir de la experiencia E *con respecto a una clase de tareas* T *y con métrica de desempeño* P*, si su desempeño en* T*, medido por* P*, mejora* E*."*

Es importante mencionar que existen diversos tipos de tareas (clasificación, traducción, regresión, detección de anomalías, síntesis y muestreo, etc), métricas de desempeño y experiencias. Por lo tanto, a continuación se revisita dicha definición en el contexto de la tarea de clasificación multiclase de imágenes, pues este es el problema que deberán resolver las RNPs en el presente estudio.

En aquel caso, la tarea T consiste en clasificar una imagen que puede pertenecer a una, y solo una, de k categorías o clases mutuamente excluyentes. Para resolver esta tarea, al algoritmo de aprendizaje se le pide producir una función $f : \mathbb{R}^n \to \{1, ..., k\}$. Cuando $y = f(\boldsymbol{x})$, el modelo le asigna a una entrada, descripta como un vector $\boldsymbol{x}$, una categoría o clase identificada por un código numérico y.

Para llevar a cabo la clasificación se necesita de una experiencia E que constituye el conjunto de imágenes a partir del cual la RNP aprende a clasificar. El aprendizaje con

aquellas imágenes constituye un subtipo de problema de lo que se conoce como aprendizaje supervisado. En el aprendizaje supervisado la RNP interactúa con un conjunto de datos que contiene características pero cada ejemplo del mismo es además asociado a una etiqueta o categoría (valor real y o vector $\boldsymbol{y}$). El algoritmo trata de predecir $\boldsymbol{y}$ a partir de $\boldsymbol{x}$, usualmente estimando $p(\boldsymbol{y}|\boldsymbol{x})$. A modo de síntesis, se puede pensar que en este aprendizaje hay un instructor o maestro que le indica a la red qué es lo que tiene que aprender.

Cabe mencionar que actualmente hay una gran variedad de conjuntos de datos etiquetados disponibles y de diversa dificultad para aprendizaje supervisado. Usualmente los conjuntos de datos contienen 2 subconjuntos: uno de entrenamiento, para entrenar al modelo de aprendizaje, y uno de prueba o evaluación, para evaluar el desempeño del mismo frente a datos nuevos y distintos a los de entrenamiento. Sin embargo, hay veces que los conjuntos de datos también vienen acompañados de un tercer subconjunto, el de validación, que permite fijar los hiperparámetros asociados al aprendizaje. Estos últimos son parámetros que controlan el comportamiento del proceso de aprendizaje en sí mismo.

Por último, la métrica de desempeño P se utiliza para cuantificar la habilidad de la RNP en la tarea para la cual fue diseñada o construida. La medida de desempeño es específica de la clase de tareas que se quiere llevar a cabo. En clasificación multiclase, la métrica frecuentemente considerada es la exactitud de clasificación del modelo. La exactitud es solo la proporción de ejemplos en los cuales el modelo produce la salida correcta. Asimismo, otra métrica equivalente es la tasa de error, es decir, la proporción de ejemplos en los cuales el modelo produce la salida incorrecta. Particularmente, interesa saber qué tan bien se desempeña la RNP frente a nuevos datos, nunca antes vistos por la misma, pues esto determina la calidad de su trabajo si se la quisiera implementar en la vida real. Por ello, se evalúa el desempeño del modelo usando datos de prueba o evaluación, que es un conjunto separado del conjunto de entrenamiento, usado para entrenar la RNP.

2.2. Neuronas Artificiales

En la Introducción del presente trabajo se presentó la neurona artificial, el modelo matemático asociado al modelo de la neurona biológica:

$$y = f\left(\sum_{k=1}^{n} w_k x_k + b\right), \tag{2.1}$$

donde $\{x_1, ..., x_n\}$ son las entradas (señales de salida de n neuronas distintas), $\{w_1, ..., w_n\}$ son los respectivos pesos o ponderaciones, b es un sesgo (constante real), f es la función de activación o transferencia usualmente elegida de forma arbitraria, e y es la salida producida por la neurona.

A continuación se presentan los tipos de neuronas más conocidos con los que son posibles construir RNPs: perceptrones (función escalón), neuronas logísticas (sigmoide, softmax y tangente hiperbólica) y neuronas rectificadas (ReLU, *Leaky* ReLU, PReLU y *Thresholded* ReLU).

2.2.1. Perceptrones

Un tipo de neurona artificial muy conocida es el perceptrón. Los perceptrones fueron desarrollados entre 1950 y 1960 por el científico Frank Rosenblatt, inspirado por los trabajos anteriores de Warren McCulloch y Walter Pitts.

La Figura 2.1 muestra el esquema de un perceptrón. Éste toma n entradas binarias y produce una única salida y, también binaria. Matemáticamente un perceptrón f se define de la siguiente manera:

$$f(\boldsymbol{x}, \boldsymbol{w}, b') \equiv \begin{cases} 0 & \text{si} \quad \sum_{k=1}^{n} w_k x_k \leq b', \\ 1 & \text{si} \quad \sum_{k=1}^{n} w_k x_k > b', \end{cases} \tag{2.2}$$

donde $\boldsymbol{x}$ y $\boldsymbol{w}$ son vectores cuyas componentes son las entradas $\{x_1, ..., x_n\}$ y los pesos $\{w_1, ..., w_n\}$ respectivamente, y b' es el valor umbral (constante real).

Es posible reescribir la Ec. (2.2) sabiendo que $\sum_{k=1}^{n} w_k x_k = \boldsymbol{w} \cdot \boldsymbol{x}$, sumando $-b'$ a ambos lados de las desigualdades y definiendo $b \equiv -b'$ como el sesgo del perceptrón:

$$f(\boldsymbol{x}, \boldsymbol{w}, b) \equiv \begin{cases} 0 & \text{si} \quad \boldsymbol{w} \cdot \boldsymbol{x} + b \leq 0, \\ 1 & \text{si} \quad \boldsymbol{w} \cdot \boldsymbol{x} + b > 0. \end{cases} \tag{2.3}$$

El sesgo b suele pensarse como una medida de cuán fácil es hacer que el perceptrón produzca como salida un 1 (neurona activada) o un 0 (neurona en reposo). Por ejemplo, para un perceptrón con un sesgo muy grande y positivo, es extremadamente fácil que produzca un 1. En cambio, si el sesgo es grande y negativo, entonces es difícil que el perceptrón produzca aquel valor como salida.

Por último, vale la pena mencionar que usualmente al perceptrón se lo puede considerar como un dispositivo de toma decisiones en base a evidencia empírica. De esta manera, al variar los valores de los parámetros (pesos y sesgo) se pueden obtener diferentes

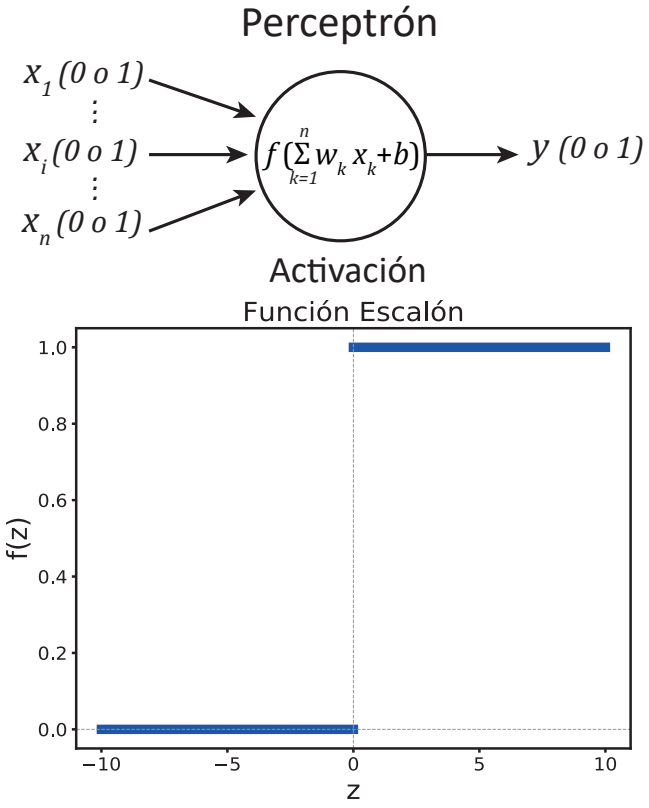


FIGURA 2.1: Esquema del perceptrón desarrollado por Frank Rosenblatt entre 1950 y 1960 (arriba) y su función de activación (abajo). Aquel recibe n entradas binarias $\{x_1, ..., x_n\}$ y retorna una única salida y, también binaria, según f.

modelos de toma de decisiones. Asimismo, es plausible que si se considera una red de perceptrones se puedan tomar decisiones de mayor complejidad y más sutiles que con un solo perceptrón.

2.2.2. Neuronas Logísticas y Rectificadas

A continuación se listan otros tipos de funciones de activación distintas a la del perceptrón: logísticas y rectificadas. Éstas son funciones populares en el diseño de arquitecturas de RNPs. La Figura 2.2 muestra las gráficas de dichas funciones y la del perceptrón.

- **Logística Sigmoide.** $\sigma : \mathbb{R} \to [0, 1]$ tal que,

$$f(z) = \sigma(z) \equiv \frac{1}{1 + e^{-z}}.$$

- **Tangente Hiperbólica (TanH).** $tanh : \mathbb{R} \to [-1, 1]$ tal que,

$$f(z) = tanh(z) \equiv \frac{e^z - e^{-z}}{e^z + e^{-z}} = 2\sigma(2z) - 1.$$

- **Softmax o Exponencial Normalizada.** $\Sigma : \mathbb{R}^k \to [-1,1]^k$ tal que,

$$f(\boldsymbol{z}) = \Sigma(\boldsymbol{z}),$$

donde:

$$\Sigma(\boldsymbol{z})_j \equiv \frac{e^{z_j}}{\sum_{i=1}^{k} e^{z_i}},$$

para $\forall j$ en $[1,k]$.

- **Unidad Lineal Rectificada (ReLU).** $max : \mathbb{R} \to \mathbb{R}^+$ tal que,

$$f(z) = max(0,z) \equiv \begin{cases} z & si \quad z > 0, \\ 0 & si \quad z \leq 0. \end{cases}$$

- ***Leaky* ReLU.** $f : \mathbb{R} \to \mathbb{R}$ tal que,

$$f(z) = \begin{cases} z & si \quad z > 0, \\ 0.01z & si \quad z \leq 0. \end{cases}$$

- **Unidad Lineal Rectificada Paramétrica (PReLU).** $f : \mathbb{R} \to \mathbb{R}$ tal que,

$$f(z) = \begin{cases} z & si \quad z > 0, \\ az & si \quad z \leq 0. \end{cases}$$

Si $a \leq 1$, $f(z) = max(z, az)$, la cual se conoce como neurona o unidad **Maxout**.

- ***Thresholded* ReLU.** $f : \mathbb{R} \to \mathbb{R}^+$ tal que,

$$f(z) = \begin{cases} z & si \quad z > a, \\ 0 & si \quad z \leq a, \end{cases}$$

donde $a \in \mathbb{R}$.

Esto concluye la breve presentación de los tipos más comunes de neuronas y sus funciones de activación. Más adelante se discute su impacto en el proceso de aprendizaje supervisado de una red.

2.3. Redes de Neuronas Artificiales

Esta sección aborda conceptos asociados a las redes neuronales artificiales: el comportamiento deseable de una red para que tenga lugar el aprendizaje en la misma y las características principales de las neuronas para facilitarlo.

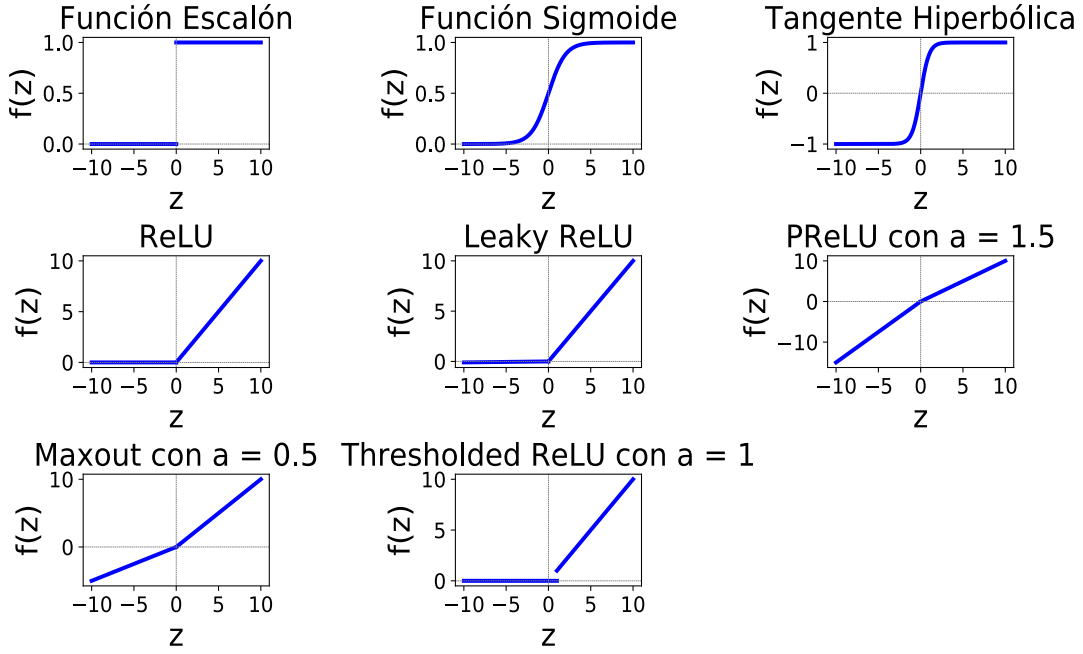


FIGURA 2.2: Gráficas de distintas funciones de activación populares.

Una red neuronal se construye mediante un conjunto de neuronas artificiales, es decir, mediante un conjunto de funciones $\{f^{(1)}, ..., f^{(k)}\}$, y conectando comúnmente la salida de cada una a las entradas de otras diferentes. De esta manera, las redes neuronales artificiales no son más que redes de funciones, típicamente representadas mediante la composición de varias funciones $f(\boldsymbol{x}) = f^{(k)}(\cdots(f^{(1)}(\boldsymbol{x})))$ como muestra la Figura 2.3. La capa más a la izquierda en esta red se llama capa de entrada, y las neuronas dentro de la capa se denominan neuronas de entrada. La capa más a la derecha o de salida contiene las neuronas de salida o, como en este caso, una única neurona de salida. Las capas intermedias se llaman capas ocultas, ya que las neuronas en estas capas no son ni entradas ni salidas. Por otro lado, los diferentes modelos de redes neuronales difieren principalmente en las funciones de activación utilizadas, el patrón de interconexión, e inclusive el tiempo de trasmisión de la información.

Supóngase que se tiene una red de neuronas artificiales diseñada para aprender a resolver algún problema de clasificación de imágenes. De esta manera, las entradas a la red son las intensidades de píxel de una imagen en donde aparece un objeto. La idea es que la red aprenda los pesos y los sesgos para que la salida de la red clasifique correctamente el objeto presente en la imagen.

Por ello, una propiedad deseable en una red neuronal que hace posible su aprendizaje (ajuste de pesos y sesgos) es la siguiente: si se realiza un pequeño cambio en algún peso (o sesgo) de la red, se quiere que aquel cause en consecuencia solo un pequeño cambio en la salida de la misma. La Figura 2.4 ilustra esta propiedad. Por lo tanto, aquello

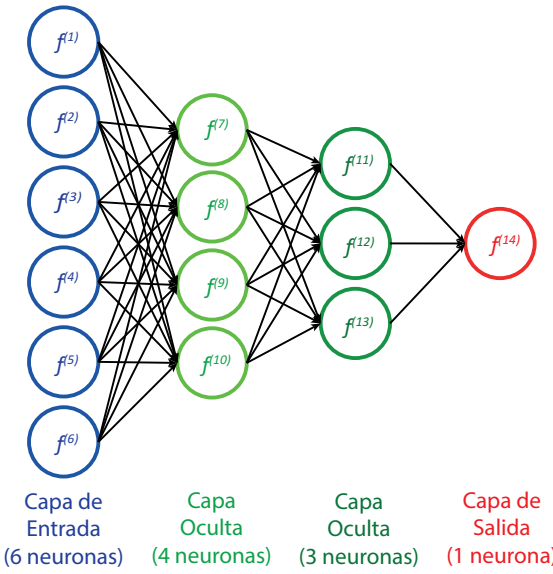


FIGURA 2.3: Esquema de una red neuronal artificial.

sugiere que se pueden modificar los pesos y sesgos para que la red se comporte cada vez más de la manera que se quiere. En consecuencia, logrando cambiar los pesos y sesgos una y otra vez para producir cada vez mejores resultados, se tendría una red que estaría aprendiendo.

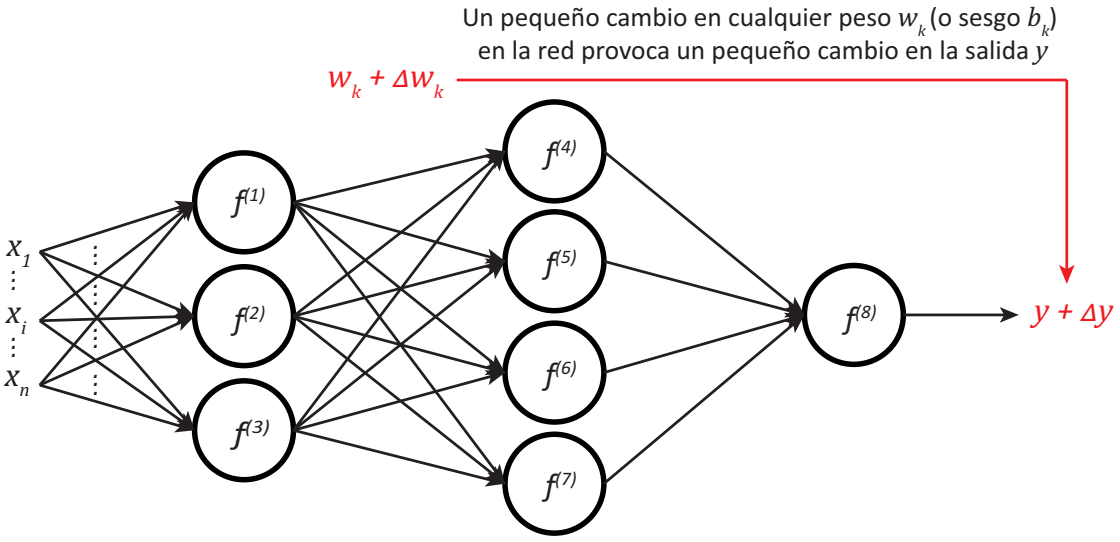


FIGURA 2.4: Comportamiento deseable de una red neuronal artificial: un pequeño cambio en un parámetro (peso o sesgo) de la red provoca un pequeño cambio en la salida.

El problema es que aquella propiedad no se logra para cualquier tipo de neurona artificial o más precisamente cuando la red presenta perceptrones. Esto se debe a que un pequeño cambio en los pesos o el sesgo de cualquier perceptrón individual en la red a veces puede hacer que la salida de ese perceptrón se invierta por completo, de 0 a 1 o viceversa. Ese cambio radical hace que sea difícil determinar cómo modificar gradualmente los

pesos y sesgos para que la red se acerque al comportamiento deseado. Este problema se soluciona mediante la introducción de otros tipos de neuronas artificiales, como las neuronas logísticas y rectificadas.

De hecho, independientemente de la definición matemática de la función de activación f, lo que realmente importa es la suavidad de la misma (usualmente tienen comportamientos similares). La suavidad de f significa que pequeños cambios Δw_j en los pesos w_j y Δb en el sesgo b producirán un pequeño cambio Δy en la salida y de la neurona. Por lo tanto, Δy se aproxima aceptablemente bien a la siguiente expresión:

$$\Delta y \approx \sum_{j=1}^{n} \frac{\partial y}{\partial w_j} \Delta w_j + \frac{\partial y}{\partial b} \Delta b, \tag{2.4}$$

donde la suma es sobre todos los pesos w_j de la neurona, y $\frac{\partial y}{\partial w_j}$ y $\frac{\partial y}{\partial b}$ denotan las derivadas parciales de la salida y con respecto a w_j y b, respectivamente.

La Ec. (2.4) establece que Δy es una función lineal de los cambios Δw_j en los pesos y Δb en el sesgo. Esta linealidad hace que sea fácil elegir pequeños cambios en los pesos y sesgos para lograr cualquier pequeño cambio deseado en la salida. Lo principal que cambia cuando se usa una función de activación diferente es que cambian los valores particulares para las derivadas parciales en la ecuación. Por consiguiente, emplear una u otra función de activación, siempre que sea suave, simplificará o complicará el álgebra de acuerdo a su definición matemática.

2.4. Redes *Feedforward*

En esta sección se presentan las redes *feedforward*, de gran importancia por sus aplicaciones en Visión Computacional. Asimismo, se comenta brevemente el Teorema de Universalidad en relación a la capacidad de representación de las redes *feedforward* como aproximadores de funciones.

Si bien existen diferentes estructuras o arquitecturas de redes neuronales artificiales, un modelo típico son las denominadas redes hacia adelante (*feedforward neural networks*, en inglés). En este trabajo se referirá a ellas como redes *feedforward* (RFs). Este modelo se lo llama de aquella manera debido a que la información fluye a través de la red hacia adelante, es decir, desde la función evaluada en $\boldsymbol{x}$ (entrada) a través de las operaciones intermedias usadas para definir f, hasta el cálculo de la salida y o $\boldsymbol{y}$. Las RFs están compuestas por unidades y conexiones (representando neuronas y sinapsis respectivamente) organizadas en diferentes capas y en donde las conexiones no forman ciclos o bucles. Las unidades en una misma capa funcionan en paralelo y cada una de ellas se

puede interpretar como una función no lineal que mapea un vector a un escalar. De esta manera, una unidad toma varias activaciones de la capa anterior como entrada para computar su propio valor de activación o salida.

Por otro lado, vale la pena mencionar dos métricas comunes para medir el tamaño de una red neuronal: el número de unidades o neuronas y la cantidad de parámetros a aprender. Asimismo, el tamaño y cantidad de capas en una red neuronal determina la capacidad de la red, esto es el espacio de funciones representables por la arquitectura de la misma. Particularmente, el espacio de funciones representables crece con el número y tamaño de las capas debido a que las neuronas individualmente colaboran a expresar diferentes funciones no lineales y de alta complejidad.

En relación a la capacidad de una RF, Hornik *et al.* (1989) establecieron que las RFs con tan solo una capa oculta y usando funciones de activación arbitrarias son capaces de aproximar cualquier función con cualquier grado de precisión deseado, siempre que haya suficientes unidades ocultas disponibles. En este sentido, las RFs multicapa son una clase de aproximadores universales. Cabe mencionar que una de las primeras versiones del teorema fue realizada por George Cybenko en 1989 para RFs con funciones de activación sigmoides.

A continuación se presentan dos tipos de RFs populares que se estudiarán en este trabajo: redes completamente conectadas o densas (RDs) y redes convolucionales (RCs).

2.4.1. Redes Densas

Esta sección se describen las redes completamente conectadas o densas a partir de un ejemplo simple.

Las RDs están formadas por un apilado de capas completamente conectadas o densas. Estas redes reciben como entrada un vector (imagen) que es transformado finalmente en otro vector (probabilidades de las categorías) a través de una serie de capas ocultas densas formadas por conjuntos de neuronas.

2.4.1.1. Capa Densa

Las capas densas se caracterizan por estar constituidas de neuronas donde cada una está totalmente conectada a todas las neuronas de la capa siguiente. Las neuronas de una misma capa funcionan completamente independiente y no comparten conexiones. La última capa completamente conectada es la capa de salida y representa las probabilidades de las clases o categorías en la tarea de clasificación.

Vale la pena introducir brevemente una notación adecuada para referirse a los parámetros de una capa densa sin ambigüedades.

Se denota w^l_{jk} al peso correspondiente a la conexión entre la neurona k de la capa $(l-1)$ y la neurona j de la capa l. Asimismo, se utiliza una notación similar para los sesgos y las activaciones de la red: b^l_j y a^l_j denotan el sesgo y la activación de la neurona j de la capa l respectivamente. Por otro lado, se sabe que la activación a^l_j está relacionada con todas las activaciones de la capa $(l-1)$ mediante la función de activación f. Por consiguiente:

$$a^l_j = f\left(\sum_{k=1}^{m} w^l_{jk} a^{l-1}_k + b^l_j\right), \tag{2.5}$$

donde la suma es sobre cada una de las m neuronas de la capa $(l-1)$.

La Ec. (2.5) se puede reescribir en forma matricial. Para ello, se define la matriz de pesos W^l para cada capa l, donde la entrada en la fila j y columna k es w^l_{jk}. De manera similar, para cada capa l se define un vector de sesgos $\boldsymbol{b}^l$ y un vector de activaciones $\boldsymbol{a}^l$, donde la componente j de cada vector son solo los valores b^l_j y a^l_j respectivamente para cada neurona j de la capa l.

Además, es necesario introducir la noción de función vectorizada. Esto implica aplicar una función a un vector $\boldsymbol{v}$ tal que la función se aplique sobre cada una de las componentes del mismo. Una función vectorizada se denota como $f(\boldsymbol{v})$, donde $f(\boldsymbol{v})_j = f(v_j)$.

Finalmente, la notación introducida permite reescribir la Ec. (2.5) en forma más compacta:

$$\boldsymbol{a}^l = f(W^l \boldsymbol{a}^{l-1} + \boldsymbol{b}^l). \tag{2.6}$$

De esta manera, la capa densa l recibe un vector de activaciones $\boldsymbol{a}^{l-1}$ de la capa anterior $(l-1)$. Este vector es transformado a través de un producto matricial con la matriz de ponderaciones W^l, asociada a la capa densa en cuestión, y sumándole luego un vector sesgo $\boldsymbol{b}^l$. Finalmente, este vector transformado representa el vector de activaciones $\boldsymbol{a}^l$ calculado por la capa l. A continuación un ejemplo de una arquitectura densa.

Considérese la tarea de clasificar imágenes que pertenecen a una, y solo una, categoría de 10 posibles, previamente definidas. Supóngase que estas imágenes son a color y de tamaño 32×32 píxeles. La Figura 2.5 muestra un esquema de una RD simple para la clasificación de aquellas imágenes y que podría tener la siguiente arquitectura:

$$[ENTRADA - 2,000\,d - 1,000\,d - 10\,d],$$

donde d hace referencia a capa densa, y el factor delante de d indica el número de neuronas en esa capa. En más detalle:

- *ENTRADA*. Vector de 3,072 componentes reales. Cada número representa el valor de la intensidad de un píxel en una imagen de 32 píxeles de ancho, 32 píxeles de alto y 3 canales de color R, G y B.
- 2,000 *d*. Vector de 2,000 componentes reales. Cada valor corresponde a una activación calculado por cada neurona de la capa.
- 1,000 *d*. Vector de 1,000 componentes reales. Cada número representa un valor de activación calculado por cada neurona de la capa.
- 10 *d*. Vector de 10 componentes reales. Cada valor corresponde a una probabilidad para cada clase o categoría.

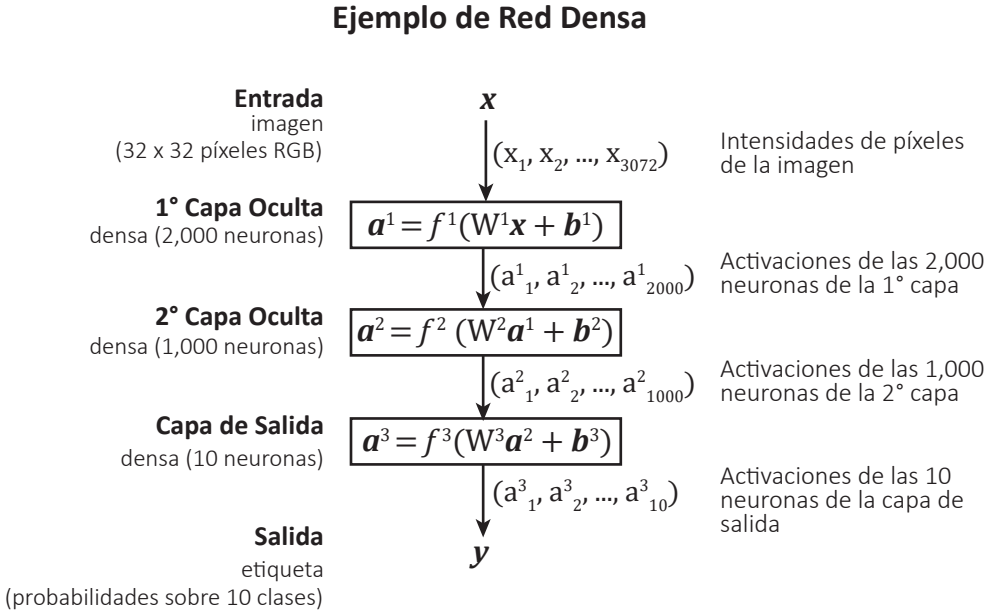


FIGURA 2.5: Ejemplo de arquitectura densa: entrada (vector imagen), capa oculta densa de 2,000 neuronas, capa oculta densa de 1,000 neuronas, y capa de salida de 10 neuronas.

Cabe mencionar que esta red presenta 2 capas ocultas completamente conectadas. Además, tiene 3,010 neuronas o unidades (2,000 + 1,000 + 10), de las cuales 3,000 (2,000 + 1,000) representan neuronas ocultas. Asimismo, la red presenta un total de 8,157,010 parámetros a determinar en el proceso de aprendizaje de la misma: $3,072 \times 2,000$ pesos y 2,000 sesgos en la primera capa oculta; $2,000 \times 1,000$ pesos y 1,000 sesgos en la segunda capa oculta; y finalmente $1,000 \times 10$ pesos y 10 sesgos en la capa de salida.

2.4.2. Redes Convolucionales

En esta sección se describen las redes convolucionales a partir de un ejemplo simple.

Las RCs aprovechan el hecho de que las entradas consisten en imágenes. En particular, a diferencia de las RDs, las capas de una RC presentan neuronas dispuestas en tres dimensiones: ancho, alto y profundidad. Por lo tanto, se tienen tensores (volúmenes) y no vectores de activaciones. Estas redes reciben como entrada un tensor (imagen) que es transformado finalmente en un vector a través de una serie de capas ocultas. A continuación un ejemplo de una arquitectura convolucional.

Nuevamente, considérese la tarea de clasificar imágenes que pertenecen a una, y solo una, categoría de 10 posibles, previamente definidas. Supóngase que estas imágenes son a color y de tamaño 32×32 píxeles. Las imágenes de entrada se representan mediante volúmenes de intensidades de píxel de tamaño $32 \times 32 \times 3$ (ancho, alto y profundidad respectivamente).

Usualmente se utilizan tres tipos principales de capas para construir una arquitectura convolucional: convolucionales, de *pooling* y densas.

La Figura 2.6 muestra una RC simple para la clasificación de aquellas imágenes y que podría tener la siguiente arquitectura:

$$[ENTRADA - c - p - 10\,d],$$

donde c hace referencia a capa convolucional, p a capa de *pooling* y d a capa densa o capa completamente conectada. El factor delante de d indica el número de neuronas en esa capa. En más detalle:

- *ENTRADA*. Tensor de tamaño $32 \times 32 \times 3$. Cada número representa el valor de la intensidad de un píxel en una imagen de 32 píxeles de ancho, 32 píxeles de alto y 3 canales de color R, G y B.
- *c*. Calcula la salida de las neuronas que están conectadas a regiones locales de la entrada. Cada neurona lleva a cabo una convolución entre una región cuadrada de pesos y la región en la imagen cuyos valores son las intensidades de píxeles, por ejemplo de 5×5, a la que están conectadas en el volumen de entrada. Esto genera un tensor de tamaño $32 \times 32 \times 12$ si se deciden usar 12 filtros con corrimiento en alto y ancho de 1 píxel y relleno con ceros de magnitud de 2 píxeles en dichas dimensiones. Por último, se aplica a cada elemento de la salida una función de activación ReLU.
- *p*. Esta capa realiza una operación de submuestreo del tensor de activaciones entrante a lo largo de las dimensiones espaciales ancho y alto. Esto podría generar un tensor de activaciones de salida de tamaño $16 \times 16 \times 12$ si se aplica sobre regiones de tamaño 2×2 y con corrimiento de 2 píxeles en ancho y alto.

- 10 *d*. Calcula las probabilidades de las clases. Esto dará como resultado un tensor de tamaño $1 \times 1 \times 10$ (vector), donde cada uno de los 10 números corresponde a una probabilidad correspondiente a una clase. Al ser una capa densa, cada neurona está conectada a todas las activaciones del volumen anterior.

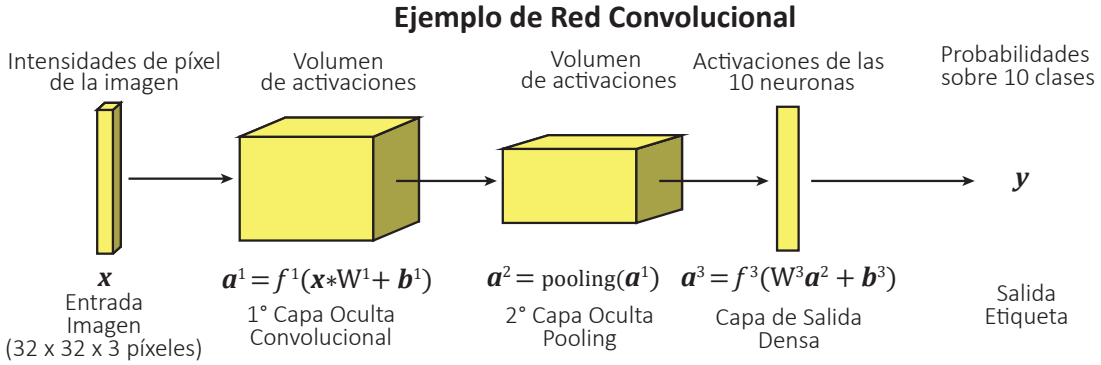


FIGURA 2.6: Ejemplo de arquitectura convolucional: entrada (tensor imagen), capa oculta convolucional, capa oculta de *pooling*, y capa de salida densa de 10 neuronas.

Cabe mencionar que esta red presenta 2 capas ocultas: una convolucional y una de *pooling*. Además, tiene 12,298 neuronas o unidades (12,288 + 10), de las cuales 12,288 ($32 \times 32 \times 12$) representan neuronas ocultas. Asimismo, la red presenta un total de 31,642 parámetros a determinar en el proceso de aprendizaje de la misma: $(5 \times 5 \times 3) \times 12$ pesos y 12 sesgos en la capa convolucional; 0 parámetros en la capa de *pooling*; y finalmente $16 \times 16 \times 12 \times 10$ pesos y 10 sesgos en la capa densa de salida.

A continuación se describen la capa convolucional y de *pooling* de una red convolucional detallando sus hiperparámetros y conectividades. Por último, se comentan tres características fundamentales que presentan las arquitecturas convolucionales: interacciones ralas, compartición de parámetros y representaciones equivariantes.

2.4.2.1. Capa Convolucional

Cada neurona de una capa convolucional está conectada a solo una pequeña región local del volumen de activaciones que ingresa a la capa. El tamaño de esta región o filtro está determinada por un hiperparámetro denominado tamaño del filtro F. La extensión del filtro a lo largo del eje de profundidad siempre es igual a la profundidad del volumen de entrada a la capa. Es importante enfatizar que las conexiones son locales en el espacio (en ancho y alto), pero siempre a lo largo de toda la profundidad del volumen de entrada.

Por ejemplo, supóngase que el volumen de activaciones de entrada a una capa tiene un tamaño de $7 \times 7 \times 3$. Si el tamaño del filtro es $F = 3$, esto es de 3×3, entonces cada neurona en la capa convolucional tendrá pesos en una región de tamaño $3 \times 3 \times 3$ en

el volumen de entrada. Esto hace un total de $3 \times 3 \times 3 = 27$ pesos, más 1 sesgo. Cabe destacar que la extensión del filtro a lo largo del eje de profundidad es 3, debido a que ésta es la profundidad del volumen de activaciones de entrada.

Con respecto al número de neuronas y tamaño del volumen de salida hay tres hiperparámetros que lo regulan:

- *Profundidad K.* Corresponde a la cantidad de filtros que se quieran utilizar, cada uno aprende a buscar algo diferente en la entrada. Por ejemplo, si la primera capa convolucional toma como entrada una imagen, entonces diferentes neuronas a lo largo de la dimensión de profundidad pueden activarse en presencia de bordes, o manchas de color.
- *Corrimiento S.* Hay que especificar el corrimiento con la que se desliza un filtro. Cuando $S = 1$ el filtro se mueve 1 píxel a la vez en ancho y alto. Cuando $S = 2$ el filtro se desliza 2 píxeles a la vez. Esto producirá volúmenes de salida más pequeños espacialmente.
- *Relleno con Ceros P.* Es conveniente rellenar el volumen de entrada con ceros alrededor del borde. El tamaño de este relleno es un hiperparámetro que permite preservar exactamente el tamaño espacial del volumen de entrada para que el ancho y alto del volumen de salida sean los mismos que los de entrada.

Es posible calcular el tamaño espacial del volumen de salida ($H2$ y $W2$) en función del tamaño del volumen de entrada ($H1$ y $W1$), el tamaño del filtro (F), el corrimiento con el que se aplica (S) y la cantidad de relleno con ceros (P) utilizado en el borde. Las expresiones para calcular el ancho y el alto del volumen de salida están dadas por:

$$W2 = (W1 - F + 2P)/S + 1,$$

$$H2 = (H1 - F + 2P)/S + 1,$$

donde se supuso mismo S (y por lo tanto mismo P) en altura que en ancho. Por ejemplo, para una entrada de $7 \times 7 \times 3$ y un tamaño de filtro de $F = 3$ con $S = 1$ y $P = 0$ se obtendría una salida de $5 \times 5 \times 3$. En cambio, con $S = 2$ se obtendría una salida de $3 \times 3 \times 3$.

En resumen, la capa convolucional realiza lo siguiente:

- Toma un volumen de activaciones de tamaño $W1 \times H1 \times D1$.
- Requiere 4 hiperparámetros:

  - Número de filtros K.
  - Tamaño de los filtros F.
  - Corrimiento S.
  - Cantidad de relleno con ceros P.
- Produce un volumen de activaciones de tamaño $W2 \times H2 \times D2$, donde:
  - $W2 = (W1 - F + 2P)/S + 1$,
  - $H2 = (H1 - F + 2P)/S + 1$,
  - $D2 = K$.
- Se aplica a cada elemento del volumen de salida una función de activación ReLU.

Cabe mencionar que con el uso compartido de parámetros, una capa convolucional introduce $F \times F \times D1$ pesos por filtro. Por lo tanto, esto hace un total de $(F \times F \times D1) \times K$ pesos y K sesgos. Asimismo, en el tensor de salida de la capa, el corte d en la dimensión profundidad (de tamaño $W2 \times H2$) es el resultado de realizar una convolución del filtro d sobre el volumen de entrada con un corrimiento de S, y luego se compensa con un sesgo.

Usualmente se usan filtros pequeños, esto es filtros de tamaños 3×3 o 5×5, usando corrimiento $S = 1$ y relleno con ceros de forma tal de que la capa convolucional no altere las dimensiones espaciales (ancho y alto) del volumen de entrada. Por ejemplo, cuando $F = 3$, se usa $P = 1$ para conservar el tamaño original de la entrada. En cambio, cuando $F = 5$, se emplea $P = 2$. En general, para un dado valor de F, se tiene que $P = (F-1)/2$ preserva el tamaño original.

Finalmente, una capa menos común en RCs y muy similar a la capa convolucional es la capa localmente conectada. La diferencia es que las ponderaciones en cada filtro no son compartidas entre neuronas.

2.4.2.2. Capa de *Pooling*

Es usual colocar periódicamente una capa *pooling* entre capas convolucionales sucesivas en una arquitectura convolucional. La función de una capa de *pooling* es reducir el tamaño espacial, en ancho y alto, del volumen de activaciones de entrada mediante cierta operación. Por lo tanto, el volumen de activaciones de salida tendrá misma profundidad pero menor tamaño, en ancho y alto, que el volumen de entrada, por lo que esta capa opera de forma independiente en cada corte de la dimensión profundidad. La Figura 2.7 muestra la forma más común de una capa de *pooling*: regiones de tamaño 2×2 aplicadas

con un corrimiento de 2 píxeles a lo largo del ancho y alto de cada corte de profundidad de la entrada. Esto descarta el 25 % de las activaciones del volumen que ingresa.

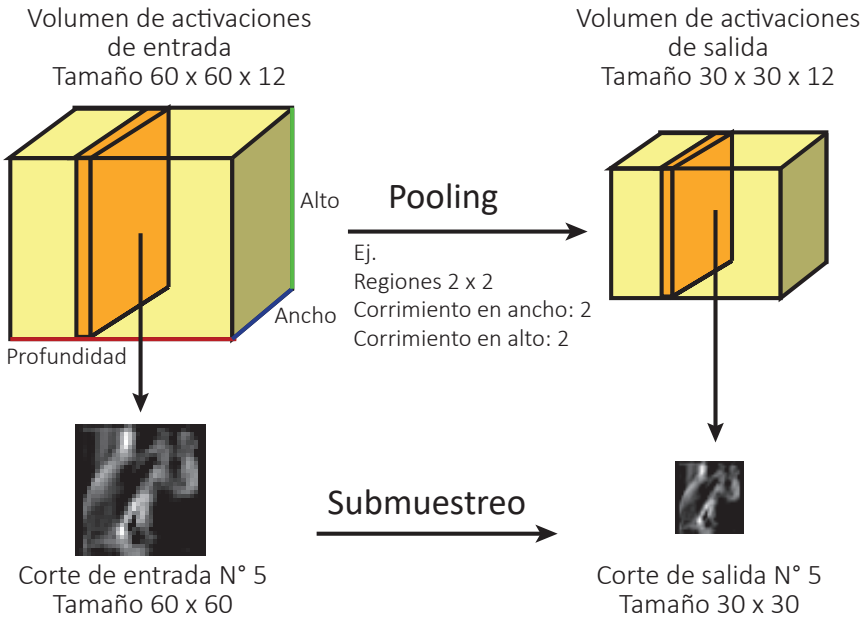


FIGURA 2.7: Ejemplo del efecto de una capa de *pooling* típica en el tamaño del volumen de activaciones de salida. Esta capa resume regiones de tamaño 2×2 con un corrimiento horizontal y vertical de 2 píxeles.

La Figura 2.8 muestra un ejemplo de las operaciones de *pooling* populares:

- *Max-Pooling*. Dado un arreglo bidimensional de números retorna el máximo valor del arreglo.
- *Average-Pooling*. Dado un arreglo bidimensional de números retorna el promedio aritmético de los valores del arreglo.

Asimismo, una operación menos común pero interesante es el *Global Average-Pooling*. Dado un arreglo tridimensional de números, retorna el promedio aritmético de los valores del arreglo de las dimensiones ancho y alto por corte de profundidad.

Cabe mencionar que la capa de *pooling* no introduce parámetros (pesos o sesgos) debido a que solo evalúa una función fija sobre pequeñas regiones en el volumen de activaciones de entrada.

En resumen, la capa de *pooling* realiza lo siguiente:

- Toma un volumen de activaciones de tamaño $W1 \times H1 \times D1$, donde $W1$ es el ancho; $H1$ es el alto; y $D1$ es la profundidad del volumen de entrada.
- Requiere 3 hiperparámetros:

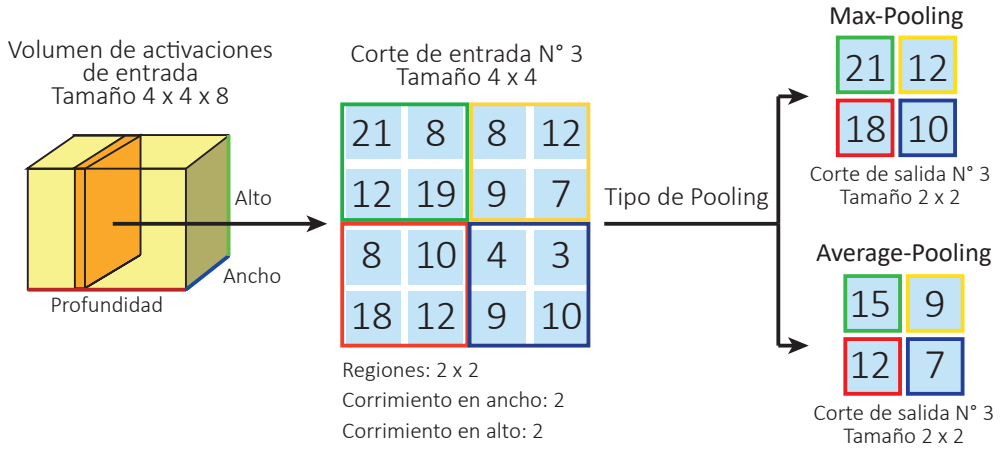


FIGURA 2.8: Ejemplo de las operaciones de *pooling* más populares: *Max-Pooling* y *Average-Pooling*.

  - Tipo de *pooling* T.
  - Tamaño de la región bidimensional R ($R \times R$).
  - Magnitudes del corrimiento S_W y S_H en ancho y alto respectivamente.
- Produce un volumen de activaciones de tamaño $W2 \times H2 \times D2$ tal que:
  - $W2 = (W1 - R)/S_W + 1$,
  - $H2 = (H1 - R)/S_H + 1$,
  - $D2 = D1$,

  donde $W2 < W1$, $H2 < H1$ y $D1 = D2$.

2.4.2.3. Tres Características Destacables

La arquitectura de una red convolucional introduce tres ideas para garantizar invariancia a las distorsiones de la entrada:

- **Interacciones Ralas.** En las RDs las neuronas de una capa densa se encuentran completamente conectadas a todas las neuronas de la capa anterior. En cambio, en las RCs las conexiones en una capa convolucional son ralas, es decir, las neuronas estás conectadas a pequeñas regiones locales de la capa de activaciones anterior.
- **Compartición de Parámetros.** En las RDs cada neurona de una capa densa calcula un producto punto entre un vector de ponderaciones, propio de esa neurona, y el vector de activaciones de la capa anterior. Por lo tanto, las neuronas de una misma capa no comparten parámetros. Sin embargo, en las RCs cada neurona de una capa convolucional comparte parámetros con las neuronas del resto de la capa

mediante un mismo filtro que se aplica en forma local en el volumen de activaciones de entrada a la capa.

- **Representaciones Equivariantes.** Los operadores de convolución son equivariantes ante traslaciones de la entrada, es decir, si la entrada cambia, la salida cambia de la misma manera. El operador de convolución no es naturalmente equivariante a otras transformaciones, tal como cambios en la escala o rotaciones. Cabe mencionar que la operación de *pooling* ayuda a hacer que las representaciones sean aproximadamente invariantes a pequeñas traslaciones. Si se traslada la entrada, la salida de la capa de *pooling* no cambia.

2.5. Aprendizaje de Redes *Feedforward*

En esta sección se comentan brevemente los requerimientos básicos implicados en el aprendizaje de tipo supervisado de redes *feedforward* para clasificación multiclase de imágenes.

Aprender una red es resolver un problema de optimización. Para ello se buscan los parámetros de la red (pesos y sesgos) Θ que minimizan una función de costo o error L para cada ejemplo $\boldsymbol{x_i}$ en el conjunto de entrenamiento $\mathbb{X}$. La optimización se lleva a cabo mediante algún algoritmo basado en el gradiente descendente. Para cualquier algoritmo de optimización basado en aquel es necesario calcular el gradiente de la función de costo. En redes neuronales dicho gradiente se calcula mediante el algoritmo de retropropagación. La Figura 2.9 muestra un esquema que resume los elementos fundamentales que hay detrás de un algoritmo de aprendizaje para RNPs.

2.5.1. Gradiente Descendente

El gradiente descendente es uno de los algoritmos más populares para llevar a cabo la optimización y por lejos la forma más común de optimizar redes neuronales. En la práctica, para encontrar los pesos w_k y los sesgos b_l de la red que minimizan la función de costo L se suele usar el algoritmo gradiente descendente estocástico (*Stochastic Gradient Descent*, en inglés, o simplemente –SGD–). El SGD permite estimar el gradiente de la función de costo ∇L calculando el gradiente $\nabla L_{\boldsymbol{x}}$ de la función de costo para un pequeño grupo (*mini batch*, en inglés) de m ejemplos de entrenamiento seleccionados aleatoriamente: $\tilde{\boldsymbol{x}}_1$, $\tilde{\boldsymbol{x}}_2$, ..., $\tilde{\boldsymbol{x}}_m$. Tomando el promedio sobre esta pequeña muestra (siempre que m sea lo suficientemente grande) se espera que el valor promedio de los gradientes $\nabla L_{\tilde{\boldsymbol{x}}_j}$ sea aproximadamente igual al promedio sobre la totalidad de los n ejemplos de entrenamiento $\nabla L_{\boldsymbol{x}}$, o sea, se puede tener una buena estimación del verdadero gradiente

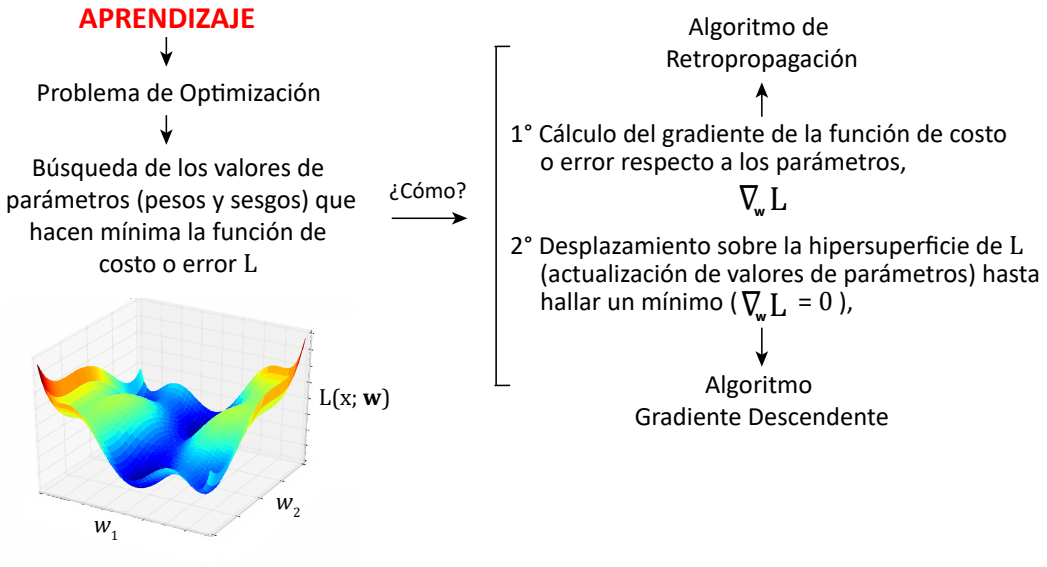


FIGURA 2.9: Esquema de los principales elementos que hacen al proceso de aprendizaje de redes neuronales artificiales.

∇L:

$$\nabla L = \frac{\sum_{i=1}^{n} \nabla L_{\boldsymbol{x}_i}}{n} \approx \frac{\sum_{j=1}^{m} \nabla L_{\tilde{\boldsymbol{x}}_j}}{m}. \tag{2.7}$$

Si w_k y b_l son los pesos y sesgos de la red, luego SGD trabaja tomando aleatoriamente un grupo de ejemplos y entrenando con ellos:

$$w_k \to w'_k = w_k - \frac{\eta}{m} \sum_{j=1}^{m} \partial \frac{L_{\tilde{\boldsymbol{x}}_j}}{\partial w_k}, \tag{2.8}$$

$$b_l \to b'_l = b_l - \frac{\eta}{m} \sum_{j=1}^{m} \partial \frac{L_{\tilde{\boldsymbol{x}}_j}}{\partial b_l}, \tag{2.9}$$

donde las sumas son sobre todos los ejemplos de entrenamiento $\boldsymbol{x}_j$ del actual *mini batch* y η es la tasa de aprendizaje. Luego, se toma otro grupo aleatoriamente y así sucesivamente hasta acabar los ejemplos de entrenamiento. Cuando esto ocurre se dice que se ha completado una época de entrenamiento (*epoch*, en inglés). A ese punto se empieza de nuevo otra época. Este algoritmo permite lograr la optimización hasta converger en un mínimo local lo suficientemente bajo como para asegurar buenas predicciones. Asimismo, para implementar el SGD en la práctica es necesario también otro bucle para generar los grupos de ejemplos de entrenamiento, y otro bucle para iterar por múltiples épocas de entrenamiento.

Por otro lado, muchas de las librerías de Aprendizaje Profundo actuales contienen implementaciones de varios algoritmos que optimizan el gradiente descendente (Ruder, 2017). Existen diversas variantes al algoritmo SGD que lo mejoran, tal como Adam (*Adaptive*

Moment Estimation, en inglés). Adam se caracteriza por ser un tipo de algoritmo con tasa de aprendizaje adaptativa. Aquel algoritmo propone ciertas reglas para adaptar de forma automática las tasas de aprendizaje controlando la rapidez de convergencia aumentando o disminuyendo este parámetro basado en el error. En esta tesis será suficiente comprender la versión presentada de SGD ya que contiene la idea fundamental detrás del gradiente descendente.

2.5.2. Retropropagación

Una forma rápida para calcular el gradiente de la función de costo ∇L, es decir, las derivadas parciales: $\partial L/\partial w_k$ y $\partial L/\partial b_l$ para cada peso w_k y sesgo b_l de la red se logra mediante el algoritmo de retropropagación o propagación hacia atrás (*backpropagation*, en inglés), propagando el error a través de aplicaciones recursivas de la regla de la cadena del Cálculo Multivariable (Rumelhart *et al.*, 1986). En particular dado un grupo de m ejemplos de entrenamiento, el siguiente algoritmo aplica una época de aprendizaje basado en SGD para RFs (Nielsen, 2015):

1. Entra un conjunto de ejemplos de entrenamiento.
2. Para cada ejemplo de entrenamiento $\boldsymbol{x}$: se calcula la correspondiente activación $\boldsymbol{a}^{\boldsymbol{x},1}$ para la capa de entrada $l = 1$ y se siguen los siguientes pasos:
   - Propagación hacia adelante: para cada capa $l = 2, 3, ..., L$ se computa $\boldsymbol{z}^{\boldsymbol{x},l} = W^l \boldsymbol{a}^{\boldsymbol{x},l-1} + \boldsymbol{b}^l$ y $\boldsymbol{a}^{\boldsymbol{x},l} = \sigma(\boldsymbol{z}^{\boldsymbol{x},l})$.
   - Error de salida $\boldsymbol{\delta}^{\boldsymbol{x},L}$: se calcula el vector $\boldsymbol{\delta}^{\boldsymbol{x},L} = \nabla_{\boldsymbol{a}} L_{\boldsymbol{x}} \bigodot {}^1 \sigma'(\boldsymbol{z}^{\boldsymbol{x},L})$.
   - Propagación hacia atrás del error: para cada $l = L-1, L-2, ..., 2$ se computa $\boldsymbol{\delta}^{\boldsymbol{x},l} = ((W^{l+1})^T \boldsymbol{\delta}^{\boldsymbol{x},l+1}) \bigodot \sigma'(\boldsymbol{z}^{\boldsymbol{x},l})$.
   - Salida: el gradiente de la función de costo para un ejemplo está dado por $\partial L_{\boldsymbol{x}}/\partial w_{jk}^{\boldsymbol{x},l} = a_k^{\boldsymbol{x},l-1} \delta_j^{\boldsymbol{x},l}$ y $\partial L_{\boldsymbol{x}}/\partial b_j^{\boldsymbol{x},l} = \delta_j^l$.
3. Gradiente descendente: para cada $l = L, L-1, ..., 2$ se actualizan los pesos y los sesgos de acuerdo a las reglas dadas por la Ec. (2.8) y Ec. (2.9), es decir:

$$W^l \to W^{l\prime} = W^l - \frac{\eta}{m} \sum_{i=1}^{m} \boldsymbol{\delta}^{\boldsymbol{x_i},l} (\boldsymbol{a}^{\boldsymbol{x_i},l-1})^T,$$

$$\boldsymbol{b}^l \to \boldsymbol{b}^{l\prime} = \boldsymbol{b}^l - \frac{\eta}{m} \sum_{i=1}^{m} \boldsymbol{\delta}^{\boldsymbol{x_i},l}.$$

[1] Dadas dos matrices A y B de tamaño $m \times n$, el producto de Hadamard de A y B, denotado como $A \bigodot B$, es una matriz de tamaño $m \times n$ con elementos dados por $(A \bigodot B)_{i,j} = A_{i,j} B_{i,j}$.

Capítulo 3

Estado del Conocimiento

En este capítulo se presenta el estado del conocimiento. El mismo busca: (a) hacer una distinción de los métodos y técnicas actuales para un entrenamiento de RNPs más eficiente y eficaz; (b) advertir los modelos de RNPs populares y estudiados en el último tiempo para clasificación multiclase de imágenes; y (c) conocer los entornos computacionales de trabajo y *software* con los que se trabaja hoy en día en Aprendizaje Profundo.

La elaboración del presente estado del conocimiento implicó la revisión de un conjunto representativo de artículos de alto impacto previamente seleccionados por uno de los tantos expertos en Visión Computacional, Rodrigo Benenson[1]. También, se hizo uso de recopilaciones de avances en Aprendizaje Profundo llevados a cabo por otros estudiosos, tales como Yoshua Bengio, Geoffrey Hinton, Yann LeCun y Andrej Karpathy, por solo mencionar algunos. Por último, este capítulo está organizado de la siguiente manera: entornos, modelos y técnicas actuales para entrenamiento eficiente de RNPs.

3.1. Técnicas de Entrenamiento

A continuación las técnicas se las agrupó en distintos aspectos centrales del entrenamiento de RNPs: preprocesamiento de datos, inicialización de parámetros, función de costo o error, actualización de parámetros, optimización de hiperparámetros y regularización.

3.1.1. Preprocesamiento de Datos

Una forma frecuentemente adoptada para destacar mejor las características de los datos, y obtener así redes con mayor desempeño, es aplicar algún tipo de normalización sobre

[1] `http://rodrigob.github.io/are_we_there_yet/build/classification_datasets_results.html`

los datos. Se ha demostrado que la técnica de Contraste Global Normalizado (CGN), que transforma el valor de cada píxel de una imagen restando la media y dividiendo por la desviación estándar de la misma, es una de las clases de preprocesamiento más conveniente para el aprendizaje de RNPs (de Andrade, 2014; Karpathy, 2017).

3.1.2. Inicialización de Parámetros

Los valores iniciales de pesos y sesgos de una RNP tienen un efecto significativo en el proceso de entrenamiento y en su convergencia, lo cual ha motivado al desarrollo de diversos métodos para este fin. Uno comúnmente sugerido es inicializar los sesgos con valor constante cero y los pesos mediante el inicializador *Glorot Normal* o también conocido como *Xavier Normal* (Glorot & Bengio, 2010; Goodfellow *et al.*, 2016). Éste otorga valores muestreando una distribución normal truncada con media cero y desviación estándar $\sqrt{2/(n_{in} + n_{out})}$, donde n_{in} y n_{out} son el número de unidades de entrada y de salida en la matriz o el tensor de pesos respectivamente. Sin embargo, en el caso particular de redes con unidades rectificadas, se recomienda el método de inicialización de pesos *He Normal* propuesto por He *et al.* (2015) específicamente para aquella clase de unidades. Asimismo, éste otorga valores muestreando una distribución normal truncada con media cero pero con desviación estándar $\sqrt{2/n_{in}}$. En la práctica esta técnica permite obtener mejores resultados en redes rectificadas con ReLUs, PReLUs y funciones de esa familia (Karpathy, 2017; Kumar, 2017).

3.1.3. Función de Costo o Error

Dentro de la variedad de estas funciones, que permiten evaluar la calidad de un conjunto particular de parámetros, se utiliza usualmente la función de verosimilitud logarítmica negativa, también denominada entropía cruzada. Esta elección suele ser la usual en los problemas de clasificación multiclase de imágenes más modernos (Nielsen, 2015).

3.1.4. Actualización de Parámetros

El gradiente descendente es uno de los algoritmos más populares para llevar a cabo la optimización y por lejos la forma más común de optimizar RNPs. En la práctica, para encontrar los pesos y los sesgos que minimizan la función de costo se suelen usar variantes mejoradas del algoritmo gradiente descendente estocástico (*Stochastic Gradient Descent*, en inglés, o simplemente –SGD–). Entre ellas se encuentra Adam (*Adaptive Moment Estimation*, en inglés) que se caracteriza por ser un algoritmo con tasa de aprendizaje adaptativa. Se ha demostrado que Adam es computacionalmente eficiente

y los resultados son comparables a los obtenidos con otros métodos de optimización estocásticos (Goodfellow *et al.*, 2016; Ruder, 2017).

3.1.5. Optimización de Hiperparámetros

El aprendizaje de RNPs involucra especificar los hiperparámetros que controlan el proceso de aprendizaje en si mismo. Esta optimización es necesaria ya que una mala configuración de hiperparámetros puede llevar a un desempeño pobre de las redes aprendidas o que el tiempo necesario para la convergencia a un buen modelo sea demasiado.

Para ello, es necesario explorar el espacio de hiperparámetros que guían el proceso de búsqueda de modo de obtener el mejor rendimiento en términos de precisión y tiempo de entrenamiento. Este proceso se lleva a cabo evaluando el desempeño de los modelos aprendidos bajo distintas configuraciones de hiperparámetros. Algunos de éstos son la tasa de aprendizaje inicial, la constante de decaimiento de la tasa de aprendizaje, el *dropout*, el tamaño de grupos y el número de épocas.

Dentro de los métodos para la exploración del espacio de parámetros se encuentra la búsqueda aleatoria (Bergstra & Bengio, 2012). Este algoritmo selecciona puntos al azar dentro del espacio de hiperparámetros, lo cual se ha mostrado que permite descubrir de manera mucho más precisa los buenos valores para los hiperparámetros más importantes.

Es importante destacar que para tener una medida significativa del desempeño de cada modelo aprendido bajo una configuración de hiperparámetros dada, es común utilizar el método de validación cruzada (*k-fold cross-validation*, en inglés). Este es un método estadístico que permite evaluar y comparar algoritmos de aprendizaje de forma rápida en base a su desempeño. Por lo general, el número de particiones en dicho algoritmo se suele fijar en $k = 10$ por ser un valor óptimo (Refaeilzadeh *et al.*, 2008).

3.1.6. Regularización

En ocasiones, las redes aprendidas son demasiado complejas y presentan lo que se conoce como sobreajuste (*overfitting*, en inglés). El sobreajuste de una red neuronal, se da cuando este es capaz de predecir (clasificar imágenes) con un alto grado de precisión aquellos datos con los que fue entrenado, pero que falla rotundamente en la predicción de datos antes no vistos. En dicha situación se dice que un modelo no tiene capacidad de generalización lo cual no es deseable desde el punto de vista del Aprendizaje Automático. Para mitigar el sobreajuste es posible reducir la complejidad de un modelo.

En el contexto de redes neuronales, algunas de las técnicas de regularización más comúnmente utilizadas son: *Dropout* y *Batch Normalization*. La primera implica dejar activa a una neurona con una probabilidad p. Esto conlleva a reducir la cantidad de conexiones entre neuronas reduciendo así la complejidad de la red. Si no se optimiza la intensidad del *dropout* (el valor de la probabilidad de que la neurona quede activa) se suele optar por el valor por defecto de 0.5 por ser un valor que brinda resultados óptimos (Srivastava *et al.*, 2015). En cambio la segunda, incrementa la estabilidad de la red normalizando las salidas (activaciones) de la capa anterior, restándoles a cada una la media del conjunto de activaciones y dividiendo luego por la desviación estándar del mismo.

3.2. Modelos de Redes Neuronales Profundas

Además de poder explorar diseñando modelos de RNPs propios o variantes de otros existentes, hay modelos de RNPs preentrenados que están disponibles para descarga libre y gratuita en distintas páginas web y para diferentes entornos de trabajo. Una RNP preentrenada se caracteriza por presentar una arquitectura definida con todos sus parámetros (pesos y sesgos) ya determinados para la tarea para la cual fue construida.

A continuación se nombran algunos de los modelos populares preentrenados que han sido estudiados y suelen mencionarse en diversos artículos del área (Benenson, 2016; Canziani *et al.*, 2016): *VGG-16*, *VGG-19*, *ResNet-50*, *Inception-v3*, *InceptionResNet-v2*, *DenseNet* y *NASNet*, disponibles en Keras[2] y con acceso a los mismos a través de TensorFlow y Theano. Además, se tienen *CaffeNet*, *AlexNet*, *R-CNN* y *GoogLeNet*, entre muchos otros, encontrados en Caffe[3]. También están *AlexNet*, *VGG-16*, *VGG-19*, *ResNet-50*, *ResNet-101*, *Inception-v3* y *GoogLeNet*, y con acceso a los modelos de Caffe y Keras, por medio de MATLAB[4]. Asimismo hay una gran variedad de modelos compartidos por usuarios de Caffe en la página de la comunidad *Model Zoo*: `https://github.com/BVLC/caffe/wiki/Model-Zoo`.

Cabe mencionar que para una misma RNP la exactitud de clasificación reportada suele variar en pequeña medida. Esto depende fundamentalmente de cómo el modelo fue entrenado (si sólo se replica la arquitectura de una RNP definida en algún estudio) y de los tipos y condiciones de los entornos de trabajo (para RNPs preentrenadas e incluso arquitecturas de RNPs para entrenar).

[2] `https://keras.io/applications`
[3] `http://caffe.berkeleyvision.org/model_zoo.html`
[4] `https://la.mathworks.com/help/nnet/ug/pretrained-convolutional-neural-networks.html`

3.3. Entornos Computacionales

Para el desarrollo, e incluso el intercambio, de RNPs existen diversos entornos computacionales de Aprendizaje Profundo que fueron desarrollados a partir de distintas fuentes. Estos entornos de código abierto y libre contienen librerías de *software* para RNPs. Entre las más populares se encuentran (Shi *et al.* 2017):

- **Keras.** Disponible desde 2015 y desarrollada como parte del proyecto ONEIROS (*Open-ended Neuro-Electronic Intelligent Robot Operating System*), y cuyo autor principal es François Chollet, de *Google*. Keras es una librería para Redes Neuronales Artificiales de alto nivel escrita en lenguaje de programación Python. Puede ejecutarse sobre los entornos TensorFlow, CNTK y Theano como soportes (*backends*, en inglés), por lo tanto provee una experiencia más rápida y universal.
- **TensorFlow.** Introducida en 2015 por *Google Brain*. Soporta lenguajes de programación C++, Python y CUDA. Se caracteriza por permitir cálculo numérico de alto rendimiento como la manipulación tensorial. Presenta una arquitectura flexible que permite una fácil implementación en gran variedad de plataformas y dispositivos, desde ordenadores de escritorio y *clusters* de servidores hasta dispositivos móviles y periféricos. Asimismo, cuenta con un sólido apoyo para Aprendizaje Automático y Profundo e incluso otras áreas científicas.
- **Theano.** Disponible desde 2017 y desarrollada por el *Laboratoire d'Informatique des Systèmes Adaptatifs* (LISA) de la *Université de Montréal*. Soporta lenguaje de programación Python. Se caracteriza por ser una librería de cómputo numérico que permite definir, optimizar y evaluar expresiones matemáticas que involucran matrices multidimensionales de manera muy eficiente.
- **Scikit-learn.** Introducida en 2007 como un proyecto en *Google Summer of Code* de David Cournapeau. Soporta lenguaje de programación Python, Cython, C y C++. Se caracteriza por ser una librería de Aprendizaje Automático con herramientas simples y eficientes para minería y análisis de datos.
- Existen otras librerías como Caffe2, CNTK, MXNet, PyTorch, Torch y Paddle.

La existencia de tales entornos computacionales son muy convenientes para la investigación en RNPs y para el diseño de aplicaciones y dispositivos eficientes para el cómputo de RNPs. Asimismo se suelen incorporar optimizadores de entornos y *software* y aceleradores de *hardware* tal como la popular librería *cuDNN*[5] de *Nvidia* que permite obtener una rápida ejecución sobre GPUs.

[5] `https://developer.nvidia.com/cudnn`

Capítulo 4

Metodología

Este capítulo trata sobre la metodología empleada para llevar a cabo los estudios realizados. Para ello, se diseñaron RNPs densas y convolucionales y se planificó el proceso de aprendizaje para entrenar y evaluar las redes en una típica tarea de clasificación multiclase de imágenes. De esta manera, se obtuvieron los datos necesarios para los estudios: la cantidad de parámetros y número de unidades ocultas de cada arquitectura, el tiempo que le demanda a cada red aprender de los datos, la exactitud de desempeño en la tarea de clasificación y los valores de pesos y sesgos aprendidos.

El aprendizaje que se planteó es de tipo supervisado ya que se entrenó cada red mediante pares entrada-salida bien definidos, es decir, imagen y etiqueta, usando un conjunto de datos. Los códigos en Python para la reproducción de los experimentos pueden encontrarse en *Github* a través del siguiente *link*: `https://github.com/agustinadinamarca/Learning-and-Analysis-of-Deep-Artificial-Neural-Networks`.

A continuación se detallan las características del conjunto de datos sobre el cual se aprendieron RNPs, la arquitectura o estructura de las redes que se propusieron para el aprendizaje, la metodología de aprendizaje y el método para la visualización de patrones en redes convolucionales.

4.1. Conjunto de Datos

Para que una red neuronal aprenda es necesario enseñarle la fuente o material de donde aprender. Por lo tanto, se puede pensar a la red como si fuese un estudiante, en donde es necesario un maestro que le enseñe el contenido en una primera instancia y luego lo evalúe por medio de un examen o prueba para ver cuánto aprendió. Con esta analogía en mente, en esta sección se presenta y describe brevemente CIFAR-10, el conjunto de datos que se utilizó para el aprendizaje de RNPs.

El nombre de CIFAR-10 proviene del hecho de que es un subconjunto de imágenes etiquetadas en el cual cada imagen pertenece a una de diez categorías posibles. Dicho subconjunto fue recuperado de un conjunto de datos recopilado por el *Canadian Institute for Advanced Research* (CIFAR). Más precisamente, CIFAR-10 es una muestra del conjunto *80 Million Tiny Images*[1] recolectada por Alex Krizhevsky, Vinod Nair y Geoffrey Hinton (Krizhevsky, 2009). Cabe mencionar que *80 Million Tiny Images* es descripto en detalle en Torralba *et al.* (2008).

Por su parte, CIFAR-10[2] consiste en 60,000 imágenes de baja resolución, particularmente de 32×32 píxeles a color (RGB). En cada una de ellas aparece un objeto principal perteneciente a una de diez clases posibles: avión, auto, ave, gato, ciervo, perro, rana, caballo, barco, o camión. La Figura 4.1 muestra una ilustración del conjunto.

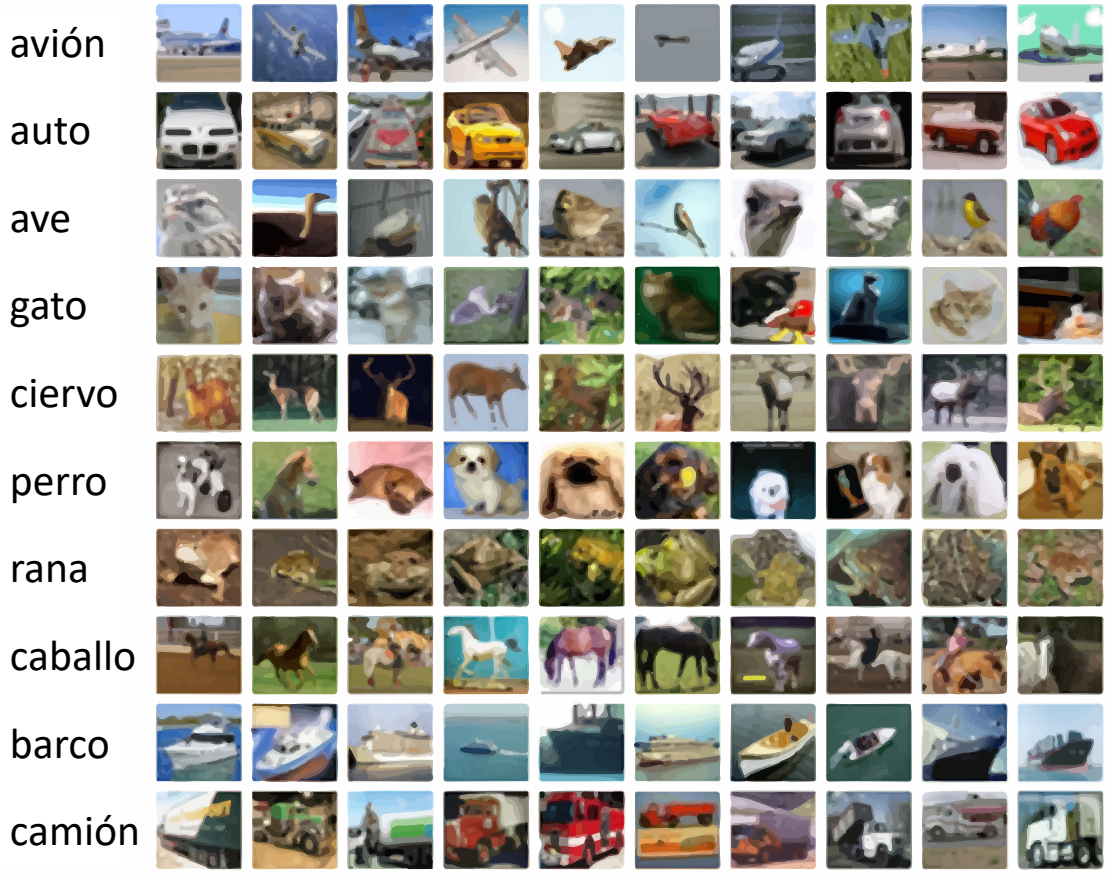


FIGURA 4.1: Ilustración de las 10 clases de objetos del conjunto de datos CIFAR-10 mediante la elección aleatoria de 10 ejemplos por clase. Adaptado de Krizhevsky (2009).

A su vez, el conjunto de aprendizaje está dividido en dos partes: 50,000 imágenes de entrenamiento y 10,000 de prueba. La Figura 4.2 muestra un diagrama ilustrativo. Cada parte presenta una misma cantidad de imágenes por clase: 5,000 para entrenamiento y

[1] `http://groups.csail.mit.edu/vision/TinyImages`
[2] `https://www.cs.toronto.edu/~kriz/cifar.html`

1,000 para prueba. Cabe mencionar que el conjunto de entrenamiento es aquel que se le muestra a la red para que aprenda, es decir, para que busque características o patrones en las imágenes que le permitan hacer la asociación correcta entre objeto y etiqueta. En cambio, el conjunto de prueba permitirá dar una noción de cómo la red aprendió luego de que fue entrenada, es decir, que tan bien fue capaz de generalizar en base a ejemplos particulares.

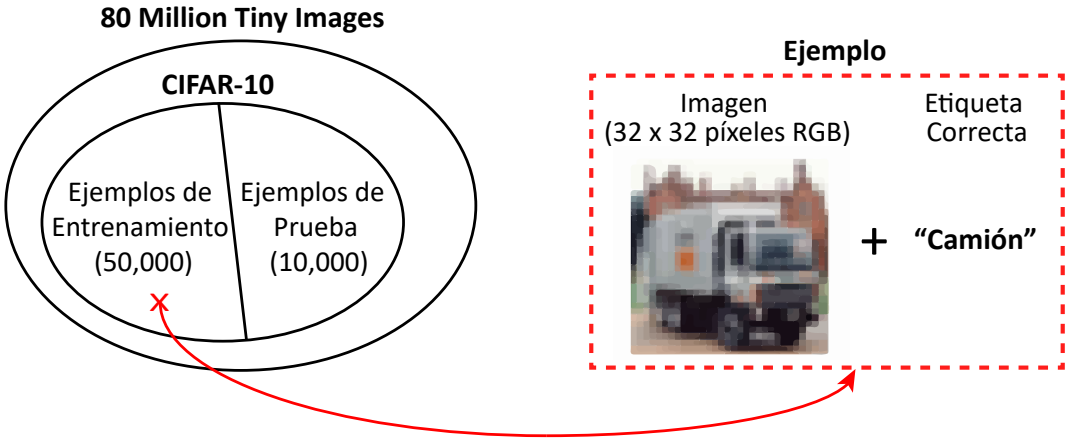


FIGURA 4.2: Diagrama del conjunto de datos CIFAR-10 como una porción del conjunto de imágenes etiquetadas *80 Millon Tiny Images*. CIFAR-10 cuenta con 50,000 ejemplos para entrenamiento y 10,000 para prueba. A la derecha, un ejemplo representativo formado por una imagen de 32×32 píxeles a color y su respectiva etiqueta correcta.

Finalmente, cabe mencionar que actualmente el conjunto de datos se encuentra disponible en tres versiones distintas: Python, MATLAB y binario. En particular, en este trabajo se usó la versión para Python ya que es el lenguaje de programación utilizado a lo largo de este trabajo. Para quien desee mayor información sobre las características de los datos para ser utilizados en aquel lenguaje vea el Apéndice A.

4.2. Arquitecturas

Se consideró aprender RNPs densas (RDs) y convolucionales (RCs). La elección de sus correspondientes arquitecturas implicó la determinación del número de capas ocultas y, así como también, la cantidad y tipo de unidades por capa oculta y capa de salida. Debido a la innumerable variedad de configuraciones estructurales que podrían tener aquellas redes, se investigó en la literatura sobre arquitecturas densas y convolucionales ya estudiadas en CIFAR-10. Asimismo, se tuvo en cuenta bibliografía referida a un conjunto de similares características, STL-10, en el cual los datos son imágenes naturales de objetos RGB y de tamaño 96×96 píxeles cada una (Coates *et al.*, 2011). Esta búsqueda

preliminar permitió orientar el diseño estructural de las redes hacia arquitecturas que han sido adecuadas para la clasificación de imágenes sobre tales conjuntos de datos.

4.2.1. Arquitecturas Densas

A continuación se describen arquitecturas de RDs existentes y también versiones propias de similares características. La Tabla 4.1 resume la arquitectura de cada red.

- **RD01**. La estructura está basada en uno de los modelos propuestos por Ba & Caruana (2013). La misma consta de dos capas ocultas conformadas por 2,000 unidades sigmoides cada una.
- **RD02**. Presenta la arquitectura de una de las redes sugeridas por Lin *et al.* (2015). Esta está formada por dos capas ocultas con 4,000 unidades ReLU cada una.
- **RD03**. Corresponde a otra estructura implementada por Lin *et al.* (2015). Esta presenta tres capas ocultas: la primera con 4,000 unidades ReLU, la segunda con 1,000 unidades lineales y la tercera con 4,000 unidades ReLU.
- **RD04**. La arquitectura tiene cinco capas ocultas. La primera presenta 2,000 unidades tanh, la segunda 500 unidades lineales, la tercera 1,000 unidades tanh, la cuarta 250 unidades lineales y la quinta 500 unidades tanh respectivamente.
- **RD05**. La estructura presenta tres capas ocultas: la primera con 3,000 unidades sigmoides, la segunda con 1,000 unidades lineales y la tercera con 2,000 unidades tanh.
- **RD06**. La arquitectura presenta cinco capas ocultas. Hay 3 capas con 1,000 unidades ReLU cada una intercaladas con una capa de 200 unidades lineales.
- **RD07**. La arquitectura presenta tres capas ocultas: la primera con 1,000 unidades tanh, la segunda con 2,000 unidades ReLU y la tercera con 3,000 unidades tanh.
- **RD08**. La estructura está formada por dos capas ocultas de 5,000 y 2,000 unidades PReLU respectivamente.
- **RD09**. La red tiene tres capas ocultas: la primera con 5,000 unidades PReLU, la segunda con 1,000 unidades lineales y la tercera con 2,000 unidades PReLU.
- **RD10**. La estructura tiene cuatro capas ocultas de 2,000 unidades cada una. La primera y la tercera con activaciones *Leaky* ReLU y la segunda y la cuarta con activaciones PReLU.

- **RD11**. La red está formada por dos capas ocultas: una con 3,000 unidades PReLU y la otra con 2,000 unidades *Thresholded* ReLU.

Cabe mencionar que cada RD recibe un vector de entrada (imagen) de 3,072 elementos, es decir, los valores de intensidad de una imagen típica de CIFAR-10. Asimismo, en todas las arquitecturas se implementó una capa de salida con 10 unidades softmax. De esta manera, la capa de salida produce una distribución de probabilidades sobre las 10 clases de objetos. Una capa de salida con este tipo de unidades y con un número igual a la cantidad de clases o categorías suele ser la opción preferencial en las tareas de clasificación multiclase (Nielsen, 2015).

Tabla 4.1: Arquitecturas Densas. Abreviaturas: k equivale a 1,000.

Red	Arquitectura
RD01 (Ba & Caruana, 2013)	2k sigmoide-2k sigmoide-10 softmax
RD02 (Lin *et al.*, 2015)	4k ReLU-4k ReLU-10 softmax
RD03 (Lin *et al.*, 2015)	4k ReLU-1k lineal-4k ReLU-10 softmax
RD04	2k tanh-0.5k lineal-1k tanh-0.25 lineal-0.5 tanh-10 softmax
RD05	3k sigmoide-1k lineal-2k tanh-10 softmax
RD06	1k ReLU-0.2k lineal-1k ReLU-0.2k lineal-1k ReLU-10 softmax
RD07	1k tanh-2k ReLU-3k tanh-10 softmax
RD08	5k PReLU-2k PReLU-10 softmax
RD09	5k PReLU-1k lineal-2k PReLU-10 softmax
RD10	2k *Leaky* ReLU-2k PReLU-2k *Leaky* ReLU-2k PReLU-10 softmax
RD11	3k PReLU-2k *Thresholded* ReLU-10 softmax

4.2.2. Arquitecturas Convolucionales

A continuación se describen arquitecturas de RCs existentes y también versiones propias de similares características. La Tabla 4.2 resume la arquitectura de cada red.

- **RC01**. La estructura está basada en uno de los modelos propuestos por Hinton *et al.* (2012). La misma consta de tres capas convolucionales seguidas por una capa de *pooling* cada una. Por último, hay una capa localmente conectada (no convolucional). Las capas convolucionales presentan 64 filtros de tamaño 5×5 cada una y con funciones de activación ReLU. Las capas de *pooling* resumen regiones de tamaño 3×3 con corrimiento 2. La primer capa de esta clase realiza *max-pooling* mientras que, la segunda y la tercera, *average-pooling* (Hinton *et al.* (2012) usa *stochastic-pooling* en la tercera). La capa localmente conectada presenta 16 filtros de tamaño 3×3. Éstos fueron aplicados con corrimiento de 1 píxel a lo largo del ancho y alto de volumen de activaciones de entrada, y sin relleno con ceros en el borde de dicho volumen.

- **RC02**. Presenta la arquitectura de una de las redes sugeridas por Zeiler & Fergus (2013). Esta está formada por tres capas convolucionales seguidas por una capa de *pooling* cada una. Las dos primeras capas convolucionales presentan 64 filtros de tamaño 5×5 cada una y con funciones de activación ReLU, mientras que, la tercera tiene 128 filtros de la misma clase que los de la primera. Asimismo, las capas de *pooling* hacen *average-pooling* con corrimiento 2 sobre regiones de tamaño 3×3.

- **RC03**. Corresponde a la estructura de *Model C* de Springenberg *et al.* (2014). La misma está conformada por siete capas convolucionales y tres capas de *pooling*. Las dos primeras capas convolucionales presentan 96 filtros de tamaño 3×3 y funciones de activación ReLU cada una. A continuación de estas capas le sigue una capa de *pooling* que realiza *max-pooling* con corrimiento 2 sobre regiones de tamaño 3×3. Luego, hay una tercera y cuarta capa convolucionales de 192 filtros de tamaño 3×3 con funciones de activación ReLU cada una. Le sigue una capa de *pooling* que hace *max-pooling* con corrimiento 2 sobre regiones de tamaño 3×3. A continuación hay tres capas convolucionales de 192, 192 y 10 filtros de tamaño 3×3, 1×1 y 1×1 respectivamente y con funciones de activación ReLU. Finalmente hay una capa de *pooling* que realiza *global average-pooling* sobre una región de tamaño 6×6.

- **RC04**. La arquitectura está formada por dos capas convolucionales y dos capas de *pooling*. La primera capa convolucional tiene 96 filtros de tamaño 5×5 con unidades ReLU. La segunda capa tiene 64 filtros de la misma clase que los de la primera. Las dos capas de pooling hacen *average-pooling* sobre regiones de tamaño 3×3 y con corrimiento 2.

- **RC05**. La estructura presenta dos capas convolucionales, dos capas de *pooling* y una capa localmente conectada. La primera capa convolucional tiene 256 filtros de tamaño 3×3 con unidades ReLU y la segunda capa convolucional tiene 128 filtros de la misma clase que los de la primera. Luego de cada capa convolucional hay una capa de *pooling* que resume regiones de tamaño 3×3 y con corrimiento 2. La primera hace *max-pooling* y la segunda hace *average-pooling*. Por último, hay una capa localmente conectada con 64 filtros de tamaño 3×3. Éstos fueron aplicados con corrimiento de 1 píxel a lo largo del ancho y alto de volumen de activaciones de entrada, y sin relleno con ceros en el borde de dicho volumen.

- **RC06**. Esta red tiene una arquitectura que presenta dos capas convolucionales, dos capas de *pooling* y dos capas completamente conectadas. La primera capa convolucional consta de 96 filtros de tamaño 5×5 con unidades ReLU, mientras que la segunda capa convolucional tiene 192 filtros de la misma clase que los de la primera. Luego de cada una de ellas hay una capa de *pooling* que resume regiones de tamaño 3×3 y con corrimiento 2. La primera hace *average-pooling* y la segunda

hace *max-pooling*. Por último, le siguen dos capas completamente conectadas de 2,000 unidades PReLU cada una.

- **RC07**. La arquitectura tiene dos capas convolucionales, dos capas de *pooling* y una capa completamente conectada. La primera capa convolucional consta de 64 filtros de tamaño 5×5 con unidades ReLU, mientras que la segunda capa convolucional tiene 128 filtros de la misma clase que los de la primera. Seguido a cada capa convolucional hay una capa de *pooling* que hace *max-pooling* y que resume regiones de tamaño 3×3 y con corrimiento 2. Finalmente, le sigue una capa completamente conectada con 1,000 unidades ReLU.

- **RC08**. La estructura presenta cinco capas convolucionales y cinco capas de *pooling*. Cada capa convolucional tiene 96 filtros de tamaño 5×5 con unidades ReLU. Luego de cada capa convolucional hay una capa de *pooling* que hace *average-pooling* con corrimiento 2 para resumir regiones de tamaño 2×2.

- **RC09**. Esta red tiene una arquitectura que presenta seis capas convolucionales y tres capas de *pooling*. Cada capa convolucional tiene 64 filtros de tamaño 3×3 con unidades ReLU. Luego, cada dos capas convolucionales hay una capa de *pooling* que hace *max-pooling* con corrimiento 2 para resumir regiones de tamaño 2×2.

- **RC10**. La arquitectura tiene dos capas convolucionales, dos capas de *pooling* y una capa completamente conectada. La primera capa convolucional consta de 32 filtros de tamaño 5×5 con unidades ReLU, mientras que la segunda capa convolucional tiene 32 filtros de la misma clase que los de la primera pero de tamaño 3×3. Seguido a cada capa convolucional hay una capa de *pooling* que hace *average-pooling* con corrimiento 2 y que resume regiones de tamaño 3×3 y 2×2 respectivamente. Finalmente, le sigue una capa completamente conectada con 1,000 unidades ReLU.

- **RC11**. La arquitectura tiene dos capas convolucionales y dos capas de *pooling*. La primera capa convolucional consta de 64 filtros de tamaño 5×5 con unidades ReLU, mientras que la segunda capa convolucional tiene 64 filtros de la misma clase que los de la primera pero de tamaño 3×3. Seguido a cada capa convolucional hay una capa de *pooling* que hace *average-pooling* con corrimiento 2 y que resume regiones de tamaño 3×3 y 2×2 respectivamente.

Cabe mencionar que el desplazamiento de los filtros en todas las RCs propuestas se aplican con corrimiento de 1 píxel a lo largo del ancho y alto de los volúmenes de activaciones. Asimismo, la magnitud del relleno con ceros en los bordes de aquellos volúmenes fue el necesario para preservar el ancho y alto de los mismos luego de la operación de convolución. Por otro lado, cada modelo convolucional recibe un tensor de

Tabla 4.2: Arquitecturas Convolucionales. Abreviaturas: c: convolucional; k: 1,000; p: *pooling*; lc: localmente conectada.

Modelo	Arquitectura
RC01	64c-p-64c-p-64c-p-16lc-10 softmax
RC02 (Zeiler & Fergus, 2013)	64c-p-64c-p-128c-p-10 softmax
RC03 (Springenberg *et al.*, 2014)	96c-96c-p-192c-192c-p-192c-192c-10c-p-10 softmax
RC04	96c-p-64c-p-10 softmax
RC05	256c-p-128c-p-64lc-10 softmax
RC06	96c-p-192c-p-2k PReLU-2k PReLU-10 softmax
RC07	64c-p-128c-p-1k ReLU-10 softmax
RC08	96c-p-96c-p-96c-p-96c-p-96c-p-10 softmax
RC09	64c-64c-p-64c-64c-p-64c-64c-p-10 softmax
RC10	32c-p-32c-p-1k ReLU-10 softmax
RC11	64c-p-64c-p-10 softmax

entrada (imagen) de tamaño $32 \times 32 \times 3$. Finalmente, al igual que en las redes densas, en todas las arquitecturas se implementó una capa de salida con 10 unidades softmax.

4.3. Metodología de Aprendizaje

Aprender una red es resolver un problema de optimización. Para ello se buscan los parámetros de la red (pesos y sesgos) Θ que minimizan una función de costo o error L para cada ejemplo $\boldsymbol{x_i}$ en el conjunto de entrenamiento $\mathbb{X}$. La optimización se lleva a cabo mediante algún algoritmo basado en el gradiente descendente. Para cualquier algoritmo de optimización basado en aquel es necesario calcular el gradiente de la función de costo. En redes neuronales dicho gradiente se calcula mediante el algoritmo de retropropagación.

El aprendizaje de RNPs se llevó a cabo en una computadora con Unidad de Procesamiento Gráfico (GPU) Nvidia *GeForce GTX 970*. Con aquel fin se diseñaron programas en lenguaje de programación Python que hacen uso de las librerías Keras (Chollet, 2015) y Scikit-learn (Pedregosa *et al.*, 2011) de Aprendizaje Profundo y Aprendizaje Automático respectivamente. A continuación se describe el procedimiento y los métodos usuales, de acuerdo al estado del conocimiento, para el aprendizaje RNPs.

4.3.1. Preprocesamiento de Datos

Debido a que el preprocesamiento de los datos mejora notablemente los resultados de muchos algoritmos de aprendizaje se aplicó Contraste Global Normalizado (CGN). Ésta técnica transforma el valor de la característica (píxel) x_{ij} a x'_{ij} restando la media $\mu_{\boldsymbol{x}}$ y

dividiendo por la desviación estándar $\sigma_{\boldsymbol{x}}$ de cada vector imagen $\boldsymbol{x}$:

$$x'_{ij} = \frac{x_{ij} - \mu_{\boldsymbol{x}}}{\sigma_{\boldsymbol{x}}}$$

Esto permite llevar los valores de las características de cada ejemplo a los de una distribución normal de media cero y desviación estándar uno. Particularmente los valores de los píxeles en el ejemplo preprocesado están en el rango [-1, 1]. La Figura 4.3 muestra el efecto del preprocesado con CGN.

FIGURA 4.3: Ilustración del preprocesamiento en un ejemplo del conjunto de datos CIFAR-10. De izquierda a derecha, un ejemplo sin preprocesar, el ejemplo preprocesado con Contraste Global Normalizado (CGN) y finalmente la diferencia entre ambas imágenes para mostrar el efecto del CGN.

4.3.2. Inicialización de Parámetros

Antes de comenzar con el aprendizaje de redes es necesario inicializar sus parámetros: pesos y sesgos. Sus valores iniciales tienen un efecto significativo en el proceso de entrenamiento y en su convergencia. Se inicializaron los sesgos con valor constante cero y los pesos mediante el inicializador *Glorot Normal* o también conocido como *Xavier Normal*. Este otorga valores muestreando una distribución normal truncada con media cero y desviación estándar $\sqrt{2/(n_{in} + n_{out})}$, donde n_{in} y n_{out} son el número de unidades de entrada y de salida en la matriz o el tensor de pesos respectivamente. Sin embargo, en el caso particular de redes con unidades rectificadas, se utilizó el método de inicialización de pesos *He Normal*. Éste otorga valores muestreando una distribución normal truncada con media cero pero con desviación estándar $\sqrt{2/n_{in}}$.

4.3.3. Función de Costo o Error

La manera de evaluar la calidad de un conjunto particular de parámetros Θ de una red está basada en que tan bien la clase predicha por la misma $y^*(\boldsymbol{x}_i)$ se aproxima a la clase

verdadera $y(\boldsymbol{x})$ para cada ejemplo $\boldsymbol{x_i}$ (imagen) del conjunto de entrenamiento $\mathbb{X}$. Aquel grado de aproximación se cuantifica mediante una función de costo o error L y se utilizó la función de verosimilitud logarítmica negativa (*cross-entropy*, en inglés):

$$L(\Theta, \mathbb{X}) = -\frac{1}{n}\sum_{i=1}^{n}\sum_{k=1}^{K}\left[y_k^{(i)} log\, h_{\boldsymbol{\theta}}(\boldsymbol{x}^{(i)})_k + (1 - y_k^{(i)}) log\,(1 - h_{\boldsymbol{\theta}}(\boldsymbol{x}^{(i)})_k)\right], \tag{4.1}$$

donde n es el número total de ejemplos en el conjunto de entrenamiento, K es el número de unidades de la capa de salida, y_k^i es la salida verdadera de la unidad k para cada ejemplo i, $h_{\boldsymbol{\theta}}(\boldsymbol{x}^{(i)})_k$ es la salida predicha por la unidad k para el ejemplo i y $\boldsymbol{\theta}$ es el conjunto de parámetros asociado a la unidad k.

4.3.4. Actualización de Parámetros

El gradiente descendente es uno de los algoritmos más populares para llevar a cabo la optimización y por lejos la forma mas común de optimizar redes neuronales.

En la práctica, para encontrar los pesos w_k y los sesgos b_l de la red que minimizan la función de costo L se suele usar el algoritmo gradiente descendente estocástico (*Stochastic Gradient Descent*, en inglés, o simplemente –SGD–). Sin embargo, muchas de las librerías de Aprendizaje Profundo actuales contienen implementaciones de varios algoritmos que optimizan el gradiente descendente (Ruder, 2017). Existen diversas variantes al algoritmo SGD que lo mejoran. El que se adoptó para el aprendizaje de RNPs fue Adam (*Adaptive Moment Estimation*, en inglés).

Adam se caracteriza por ser un tipo de algoritmo con tasa de aprendizaje adaptativa por lo que no es necesario optimizar la tasa de aprendizaje, uno de los hiperparámetros más difíciles de optimizar debido a su gran impacto en el desempeño de los modelos de aprendizaje (Goodfellow *et al.*, 2016; Bengio, 2012). Aquel algoritmo propone ciertas reglas para adaptar de forma automática las tasas de aprendizaje controlando la rapidez de convergencia aumentando o disminuyendo este parámetro basado en el error. Los parámetros utilizados en Adam son los que recomiendan Kingma & Ba (2015) y que en la implementación del algoritmo en Keras son los valores por defecto (lr=0.001, beta_1=0.9, beta_2=0.999, epsilon=10^{-8}, decay=0.0).

4.3.5. Retropropagación

Una forma rápida para calcular el gradiente de la función de costo ∇L, es decir, las derivadas parciales: $\partial L/\partial w_k$ y $\partial L/\partial b_l$ para cada peso w_k y sesgo b_l de la red se logra

mediante el algoritmo de retropropagación o propagación hacia atrás (*backpropagation*, en inglés).

4.3.6. Optimización de Hiperparámetros

El aprendizaje de redes neuronales involucra además especificar parámetros que controlan el comportamiento del proceso de aprendizaje en sí mismo, los hiperparámetros. Ejemplos de ellos son la tasa de aprendizaje, el número de épocas, el número de grupos, entre muchos otros. Es común explorar el espacio de estos parámetros con el fin de obtener el mejor rendimiento en términos de precisión y tiempo de entrenamiento. Es frecuentemente no obvio qué valores habría que elegir, por ello, para resolver este asunto se necesita de un conjunto de validación de ejemplos que el algoritmo de aprendizaje no haya observado antes. Pues en la práctica utilizar el conjunto de prueba llevaría a que el modelo memorice los datos y se obtenga un desempeño optimista de lo que realmente podría dar.

Para hallar los valores óptimos de hiperparámetros para una dada arquitectura de RNP se realizó lo siguiente:

1. Se identificaron los hiperparámetros a optimizar.
2. Se definió un espacio de hiperparámetros acotado a explorar.
3. Se determinó la mejor combinación de hiperparámetros de la RNP mediante una combinación típica de dos algoritmos: validación cruzada y búsqueda aleatoria.

En primer lugar, debido al costo computacional que conlleva realizar una optimización en espacios multiparamétricos, se decidió que los hiperparámetros a optimizar fueran: (i) el tamaño de grupos y (ii) el número de épocas para cada modelo. Los hiperparámetros más comunes de optimizar en el contexto de redes neuronales incluyen la tasa de aprendizaje inicial, la tasa de decaimiento de la tasa de aprendizaje y los valores de regularización como el *dropout*, entre otros. Los hiperparámetros más sensibles, vinculados al tipo de optimizador, como la tasa de aprendizaje, se ajustan automáticamente por ser Adam un algoritmo de tasa de aprendizaje adaptativa. Dicha característica hace que no sea necesario incluir la tasa de aprendizaje dentro de la búsqueda de hiperparámetros. Este es otro de los motivos por el cual se seleccionó Adam como optimizador en este trabajo.

En segundo lugar, es necesario definir un espacio de búsqueda que sea acotado, es decir, hay que fijar los rangos de valores en los que se cree que el modelo puede funcionar con mayor éxito. En base a Karpathy (2017) se exploraron los hiperparámetros en escala

logarítmica debido a que presentan efectos multiplicativos en la dinámica del aprendizaje. Por lo tanto, el tamaño de grupos y el número de épocas se exploraron en rangos típicos, {30, 100, 500, 1,000, 3,000, 5,000, 8,000} y {10, 30, 100, 300, 800, 1,000, 1,500} respectivamente.

En tercer lugar, fue necesario determinar la mejor combinación de hiperparámetros dentro del espacio de búsqueda con la que se obtenga el mejor desempeño para cada RNP. Para cada configuración de hiperparámetros y para una dada arquitectura de RNP, se aplicó el algoritmo de validación cruzada (*k-fold cross-validation*, en inglés, o simplemente –CV–). Éste particionó el conjunto de entrenamiento (50,000 imágenes) en k subconjuntos de igual tamaño. Uno de los subconjuntos se utilizó como datos de prueba y los *k-1* subconjuntos restantes como datos de entrenamiento. De esta manera, se procedió a entrenar y evaluar la RNP. El proceso de CV es repetido durante k iteraciones con cada uno de los posibles subconjuntos de datos como conjunto de prueba. Por lo tanto, para una RNP con una dada configuración de hiperparámetros se la entrenó y evaluó un total de k veces. Al finalizar con las k iteraciones se obtuvieron k medidas de desempeño que se promediaron aritméticamente. El número de particiones k se fijó en 10 por ser una cantidad óptima (Refaeilzadeh *et al.*, 2008). Cabe mencionar que cada configuración de hiperparámetros fue seleccionada al azar en el espacio de hiperparámetros mediante el algoritmo de búsqueda aleatoria. La Figura 4.5 ilustra el procedimiento completo de la optimización de hiperparámetros donde se usa búsqueda aleatoria y CV con $k = 5$.

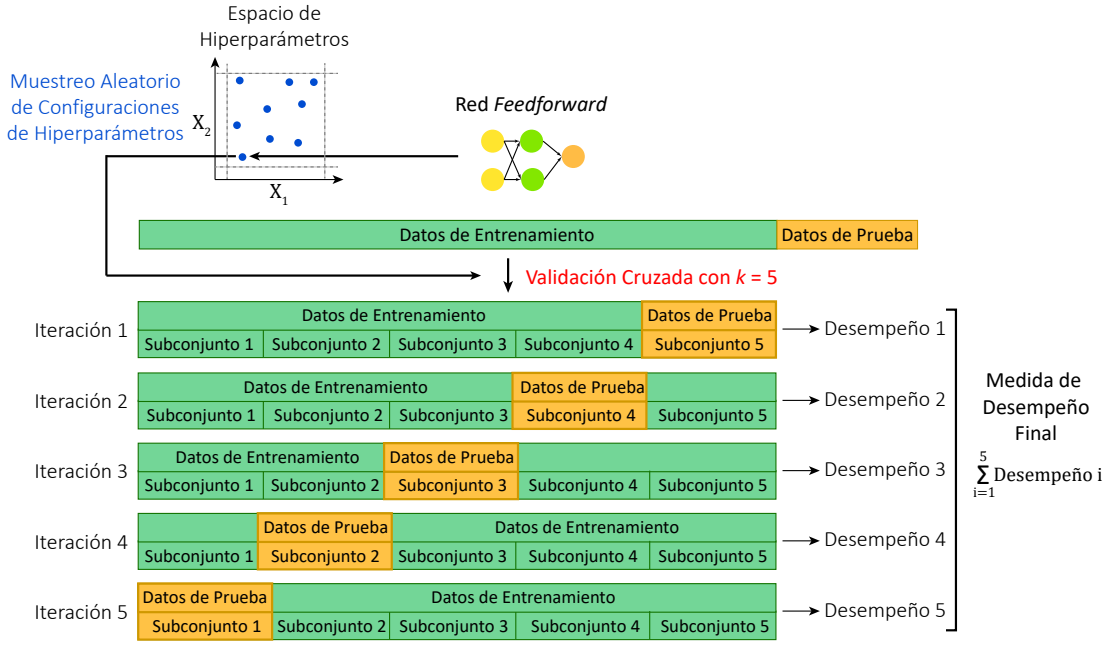


FIGURA 4.4: Ilustración del método de Validación Cruzada con número de particiones k igual a 5.

El algoritmo de CV se aplicó a un conjunto de configuraciones de hiperparámetros el cual fue seleccionado mediante el algoritmo de búsqueda aleatoria. Dicho algoritmo tomó al azar una cantidad de puntos en el espacio de hiperparámetros y en cada uno se aplicó CV a la RNP. Más precisamente la implementación que se usó fue *RandomizedSearchCV* de la librería Scikit-learn. La Figura 4.4 ilustra la optimización de hiperparámetros donde se usa búsqueda aleatoria y CV con $k = 5$. Luego, se compararon las medidas de desempeño de la RNP en los distintos puntos y se eligió la mejor configuración con la que se obtiene mejor desempeño. Una vez que se determinó cuál es la mejor combinación de hiperparámetros, se tomó el conjunto de entrenamiento original (50,000 imágenes) y se entrenó el modelo. Finalmente, se evaluó su desempeño sobre el conjunto de prueba original (10,000 imágenes).

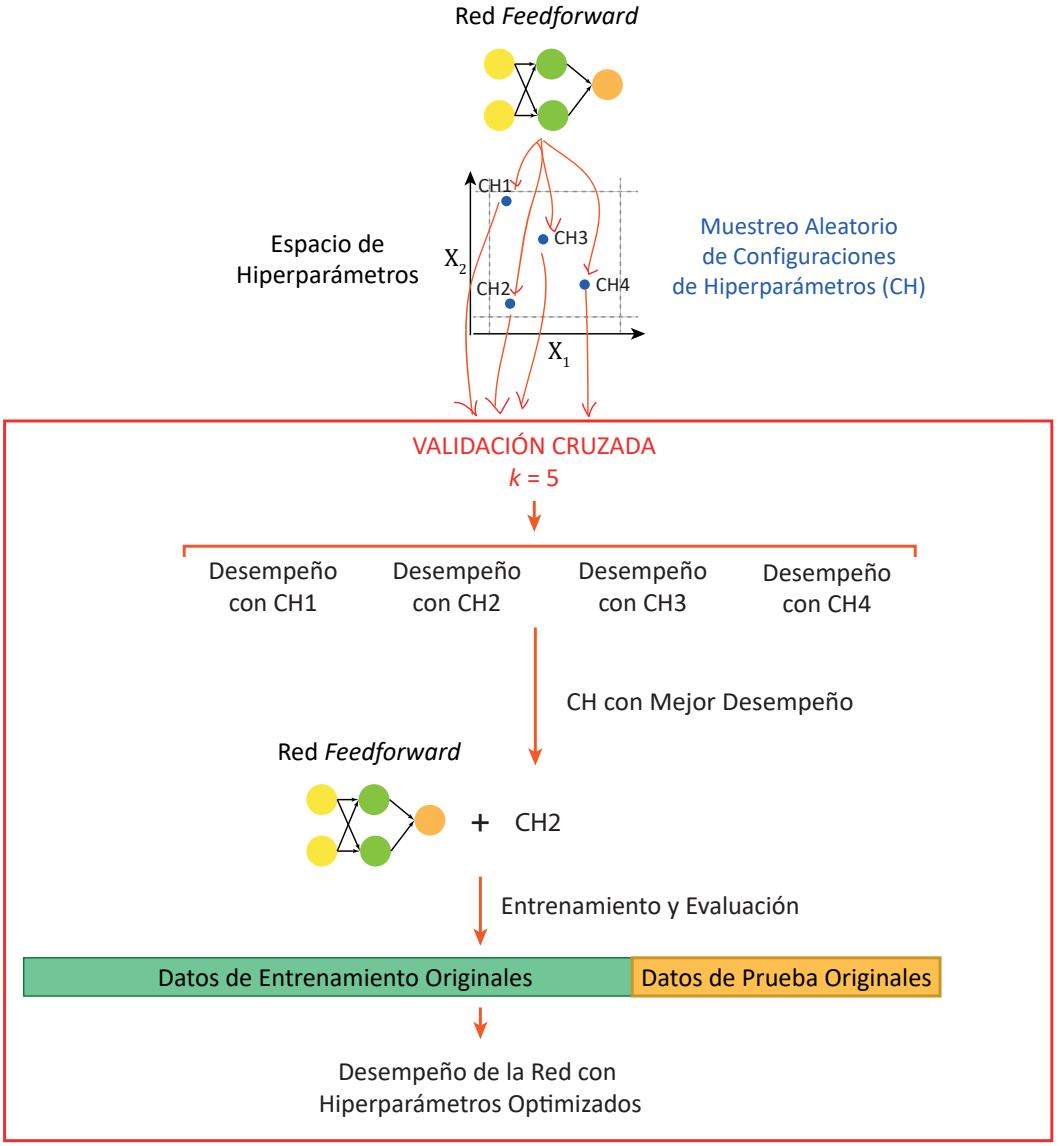


FIGURA 4.5: Ilustración del procedimiento completo utilizado para optimizar hiperparámetros en redes neuronales profundas.

En el Apéndice B se especifican las mejores configuraciones de hiperparámetros para el aprendizaje de cada RNP.

4.3.7. Técnicas de Regularización

Las RNPs suficientemente grandes pueden llegar a memorizar cualquier dato. Este problema denominado sobreajuste (*overfitting*, en inglés) ocurre cuando el modelo aprendido tiene una capacidad de predecir con mucha exactitud aquellos datos con los que fue entrenado pero tiene una capacidad muy pobre de predicción sobre datos que no ha visto anteriormente. Por lo tanto, la RNP aprendida es incapaz de generalizar. En los experimentos se utilizaron las siguientes técnicas comúnmente adoptadas para reducir aquel problema:

- **Dropout**. Este método hace cero las activaciones de las unidades ocultas de forma estocástica y con probabilidad p para cada ejemplo de entrenamiento y para cada época de entrenamiento. En el caso más simple, todas las unidades son retenidas con una misma probabilidad. En el experimento se pudo haber explorado diferentes probabilidades para las diferentes capas pero se optó por fijarlas en 0.5. De acuerdo a Srivastava *et al.* (2014), dicho valor parece estar cerca del óptimo para una amplia gama de redes y tareas.
- **Batch Normalization**. Es un método desarrollado por Ioffe & Szegedy (2015) y consiste en normalizar las salidas de las capas de una red. En la práctica, en algunos casos elimina la necesidad de utilizar *dropout* y suele usarse inmediatamente después de capas completamente conectadas y antes de funciones de activación no lineales.

En el Apéndice B se especifica con detalle las capas en las redes donde se implementó las técnicas de regularización de *dropout* y *batch normalization*.

4.4. Visualización de Patrones en Redes Convolucionales

Para poder ver y analizar los patrones aprendidos por las RCs se utilizó la librería Quiver[3] implementada en lenguaje de programación Python para Keras. Quiver permite la visualización interactiva de las características aprendidas en redes convolucionales profundas. La Figura 4.6 muestra una ilustración de la consola de visualización que permite una simple interacción entre el usuario y la red a explorar.

[3] `https://keplr-io.github.io/quiver`

Para analizar los patrones aprendidos por las convoluciones se tomaron las redes neuronales aprendidas (con los pesos y sesgos ya determinados en el proceso de entrenamiento) y se alimentaron sus entradas con imágenes del conjunto de prueba. Se visualizaron con Quiver los patrones de activación de las neuronas de las capas convolucionales.

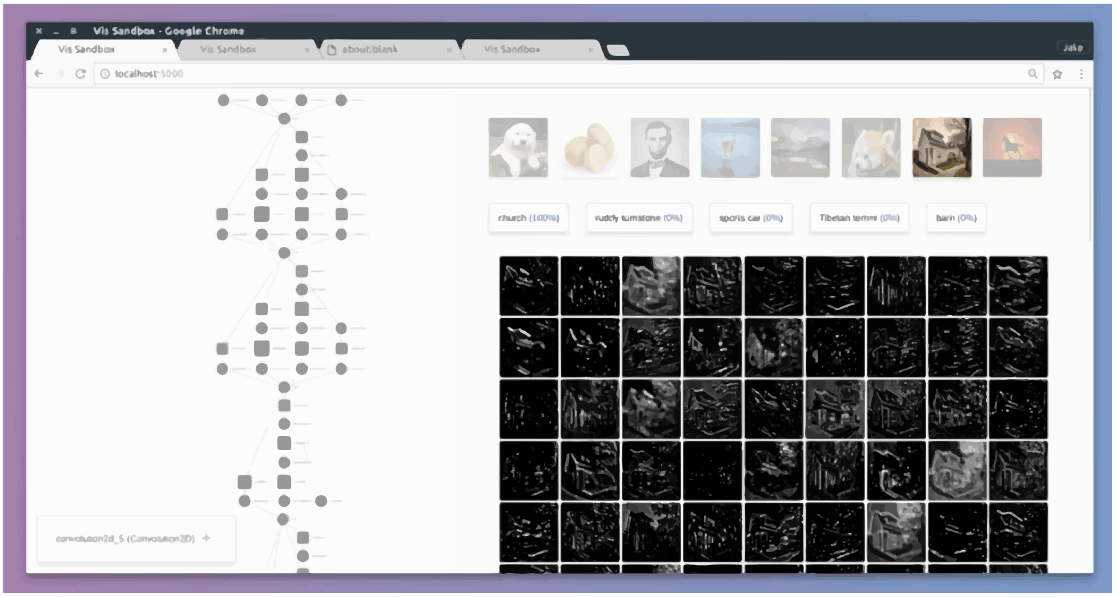


FIGURA 4.6: Ilustración de la consola de visualización de Quiver. Este *software* permite explorar las activaciones de cada capa de una red convolucional. Se observa: un esquema del modelo de aprendizaje (izquierda), imágenes de entrada (arriba a a la derecha) y activaciones de una capa convolucional (abajo a la derecha).

Capítulo 5

Resultados

Este capítulo da cuenta en qué medida las RDs y RCs, son más eficaces y eficientes al compararlas unas frente a las otras a través de la medición de sus desempeños en términos de exactitud de clasificación y tiempo de aprendizaje. Asimismo, da cuenta de la relación entre eficacia y eficiencia, y de cada una de estas dos variables frente a características estructurales de las redes, tal como el número de unidades ocultas y la cantidad de parámetros a aprender. Por último, muestra la identificación y caracterización de los patrones aprendidos por las convoluciones en las RCs.

Aquellos estudios permiten demostrar las hipótesis de esta investigación. Brevemente, las mismas sostienen que las RCs son siempre más eficaces y eficientes que las RDs en la tarea de clasificación multiclase de imágenes. Asimismo, que el número de unidades ocultas en una RNP tiene una correlación positiva frente a su exactitud de clasificación y tiempo de aprendizaje. En cambio, el número de parámetros de una red presenta una correlación negativa frente a aquellas. Por otro lado, se espera que los patrones capturados por las convoluciones en las RCs sean cada vez más abstractos y menos interpretables en capas cada vez más profundas.

Cabe mencionar que medir variables asociadas al aprendizaje de RNPs y conocer sus comportamientos forma parte de los componentes necesarios para cualquier estudio con redes neuronales. Por lo tanto, ésto permite construir modelos de RNPs que sean lo más eficaces y eficientes posibles para cualquier tarea o actividad para la cual fueron diseñados.

Por su parte, los resultados de los estudios llevados a cabo en este trabajo se presentan mediante gráficos de barras, de cajas y de dispersión. Éstos fueron realizados con seaborn[1], una librería de Python para la visualización estadística de datos.

[1] `https://seaborn.pydata.org`

Finalmente, el capítulo se desarrolla en cinco partes. En la primera se consignan los resultados de eficacia, en la segunda los resultados de eficiencia, y en la tercera los correspondientes al número de parámetros y unidades ocultas. Luego, en la cuarta se muestran las correlaciones entre desempeño y características estructurales. Por último, se presenta el estudio de identificación y caracterización de los patrones capturados por las convoluciones en las RCs.

5.1. Eficacia

El objetivo de esta sección es comparar las exactitudes logradas tras el aprendizaje de las RDs y RCs. Por ello, la Figura 5.1 muestra a través de un gráfico de barras un *ranking* de las exactitudes de clasificación logradas por las RNPs. Además, se indica a modo de referencia la exactitud de clasificación promedio alcanzada por una persona en CIFAR-10: ~ 94 %[2].

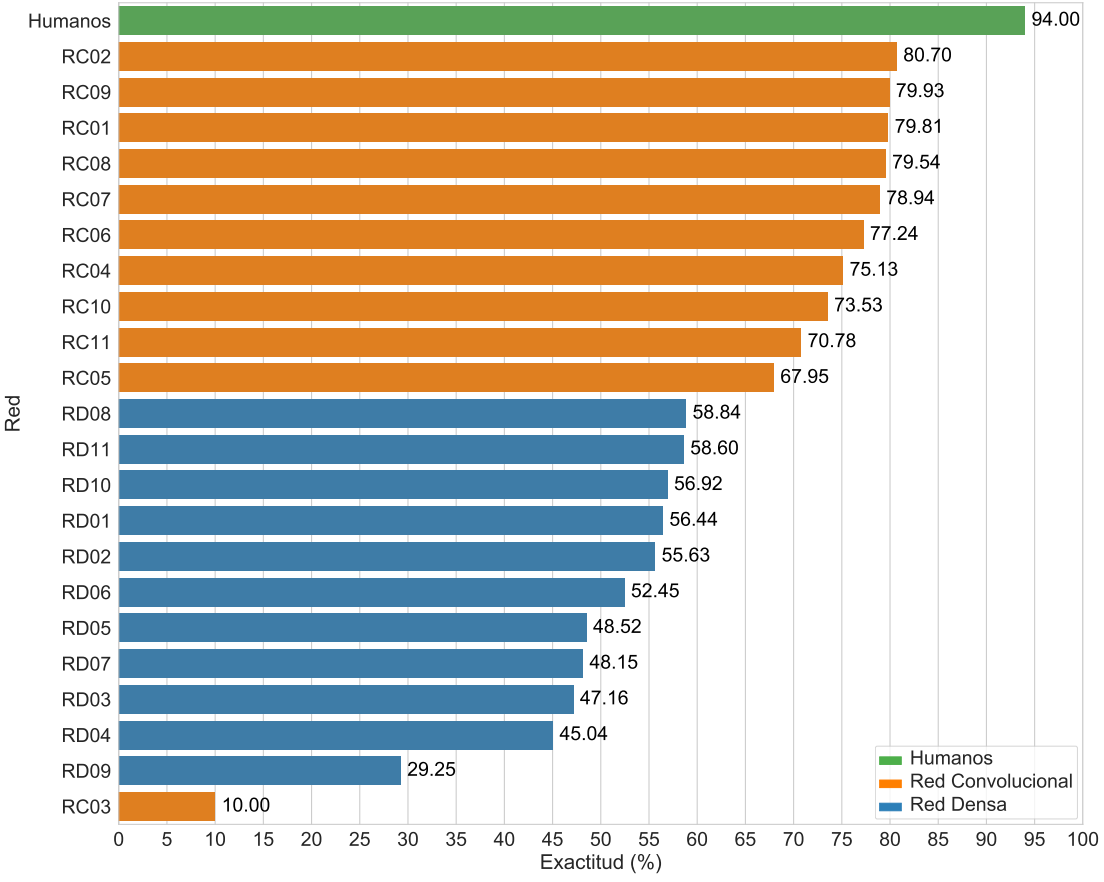


FIGURA 5.1: Eficacia. Gráfico de barras de las exactitudes de clasificación de las redes profundas densas (RDs) y convolucionales (RCs). Se indica además el desempeño promedio de una persona (Humanos) para el mismo conjunto de datos.

[2] `http://karpathy.github.io/2011/04/27/manually-classifying-cifar10`

Los resultados de la Figura 5.1 muestran que de un total de 22 RNPs aprendidas, el máximo desempeño logrado en términos de exactitud de clasificación es de 80.70 % por la red convolucional RC02. Este valor pone de manifiesto que en promedio una persona clasifica un 13.30 % mejor las imágenes que aquel modelo. Asimismo, se observa que 10 de las 11 RCs aprendidas presentan mayores desempeños que las RDs, logrando aquellas exactitudes entre un 67.95 % y un 80.70 %. La excepción es la red RC03 que obtuvo una exactitud de un 10.00 %. Las RDs aprendidas lograron desempeños entre un 29.25 % y un 58.84 %. Por lo tanto, hay una brecha de un 9.11 % entre el mejor modelo denso (RD08 con 58.84 %) y el de menor desempeño convolucional (RC05 con 67.95 %), si se exceptúa la red RC03. En general, aquello indica que las RCs tienen una mayor capacidad de aprender patrones complejos, lo que las hace capaces de clasificar mejor que las RDs.

Por otro lado, la Figura 5.2 indica las exactitudes de clasificación alcanzadas por las RDs y RCs mediante un diagrama de cajas. Este gráfico proporciona una visión general de la simetría de las distribuciones de los datos. Se puede apreciar la existencia de un valor atípico en las RDs y otro en las RCs correspondientes a redes que obtuvieron desempeños menores que el 91 % de las redes restantes en cada caso. La distribución de los datos en las RCs es asimétrica respecto a la mediana, mientras que para las RDs la distribución es prácticamente simétrica respecto de aquel punto. En términos generales, el gráfico muestra que las RCs presentan mejores desempeños en términos de exactitud que las RDs.

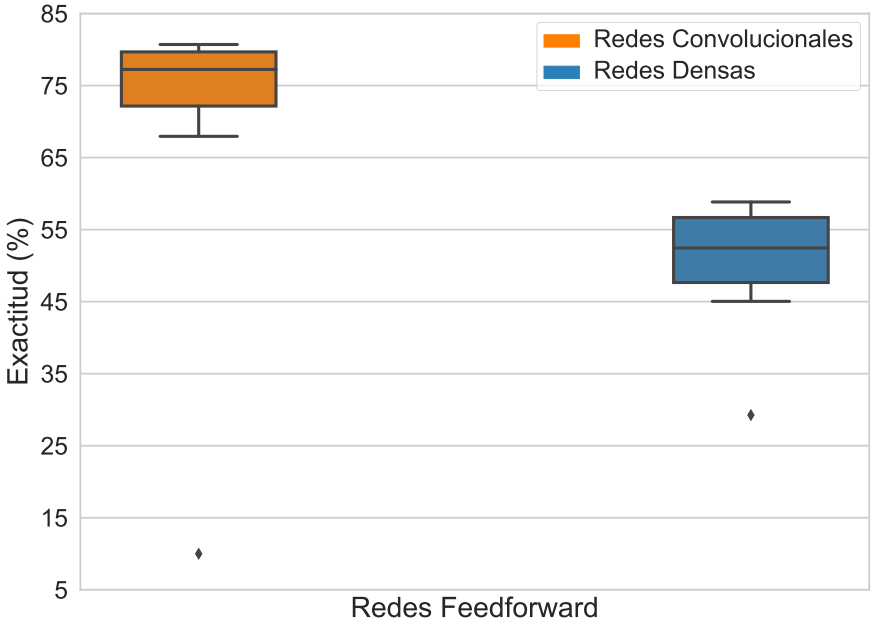


FIGURA 5.2: Eficacia. Diagrama de cajas para las exactitudes de clasificación de las redes profundas densas (RDs) y convolucionales (RCs).

Cabe mencionar que 5 de las redes aprendidas (RD01, RD02, RD03, RC02 y RC03) corresponden a arquitecturas estudiadas en la literatura. La Figura 5.3 indica, además de la exactitud lograda en este trabajo, la exactitud reportada en otros estudios. En todos los casos se observan discrepancias entre ambas exactitudes. Esto es debido a que en este trabajo se priorizó replicar las arquitecturas de red. Es importante destacar que los rangos de hiperparámetros utilizados no corresponden a los utilizados en los respectivos estudios, a lo cual se atribuyen las discrepancias entre los resultados publicados y los obtenidos en este trabajo. Como puede observarse, en algunos casos (RD01 y RD02) dichas discrepancias favorecen a las redes aprendidas en esta tesis.

La Figura 5.3 muestra que las redes RC02 y RC03 obtuvieron desempeños más bajos a los reportados en otros trabajos, específicamente un 0.06 % y un 80.26 % menos respectivamente. Por su parte, los modelos densos RD01 y RD02 lograron exactitudes de clasificación un 14.24 % y un 1.71 % mayores a las reportadas en la literatura en cada caso. Sin embargo, la red RD03 alcanzó un desempeño 9.68 % menor al indicado en otros estudios.

Los resultados obtenidos en este trabajo para los modelos RD01 y RD02 se consideran positivos ya que el aprendizaje implementado incrementó sus desempeños indicados en la literatura (en un 14.24 % y un 1.71 % más respectivamente). Asimismo, el resultado de la red RC02, a pesar de que fue un 0.06 % menor, se considera satisfactorio. En cambio, el desempeño logrado por los modelos RD03 y RC03 fue degradado (en un 9.68 % y un 80.26 % menos en cada caso), principalmente el de la red RC03. Esto se debe a que los métodos de aprendizaje y las parametrizaciones implementadas no fueron óptimas para estas redes. Específicamente, en la literatura se hizo uso de SGD (versión optimizada), adaptaciones de la tasa de aprendizaje previamente determinadas, uso de constantes de decaimiento específicas e implementaciones de *dropout* en la imagen de entrada.

Por otro lado, en el Apéndice C se muestran las matrices de confusión asociadas a cada RD y RC aprendidas en este trabajo. Cada matriz de confusión M presenta tamaño 10×10, donde 10 es el número de clases o categorías en CIFAR-10. Estas matrices permiten visualizar el desempeño de un algoritmo de clasificación. Por columnas se representa el número de predicciones para cada clase, y por filas se representa la cantidad real de instancias en cada clase. En el contexto de este trabajo, cada elemento $M_{i,j}$ de la matriz representa la cantidad de imágenes de la clase i clasificadas como pertenecientes a la clase j. Un clasificador perfecto tiene una matriz de confusión diagonal, es decir, todos los elementos fuera de la diagonal principal toman exactamente el valor 0.

La Figura 5.4 muestra las matrices de confusión de las mejores y peores RDs y RCs en cuanto a desempeño: los modelos RD08 (58.84 %) y RC02 (80.70 %) en el primer caso, y las redes RD09 (29.25 %) y RC03 (10.00 %) en el segundo.

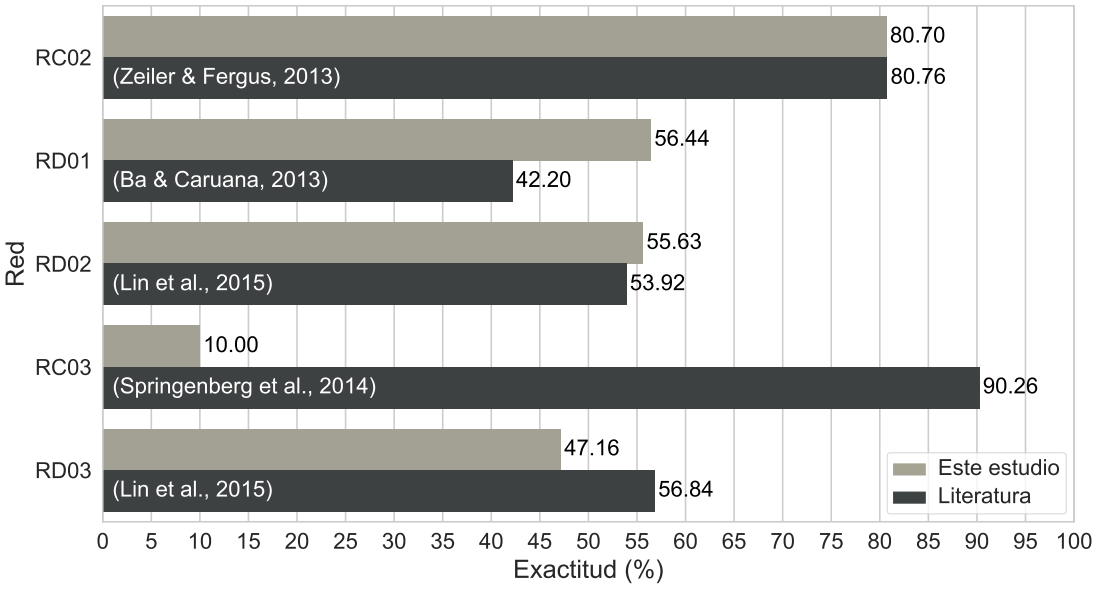


FIGURA 5.3: Comparación entre las exactitudes obtenidas en este trabajo y las alcanzadas en la literatura para las cinco arquitecturas de redes *feedforward* que fueron extraídas de otros estudios.

Por último, vale la pena comentar los resultados del aprendizaje que plasman las matrices de confusión de la Figura 5.4, por ejemplo, las de los modelos con mejor desempeño, RD08 (58.84 %) y RC02 (80.70 %). En primer lugar, las diferencias se ven reflejadas en los valores y colores de las entradas de las matrices. Por ejemplo, para RD08 se ve que los valores en la diagonal (número de imágenes de cada clase que fueron clasificadas correctamente) son más bajos que en RC02. También, aquello se ve en que los colores de las celdas de la diagonal tienden a ser más claros para RD08 que para RC02. A su vez, para RD08 los colores fuera de la diagonal tienden a ser más oscuros que los correspondientes a RC02. Asimismo, en ambos casos, lo que mejor saben clasificar estos modelos son barcos, pues de 1,000 barcos vistos, 814 y 901 fueron clasificados como tal por RD08 y RC02 respectivamente. Por otro lado, lo que más le cuesta aprender a RD08 son aves (418/1,000) y a RC02 gatos (674/1,000). En ambos casos, las clases más confundidas son perros con gatos, 220/1,000 para RD08 y 139/1,000 para RC02. Un análisis similar se puede lograr comparando las matrices de las redes RD09 y RC03. A grandes rasgos, RD09 clasifica cualquiera de las diez clases como pertenecientes a una de dos categorías posibles (auto o caballo). En cambio, RC03 clasifica las diez clases como pertenecientes a solo una categoría (barco).

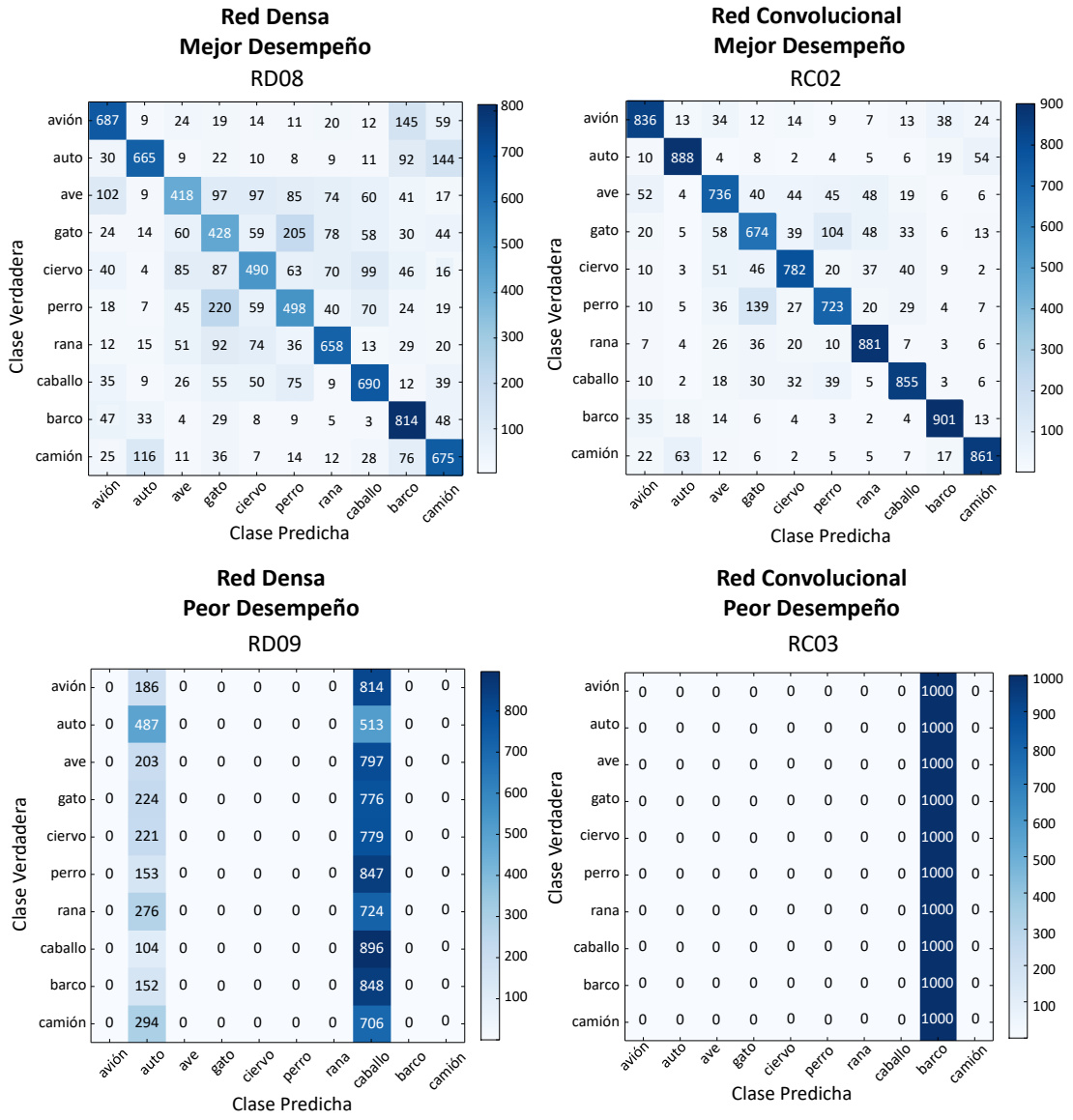


FIGURA 5.4: Matrices de confusión correspondientes a las redes densas y convolucionales con mejor y peor desempeño: RD08 (58.84 %) y RC02 (80.70 %) en el primer caso, y las redes RD09 (29.25 %) y RC03 (10.00 %) en el segundo. Las diferencias entre matrices se ven reflejadas en los valores y colores de las entradas de las mismas. Puede observarse que para el peor caso de red densa (RD09), ésta predice que todas las imágenes contienen caballos o autos. Para el caso de la peor red convolucional (RC03) predice que absolutamente todas las imágenes corresponden a barcos.

5.2. Eficiencia

En esta sección se pretenden comparar los tiempos de aprendizaje alcanzados por las RDs y RCs. Por ello, la Figura 5.5 muestra un gráfico de barras con el *ranking* de los tiempos de aprendizaje alcanzados por las RNPs aprendidas. Los resultados indican que a la RNP que le demandó menor tiempo aprender a clasificar imágenes fue la red densa RD04 con 1.6 min. En cambio, el modelo convolucional más eficiente fue RC05

con 31 min. Más precisamente, 9 de las 11 RDs alcanzaron tiempos entre 1.6 min y 24.6 min, exceptuando las redes RD08 y RD03 con 68.5 min y 83.2 min respectivamente. En cambio, a las RCs les llevó más tiempo aprender que al otro tipo de redes, alcanzando tiempos entre 31 min y 240 min. Por lo tanto, 9 de las 11 RDs fueron mínimamente 6.4 min más eficientes que las RCs aprendidas.

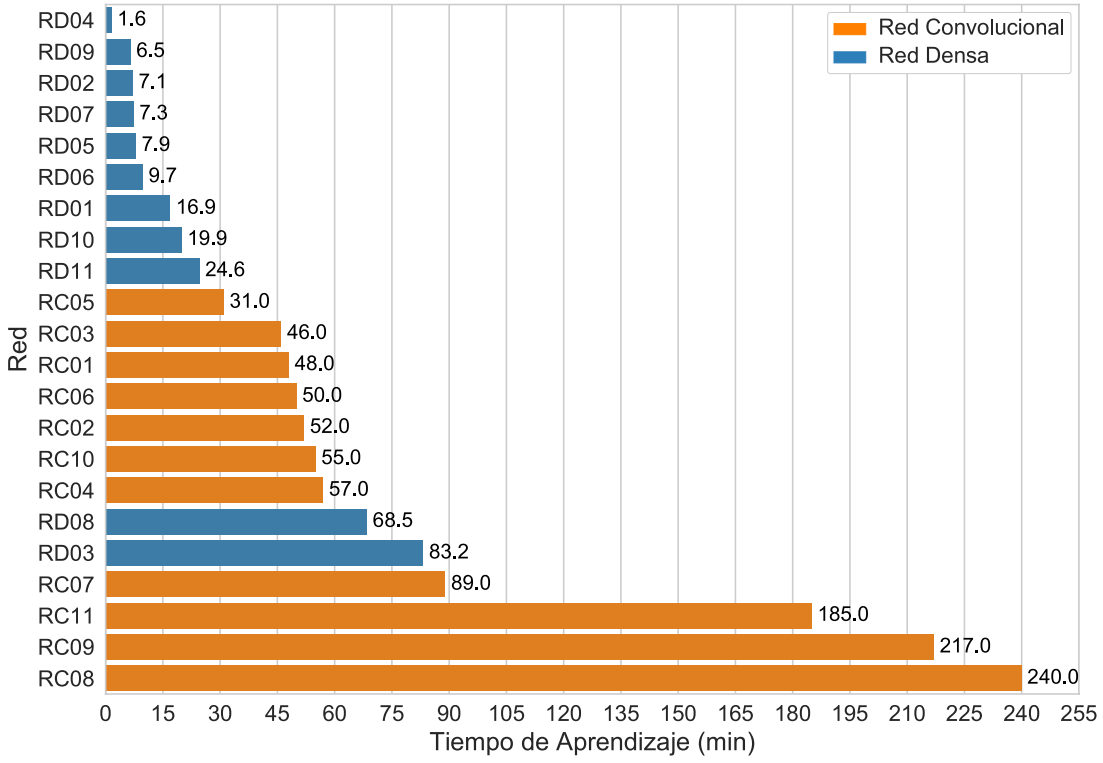


FIGURA 5.5: Eficiencia. Gráfico de barras para el tiempo de aprendizaje, en minutos, de las redes profundas densas (RDs) y convolucionales (RCs).

Por su parte, la Figura 5.6 indica los tiempos de aprendizaje alcanzados por las RDs y RCs mediante un diagrama de cajas. En general, se observa que a las RDs aprendidas les demanda menor tiempo aprender a clasificar imágenes que a las RCs. Se puede apreciar la existencia de 2 valores atípicos en las RDs correspondientes a modelos densos que obtuvieron tiempos de aprendizaje mayores que el 82 % de las RDs restantes. Asimismo, el 50 % de los datos de las distribuciones de ambos modelos presentan una asimetría similar respecto a la mediana o cuartil Q2. Precisamente, un 25 % de los datos, para cada tipo de red, está contenido entre los cuartiles Q2 (50 %) y Q1 (25 %), y otro 25 % de los mismos se encuentra entre los cuartiles Q3 (75 %) y Q2 (50 %), con la particularidad de que en ambos se observa que $(Q3 - Q2) \gg (Q2 - Q1)$.

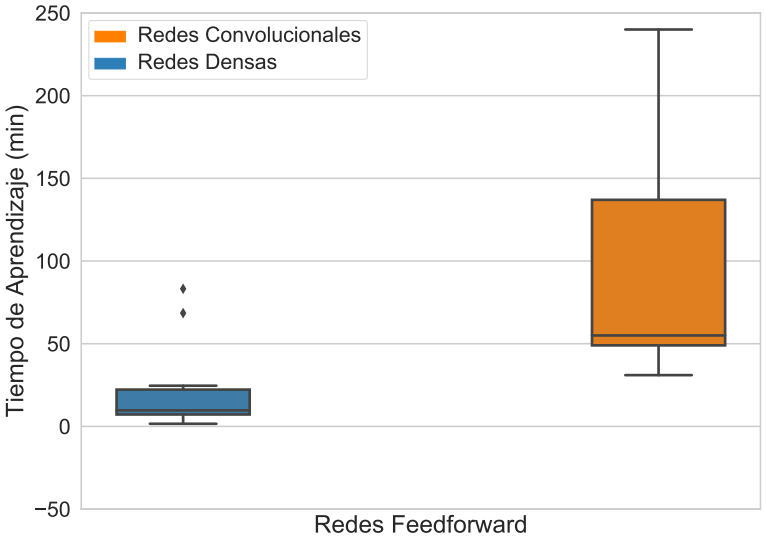


FIGURA 5.6: Eficiencia. Diagrama de cajas para los tiempos de aprendizaje, en minutos, de las redes profundas densas (RDs) y convolucionales (RCs).

5.3. Características Estructurales

El objetivo de esta sección es comparar características estructurales, como la cantidad de parámetros y el número de unidades ocultas, entre RDs y RCs. Por ello, la Figura 5.7 muestra un diagrama de barras con las cantidades de parámetros en las RNPs aprendidas. En el gráfico puede observarse que las RCs tienen una cantidad de parámetros entre 0.1 millones y 29.1 millones, mientras que las RDs, entre 3.9 millones y 28.3 millones. Asimismo, 9 de las 11 RCs son modelos que tienen menores cantidades de parámetros que las RDs. El modelo denso con menor número de parámetros es RD06 con 2.9 millones más que la de menor cantidad de las RCs, la red RC11.

Por su parte, la Figura 5.8 indica las cantidades de parámetros aprendidos por las RNPs mediante un diagrama de cajas. En aquel se observa que las RDs presentan mayor cantidad de parámetros que las RCs. Se puede apreciar la existencia de 2 valores atípicos en las RCs correspondientes a modelos convolucionales con una cantidad de parámetros mayor al que posee el 82 % de las RCs restantes. Estos valores atípicos corresponden a modelos convolucionales con una o dos capas completamente conectadas además de las capas convolucionales y la densa de salida. La distribución de parámetros en las RDs y RCs son asimétricas respecto a las medianas.

Las diferencias observadas en las figuras 5.7 y 5.8 entre ambos tipos de modelo se deben a las configuraciones de sus respectivas arquitecturas. Precisamente, la arquitectura convolucional admite conectividad local, es decir, cada unidad está conectada solo a una pequeña región de la capa anterior. Además, en esta estructura existe la compartición

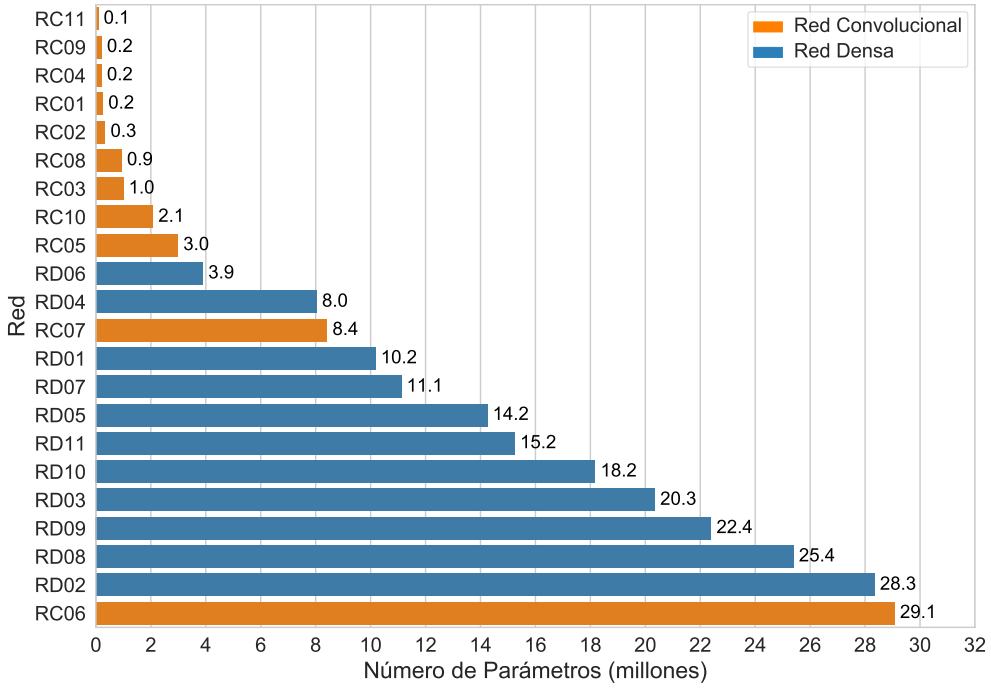


FIGURA 5.7: Diagrama de barras de las cantidades de parámetros, en millones, de las redes profundas densas (RDs) y convolucionales (RCs).

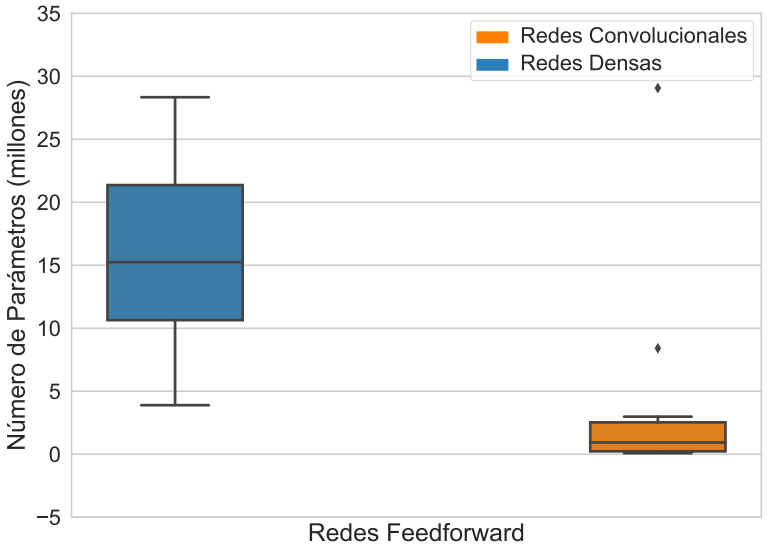


FIGURA 5.8: Diagrama de cajas de las cantidades de parámetros, en millones, de las redes profundas densas (RDs) y convolucionales (RCs).

de parámetros entre unidades de una misma capa. En cambio, la arquitectura densa presenta conectividad total entre unidades de una capa a la siguiente, y presenta un conjunto diferente de parámetros para cada neurona de una capa. Aquellas características de la arquitectura convolucional permiten la reducción del número de parámetros con respecto a las arquitecturas densas.

Por otro lado, la Figura 5.9 muestra un gráfico de barras del número de unidades ocultas

en las RDs y RCs. En el mismo se observa que las RDs tienen entre 3.4 mil y 9 mil neuronas, mientras que las RCs entre 41.96 mil y 320.13 mil unidades. Las RCs aprendidas tienen un mayor número de neuronas que las RDs, mínimamente la diferencia es de 32.96 mil unidades.

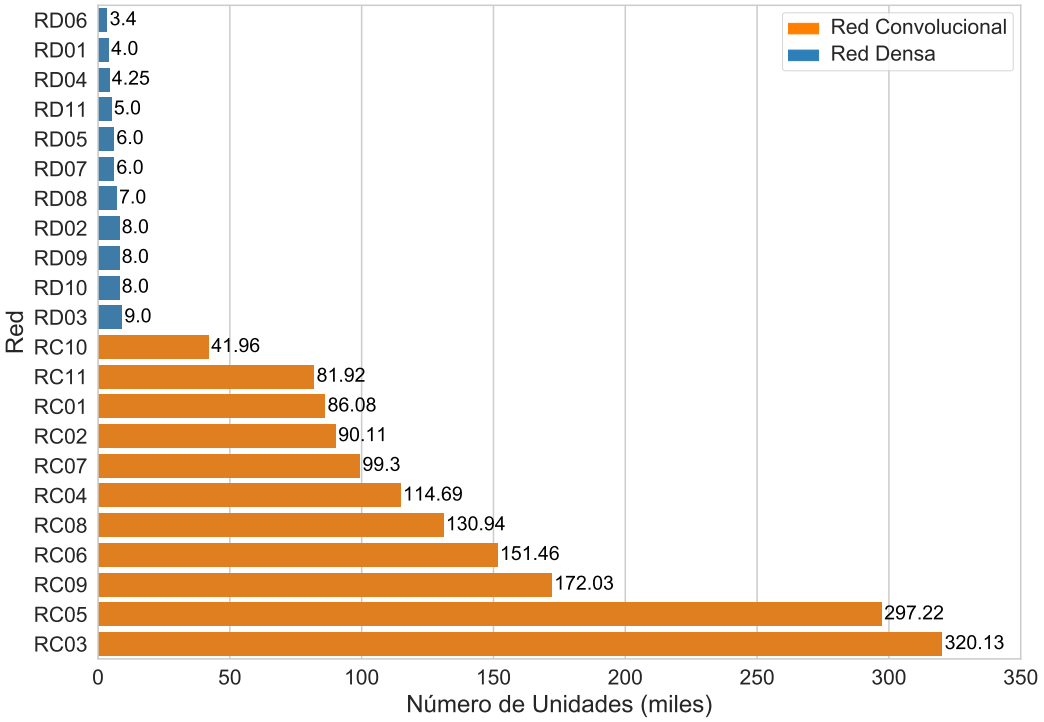


FIGURA 5.9: Número de unidades ocultas, en las redes profundas densas (RDs) y convolucionales (RCs).

Por otro lado, la Figura 5.10 indica el número de unidades ocultas (en escala logarítmica) de las RNPs mediante un diagrama de cajas. En términos generales, se observa que las RCs presentan mayor cantidad de neuronas que las RDs aprendidas. Se puede apreciar la existencia de 2 valores atípicos en las RCs correspondientes a modelos convolucionales con una cantidad de unidades ocultas mayor al que posee el 82 % de las RCs restantes. Estos valores atípicos corresponden a modelos convolucionales con mayor número de capas convolucionales que el resto. La distribución de unidades ocultas en los modelos densos es bastante simétrica respecto a la mediana, mientras que la distribución en los convolucionales es asimetría respecto al mismo punto.

Por último, la Figura 5.11 muestra un diagrama de cajas del cociente entre el número de unidades ocultas y la cantidad de parámetros de cada RNP. El gráfico condensa las características estructurales de las RNPs determinadas por sus arquitecturas. Se observa que las RDs aprendidas se destacan por presentar una cantidad de parámetros mucho mayor que la cantidad de unidades ocultas. En cambio, en las RCs predomina el número de unidades ocultas frente a la cantidad de parámetros.

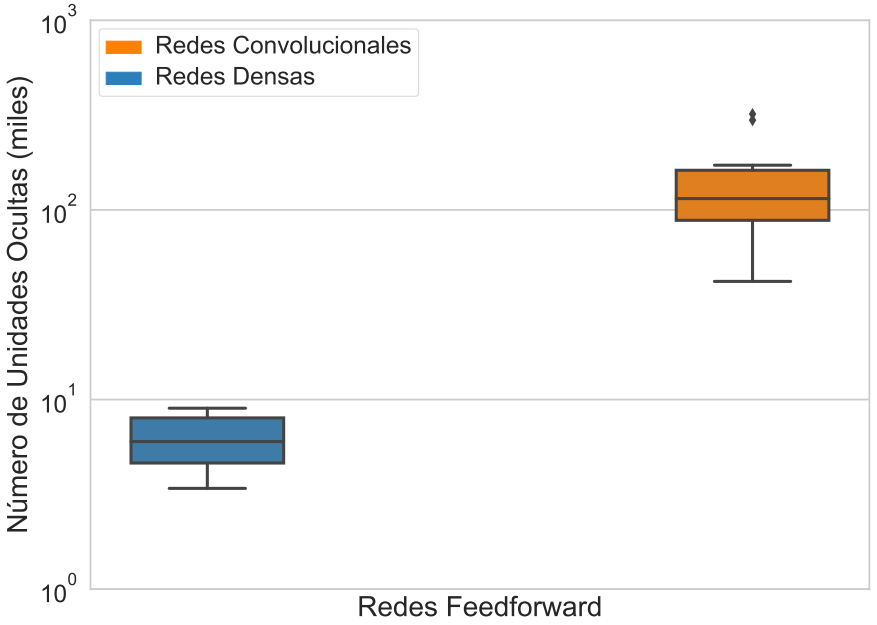


FIGURA 5.10: Diagrama de cajas de la cantidad de unidades ocultas en las redes profundas densas (RDs) y convolucionales (RCs). El eje y se encuentra en escala logarítmica.

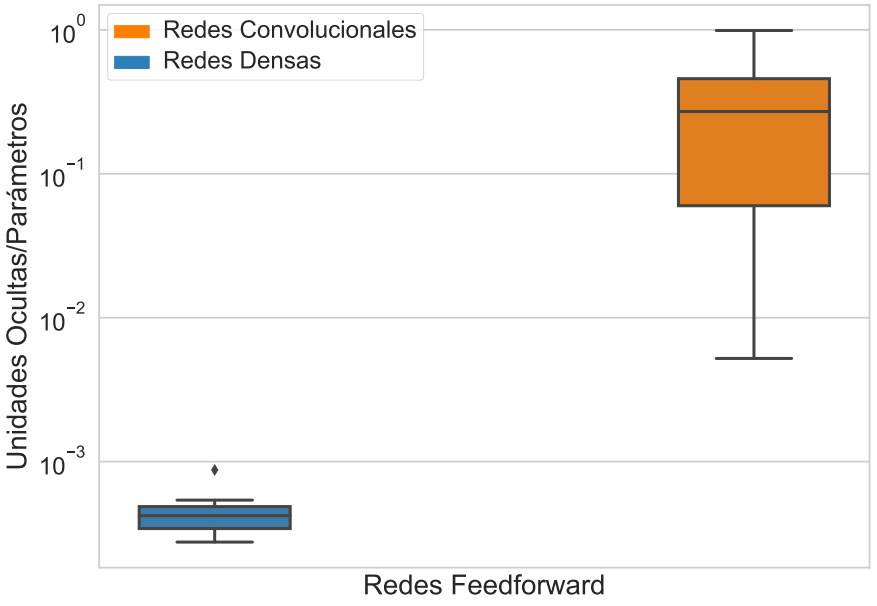


FIGURA 5.11: Diagrama de cajas del cociente entre el número de unidades ocultas y la cantidad de parámetros en las redes profundas densas (RDs) y convolucionales (RCs). El eje y se encuentra en escala logarítmica.

5.4. Correlaciones entre Características y Desempeño

En esta sección se pretende analizar la correlación entre la eficacia y eficiencia de las RNPs. Asimismo, se analizan como impactan diferentes características de las distintas

arquitecturas (cantidad de parámetros y número de unidades ocultas) en exactitud y tiempo de aprendizaje de las redes.

La Figura 5.12 muestra un gráfico de dispersión de la exactitud de clasificación en función del tiempo de aprendizaje para las RDs y RCs.

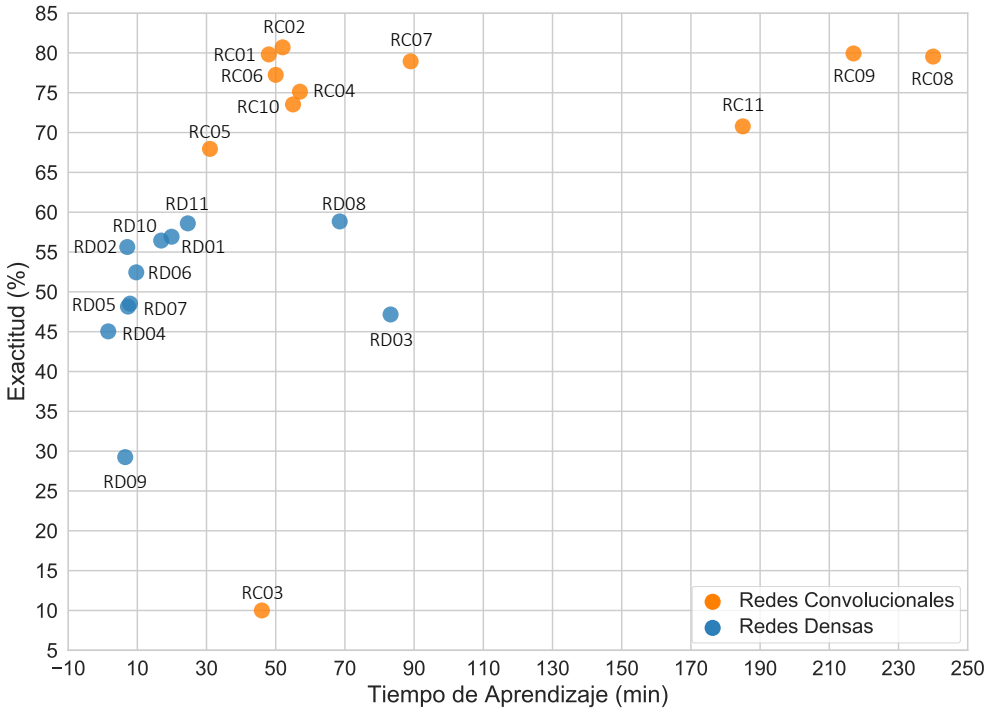


FIGURA 5.12: Correlación entre la exactitud de clasificación (en %) y el tiempo de aprendizaje (en min) en las redes densas (RDs) y convolucionales (RCs).

En la Figura 5.12 se observa que no hay correlación fuerte entre eficacia y eficiencia: hay redes con similar eficacia que presentan una notable diferencia en eficiencia y viceversa. Esto ocurre de manera más marcada en las RCs que en las RDs. Ejemplo de ello en las RCs son las redes RC02 (80.70 %) y RC01 (79.81 %) con las redes RC09 (79.93 %) y RC08 (79.54 %) que presentan una eficiencia en promedio de 50 min y 229 min respectivamente, haciendo una discrepancia de 179 min. En cambio, en las RDs se tiene la red RD01 (56.44 %) con la red RD08 (55.63 %) que presentan una eficiencia de 16.9 min y 68.5 min respectivamente, haciendo una diferencia de 25.8 min. Asimismo, hay redes con similar eficiencia que presentan una marcada diferencia en eficacia. Por último, se observa que hay un grupo de RCs que son las redes aprendidas con mayores eficacia y eficiencia, entre ellas, la red RC02.

La Figura 5.13 muestra un gráfico de dispersión de la exactitud de clasificación y la cantidad de parámetros en las RNPs. Se observa una correlación pobre entre aquellas cantidades. 9 de las 11 RCs tienen menos parámetros que las RDs, de estas 8 se desempeñaron mejor que las RDs. Asimismo, 2 de 11 RCs tenían una cantidad de parámetros

comparable a las RDs e inclusive se desempeñaron mejor que cualquier RD. Cabe mencionar que entre redes de una misma clase de modelo, ya sea entre redes RDs o entre redes RCs, tampoco hubo correlación fuerte entre exactitud y número de parámetros, pues hubieron redes con distinta exactitud y similar número de parámetros y viceversa. Ejemplo de ello son las redes RC04 (75.13 %) con RC09 (79.93 %) o RC01 (79.81 %), o bien, RC06 (77.24 %) con RD09 (29.25 %).

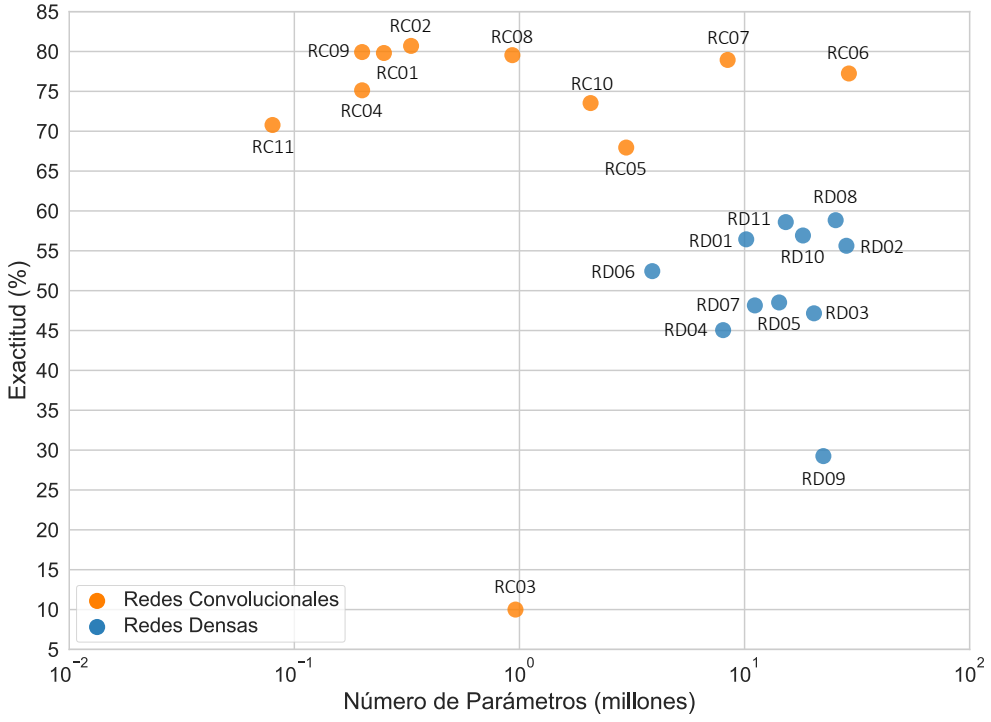


FIGURA 5.13: Correlación entre la exactitud de clasificación (en %) y la cantidad de parámetros (en millones) en las redes densas (RDs) y convolucionales (RCs). El eje x está en escala logarítmica.

La Figura 5.14 muestra un gráfica de dispersión de la exactitud de clasificación y el número de unidades ocultas en las RNPs. Los resultados sugieren que no hay una correlación clara entre exactitud de clasificación y número de unidades ocultas dentro de cada tipo de red. Las RDs tienen entre 3.4 mil y 9 mil unidades, mientras que las RCs entre 81.9 mil y 320.1 mil neuronas. Las 11 RCs aprendidas presentaban mayor número de unidades ocultas que las RDs y de las cuales 10 tuvieron mejor desempeño que las RDs. Asimismo, se observan casos en los cuales modelos con la misma cantidad de unidades ocultas se desempeñan diferente, como es el caso del las redes RD09 (29.25 %) con RD10 (56.92 %) y RD02 (55.63 %). Así como modelos con distinta cantidades de unidades tienen casi misma exactitud, como las redes RC01 (79.81 %) y RC09 (79.93 %).

La Figura 5.15 muestra la correlación entre el tiempo de aprendizaje y la cantidad de parámetros de las RNPs. Estos sugieren que no hay una relación fuerte entre aquellas cantidades dentro de cada tipo de modelo. Más aun, 9 de 11 RCs tenían menor cantidad

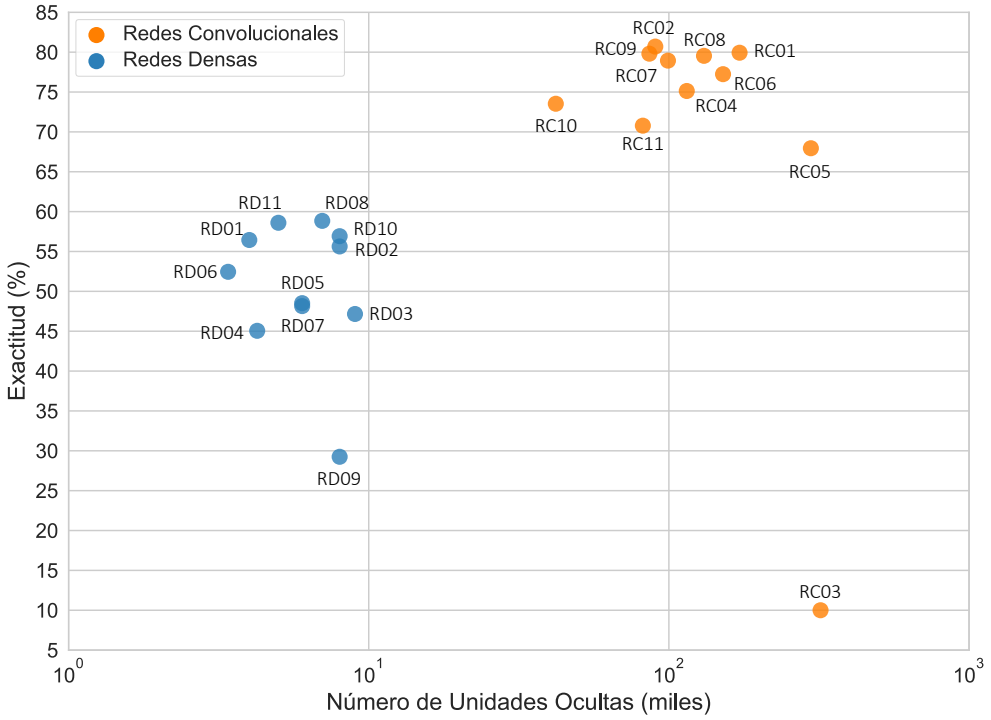


FIGURA 5.14: Correlación entre la exactitud de clasificación (en %) y el número de unidades ocultas (en millones) en las redes densas (RDs) y convolucionales (RCs). El eje x está en escala logarítmica.

de parámetros que las RDs de las cuales 3 fueron las que más tiempo les tomó aprender y 6 demoraron más de 9 RDs. Por último, 2 de 11 RCs tenían una cantidad de parámetros comparable al de las redes RDs pero se demoraron más que 9 de las 11 RDs.

La Figura 5.16 muestra la correlación entre el tiempo de aprendizaje y el número de unidades ocultas de las RNPs. Los resultados indican que no hay una relación clara entre aquellas variables dentro de cada tipo de modelo. Las 11 RCs aprendidas tenían mayor número de unidades ocultas que las RDs, de las cuales 4 demoraron aprender más que las 11 RDs y 7 más que 9 de las RDs.

5.5. Patrones Aprendidos por Convoluciones

En esta sección se identifican y caracterizan los patrones capturados por las convoluciones en las RCs. El estudio se hace sobre las RCs y no sobre las RDs debido a que estas últimas presentan todas las unidades de una capa totalmente conectadas a todas las unidades de la capa siguiente. Por lo tanto, no hay un arreglo espacial de unidades que permita organizarlas en una imagen para ser visualizadas.

Para estudiar los patrones aprendidos por las convoluciones en las RCs se tomó como base la red RC02 correspondiente al modelo convolucional de mejor desempeño en términos

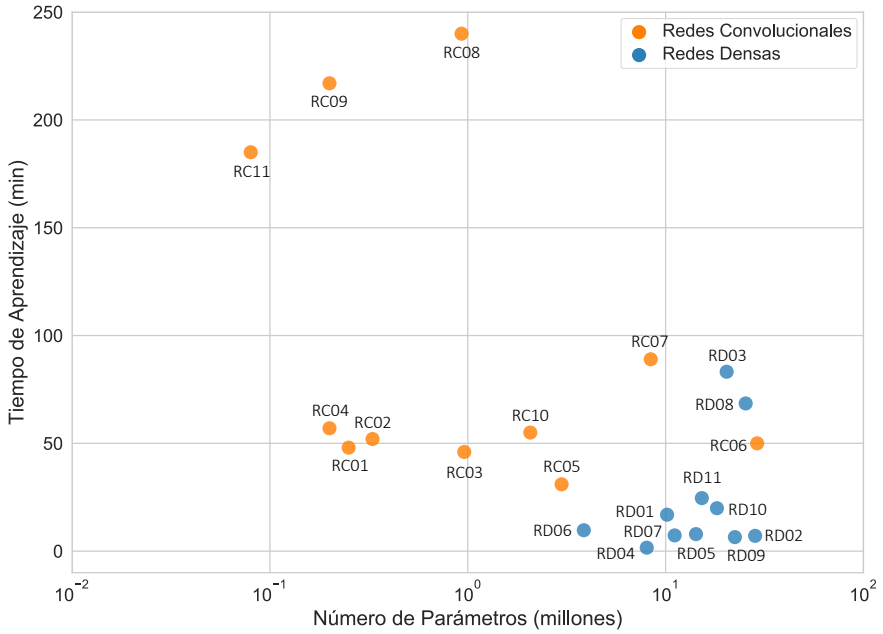


FIGURA 5.15: Correlación entre el tiempo de aprendizaje (en min) y la cantidad de parámetros (en millones) en las redes densas (RDs) y convolucionales (RCs). El eje x está en escala logarítmica.

de exactitud. A modo de ejemplo, se llevó a cabo el análisis sobre un conjunto reducido de imágenes de entrada. Los resultados se muestran a través de visualizaciones de las distintas capas de la red, empezando desde la entrada hasta la salida de la misma. Estas visualizaciones fueron caracterizadas necesariamente en forma subjetiva debido a la naturaleza del estudio.

La Figura 5.17 muestra el esquema de la red RC02 en la interfaz de visualización de Quiver. Asimismo, presenta una imagen de CIFAR-10, tomada al azar del conjunto de prueba, que fue utilizada como entrada de la red para visualizar los patrones aprendidos. Cabe mencionar que el modelo RC02, descripto en el Capítulo 4, cuenta con tres capas convolucionales con 64, 64 y 128 filtros respectivamente, todos de tamaño 5×5 y con funciones de activación ReLU.

En primer lugar, la Figura 5.18 muestra las activaciones en la primera capa convolucional del modelo RC02. La misma está formada por 64 filtros, por lo tanto, hay 64 imágenes, de tamaño 32×32 píxeles, producto de la convolución de cada uno con la imagen de entrada. A grandes rasgos, los distintos filtros parecen capturar contornos, bordes y contrastes.

Asimismo, la Figura 5.19 muestra diez imágenes de CIFAR-10 seleccionadas al azar, cada una correspondiente a cada categoría y las respectivas imágenes filtradas por tres filtros distintos de la primera capa convolucional. De esta manera, se observa el efecto

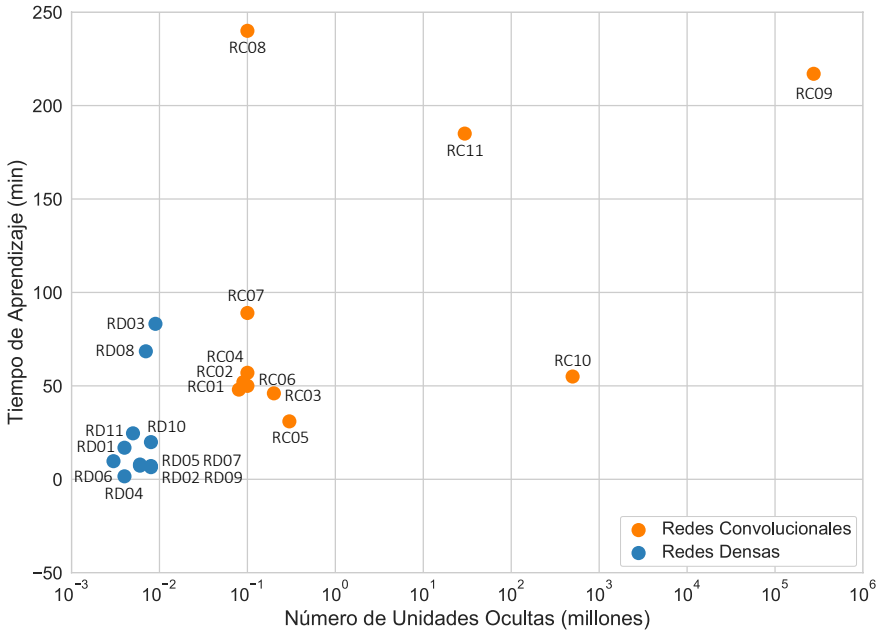


FIGURA 5.16: Correlación entre el tiempo de aprendizaje (en min) y el número de unidades ocultas (en millones) en las redes densas (RDs) y convolucionales (RCs). El eje x está en escala logarítmica.

de tres filtros sobre diez imágenes de entrada diferentes y permite identificar y describir subjetivamente las características activadas por cada uno.

En segundo lugar, la Figura 5.20 muestra las activaciones en la segunda capa convolucional del modelo RC02. La misma presenta los patrones capturados por los 64 filtros de esta capa. Básicamente, se empiezan a volver menos inteligibles las activaciones de las unidades. Estas activaciones dependen de las activaciones de la capa anterior y de los pesos asociados con lo cual en esta segunda capa convolucional puede decirse que está modelando patrones más abstractos que los de la capa anterior.

En tercer lugar, la Figura 5.21 muestra las activaciones en la tercera capa convolucional del modelo RC02. La misma presenta los patrones capturados por los 128 filtros de esta capa. Las activaciones son aún más abstractas que en la segunda capa convolucional. Hay muchas zonas de baja señal (regiones negras) y las activaciones se agrupan de forma más localizada. Por lo tanto, no es posible determinar a qué corresponde exactamente cada una de ellas.

Por último, la Figura 5.22 muestra las activaciones en la capa de salida (capa softmax) del modelo RC02. La salida de la capa softmax puede visualizarse como un vector de 10 píxeles en donde la intensidad de cada uno representa la probabilidad de que la imagen de entrada pertenezca a la clase correspondiente. Dicho vector ya no está representando patrones concretos de la imágenes sino que está representando directamente un concepto abstracto, es decir, el objeto identificado en la imagen de entrada. Puede decirse que,

desde la capa de entrada a la de salida, las capas de la red, las capas de la red son capaces de capturar patrones cada vez más abstractos que se van componiendo de los patrones detectados de menor abstracción en las capas que las preceden. Para la imagen del camión, la predicción del modelo RC02 fue que es un camión con una probabilidad de 1.

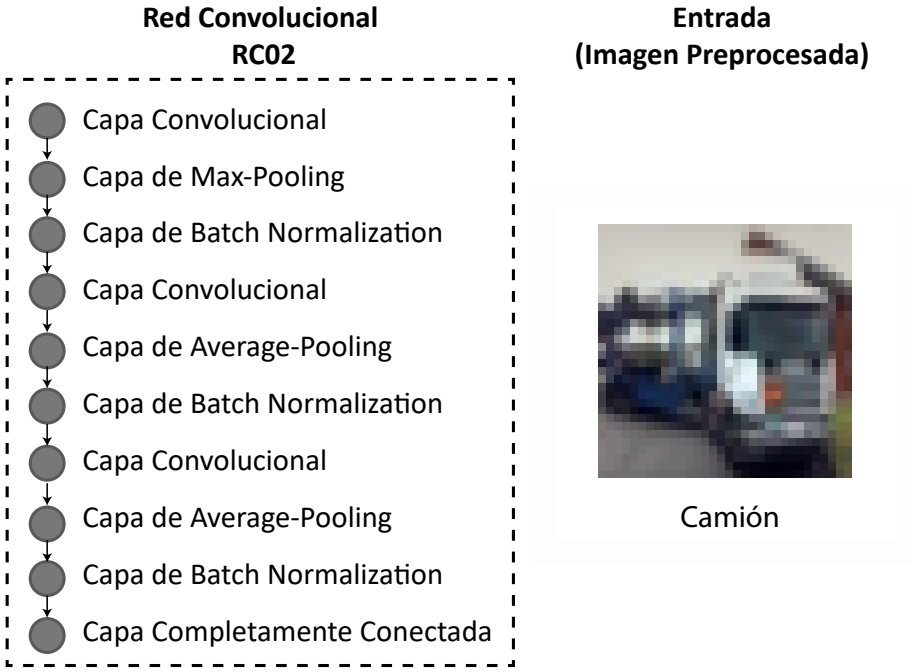


FIGURA 5.17: Esquema del modelo convolucional RC02 en la interfaz de Quiver (izquierda) e imagen seleccionada para alimentar la entrada de la red y visualizar los patrones aprendidos por las convoluciones (derecha).

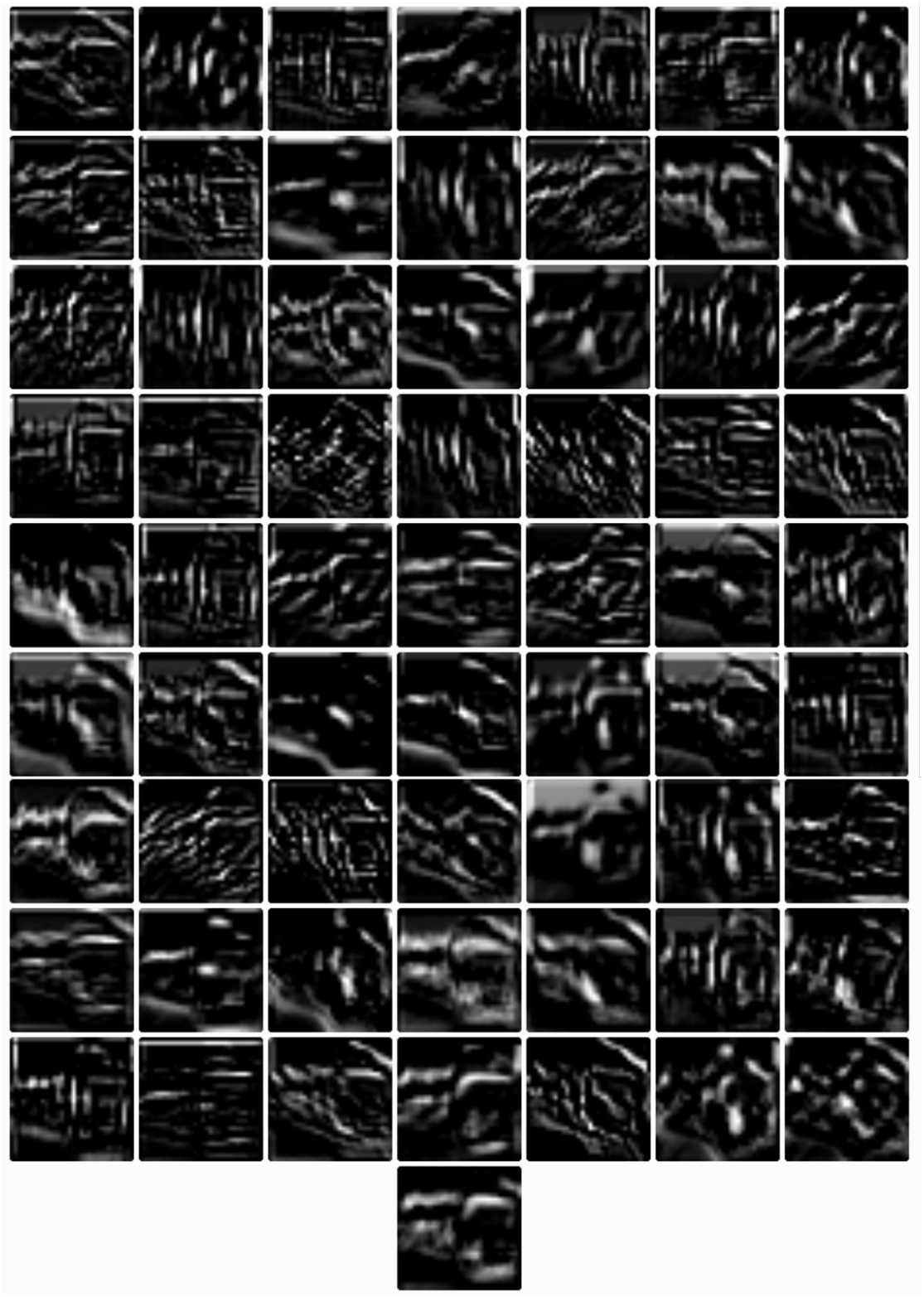


FIGURA 5.18: Activaciones de la primera capa convolucional de la red RC02. La capa genera 64 imágenes (32×32 píxeles) correspondientes a los 64 filtros que constituyen la misma.

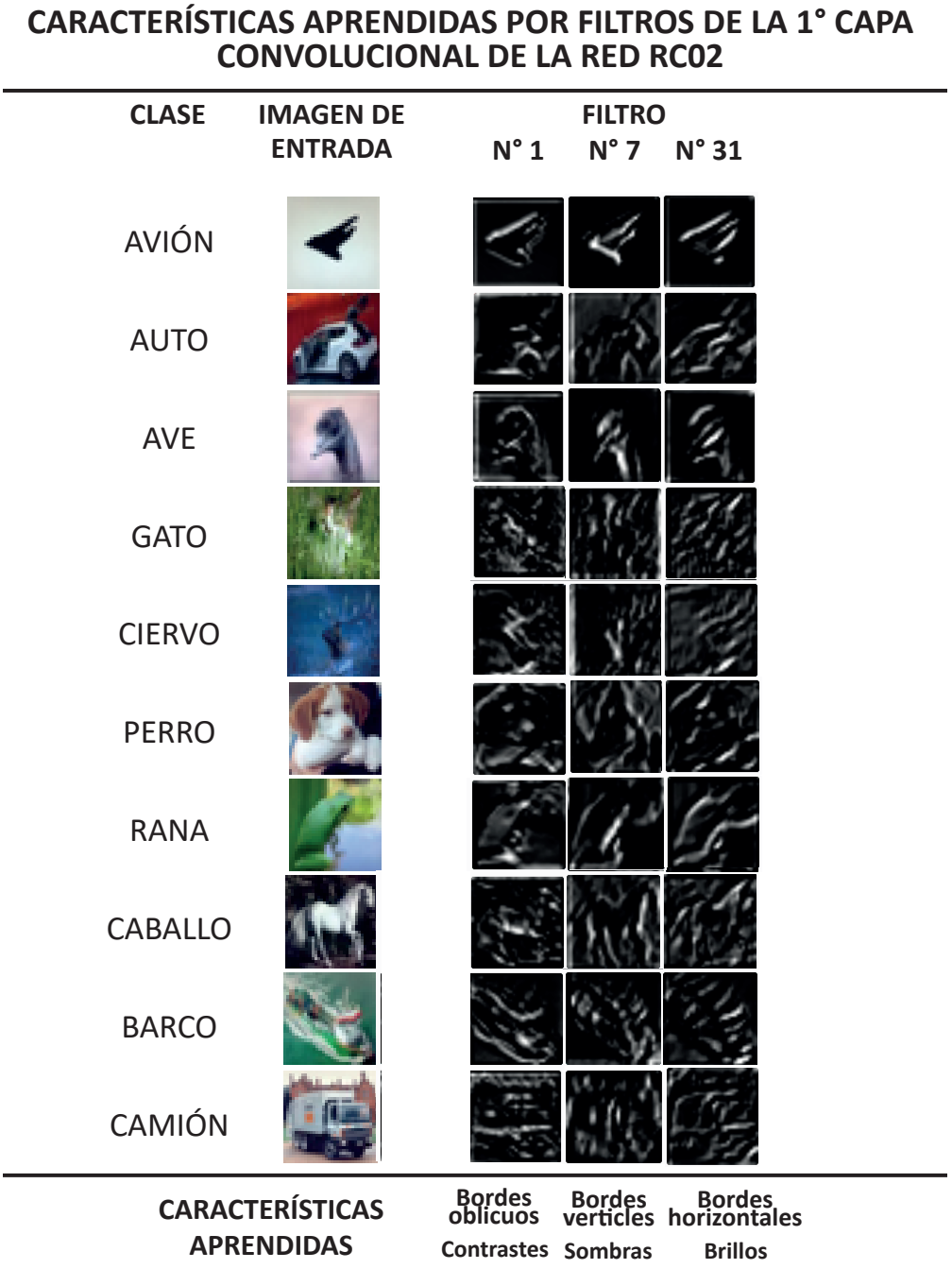


FIGURA 5.19: Patrones aprendidos por tres filtros distintos de la primera capa convolucional del modelo RC02. Se muestran las imágenes filtradas para una imagen de cada categoría de CIFAR-10.

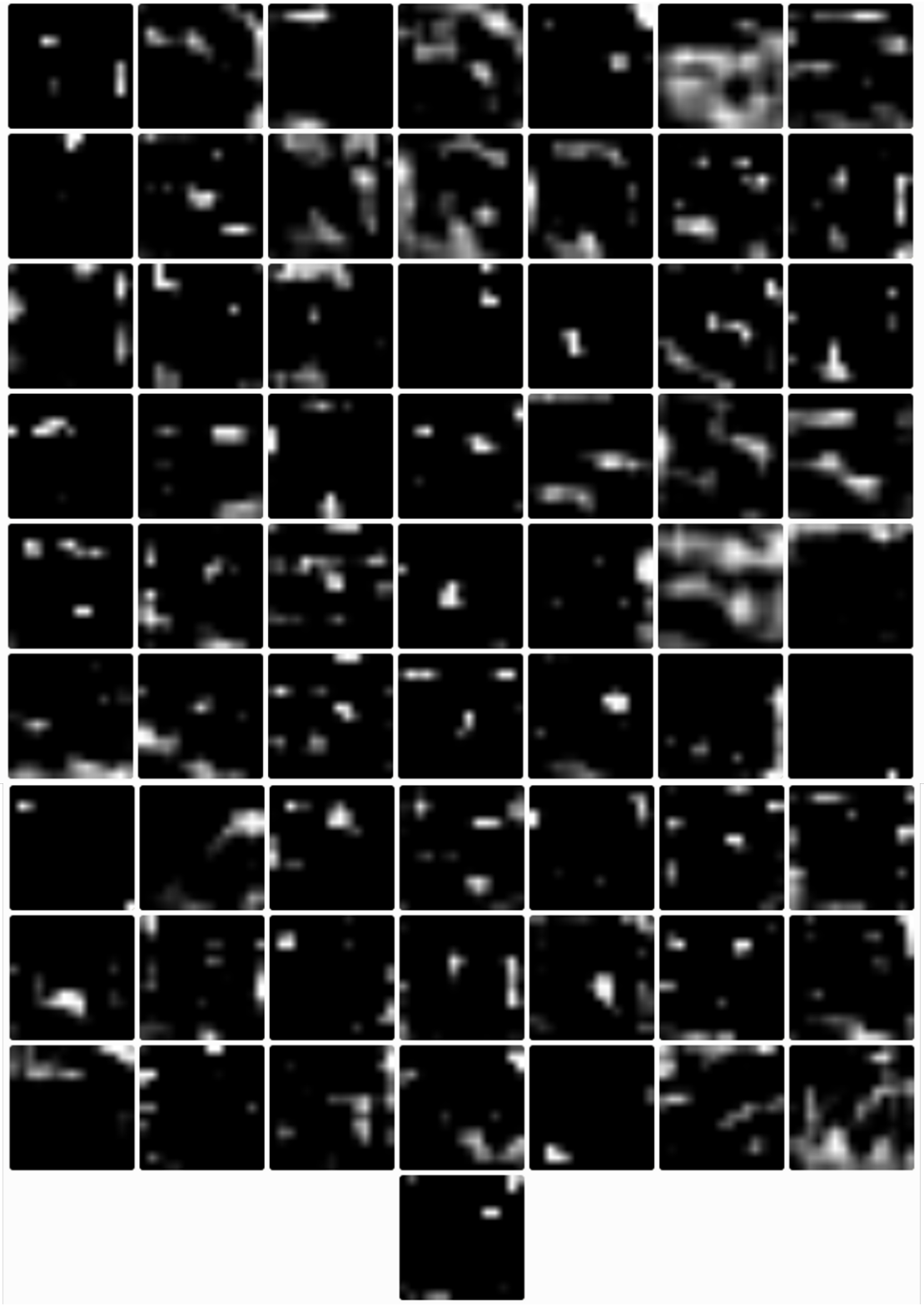


FIGURA 5.20: Activaciones de la segunda capa convolucional de la red RC02. La capa genera 64 imágenes (16×16 píxeles) correspondientes a los 64 filtros que constituyen la misma. Las activaciones son patrones más abstractos y están presentes ciertas combinaciones de patrones detectados en la capa anterior.

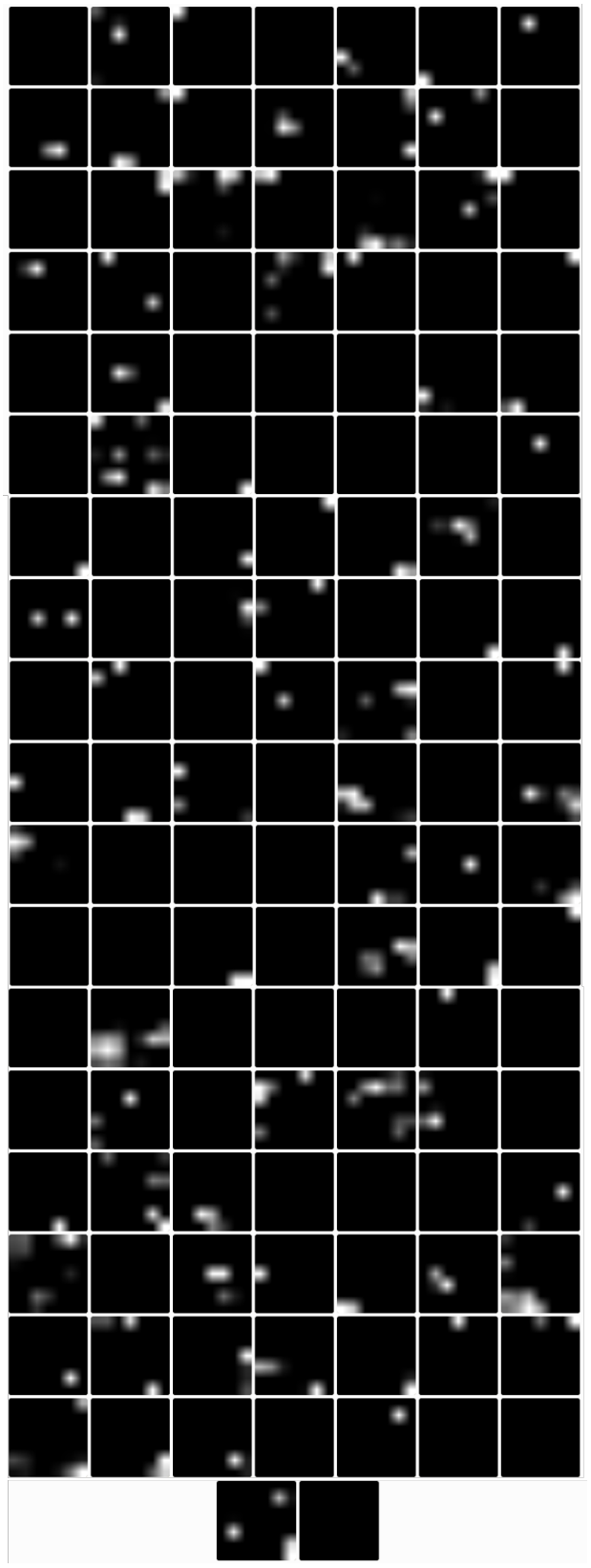


FIGURA 5.21: Activaciones de la tercera capa convolucional de la red RC02. La capa genera 128 imágenes (8×8 píxeles) correspondientes a los 128 filtros que constituyen la misma. Las activaciones son aún más abstractas que en capas anteriores y están agrupadas de forma más localizada. No es posible determinar a qué corresponde exactamente cada una de ellas.

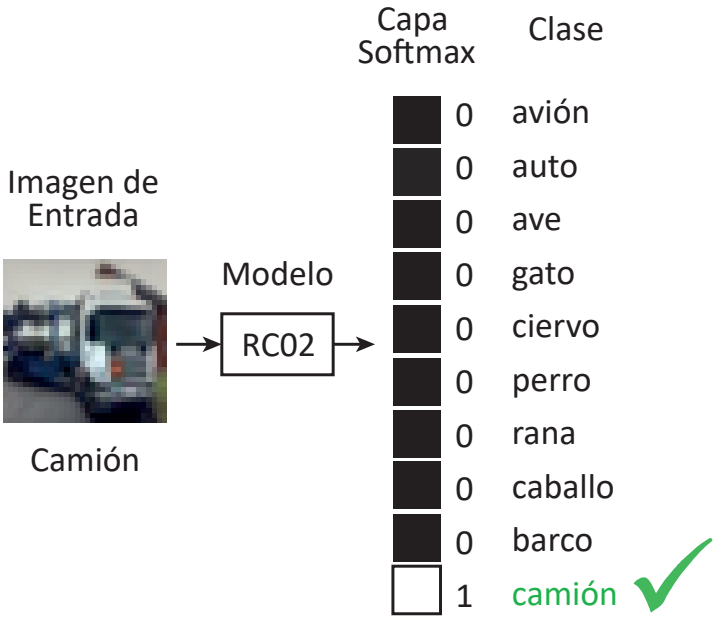


FIGURA 5.22: Activaciones de la capa de salida (capa softmax) de la red RC02. La salida de esta capa es simplemente un vector de 10 componentes, cada una correspondiente a una clase de CIFAR-10, donde el valor de cada componente representa la probabilidad de pertenecer a esa clase.

Capítulo 6

Conclusiones

Las RNPs han logrado un desempeño muy notable en tareas de IA pero es bien sabido que el entrenamiento de estos modelos viene acompañado de un alto costo y complejidad computacional. Existe una amplia variedad de RNPs dependiendo de su aplicación: cada una tiene una arquitectura diferente en cuanto a número, tipo y forma de capas, y conexiones entre capas. Por lo tanto, muchos modelos de RNPs han sido desarrollados en el último tiempo, e incluso mejorados, para lograr mayor eficacia y eficiencia en las tareas para las cuales fueron diseñados. Para la clasificación de imágenes multiclase hay dos modelos de RNPs populares: RDs y RCs. En consecuencia, al implementar una red para una determinada tarea es necesario entender las variaciones en los modelos de redes y posibles tendencias en los resultados para incorporar una correcta flexibilidad a cualquier dispositivo de RNPs eficiente y eficaz.

Esta tesis busca medir qué tan eficaces y eficientes son distintas configuraciones de RDs al clasificar objetos en imágenes frente a las RCs. Asimismo, busca identificar la correlación entre la eficacia y eficiencia de las redes y características estructurales de las mismas, tales como, el número de unidades ocultas y el número de parámetros a aprender. Por último, busca identificar y caracterizar los patrones capturados por las convoluciones en las RCs.

La tesis se desarrolló en tres grandes bloques: uno introductorio, uno de resultados y uno de conclusiones. En el primero se identificó la temática y se informó acerca del estado del conocimiento en el tema del aprendizaje de redes neuronales profundas. Asimismo, con la identificación de la situación problemática y con la formulación del problema de investigación se buscaron diseñar RNPs y metodologías actuales para su entrenamiento. Se cerró este bloque con las preguntas, objetivos e hipótesis, y se realizó el diseño metodológico. En el segundo bloque de esta tesis se consignaron los resultados de la investigación. Se dividió en cinco partes, de acuerdo a los distintos aspectos a estudiar

y que permitieran demostrar las hipótesis. El último bloque corresponde a este capítulo de conclusiones, el cual sintetiza la información contenida en esta investigación.

Para cumplir con los objetivos y corroborar las hipótesis fue necesario: aprender RDs y RCs que clasifiquen objetos en imágenes, determinar las exactitudes de clasificación y tiempos de aprendizaje, y comparar los desempeños de los modelos densos frente a los alcanzados por los modelos convolucionales.

La metodología diseñada para aprender RNPs involucró lo siguiente: seleccionar un conjunto de datos (CIFAR-10), construir RNPs densas y convolucionales adecuadas para clasificar imágenes, y elegir las técnicas óptimas para el proceso de aprendizaje.

Como conjunto de datos se utilizó CIFAR-10, un conjunto de imágenes a color y de 32×32 píxeles. Aparece un único objeto por imagen que pertenece a una de diez clases posibles. Con respecto a las arquitecturas de las RNPs, se construyeron 22 redes: 11 densas y 11 convolucionales. De las cuales, 5 fueron basadas en redes de estudios publicados y el resto fueron variantes similares que involucraban distinto número y tipo de capas y de unidades ocultas. Por otro lado, dentro de las técnicas usadas para el aprendizaje estas incluyeron CGN como preprocesamiento de datos, uso de *dropout* y *batch normalization* como técnicas de regularización, uso del algoritmo Adam como algoritmo gradiente descendente, optimización de hiperparámetros con validación cruzada y búsqueda aleatoria del número de épocas y el tamaño de grupos.

Los resultados de esta investigación fueron interesantes y se resumen a continuación:

- La RNP más eficaz fue la red convolucional RC02 logrando una exactitud de clasificación de 80.70 %.
- La red densa más eficaz fue RF08 con exactitud de clasificación de 58.84 %.
- De las 22 RNPs aprendidas ninguna supera en desempeño al promedio de una persona (94 %), pues como mínimo las redes fueron un 13.3 % menos eficaces.
- 10 de las 11 RCs fueron mínimamente un 9.11 % más eficaces que las RDs aprendidas. Esto las hace aquellas con mayor capacidad de capturar patrones abstractos y generalizar mejor de los datos.
- 5 de las 22 RNPs cuyas arquitecturas fueron basadas en redes ya estudiadas obtuvieron desempeños tanto similares como diferentes a los reportados. Las exactitudes discreparon en: RC02 (-0.06 %), RC03 (-80.26 %), RD01 (+14.24 %), RD02 (+1.71 %) y RD03 (-9.68 %). Los resultados de RD01 y RD02 se consideran positivos ya que el aprendizaje implementado incrementó su desempeño reportado en otros estudios. El resultado de RC02 se considera satisfactorio. En cambio,

el desempeño logrado con RC03 y RD03 fue degradado. Esto se debe a que el aprendizaje implementado no fue óptimo para estas redes.

- La RNP más eficiente fue la RD RD04 demandándole aprender a clasificar imágenes 1.6 min.
- La RC más eficiente fue RC05 con 31 min.
- 9 de las 11 RDs fueron mínimamente 6.4 min más eficientes que las RCs aprendidas.
- El 82 % de las RCs tienen menos parámetros que aprender que las RDs.
- Todas las RCs aprendidas tienen un mayor número de unidades ocultas que las RDs.
- La corrrelación entre exactitud de clasificación y cantidad de parámetros aprendidos no es clara dentro de cada tipo de modelo, pues hubieron redes con distinta exactitud y similar número de parámetros y viceversa en cada caso. A grandes rasgos, considerando el conjunto de RNPs aprendidas, se observa que hay una tendencia en que a mayor cantidad de parámetros la eficacia es menor.
- La correlación entre exactitud de clasificación y número de unidades ocultas no fue clara dentro de cada tipo de modelo, pues hubieron redes con distinta exactitud y similar número de unidades y viceversa. En términos generales, considerando el conjunto de RNPs aprendidas, se observa que hay una tendencia en que a mayor número de unidades ocultas la eficacia es mayor.
- Se observa una correlación poco clara entre tiempo de aprendizaje y cantidad de parámetros aprendidos dentro de cada tipo de red. Sin embargo, en general existe una tendencia en que a mayor cantidad de parámetros menor es la eficiencia.
- La correlación entre el tiempo de aprendizaje y número de unidades ocultas es poco clara dentro de cada tipo de modelo. Sin embargo, a grandes rasgos existe una tendencia en que a mayor cantidad de unidades ocultas menor es la eficiencia.
- Los patrones aprendidos por las convoluciones en RCs muestran que son cada vez menos interpretables e identificables al pasar de una capa convolucional a otra. De forma subjetiva se puede afirmar que los filtros de la primera capa capturaron distintas orientaciones de bordes, contornos y contrastes.

Estos resultados permiten entender el impacto de las arquitecturas de los modelos de RNPs en el desempeño de tareas o actividades específicas. Esto es de gran importancia para lograr incorporar una correcta y adecuada flexibilidad en cualquier dispositivo de redes neuronales profundas eficiente y eficaz.

Apéndice A

Versión para Python de CIFAR-10

En este apéndice se especifica con detalle el formato de los datos de CIFAR-10 en su versión para Python.

El conjunto de datos está dividido en cinco grupos para entrenamiento y un grupo para prueba, cada uno con 10,000 imágenes. El grupo para prueba contiene exactamente 1,000 ejemplos de cada clase seleccionados al azar. Los grupos para entrenamiento contienen las imágenes restantes en orden aleatorio. Algunos grupos para entrenamiento pueden contener más imágenes de una clase que de otra pero, en conjunto, contienen exactamente 5,000 ejemplos de cada una (Krizhevsky, 2009).

El archivo disponible de forma libre y gratuita en `https://www.cs.toronto.edu/~kriz/cifar.html` contiene los archivos *data_batch_1*, *data_batch_2*, ..., *data_batch_5*, así como también, *test_batch*. Cada uno de estos archivos es un objeto pickled de Python producido por el paquete cPickle. De esta manera, mediante una rutina pueden abrirse estos archivos y retornar un objeto diccionario de Python.

Cada uno de los archivos, correspondiente a un grupo de ejemplos, contiene un objeto diccionario con los siguientes elementos:

- ***data.*** Presenta los datos, un arreglo de la librería NumPy de $10,000 \times 3,072$ elementos de tipo *uint8s*. Cada fila de la matriz almacena una imagen a color de 32×32 píxeles. Las primeras 1,024 entradas contienen los valores de intensidades correspondientes al canal rojo, las 1,024 siguientes al verde y las 1,024 finales al azul. La imagen se almacena de modo que las primeras 32 entradas de la matriz son

los valores de canal rojo de la primera fila de la imagen, las siguientes 32 entradas son los valores del mismo canal de la segunda fila, y así sucesivamente.

- ***labels.*** Contiene las etiquetas, una lista de 10,000 números en el rango [0, 9]. El número en el índice i indica la etiqueta de la *i-ésima* imagen en los datos de la matriz.

Asimismo, hay otro archivo, llamado *batches.meta*. También contiene un objeto diccionario de Python y tiene el siguiente elemento:

- ***label_names.*** Presenta los nombres de las etiquetas, una lista de 10 elementos que da nombres significativos a las etiquetas numéricas en el conjunto de etiquetas descrito anteriormente. Por ejemplo: `label_names [0] == "airplane"`, etc.

Apéndice B

Hiperparámetros

En este apéndice se especifican los valores de hiperparámetros involucrados en el proceso de aprendizaje de cada RNP.

Los hiperparámetros son parámetros que controlan el comportamiento del proceso de aprendizaje en sí mismo. Como se ha comentado, hay diversos hiperparámetros que fijar, entre ellos, tamaño de grupos, número de épocas, valores de *dropout*, etc. Por lo tanto, debido a la complejidad y costo computacional que requiere una optimización en un espacio de alta dimensión, se optimizaron solo dos hiperparámetros (tamaño de grupos y número de épocas) y el resto se fijó según recomendaciones de estudios publicados.

La Tabla B.1 muestra los valores de tamaño de grupos y número de épocas de las RDs de este trabajo, mientras que la Tabla B.2 muestra los valores correspondientes para las RCs. Se optimizaron aquellas cantidades mediante búsqueda aleatoria y validación cruzada con valor típico de particiones igual a 10. Para mayor detalle sobre los hiperparámetros y los métodos empleados para su optimización consultar el Capítulo 4.

Además, en las RDs se usó *dropout* como método de regularización y en las RCs se empleó *batch normalization*. En particular, se implementó *dropout* con valor óptimo recomendado 0.5 en las capas ocultas completamente conectadas de las redes RD01, RD05, RD06, RD07, RD08, RD10, RD11 y en la capa localmente conectada de RC01 y RC05. Además, se usó *dropout* del mismo valor en las capas ocultas no lineales de RD09. Por otro lado, se aplicó *batch normalization* luego de la primera y segunda capa de *pooling* en RC01, y luego de cada capa del mismo tipo en RC02, RC04, RC06 y RC07.

Tabla B.1: Hiperparámetros optimizados en las redes densas: tamaño de grupos y número de épocas. Exploración del espacio de parámetros por búsqueda aleatoria y selección del mejor modelo mediante validación cruzada con 10 particiones.

Red	Tamaño de Grupos	Número de Épocas
RD01	500	300
RD02	5,000	100
RD03	10,000	1,000
RD04	500	30
RD05	500	100
RD06	100	100
RD07	500	100
RD08	5,000	1,000
RD09	8,000	100
RD10	5,000	300
RD11	5,000	500

Tabla B.2: Hiperparámetros optimizados en las redes convolucionales: tamaño de grupos y número de épocas. Exploración del espacio de parámetros por búsqueda aleatoria y selección del mejor modelo mediante validación cruzada con 10 particiones.

Red	Tamaño de Grupos	Número de Épocas
RC01	300	100
RC02	300	100
RC03	500	30
RC04	800	100
RC05	30	30
RC06	800	30
RC07	30	100
RC08	300	300
RC09	30	100
RC10	30	100
RC11	30	300

Apéndice C

Matrices de Confusión

En este apéndice se describe qué es una matriz de confusión y su utilidad en la tarea de clasificación multiclase. Asimismo se presentan las matrices correspondientes asociadas a cada una de las RNPs aprendidas.

Una matriz de confusión, o también denominada matriz de error, C es una herramienta de aprendizaje y clasificación estadística. En éstas el elemento $C_{i,j}$ es igual al número de observaciones conocidas que están en el grupo i pero que fueron predichas pertenecer al grupo j. Por lo tanto, aquellas matrices permiten la visualización del desempeño de un algoritmo.

En particular, en los algoritmos de aprendizaje supervisado se tiene acceso a las etiquetas de los datos de entrada en las etapas de entrenamiento y evaluación del algoritmo. En clasificación multiclase la entrada tiene que ser clasificada en una, y solo una, de k clases mutuamente excluyentes (c_1, ..., c_k). Éste caso representa el tipo de clasificación involucrada en el presente trabajo de investigación, en el cual RNPs aprenden a clasificar imágenes de objetos pertenecientes a una sola categoría de diez posibles.

Uno de los beneficios de las matrices de confusión es que facilitan evaluar si el modelo de aprendizaje está confundiendo clases o no. Esto sugiere que tan bueno es el modelo para generalizar a partir de los datos aprendidos.

Por lo tanto, una vez evaluadas las RNPs con los datos del conjunto para prueba, se utilizó en este trabajo `sklearn.metrics.confusion_matrix` de la librería scikit-learn. Esta función permitió calcular las matrices de confusión (normalizadas y no normalizadas) asociadas a cada red para evaluar la exactitud de clasificación. Cabe recordar que el conjunto de aprendizaje CIFAR-10 cuenta con 10,000 imágenes para prueba de las cuales hay 1,000 de cada clase. Las figuras C.1 y C.2 muestran las matrices de confusión

asociadas a las RDs. Mientras que las figuras C.3 y C.4 muestran las matrices correspondientes a las RCs. En todos los casos las matrices no se encuentran normalizadas por la cantidad de ejemplos que conforman las clases.

En particular, aquellas matrices son cuadradas de tamaño igual al número de clases o categorías, en este caso tienen tamaño 10×10. Las filas indican las clases verdaderas y las columnas las clases predichas. Por lo tanto, los elementos de la diagonal de la matriz representan el número de ejemplos de los cuales la etiqueta predicha es igual a la etiqueta verdadera, mientras que los elementos fuera de la diagonal son aquellos que han sido mal clasificados por el modelo. Los valores de la diagonal de la matriz de confusión con valores más altos, indican muchas predicciones correctas.

A modo de ejemplo, considere la primera matriz de la Figura C.1 asociada a la red RD01. La tercera fila corresponde a la clase *ave*. Si se recorren las columnas a lo largo de esa fila se puede saber cuántas de las 1,000 imágenes de aves fueron clasificadas en la categoría: *avión* (73), *auto* (17), *ave* (425), *gato* (101), *ciervo* (114), *perro* (75), *rana* (54), *caballo* (64), *barco* (16) y *camión* (21). Luego, se puede notar que las aves suelen ser mayormente confundidas con gatos y ciervos.

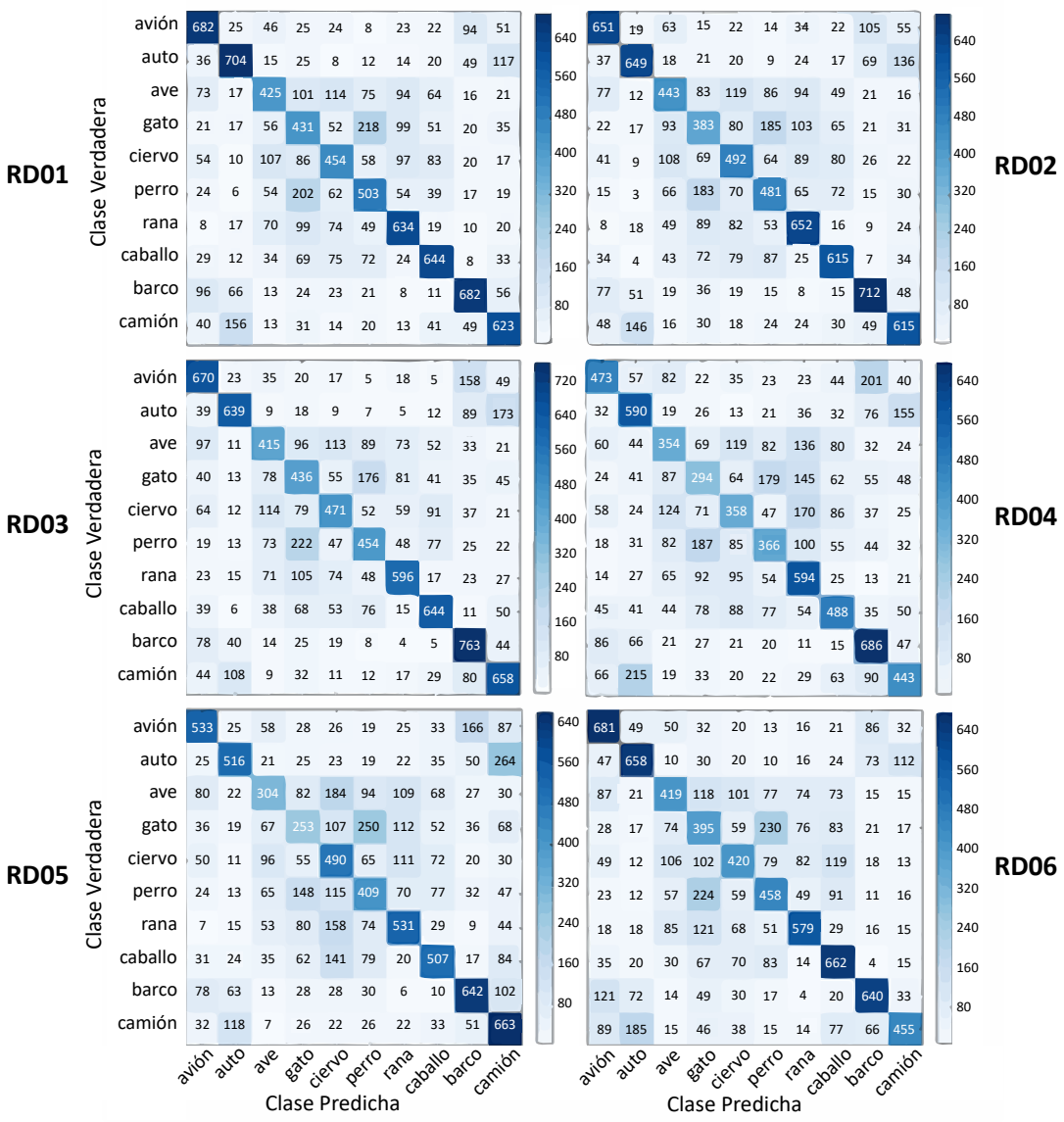


FIGURA C.1: Matrices de confusión sin normalizar asociadas a las redes densas RD01, RD02, RD03, RD04, RD05 y RD06.

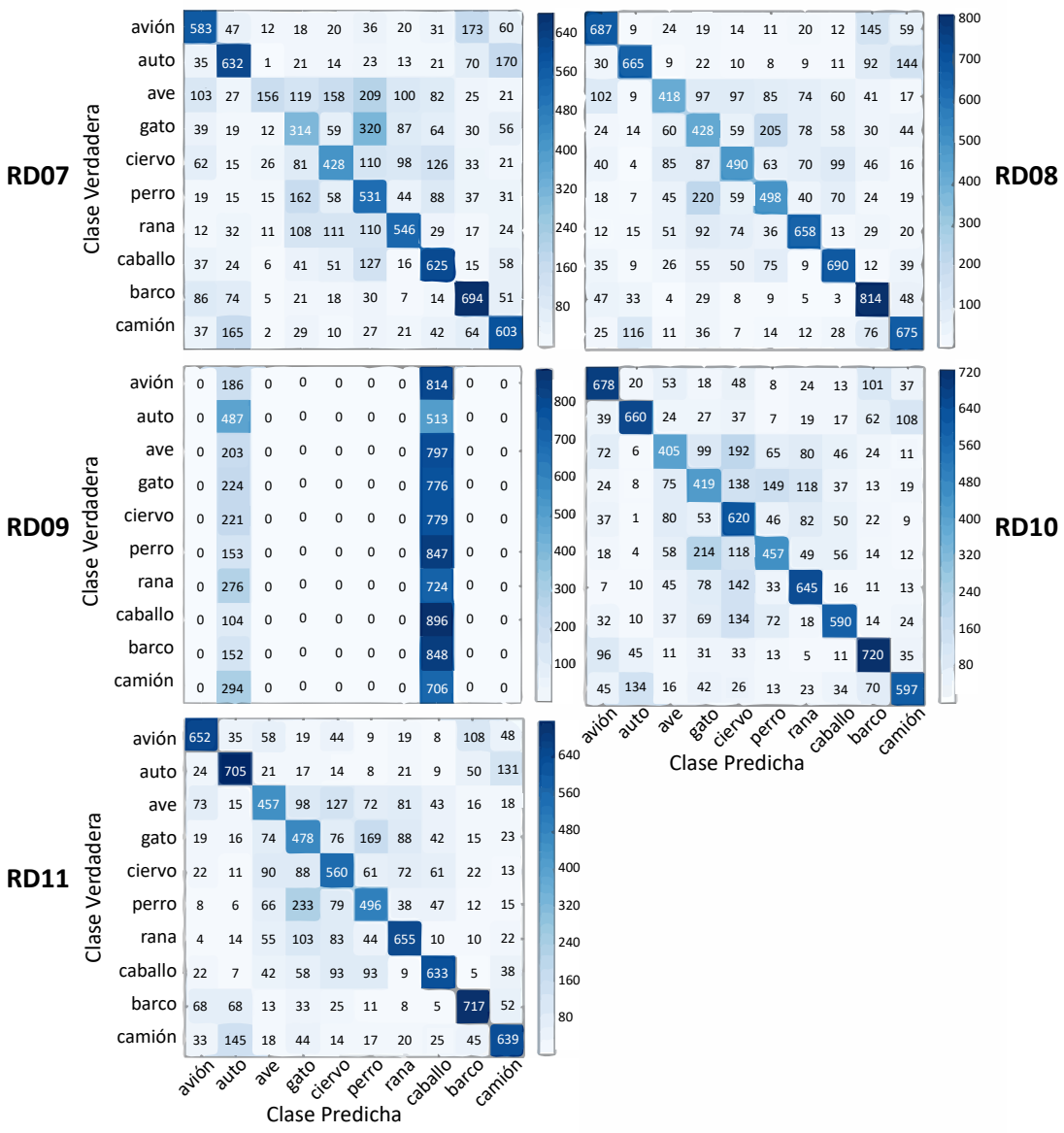


FIGURA C.2: Matrices de confusión sin normalizar asociadas a las redes densas RD07, RD08, RD09, RD10 y RD11.

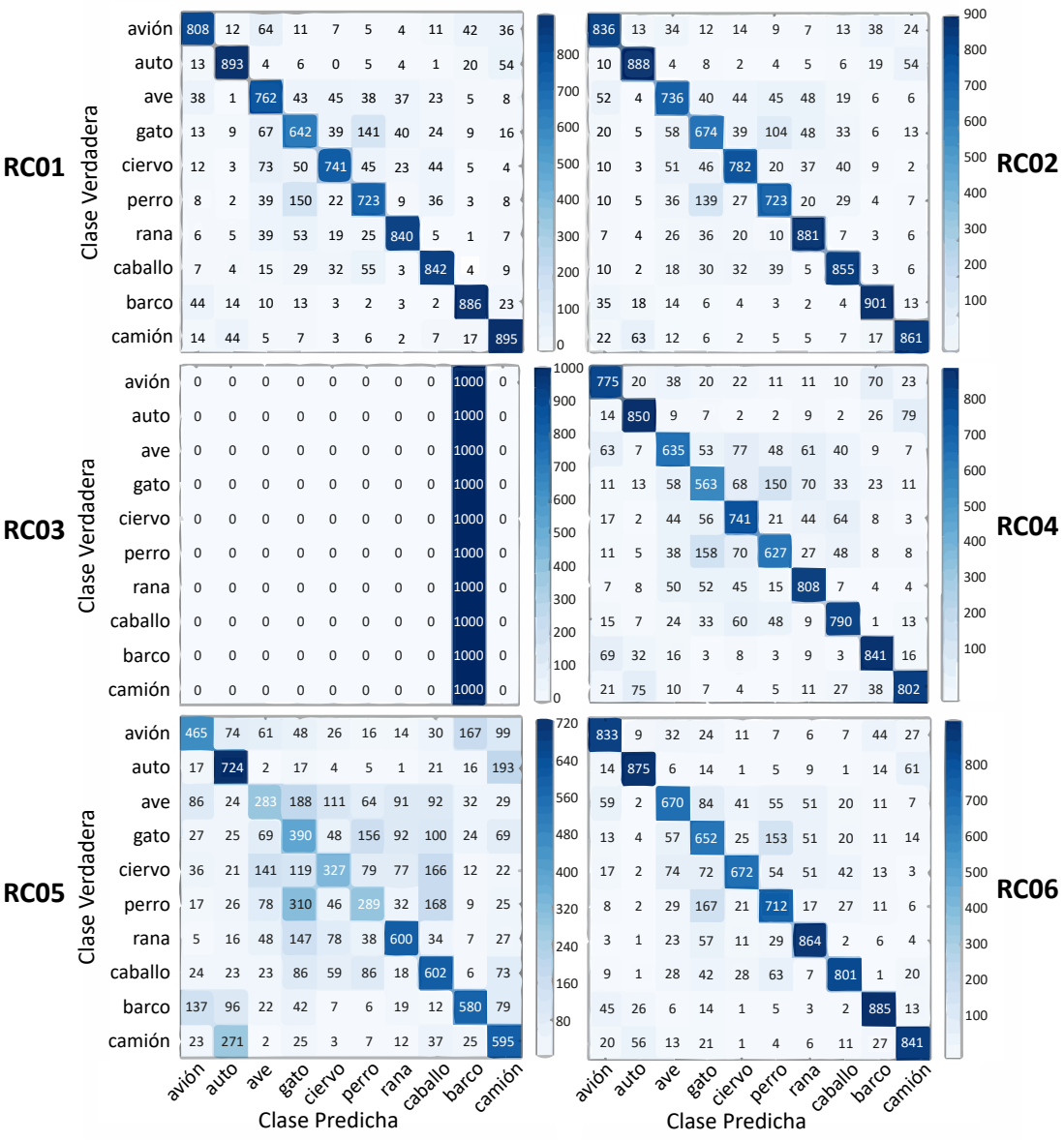


FIGURA C.3: Matrices de confusión sin normalizar asociadas a las redes convolucionales RC01, RC02, RC03, RC04, RC05 y RC06.

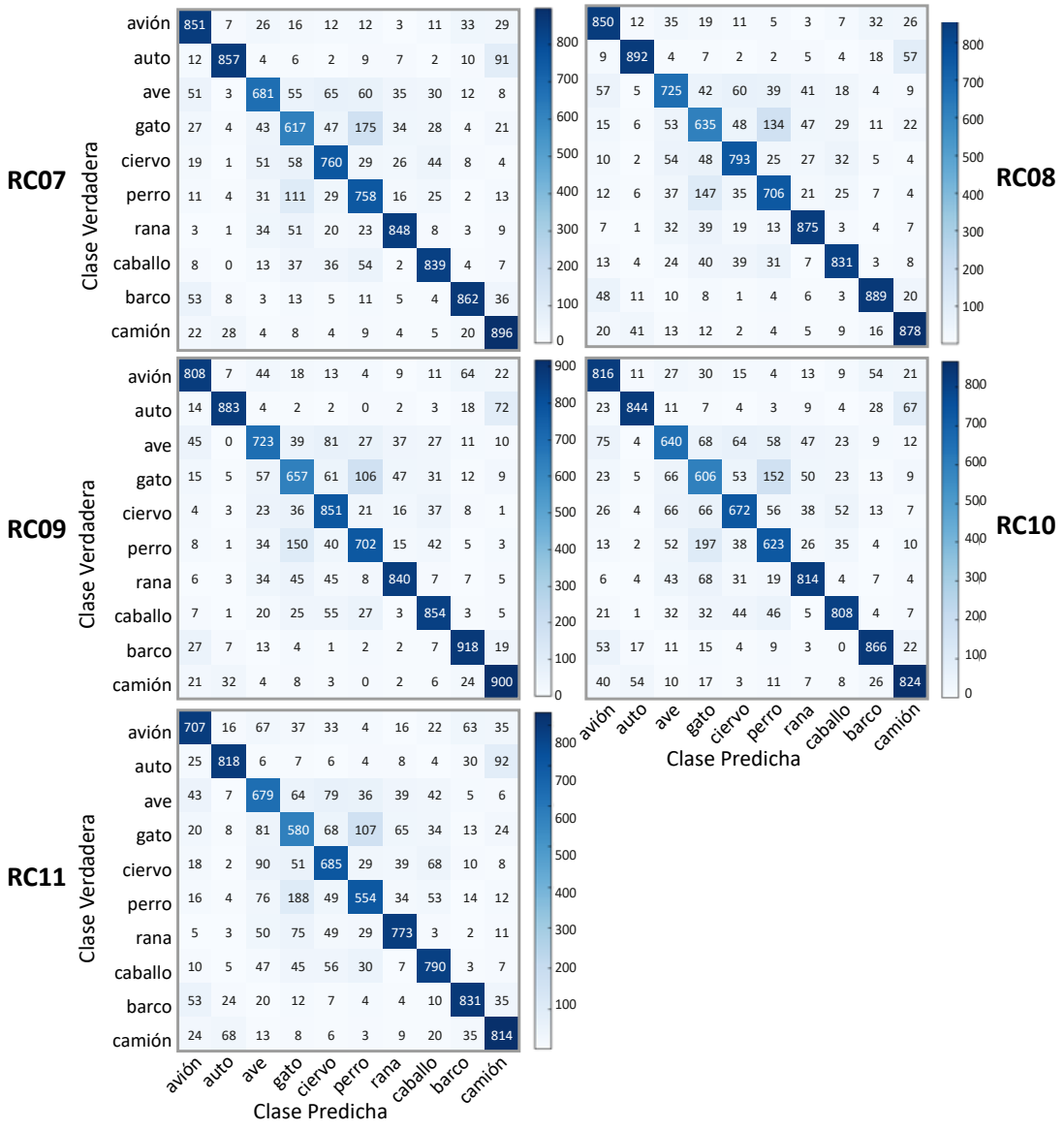


FIGURA C.4: Matrices de confusión sin normalizar asociadas a las redes convolucionales RC07, RC08, RC09, RC10 y RC11.

Bibliografía

Andrade, A. de (2014). Best Practices for Convolutional Neural Networks Applied to Object Recognition in Images [online]. Recuperado de `https://pdfs.semanticscholar.org/e695/4bd33ec4ee682049b64cc6c7fd5c06ffc62a.pdf`

Azevedo, F. A. C., Carvalho, L. R. B., Grinberg, L. T. , Farfel, J. M., Ferretti, R. E. L., Leite, R. E. P., Filho, W. J., Lent, R. y Herculano-Houzel, S. (2009). Equal numbers of neuronal and nonneuronal cells make the human brain an isometrically scaled-up primate brain. *The Journal of Comparative Neurology*, *513*(5), 532-541. DOI: 10.1002/cne.21974.

Ba, J. y Caruana, R. (2013). Do Deep Nets Really Need to be Deep?. *Computing Research Repository*, *abs/1312.6184*. Recuperado de `http://arxiv.org/abs/1312.6184`

Bechtel, M. G., McEllhiney, E., Kim, M. y Yun, H. (2018). DeepPicar: A Low-cost Deep Neural Network-based Autonomous Car. *Computing Research Repository*. Recuperado de `https://arxiv.org/pdf/1712.08644.pdf`

Benenson, R. (2016). *Classification Datasets Results* [online]. Recuperado de `http://rodrigob.github.io/are_we_there_yet/bild/classification_datasets_results.html#43494641522d3130`

Bengio, Y. (2012). Practical Recommendations for Gradient-Based Training of Deep Architectures. *Computing Research Repository*, *abs/1206.5533*. Recuperado de `http://arxiv.org/abs/1206.5533`

Bergstra, J. y Bengio, Y. (2012). Random Search for Hyper-Parameter Optimization. *Journal of Machine Learning Research*, *13*(1), 281-305.

Canziani, A., Paszke, A. y Culurciello, E. (2016). An Analysis of Deep Neural Network Models for Practical Applications. *Computing Research Repository*, *abs/1605.07678*. Recuperado de `http://arxiv.org/abs/1605.07678`

Chernykh, V., Sterling, G. y Prihodko, P. (2017). Emotion Recognition From Speech With Recurrent Neural Networks. *Computing Research Repository*, *abs/1701.08071*. Recuperado de `https://arxiv.org/pdf/1701.08071.pdf`

Chollet, F. (2015). Keras. *GitHub repository*. Recuperado de `https://github.com/fchollet/keras`

Coates, A., Ng, A. Y. y Lee, H. (2011). An Analysis of Single Layer Networks in Unsupervised Feature Learning. *Journal of Machine Learning Research - Proceedings Track*, *15*(1), 215-223. DOI: 10.1.1.186.4059.

Cybenko (1989). Approximation by Superpositions of a Sigmoidal Function. *Mathematics of Control, Signal and Systems*, *2*, 303-314. Recuperado de `http://www.dartmouth.edu/~gvc/Cybenko_MCSS.pdf`

Esteva, A., Kuprel, B., Novoa, R. A., Ko, J., Swetter, S. M., Blau, H. M. y Thrun, S. (2017). Dermatologist-level classification of skin cancer with deep neural networks. *Nature*, *542* (7639), 115–118. DOI: 10.1038/nature21056.

Glorot, X. y Bengio, Y. (2010). Understanding the difficulty of training deep feedforward neural networks. *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics, en PMLR*, *9*, 249-256.

Goodfellow, I., Bengio, Y. y Courville, A. (2016). *Deep Learning* [online]. Recuperado de `http://www.deeplearningbook.org`

Goyal, S. y Benjamin, P. (2014). Object Recognition Using Deep Neural Networks: A Survey. Computer Vision and Pattern Recognition. *Computing Research Repository*, *abs/1412.3684*. Recuperado de `https://arxiv.org/abs/1412.3684`

He, K., Zhang, X., Ren, S. y Sun, J. (2015). Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification. *Proceedings of the 2015 IEEE International Conference on Computer Vision, en IEEE Computer Society*, 1026-1034. DOI: 10.1109/ICCV.2015.123.

Hinton, G. E., Srivastava, N., Krizhevsky, A., Sutskever, I. y Salakhutdinov, R. R. (2012). Improving neural networks by preventing co-adaptation of feature detectors. *Computing Research Repository*, *abs/1207.0580*. Recuperado de `http://arxiv.org/abs/1207.0580`

Hornik, K., Stinchcombe, M. y White, H. (1989). Multilayer feedforward networks are universal approximators. *Neural Networks*, *2*(5), 359-366.

Ioffe, S. y Szegedy, C. (2015). Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. *Proceedings of the Thirty-two International Conference on International Conference on Machine Learning, en JMLR*, *37*, 448-456.

Karpathy, A. (2017). *CS231n: Convolutional Neural Networks for Visual Recognition, Spring 2017* [online]. Recuperado de `http://cs231n.github.io`

Kingma, D. P. y Ba, J. L. (2015). Adam: A Method For Stochastic Optimization. *Computing Research Repository, abs/1412.6980*. Recuperado de `https://arxiv.org/abs/1412.6980`

Krizhevsky, A. (2009). *Learning Multiple Layers of Features from Tiny Images* [online]. Recuperado de `https://www.cs.toronto.edu/~kriz/learning-features-2009-TR.pdf`

Kumar, S. K. (2017). On weight initialization in deep neural networks. *Computing Research Repository, abs/1704.08863*. Recuperado de `https://arxiv.org/pdf/1704.08863.pdf`

Lin, Z., Memisevic, R. y Konda, K.R. (2015). How far can we go without convolution: Improving fully-connected networks. *Computing Research Repository, abs/1511.02580*. Recuperado de `http://arxiv.org/abs/1511.02580`

Liu, B., Wang, M., Foroosh, H., Tappen, M. y Penksy, M. (2015). Sparse Convolutional Neural Networks. *2015 IEEE Conference on Computer Vision and Pattern Recognition* (CVPR), 806-814. DOI: 10.1109/CVPR.2015.7298681.

Livni, R. , Shalev-Shwartz, S. y Shamir, O. (2014). On the Computational Efficiency of Training Neural Networks. *Computing Research Repository, abs/1410.1141*. Recuperado de `https://arxiv.org/pdf/1410.1141.pdf`

London, M. y Häusser, M. (2005). Dendritic Computation. *Annual Review of Neuroscience, 28*(1), 503-532. DOI: 10.1146/annurev.neuro.28.061604.135703

Mitchell, T. M. (1997). *Machine Learning* [online]. Recuperado de `http://www.cs.ubbcluj.ro/~gabis/ml/ml-books/McGrawHill%20-%20Machine%20Learning%20-Tom%20Mitchell.pdf`

Mohsen, H., El-Dahshan, E. A., El-Horbaty, E. M. y Salem, A. M. (2017). Classification using deep learning neural networks for brain tumors. *Future Computing and Informatics Journal*. DOI: 10.1016/j.fcij.2017.12.001.

Nielsen, M. A. (2015). *Neural Networks and Deep Learning* [online]. Recuperado de `http://neuralnetworksanddeeplearning.com`

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D.,

Brucher, M., Perrot, M. y Duchesnay, E. (2011). Scikit-learn: Machine Learning in Python. *Journal of Machine Learning Research*, *12*, 2825-2830. DOI: $< hal - 00650905v2 >$.

Refaeilzadeh, P., Tang, L. y Liu, H. (2008). *Cross-Validation* [online]. Recuperado de `http://leitang.net/papers/ency-cross-validation.pdf`

Rojas, R. (1996). *Neural Networks: A Systematic Introduction* [online]. Recuperado de `https://page.mi.fu-berlin.de/rojas/neural`

Ruder, S. (2017). An overview of gradient descent optimization algorithms. *Computing Research Repository*, *abs/1609.04747*. Recuperado de `https://arxiv.org/pdf/1609.04747.pdf`

Rumelhart, D. E., Hinton, G. E. y Williams, R. J. (1986). Learning representations by back-propagating errors. *Nature*, *323*, 533-536. DOI: 10.1038/323533a0.

Schmidhuber, J. (2015). Deep learning in neural networks: An overview. *Journal of Neural Networks*, *61*(1), 85-117. DOI: 10.1016/j.neunet.2014.09.003.

Shi, S., Wang, Q., Xu, P. y Chu, X. (2017). Benchmarking State-of-the-Art Deep Learning Software Tools. *Computing Research Repository,abs/1608.07249*. Recuperado de `https://arxiv.org/pdf/1608.07249.pdf`

Springenberg, J. T., Dosovitskiy, A., Brox, T. y Riedmiller, M. (2014). Striving for Simplicity: The All Convolutional Net. *Computing Research Repository*, *abs/1412.6806*. Recuperado de `http://arxiv.org/abs/1412.6806`

Srivastava, N., Hinton, G., Frizhevsky, A., Sutskever, I. y Salakhutdinov, R. (2014). Dropout: A Simple Way to Prevent Neural Networks from Overfitting. *Journal of Machine Learning Research*, *15*(1), 1929-1958.

Sze, V., Chen, Y. H., Yang, T. J. y Emer, J. S. (2017). Efficient Processing of Deep Neural Networks: A Tutorial and Survey. *Computing Research Repository*, *abs/1703.09039*. Recuperado de `https://arxiv.org/pdf/1703.09039.pdf`

Torralba, A., Fergus, R. y Freeman, W. T. (2008). 80 million tiny images: a large dataset for non-parametric object and scene recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *30*(11), 1958-1970. DOI: 10.1109/TPAMI.2008.128.

Zeiler, M. D. y Fergus, R. (2013). Stochastic Pooling for Regularization of Deep Convolutional Neural Networks. *Computing Research Repository*, *abs/1301.3557*. Recuperado de `https://arxiv.org/pdf/1301.3557.pdf`