Proyecto Fin de Grado
Ingeniería Electrónica, Robótica y Mecatrónica

Clasificador de Señales de Tráfico Basado en Redes Neuronales Convolucionales para Robots Móviles

Autor: Eduardo Miguel Moscosio Navarro
Tutores: Daniel Gutiérrez Reina

Dpto. de Electrónica
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla

Sevilla, 2020

Proyecto Fin de Grado
Ingeniería Electrónica, Robótica y Mecatrónica

Clasificador de Señales de Tráfico Basado en Redes Neuronales Convolucionales para Robots Móviles

Autor:
Eduardo Miguel Moscosio Navarro

Tutor:
Daniel Gutiérrez Reina
Profesor Contratado

Dpto. de Electrónica
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla
Sevilla, 2020

Proyecto Fin de Carrera: Clasificador de Señales de Tráfico Basado en Redes Neuronales Convolucionales para Robots Móviles

Autor: Eduardo M. Moscosio Navarro

Tutor: Daniel Gutiérrez Reina

El tribunal nombrado para juzgar el Proyecto arriba indicado, compuesto por los siguientes miembros:

Presidente:

Vocales:

Secretario:

Acuerdan otorgarle la calificación de:

Sevilla, 2020

El Secretario del Tribunal

A mi madre, Mª Carmen.

A mi padre, Eduardo.

A mi hermana, Paula.

Agradecimientos

Es mi deseo expresar mis más sinceros agradecimientos a todas las personas que me han ayudado durante toda mi trayectoria haciendo que haya sido posible realizar el presente proyecto.

En primer lugar, agradecer a mi tutor por todo el apoyo y ayuda recibidos de él, que sin duda son parte fundamental del proyecto.

Gracias a toda mi familia, en especial a mis padres, Eduardo y Mª Carmen, y a mi hermana Paula, por ser mi gran apoyo durante toda la carrera y ayudarme a seguir adelante en los momentos más duros ya que, si no hubieran creído en mí, el transcurso de estos años hubiera sido mucho más complicado de lo que fue, y espero que se sientan orgullosos de mí.

También agradecer a todos mis amigos por entenderme y apreciar el sacrificio que he hecho durante todos estos años para llegar hasta donde estoy hoy.

Finalmente, agradecer a todos mis compañeros de clase, en especial a Alejandro Mendoza, por ser parte fundamental de mi vida universitaria y un buen amigo que siempre me ha dado buenos consejos y ayuda cuando lo he necesitado.

Eduardo Miguel Moscosio Navarro

Sevilla, 2020

Resumen

En la actualidad, se experimenta un auge tecnológico en el ámbito de la robótica. Dicho avance se debe en parte al desarrollo de técnicas de reconocimiento de patrones, memoria, etc., para elaborar lo que se conoce como redes neuronales.

Hay muchísimos tipos de redes neuronales, por lo que se decidió resolver un problema actual para mejorar la experiencia de conducción de vehículos.

El proyecto trata de un sistema que implique redes neuronales convolucionales aplicadas a la detección de señales de tráfico. De esta manera, a lo largo del trabajo se estudiará en primer lugar una visión general de algunas de las redes neuronales más usadas, para centrarse a continuación en cada una de las fases de desarrollo de toda la red, empezando por la arquitectura de dichas redes, pasando por el proceso de entrenamiento, y terminando con el proceso de test.

Abstract

Today, there is a technological boom in the field of robotics. This progress is due in part to the development of pattern recognition techniques, memory, etc., to produce what are known as neural networks.

There are many types of neural networks, so it was decided to solve a current problem to improve the driving experience.

The project deals with a system involving convolutional neural networks applied to the detection of traffic signs. In this way, throughout the work, a general vision of some of the most used neural networks will be studied first, to then focus on each of the development phases of the whole network, starting with the architecture of these networks, going through the training process, and ending with the test process.

Índice

Agradecimientos **ix**

Resumen **xi**

Abstract **xiii**

Índice **xiv**

Índice de Tablas **xvi**

Índice de Figuras **xviii**

Notación **xxii**

1 Introducción **1**
1.1 Motivaciones *1*
1.2 Objetivos *2*
1.3 Tendencias *2*
1.4 Mercado e importancia *6*
1.4.1 Impacto en Visión Artificial 6
1.4.2 Sector financiero 8
1.4.3 Sector cultural 9
1.4.4 Sector sanitario 11
1.4.5 Sector automovilístico 12
1.4.6 Sector de la robótica 13
1.5 Comentarios finales *14*

2 Estado del Arte **17**
2.1 Algunas arquitecturas famosas de redes convolucionales *17*
2.1.1 LeNet-5 17
2.1.2 AlexNet 19
2.1.3 Visual Geometry Group (VGG) 20
2.1.4 Network in Network (NiN) 21
2.1.5 GoogLeNet 22
2.1.6 Residual Networks (ResNet) 24

2.1.7 DenseNet 26
2.2 Aplicación a las señales de tráfico 28
2.3 Comentarios finales 33

3 Metodología 35
3.1 Machine Learning 35
3.2 Tipos de redes neuronales 38
3.2.1 Según la topología 38
3.2.2 Según el algoritmo de entrenamiento 41
3.3 Funciones de activación y algoritmos de optimización 44
3.3.1 Funciones de activación 44
3.3.2 Algoritmos de optimización 48
3.4 Planteamiento del problema 52
3.5 Dataset 53
3.6 Descripción general de las partes de la arquitectura 54
3.6.1 Etapa convolucional 54
3.6.2 Etapa Fully Connected 55
3.6.3 Arquitectura final de la red 56
3.7 Procesos de Entrenamiento, Validación y Testeo 58
3.7.1 Entrenamiento 59
3.7.2 Validación 60
3.7.3 Testeo 61
3.8 Comentarios finales 62

4 Resultados 64
4.1 Implementación del diseño 64
4.2 Preprocesado de las imágenes del dataset 66
4.3 Resultados para primeros experimentos 67
4.3.1 Entrenamiento y validación 68
4.3.2 Resultados de test 71
4.4 Experimentos con Dropout 75
4.4.1 Dropout 1 76
4.4.2 Dropout 2 80
4.4.3 Dropout 3 87
4.5 Experimentos de ajuste de hiperparámetros 94
4.6 Discusión de los resultados 101
4.7 Implementación en sistema embebido 105

5 Conclusiones y Futuros Trabajos 110
5.1 Conclusiones del proyecto 110
5.2 Futuros trabajos 111

6 Bibliografía 115

ÍNDICE DE TABLAS

Tabla 2-1: Redes VGG según el número de capas 21
Tabla 2-2: Arquitectura GoogleNet con bloques Inception 23
Tabla 2-3: Arquitecturas de ResNet 25
Tabla 2-4: Arquitecturas de DenseNet según el número de capas, indicando para cada una el growth rate correspondiente (k) [31] 27
Tabla 2-5: Estructura del APM [38] 32
Tabla 3-1: División de imágenes del dataset para experimentos de training y test 59
Tabla 3-2: Distribución final de imágenes del dataset 61
Tabla 4-1: Resultados de los experimentos 75
Tabla 4-2: Resultados experimentales 101
Tabla 4-3: Especificaciones de sistema embebido 106

ÍNDICE DE FIGURAS

Figura 1-1: Arquitectura von Neumann 3
Figura 1-2: Arquitectura monolítica y MCM para GPU NVIDIA 4
Figura 1-3: Arquitecturas de TPU v2 y TPU v3 5
Figura 1-4: Primitivas de computación de CPU, GPU y TPU 5
Figura 1-5: Sensores y cámaras de visión artificial 6
Figura 1-6: Ejemplo de eficacia de Deep Learning aplicado a Visión Artificial [12] 7
Figura 1-7: Usos de Visión Artificial clásica y con Deep Learning [12] 8
Figura 1-8: Tecnologías que aplican IA al sector financiero según su estado de maduración 8
Figura 1-9: Escenarios originales (Izquierda) vs Escenarios remasterizarizados con IA (Derecha) 9
Figura 1-10: Personalización de portadas de Netflix 10
Figura 1-11: Recreación de Carrie Fisher como 'Leia Organa' para Rogue One: una historia de Star Wars (G. Edwards, 2016) y rejuvenecimineto de Kurt Russell como 'Ego' Guardianes de la Galaxia Vol. 2 (J. Gunn, 2017) 10
Figura 1-12: Watson Health de IBM y Biobank 11
Figura 1-13: Funcionamiento de Corti en una llamada de emergencia 12
Figura 1-14: Sistema ADAS con IA de Hyundai que aprende del manejo del usuario [21] 12
Figura 1-15: Robot Mulán trabajando en el restaurante Crensa de Valencia 13
Figura 1-16: Brazos robóticos UR3 resolviendo un cubo de Rubik 14
Figura 2-1: Arquitectura de LeNet-5 [25] 18
Figura 2-2: Conexiones entre capas S2 y C3 de LeNet-5 [25] 18
Figura 2-3: Comparación entre arquitectura LeNet (izquierda) y AlexNet (derecha) [24] 19
Figura 2-4: Arquitectura AlexNet dividida para poder ser entrenada con GPU en 2012 [26] 20
Figura 2-5: AlexNet usando bloques VGG [24] 20
Figura 2-6: Comparación de AlexNet, VGG y NiN [24] 22
Figura 2-7: Arquitectura del bloque Inception de GoogLeNet [24] 22
Figura 2-8: Arquitectura de la red GoogLeNet usando el bloque Inception [24] 23
Figura 2-9: Modelo GoogLeNet usado para el entrenamiento en la ILSVRC de 2014 [29] 24
Figura 2-10: Bloque Residual de ResNet [24] 24
Figura 2-11: Comparación entre VGG-19, PlainNet y ResNet de 34 capas 25
Figura 2-12: Diferencia entre ResNet (izquierda) y DenseNet (derecha) [24] 26
Figura 2-13: Conexiones densas en DenseNet [24] 26
Figura 2-14: Dense Block de 5 capas con tasa de crecimiento k=4 [31] 27
Figura 2-15: Etapas de AMAP para detección de señales de tráfico [34] 29
Figura 2-16: Cámara MPC3 de Bosch [35] 30

Figura 2-17: Arquitectura de bloques del sistema de detección y clasificación de señales del documento [37] ________ 30
Figura 2-18: Esquema global del sistema [38] ________ 32
Figura 3-1: Deep Learning en IA ________ 36
Figura 3-2: Diagrama de funcionamiento de algoritmo aprendizaje supervisado ________ 36
Figura 3-3: Diagrama de funcionamiento de algoritmo aprendizaje no supervisado ________ 37
Figura 3-4: Diagrama de funcionamiento de algoritmo Reinforcement Learning ________ 37
Figura 3-5: Modelo de un perceptrón [40] ________ 38
Figura 3-6: Red monocapa autoconcurrente [41] ________ 39
Figura 3-7: Modelo de red de Hopfield con 3 neuronas, siendo xi *las entradas,* yi *las salidas, y* wij *los pesos de la red* _ 39
Figura 3-8: Estructura de perceptrón multicapa con 3 capas ________ 40
Figura 3-9: Ejemplo de red tipo ART1 ________ 41
Figura 3-10: Esquema resumen de las topologías de redes neuronales [42] ________ 41
Figura 3-11: Funcionamiento de aprendizaje supervisado y backpropagation [39] ________ 42
Figura 3-12: Estructura general competitiva (izquierda) y particularización para 2 neuronas (derecha) ________ 43
Figura 3-13: Función de activación lineal ________ 45
Figura 3-14: Función umbral con umbral = 0 ________ 45
Figura 3-15: Función sigmoide ________ 46
Figura 3-16: Función tangente hiperbólica ________ 46
Figura 3-17: Función ReLU con umbral de entrada de valor 0 ________ 47
Figura 3-18: Función Leaky ReLU, variación de la ReLU clásica [47] ________ 47
Figura 3-19: Función Softmax [47] ________ 48
Figura 3-20: Algoritmo de Gradiente Descendente ________ 49
Figura 3-21: Ecuaciones del algoritmo AdaGrad ________ 49
Figura 3-22: Ecuaciones del algoritmo RMSprop ________ 50
Figura 3-23: Función de loss con varios mínimos [39] ________ 51
Figura 3-24: Comparación entre la actualización del paso con el momentum clásico y con el Nesterov ________ 51
Figura 3-25: Deducción de la fórmula de Adam a partir del RMSprop y la introducción del hiperparámetro momentum 52
Figura 3-26: Señales de tráfico a clasificar ________ 53
Figura 3-27: Esquema de la red diseñada ________ 57
Figura 3-28: Modelo de la red desarrollada en Keras ________ 58
Figura 3-29: Dropout [50] ________ 60
Figura 3-30: Funcionamiento de Dropout [50] ________ 60
Figura 3-31: Matriz de confusión genérica [39] ________ 62
Figura 4-1: Etapas de división de imágenes del dataset ________ 66
Figura 4-2: Redimensión de imágenes a 64x64 píxeles ________ 66
Figura 4-3: Conversión a escala de grises ________ 67
Figura 4-4: Normalización de píxeles de la imagen en el rango [0,1] ________ 67
Figura 4-5: Train vs Validation con optimizador SGD y 50 épocas ________ 68
Figura 4-6: Train vs Validation con optimizador SGD y 100 épocas ________ 69
Figura 4-7: Train vs Validation con optimizador RMSprop y 50 épocas ________ 70
Figura 4-8: Train vs Validation con optimizador Adam y 50 épocas ________ 71
Figura 4-9: Matriz de confusión de SGD con 50 epochs ________ 72
Figura 4-10: Matriz de confusión de SGD con 100 epochs ________ 72
Figura 4-11: Resultados de test SGD para 50 (izquierda) y 100 epochs (derecha) ________ 73
Figura 4-12: Matriz de confusión RMSprop 50 epochs ________ 73
Figura 4-13: Resultados de test de RMSprop (izquierda) y Adam (derecha) con 50 epochs ________ 74
Figura 4-14: Matriz de confusión Adam 50 epochs ________ 74
Figura 4-15: Train vs Validation con optimizador RMSprop, 200 épocas, y Dropout 1 ________ 77
Figura 4-16: Train vs Validation con optimizador Adam, 200 épocas, y Dropout 1 ________ 78
Figura 4-17: Test con optimizador Adam, 200 épocas, y Dropout 1 ________ 79
Figura 4-18: Matriz de confusión Adam 200 epochs Dropout 1 ________ 79
Figura 4-19: Train vs Validation con optimizador RMSprop, 200 épocas, y Dropout 2 ________ 80
Figura 4-20: Train vs Validation con optimizador Adam, 200 épocas, y Dropout 2 ________ 81
Figura 4-21: Train vs Validation con optimizador RMSprop, 50 épocas, y Dropout 2 ________ 82
Figura 4-22: Train vs Validation con optimizador Adam, 50 épocas, y Dropout 2 ________ 83
Figura 4-23: Test RMSprop (izquierda) y Adam (derecha) para Dropout 2 con 200 epochs ________ 84
Figura 4-24: Matriz de confusión RMSprop 200 epochs Dropout 2 ________ 84
Figura 4-25: Matriz de confusión Adam 200 epochs Dropout 2 ________ 85
Figura 4-26: Test RMSprop (izquierda) y Adam (derecha) para Dropout 2 con 50 epochs ________ 85

Figura 4-27: Matriz de confusión RMSprop 50 epochs Dropout 2 86
Figura 4-28: Matriz de confusión Adam 50 epochs Dropout 2 86
Figura 4-29: Train vs Validation con optimizador Adam, 200 épocas, y Dropout 3 87
Figura 4-30. Train vs Validation con optimizador RMSprop, 200 épocas, y Dropout 3 88
Figura 4-31: Train vs Validation RMSprop 50 epochs y Dropout 3 89
Figura 4-32: Train vs Validation Adam 50 epochs y Dropout 3 90
Figura 4-33: Test con optimizador RMSprop (izquierda) y Adam (derecha), 50 épocas, y Dropout 3 91
Figura 4-34: Matriz de confusión RMSprop 50 epochs Dropout 3 91
Figura 4-35: Matriz de confusión Adam 50 epochs Dropout 3 92
Figura 4-36: Matriz de confusión RMSprop 200 epochs Dropout 3 92
Figura 4-37: Test con optimizador RMSprop (izquierda) y Adam (derecha), 200 épocas, y Dropout 3 93
Figura 4-38: Matriz de confusión Adam 200 epochs Dropout 3 93
Figura 4-39: Modelo con Adam 50 epochs modificación 1 94
Figura 4-40: Train vs Validation Adam 50 epochs modificación 1 95
Figura 4-41: Train vs Validation Adam 50 epochs modificación 2 96
Figura 4-42: Modelo con Adam 50 epochs modificación 2 96
Figura 4-43: Modelo Dropout con Adam 50 epochs modificación 3 97
Figura 4-44: Train vs Validation Adam 50 epochs Dropout modificación 3 98
Figura 4-45: Resultados de test de la modificación de red 1, 2 y 3 de izquierda a derecha respectivamente 99
Figura 4-46: Matriz de confusión de modificación 1 99
Figura 4-47: Matriz de confusión de modificación 2 100
Figura 4-48: Matriz de confusión de modificación 3 100
Figura 4-49: Matriz de confusión del modelo final 103
Figura 4-50: Resultados de test de modelo final 104
Figura 4-51: Imágenes de la clase 38 confundidas con la clase 1 105
Figura 4-52: Imágenes de la clase 23 confundidas con la clase 20 105
Figura 4-53: Montaje de Raspberry Pi 3 con cámara 106
Figura 5-1: Arquitectura YOLO de 24 capas [54] 112
Figura 5-2: YOLOv3 sobre el dataset COCO [54] 113

Notación

IA	Inteligencia Artificial
CPU	Unidad Central de Procesamiento
GPU	Unidad de Procesamiento Gráfico
TPU	Unidad de Procesamiento Tensorial
API	Interfaz de Programación de Aplicaciones
CNN	Redes Neuronales Convolucionales
s	Segundos
ms	Milisegundos
Km/h	Kilómetros por hora
e	Número e
sen	Función seno
tg	Función tangente
arctg	Función arco tangente

1 INTRODUCCIÓN

El misterio de la vida no es un problema a resolver, sino una realidad a experimentar.

- Frank Herbert -

Actualmente, el uso de la Inteligencia Artificial está en auge, siendo capaz de conseguir cosas que hace algunas décadas parecerían de ciencia ficción. Campos como la aviación, medicina o seguridad se han visto afectados por esta revolución, cambiando en muchas ocasiones el panorama actual de dichas áreas.

La automoción tampoco se queda atrás en cuanto a aplicación de dicha tecnología. Existen ya coches que pueden aparcar solos con ayuda de cámaras y sensores que le permiten conocer el entorno, y posiblemente en algunos años sean capaces de conducir de manera autónoma. Pero, para que esto ocurra, primero se debería tener un sistema que sea igual o más fiable que una persona para no poner en riego vidas humanas. Una de las áreas de investigación que está avanzando a pasos agigantados es la visión artificial. La visión artificial es básicamente la capacidad que un ordenador tiene para "ver" el entorno que lo rodea, y para desarrollarla una de las herramientas son las redes neuronales.

Aunque por su nombre parezca que se trata de una red de auténticas neuronas como las que forman el cerebro, nada más lejos de la realidad, se le da ese nombre porque se trata de una red de parámetros con unos valores o pesos que pueden "aprender", es decir, ajustarse de manera que puedan reconocer diferentes características que tienen unos datos de entradas, que pueden ser matrices, vectores, etc., para obtener una salida que haga una predicción de la situación del sistema. Esto se puede aplicar a imágenes, ya que una imagen no es más que una matriz de valores codificados para que representen un color determinado. No es ciencia ficción, son matemáticas, programación, y un poco de arte, ya que el diseño de una red neuronal actualmente se podría considerar un auténtico arte.

De este modo, se crearon unas redes algo diferentes a las demás, que estaban optimizadas para trabajar con imágenes y audio, y son las redes convolucionales. Aunque estas redes no solo se pueden utilizar con imágenes, tal como se acaba de decir, es dicho uso el que se tratará más en profundidad, ya que son las que componen el grueso de este proyecto, en el que se construirá una red neuronal convolucional que sea capaz de clasificar diferentes señales de tráfico con el fin de implementarlo a un futuro sistema de coche autónomo.

Para comenzar, se explicará por qué se ha hecho este trabajo, los objetivos que se pretenden conseguir, la situación de mercado y tendencias actuales en este campo.

1.1 Motivaciones

Principalmente, el desarrollo de este trabajo ha sido impulsado por la curiosidad de conocer más el ámbito de la IA y en concreto, de las redes neuronales, ya que no se ha visto nada de esta temática a lo largo del grado y se considera que actualmente está en auge creciente, siendo una parte importante en el campo de la robótica, que son los estudios cursados por el autor del presente trabajo.

Adquirir estos conocimientos servirá para tener una base de partida en la que poder apoyarse en futuros proyectos que apliquen esta disciplina, pudiendo sacar adelante dicho trabajo haciéndolo lo mejor posible, además de poder asimilar con mayor facilidad conocimientos de este tipo si dicho proyecto necesitase una mayor cantidad de conocimientos en esta área. De esta forma, la red elaborada en este trabajo podría ser considerada como los primeros pasos para conseguirlo y ser un mejor profesional en esta disciplina que es la Inteligencia Artificial.

1.2 Objetivos

Los objetivos que se desean alcanzar en el proyecto se enumeran a continuación:

- Conocimiento básico de Deep Learning y redes neuronales.
- Conocimiento del lenguaje Python y las librerías más comunes desarrolladas para redes neuronales.
- Desarrollo de una red neuronal convolucional tipo CNN aplicada a reconocimiento de señales de tráfico.
- Implementación final de la red anterior de modo que pueda probarse su eficacia en unos datos que nunca haya visto.
- Implementación en sistema embebido real, en este caso, una Raspberry Pi 3 modelo B.

Una vez que se han enumerado los objetivos a alcanzar en este trabajo, se explicará en las siguientes secciones la historia de las redes neuronales hasta la actualidad, así como la repercusión que tiene en las diferentes áreas de mercado.

1.3 Tendencias

En esta sección se tratarán las diferentes tendencias y etapas de evolución de las redes neuronales a lo largo de la historia hasta la época reciente. Para realizarlo, se ha hecho uso las páginas [1] y [2].

Las redes neuronales parecen ser muy recientes debido al auge del que actualmente gozan, pero la primera red neuronal data de finales de los años 50, cuando el científico Frank Rosenblatt desarrolló el modelo del perceptrón, inspirado en los estudios biológicos de Warren McCulloch y Walter Pitts. Lo que intentaba era emular de la manera más sencilla posible el funcionamiento conocido de una neurona en aquella época, de modo que la neurona artificial era en realidad una función "todo-nada" capaz de distinguir entre dos opciones, asignándole el valor '0' a una y el valor '1' a otra. Su funcionamiento por tanto estaba limitado por la suma de los pesos de las entradas de modo que, si se superaba el umbral de la función binaria de la neurona en cuestión, sacaba un '1' como salida, y un '0' en caso contrario. A mediados de la década de los 60 surgió el concepto de perceptrón multicapa, que es la evolución natural del perceptrón de Rosenblatt. Consiste en coger varios perceptrones simples y ponerlos en la misma capa, formando varias capas de neuronas que se conectan entre sí.

En estos dos primeros casos, el entrenamiento era bastante tedioso, ya que al ser una función "todo-nada", el ajuste de parámetros era muy largo de hacer. Esto cambió con la llegada de diferentes funciones de decisión, en concreto con la función sigmoide, a principios de la década de 1980. Tener esta función dentro del perceptrón permitió que pequeñas variaciones de los parámetros influyeran en la salida poco a poco, en lugar de tener que cambiar el valor del parámetro por otro radicalmente distinto. Esto permitió también que el ajuste de pesos dejara de ser manual y pudiera empezar a hacerse de manera automática.

El siguiente avance fue la aparición de las redes de tipo Feed Forward, donde las salidas de una de las capas de la red neuronal funcionaban como entradas de la capa siguiente, apareciendo el término "*fully-connected*". Esto acarreó la aparición de algoritmos de entrenamiento automático muy eficaces como el "*backpropagation*" a mediados de 1980, permitiendo entrenar los pesos de la red de manera supervisada pero automática. Fue a final de los 80 cuando surgió el tipo de red con el que se trabajará en este proyecto, conocidas como redes convolucionales, o CNN, desarrolladas por Yann LeCun. Son muy usadas actualmente en problemas de procesado de audio e imagen. En las redes tradicionales "*fully-connected",* o densamente conectadas, si se querían procesar imágenes, había que poner una neurona por cada pixel de la imagen de modo que se pudieran extraer características de toda la imagen. Esto era inviable, ya que para imágenes de tamaños muy grandes con un número excesivo de píxeles aumentaban de manera exponencial el número de parámetros a entrenar, y la

capacidad de cómputo no era suficiente en aquella época, por lo que tardaría meses, e incluso años, en entrenar toda la red. Las CNN permitían aplicar el reconocimiento de características a partes de la imagen en lugar de a su totalidad concentrando las características en un único peso, además de usar capas de pooling para reducir el tamaño del mapa de características que se crea. Esto permite obtener una matriz de pesos con todas las características de la imagen pero con un tamaño muchísimo menor que dicha imagen, de forma que al pasarle esas caracterísiticas a una "*fully-connected*" clásica el entrenamiento ya sí que era posible de hacer. Para un audio se haría de manera análoga.

Cada capa convolucional era capaz de reconocer características más escondidas en la imagen o audio de entrada. Esta forma de pensar en el aprendizaje abrió un abanico de posibilidades que, junto con la aparición a finales de la década de 1990 de redes que tenían celdas de memorias para almacenar loops para poder aprender hacia las capas de atrás, nació la disciplina conocida como Deep Learning.

La última revolución que ha surgido en este campo ha sido la aparición de unidades de procesamiento como las GPU's, que engrandecieron la capacidad de cómputo con cálculos paralelos, haciendo el entrenamiento de las redes mucho más rápidos y eficientes que de la manera en la que lo hacían las CPU's. Más tarde, aparecieron también las TPU's, que son exclusivamente dedicadas al trabajo con Deep Learning, de modo que los resultados son todavía mejores que con las GPU's.

El término CPU surgió a principios de la década de 1960 y se corresponde con las siglas de "*Central Processing Unit*" (o "*Unidad Central de Procesamiento*" en español). Se puede definir como "*un procesador hardware que se diseña con propósito general, de modo que su memoria y caché sean óptimas para cualquier problema de programación de carácter general*", según se especifica en [3]. Se compone básicamente de una unidad aritmético-lógica (ALU) que se encarga de realizar todas las operaciones matemáticas sencillas, los registros que usa la ALU para realizar dichas operaciones, y una unidad de control que extrae las instrucciones almacenadas en memoria, de modo que las interpreta y las ejecuta [4]. El esquema básico de una CPU con arquitectura de von Neumann como la mostrada en la Figura 1-1[1]:

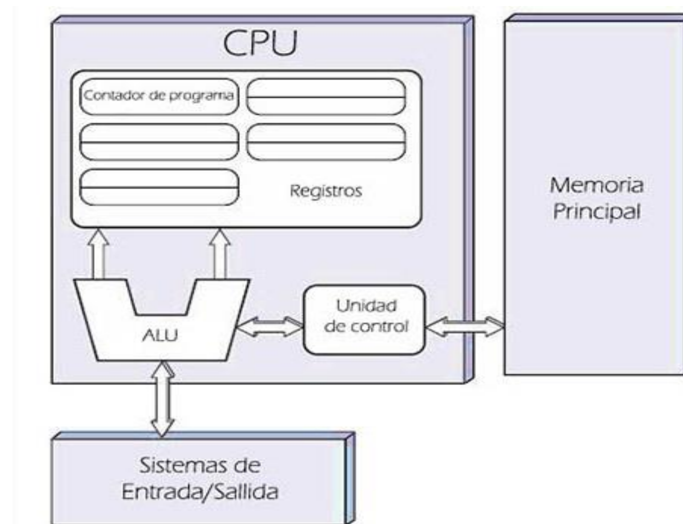


Figura 1-1: Arquitectura von Neumann

Los tipos de CPU se clasifican según el número de núcleos que posea, pudiendo llegar a haber CPU's con más de 12 núcleos. Sin embargo, aunque tenga muchos núcleos no quiere decir que sea más rápido que otra que tenga menos núcleos, ya que influyen varios factores, como el diseño de dicho núcleo y la velocidad de operación. Respecto al primer factor, se puede dar el caso de un núcleo físico que sea capaz de ejecutar 2 hilos de ejecución, actuando así como si fueran 2 núcleos en lugar de 1. El segundo factor lo marca la frecuencia del reloj, de modo que a mayor frecuencia, mayor número de acciones por segundo, y menor es el tiempo que tarda en realizar las operaciones correspondientes a una instrucción [4].

[1] Fuente: https://www.lifeder.com/arquitectura-von-neumann/

El término GPU corresponde a las siglas "*Graphics Processing Unit*" (o *Unidad de Procesamiento Gráfico* en español), y se pude definir como "*un coprocesador dedicado al procesamiento de gráficos u operaciones de coma flotante, para aligerar la carga de trabajo del procesador central en aplicaciones como los videojuegos o aplicaciones 3D interactivas*" según se dice en [5]. Al estar dedicado, ya no tiene propósito general como tenía la CPU, de modo que realizará las operaciones correspondientes de manera más optimizada y rápida que las CPU's, ya que tiene mayor capacidad de trabajo en paralelo debido a sus núcleos que le permiten procesar las imágenes más rápido. Para su diseño, se usan muchos núcleos con bajas frecuencias de reloj, de forma contraria a las CPU's, que tienen pocos núcleos con altas frecuencias de reloj. La mayor parte de dichos núcleos de la GPU realizan funciones independientes, pero que se pueden agrupar en 2 grupos según su finalidad global, que puede ser procesamiento de píxeles o procesamiento de vértices [6].

La estructura de las GPU's ha evolucionado desde una de tipo monolítico con una única GPU centralizada, hasta una estructura de tipo "*Multi-Chip-Module*" (MCM), que divide el trabajo entre varias GPU's en lugar de tener una sola GPU centralizada. Esto ha ocurrido porque la Ley de Moore ha llegado a un punto de desaceleración, donde el número de transistores posibles por unidad de espacio ya no crece a un ritmo tan alto, por lo que el rendimiento de la estructura monolítica llegó a su techo, de modo que se evolucionó a una estructura MCM para que el rendimiento siguiera subiendo a pesar de la ley de Moore. Esta division en módulos GPU's más básicos unida a un alto ancho de banda y una buena eficiencia energética permiten grandes mejoras de rendimiento [7]. Para ilustrar gráficamente ambas estructuras, se añade la Figura 1-2[2] con el paso de una a la otra:

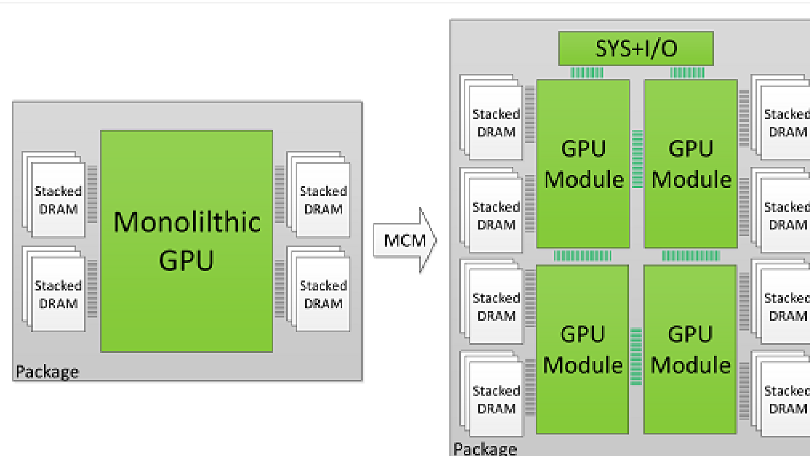


Figura 1-2: Arquitectura monolítica y MCM para GPU NVIDIA

Por último, las TPU o *Unidades de Procesamiento Tensorial* ("*Tensor Processing Unit*"), desarrolladas por Google en 2016, son dispositivos que se definen según su creador como "*los circuitos integrados personalizados específicos de aplicaciones (ASIC) de Google que se usan para acelerar las cargas de trabajo de aprendizaje automático*" según el documento de Google [8].

Cada núcleo de la TPU tendrá unidades escalares, vectoriales y matriciales (MXU), realizando 16,000 multiplicaciones y acumulaciones por ciclo. También tendrá cada núcleo una memoria de gran ancho de banda (HBM). En la Figura 1-3 se proporciona la estructura de las TPUv2 y TPUv3.

[2] Fuente: https://hardzone.es/app/uploads-hardzone.es/2020/03/MCM-NVIDIA.png

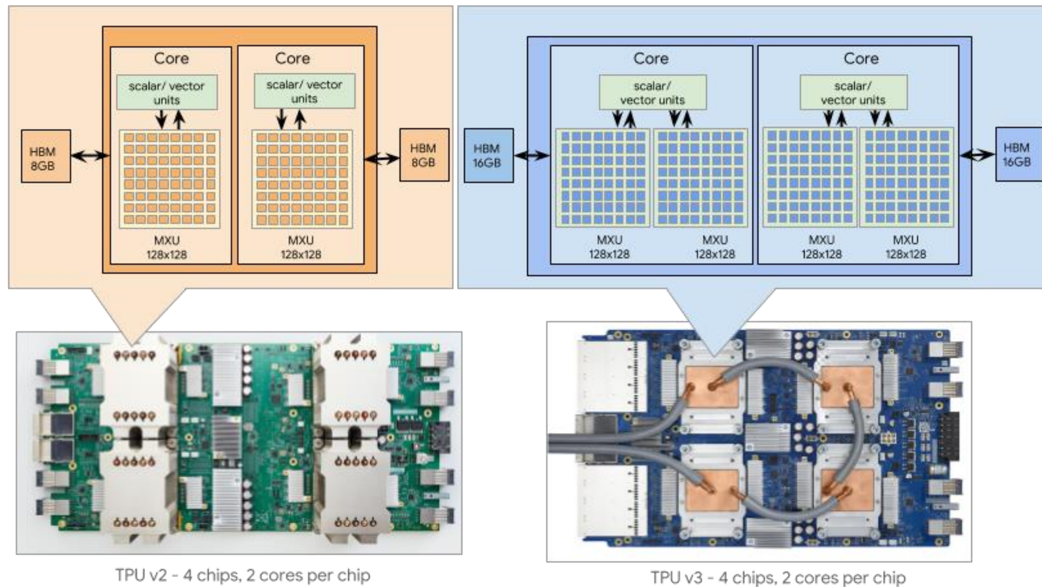


Figura 1-3: Arquitecturas de TPU v2 y TPU v3

Estos dispositivos pueden tener varias configuraciones, ya sea usando una sola TPU que soporte toda la carga de trabajo, o conectando varias TPU a través de redes dedicadas de alta velocidad de modo que se reparten la carga de trabajo [9].

Si se comparan la CPU, GPU y TPU desde el punto de vista computacional:

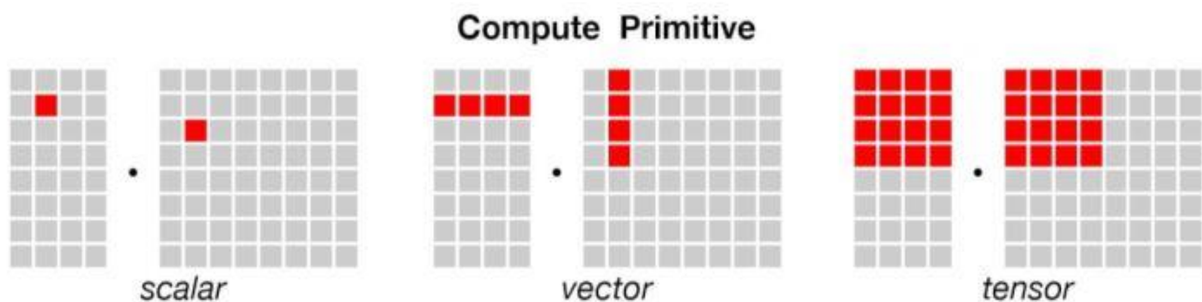


Figura 1-4: Primitivas de computación de CPU, GPU y TPU

Se ve en la Figura 1-4[3] como la CPU solo puede multiplicar escalares, por lo que la dimensión de los datos puede ser como máximo de 1x1 a la vez. La GPU en cambio, al tener las primitivas de computación que detectan vértices y píxeles, pueden operar con vectores directamente, de modo que el tamaño de dato máximo ahora será de 1xN siendo N la longitud del vector. Por último, la TPU usa tensores de tamaño N, siendo el tamaño máximo del dato de NxN [3].

Lo que se puede concluir de dicha comparación, es que al trabajar con matrices, como son las imágenes que se van a usar en el proyecto, las CPU's tendrán un funcionamiento más general que hará que sea el peor para trabajar en el caso concreto de las imágenes, haciendo que los entrenamientos de las redes sean más largos. La GPU funcionará mejor al estar optimizada para ello, de modo que operará con las imagenes de manera más rápida que la CPU. Por último, la TPU es la que mejor funcionaría ya que está optimizada para uso de Deep Learning, pudiendo obtener todos los cálculos de la imagen prácticamente en una operación debido a que puede

[3] Fuente: https://petamind.com/quick-benchmark-colab-cpu-gpu-tpu-xla-cpu/

usarse un tensor de las mismas dimensiones que la propia imagen, reduciendo bastante los tiempos de entrenamiento respecto a la CPU y GPU.

Estas serían las tendencias de evolución que han ido sufriendo las redes neuronales hasta la actualidad, así como el cambio también en las capacidades hardware de procesamiento para dichas redes. En el apartado siguiente se verá el estado actual de mercado de dichas redes en diferentes áreas comerciales.

1.4 Mercado e importancia

Para finalizar este capítulo, se explorará el mercado que abre la inteligencia artificial en la actualidad, de modo que se pueda apreciar el impacto que ha tenido en la sociedad. De igual forma, se explorará el impacto que han tenido las redes neuronales en la Visión Artificial, una de las bases de este proyecto.

1.4.1 Impacto en Visión Artificial

Según la Wikipedia, la Visión Artificial se define como "*una disciplina científica que incluye métodos para adquirir, procesar, analizar y comprender las imágenes del mundo real con el fin de producir información numérica o simbólica para que puedan ser tratados por un ordenador*" [10]. Es decir, que es el equivalente a la visión humana intentando ser imitada por una computadora.

El análogo lógico al ojo humano teniendo lo anterior en cuenta sería una cámara, de manera que pudiera traducir el entorno capturado a unidades interpretables por un ordenador. Pero en realidad la Visión Artificial no solo usa cámaras. Se da en [11] una idea comercial del número de dispositivos que se pueden usar en Visión Artificial, entre ellos los sensores de visión, que tienen menores funcionalidades que una cámara y únicamente pueden indicar funciones binarias de éxito o fallo.

Las cámaras inteligentes en cambio tienen mayor potencia de cálculo y resolución, y la capacidad de aportar matices del entorno que capturan, ya que además de incluir sensores de visión están dotadas de memoria y procesador. A pesar de ello, lo más normal es que se requiera un sistema externo de entrada/salida que ayude a procesar dicha información.

Además de los dos dispositivos anteriores, también existen sistemas avanzados, destinados a problemas más complejos, por lo que tendrán mayores prestaciones que las cámaras inteligentes o los sensores de visión estándares. Se pueden apreciar algunos de los dispositivos anteriores en la Figura 1-5[4].

Figura 1-5: Sensores y cámaras de visión artificial

Tradicionalmente, se han empleado los elementos anteriores en conjunto con una enorme base de datos que contiene una lista de patrones y plantillas de los objetos que se deseen reconocer y "ver" con el sistema de Visión Artificial, y mediante la comparación del objeto capturado con los que tiene en su biblioteca puede obtener una

[4] Fuente: http://www.satvision.es/productos/

identificación, medida, o aquello para lo que se haya programado el sistema en cuestión. Esto tiene un problema muy reconocible, y es que los algoritmos y programas desarrollados con esta técnica son demasiado dependientes de las bases de datos anteriores, por lo que será muy poco flexible en general ante cambios con respecto a su patrón almacenado y ante la introducción a la detección de nuevos objetivos.

En [12] se da un pequeño análisis del impacto de la IA sobre la Visión Artificial, siendo aquí donde el Deep Learning con las redes neuronales hace acto de aparición para solventar este problema y cambiar el paradigma de la Visión Artificial. Gracias a la arquitectura de capas de neuronas que intentan simular el funcionamiento humano, se desarrollan en cada capa algoritmos que son capaces de reconocer características específicas de los objetos que se desea reconocer, de manera que si se entrena con una buena base de datos de imágenes, no será necesaria ninguna base de datos posterior con la que comparar lo que capture la cámara, ya que ahora ha obtenido un conocimiento generalizado en los pesos de la red y es capaz de reconocer las características de los objetos deseados que se encuentren en el entorno, siendo muy flexible a fallos y ruidos en la imagen respecto a los patrones que se usaron para entrenarse. Además, podrían adaptarse a un nuevo tipo de objetivo de reconocimiento más rápido que con el método clásico, donde habría que reprogramar todo el algoritmo de detección.

Un ejemplo clásico de lo anterior es la aplicación de Deep Learning al problema de Visión Artificial de reconocimiento de caracteres, donde para reconocer una única letra de manera correcta la base de datos del método de reconocimiento clásico debería ser enorme para contener la mayor cantidad de variaciones de dicha letra, y aún así no asegura un funcionamiento correcto si la escritura de esa letra tiene fallos, mientras que con Deep Learning es capaz de reconocer la letra incluso si la escritura es defectuosa, siempre y cuando conserve las características básicas de dicha letra. En la Figura 1-6 siguiente se observa hasta dónde puede llegar el reconocimiento usando Deep Learning sobre un código de barras que está doblado, y aún así reconoce los números de manera correcta, cosa que no podría hacerse con el método clásico de Visión Artificial.

Figura 1-6: Ejemplo de eficacia de Deep Learning aplicado a Visión Artificial [12]

A pesar de las mejoras que produce el Deep Learning en cuanto a la generalización de las características de los objetos detectados, hay veces en las que la visión tradicional sigue siendo mejor, como en la toma de medidas o en los procesos de calibración, ya que al estar basadas en algoritmos programados con reglas específicas para el proceso concreto se obtendrán datos más exactos que si se usa un método más general como es el que implementa Deep Learning. Sin embargo, sí que se podría usar esta última para inspeccionar las zonas concretas en las que se deseen tomar medidas y trabajaría mejor que la visión tradicional, o simplemente no se usa este método en situaciones en las que no merece la pena por la simplicidad del proceso, como en el reconocimiento binario de presencia o no de una pieza en un sitio, donde es más rentable usar la visión clásica. La clave está en emplear cada tipo cuando sea necesario en cada caso. En la Figura 1-7 se pueden ver algunas aplicaciones de cada uno de los dos tipos de visión, así como algunos ejemplos donde estaría bien que trabajasen en conjunto.

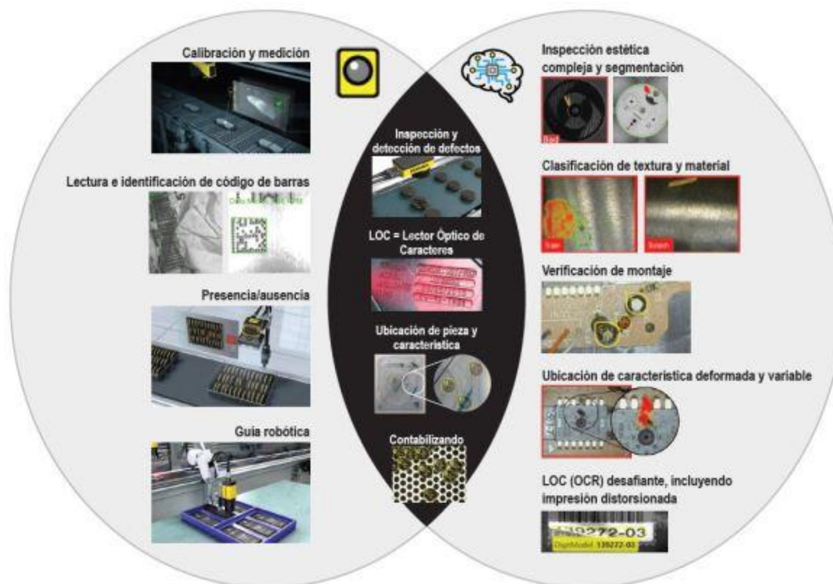


Figura 1-7: Usos de Visión Artificial clásica y con Deep Learning [12]

A la vista de las posibilidades que ofrece la dupla Deep Learning – Visión Artificial, se han podido hacer avances tecnológicos en campos como la medicina, la robótica o la automoción, donde se han podido crear aparatos que puedan ayudar tanto a profesionales como a usuarios a desarrollar tareas en el trabajo y en la vida cotidiana. Este impacto tan grande se verá reflejado a continuación al indagar en algunos de los sectores de mercado en la actualidad, especialmente en automoción y robótica, haciendo posible el desarrollo de vehículos autónomos o la manipulación de elementos delicados usando brazos mecanizados. Por eso, se ha creído necesario realizar este apartado antes de desarrollar cada sector de manera individual.

1.4.2 Sector financiero

Desde la crisis financiera de 2007, se ha dado una revolución digital en el sector financiero que, hoy en día, ha empezado ya a implementar técnicas de Inteligencia Artificial, que permiten incrementar los beneficios ofreciendo mejores servicios al usuario, según se dice en [13]. En la Figura 1-8 se muestran algunas de las tecnologías basadas en IA que se están empleando actualmente:

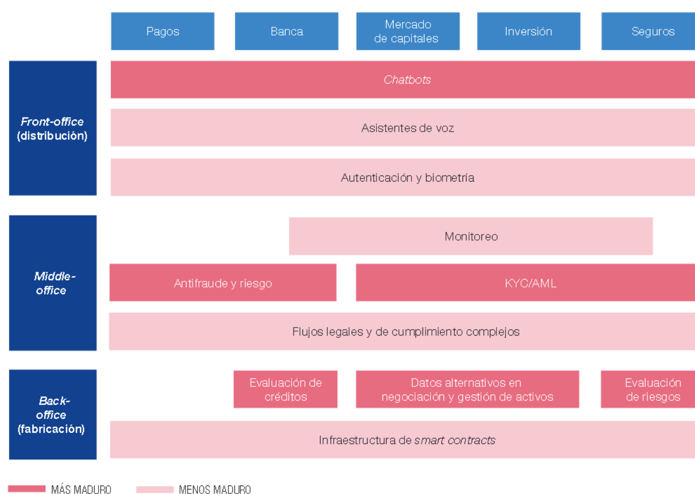


Figura 1-8: Tecnologías que aplican IA al sector financiero según su estado de maduración

Algunas de las aportaciones de la IA al sector financiero según el grupo financiero BBVA [14] son las siguientes:

- **Gestión de activos**: Plataformas de IA como **Kensho** o **Dataminr** usan algoritmos de inteligencia artificial para gestionar los activos financieros. Dataminr se especializa en detectar patrones a través de redes sociales, y Kensho es capaz de correlacionar noticias de actualidad con el mercado. De esta forma, sería posible predecir la evolución que tendrá un activo en el tiempo y actuar preventivamente para paliar posibles daños económicos, o bien actuar en consecuencia para sacarle mayor rentabilidad a dicho activo.
- **Detección de fraude y cumplimiento de la normativa**: Empresas como **Trifacta** y **Nice Actimize** emplean 'machine learning' e IA para aumentar la precisión de los métodos de detección de incumplimiento de normativas, ya que hasta ahora se cometían más fallos en la detección, provocando una mayor carga de trabajo en la entidad bancaria. Esto hace entonces que el dinero de los usuarios y de la propia entidad estén mejor protegidos ante fraudes, ciberataques, etc.
- **Atención al cliente**: Hay diferentes métodos de atención al cliente que emplean IA, como por ejemplo los chatbots o los roboadvisors. Los chatbots permiten contestar a los usuarios a sus preguntas como si de una persona se tratase, liberando tiempo y recursos para que los trabajadores puedan dedicárselo a otras áreas del sector financiero, según [15]. Los roboadvisors, en cambio, son asesores virtuales que aconsejan a los clientes y se encargan de la gestión de patrimonios, planificación financiera, y creación de carteras de activos de los clientes, según se indica en [13]. Casos reales de todo esto serían **Pefín** (*'Personale Finance Intelligence'*) y **Kasisto**, que emplean 'big data', 'machine learning' y una serie de algoritmos.

En definitiva, la IA ha provocado una gran revolución en este sector, cuyo progreso habrá que seguir muy de cerca durante los próximos años.

1.4.3 Sector cultural

En lo referente a este sector se pueden englobar áreas como el cine, música, plataformas de streaming, videojuegos, etc. Se verá a continuación como la IA está influyendo en dichas áreas, estando implantada en algunos casos desde hace mucho, como en el caso de los videojuegos.

Tal como se dice en [16], la IA en videojuegos actualmente ha evolucionado y ya no solo se emplea en los NPC's (*'Non Playable Characters'*), sino que se puede usar también para remasterizar videojuegos, o incluso crearlos desde cero. En el primero de los usos, apareció una herramienta desarrollada por **Ian Goodfellow** en el MIT, conocida como **GAN**, y que Nvidia adoptó para crear una IA llamada **StyleGAN**. En la Figura 1-9 se aprecia el efecto de dicha IA sobre el juego Final Fantasy 7, viendo como los gráficos mejoran considerablemente.

Figura 1-9: Escenarios originales (Izquierda) vs Escenarios remasterizarizados con IA (Derecha)

Otra de las zonas donde se empieza a usar la IA es en servicios de streaming, donde se puede disfrutar del visionado de películas, series, música, podcast, etc. Por ejemplo, la plataforma de Netflix es bastante famosa entre los usuarios, que tienen la dificultad de elegir algo que les pueda gustar de su enorme catálogo, pasando una mayor cantidad de tiempo escogiendo una película que viéndola.

Según el artículo [17], para solucionar este problema Netflix implementó un algoritmo que tiene en cuenta visionados previos, gustos personales, actores favoritos, etc., de modo que puede cambiar las portadas de los contenidos de la plataforma para ser más llamativa para el usuario. De este modo, si se han visto películas de ciencia ficción, Netflix cambiará las portadas de su contenido para que evoquen dicho género. Se ve en la Figura 1-10 un ejemplo de las diferentes portadas que adopta la serie 'Stranger Things' (2016) según la preferencia del usuario por un tipo de contenido u otro.

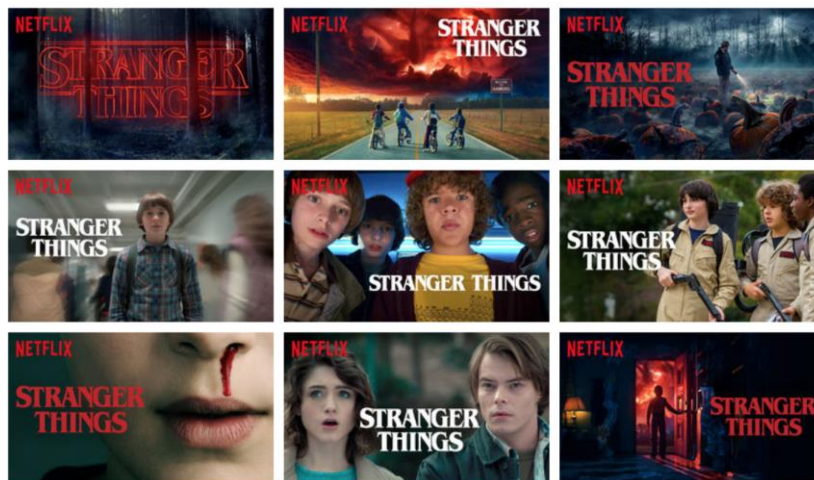


Figura 1-10: Personalización de portadas de Netflix

Figura 1-11: Recreación de Carrie Fisher como 'Leia Organa' para Rogue One: una historia de Star Wars (G. Edwards, 2016) y rejuvenecimineto de Kurt Russell como 'Ego' Guardianes de la Galaxia Vol. 2 (J. Gunn, 2017)

Finalmente, en el cine y la televisión, sistemas como la IA **Shapeshifter**, de Gradient Effects que, según se dice en [18], permite emplear la Visión Artificial para filmar escenas con los actores y que se modifique el aspecto de dicho actor de manera automática sin que nadie tenga que modelarlo en ordenador, por lo que abarata costes en la producción. Esto ha ocurrido recientemente en algunas superproducciones de Hollywood, como la saga Star Wars o el exitoso Universo Cinematográfico de Marvel, llegando a obtener resultados muy realistas, tal como se aprecia en la Figura 1-11[5].

Estas serían algunas de las aplicaciones de la IA a este sector, pero hay muchas más, por lo que también aquí está teniendo una creciente importancia su uso, llegando a revolucionar en el futuro toda la industria.

1.4.4 Sector sanitario

Este campo es uno de los más importantes en cuanto a IA se refiere, ya que abre un sinfín de posibilidades, siendo todas sus consecuencias mucho más directas que en las de los sectores anteriores, puesto que impacta directamente en la salud de las personas.

En el artículo [19], se dice que algunas de las aplicaciones se dan en farmacología, radiología u oncología, pudiendo realizar fármacos que tratan el ébola en unos días o detectar cáncer como el glaucoma con un análisis de retina y un cáncer de mama con biopsias y un 90% de acierto. También en neurología se usa para conocer mejor el cerebro y aplicar esos conocimientos a tratar a enfermedades como Parkinson o Alzheimer. En cirugía ya existen equipos que pueden realizar operaciones de manera autónoma, pero siempre con la supervisión de un profesional cualificado.

Uno de esos ejemplos es la IA de IBM conocida como Watson Health, que es capaz de detectar 13 tipos de cáncer gracias a su entrenamiento con millones de textos médicos. En España, en el Instituto de Física Corpuscular también están desarrollando tecnología que ayuda a los médicos a detectar cáncer de pecho con mamografías. Biobank es uno de los sistemas que facilita a los médicos el acceso a los expedientes médicos de los pacientes para poder hacer un buen diagnóstico y tratamiento de las enfermedades. En la Figura 1-12[6] se ve un esquema de algunas características de Watson Health, así como un esquema de Biobank.

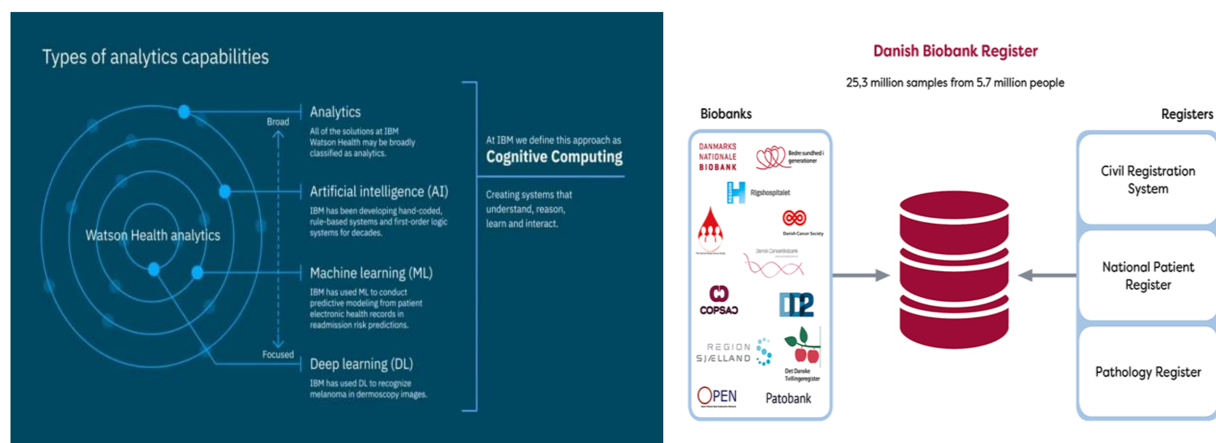


Figura 1-12: Watson Health de IBM y Biobank

Corti es una IA holandesa que, mediante el análisis de las llamadas de emergencia, es capaz de detectar pistas sobre un posible ataque al corazón por parte del interlocutor, tanto si él lo sabe como si no. Es capaz de leer entre líneas y fijarse en los detalles, informando de todo al personal de emergencia, agilizando enormemente el tratamiento de ese tipo de casos y salvando vidas [20]. En la Figura 1-13[7] se muestra en funcionamiento.

[5] Fuente: https://www.ayojon.mx/nota/20289.actores-que-fueron-digitalmente-rejuvenecidos-en-peliculas
[6] Fuentes: https://www.ibm.com/es-es/partnerworld/watson-health y https://www.danishnationalbiobank.com/danish-biobank-register
[7] Fuente: https://andro4all.com/2018/04/inteligencia-artificial-detecta-fallos-corazon

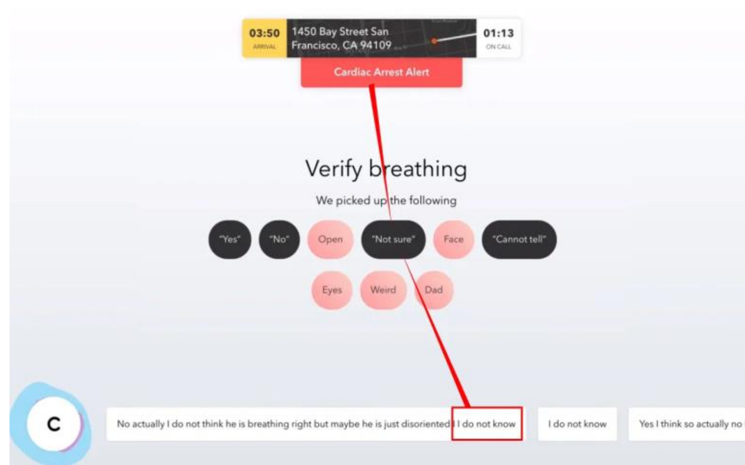


Figura 1-13: Funcionamiento de Corti en una llamada de emergencia

En definitiva, estas son algunas de la infinidad de aplicaciones y estudios actualmente que indican que la inteligencia artificial está revolucionando este sector, y seguirá haciéndolo en el futuro, permitiendo que se pierdan menos vidas humanas y mejore la calidad de vida considerablemente.

1.4.5 Sector automovilístico

Actualmente, hay muchísimos coches que integran asistentes virtuales entre sus servicios, conocidos como ADAS, de modo que ayudan al conductor a mejorar su experiencia de conducción aumentando la seguridad, ya que mediante técnicas de Machine Learning y Deep Learning puede aprender de sus hábitos y mediante el uso de Visión Artificial con una cámara y sensores podrían reconocerse situaciones de peligro e incluso actuar en consecuencia, como frenar cuando un peatón está cruzando la calle. Es el caso de Hyundai, que implementó un control de velocidad que aprende del conductor humano a la hora de acelerar y mantener la distancia de seguridad, de modo que al usarlo el usuario no lo sintiera muy diferente a como lo haría él mismo, tal como se dice en [21].

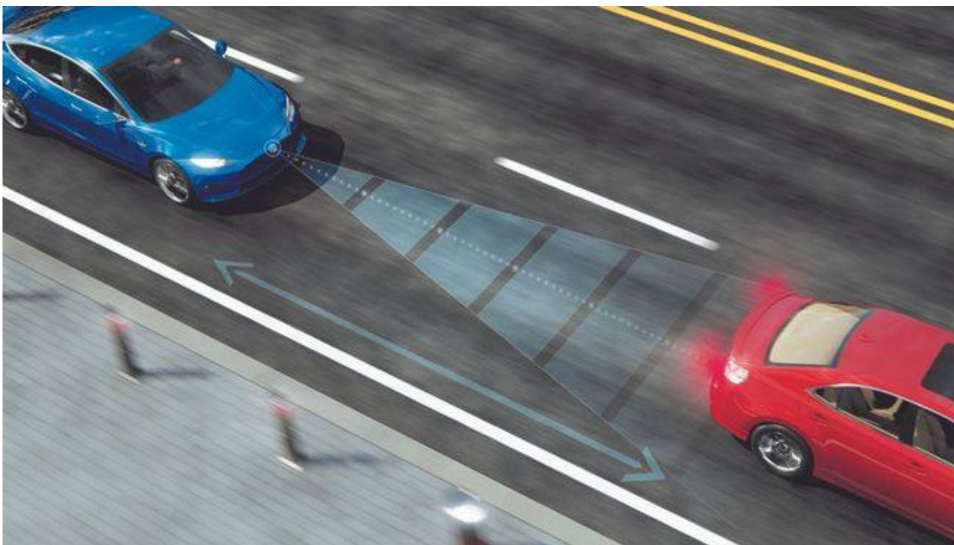


Figura 1-14: Sistema ADAS con IA de Hyundai que aprende del manejo del usuario [21]

Además, hay diseños experimentales con Deep Learning que son muy prometedores, como el que está desarrollando **Wayve**, cuyo objetivo es hacer que un coche conduzca autónomamente por una carretera sin salirse, cosa que consiguieron tras 12 intentos, con un humano corrigiendo el rumbo constantemente, y el coche lo consiguió aprender en aproximadamente 20 minutos. De esa forma, el coche podría conducir en la realidad usando GPS, cámaras, sensores, y su habilidad para aprender constantemente usando *feedback*.

Todo esto podría ser la antesala para, en un futuro, tener coches que conduzcan de manera autónoma, unificando los sistemas anteriores con otros sistemas como, por ejemplo, el sistema de detección de señales de tráfico tratado en este proyecto, y sistemas también de control automático, aunque al momento de redactar estas palabras aún no ha sido desarrollado un sistema de conducción autónoma total que sea aprobado por la comunidad científica, automovilística y la dirección de tráfico.

1.4.6 Sector de la robótica

La robótica abarca un ámbito enorme de la tecnología que no se limita a robots mecánicos o drones, sino que hay sistemas robóticos que ni siquiera tienen un 'cuerpo físico', ya que son totalmente software, como por ejemplo los asistentes virtuales o los chatbots. Realmente, la robótica se podría aplicar a cualquiera de los sectores anteriores ya sea en un más bajo o alto nivel. Por tanto, se puede tener un robot que funcione siguiendo las reglas indicadas previamente sin la capacidad de aprender sobre la marcha, pero es cuando se produce la unión entre IA y robótica que permite la apertura de muchas puertas nunca imaginadas, siendo una de las revoluciones producidas actualmente en este campo.

Tal como se cuenta en [22], uno de los usos más comunes de esta unión se da en el sector industrial, utilizándose en las cadenas de montaje para tareas tediosas para el humano, como por ejemplo el montaje y embalado del producto. Si para dichas tareas se aplican robots con IA, se ahorra tiempo y aumenta la eficacia y producción de la planta. Además, puede monitorizar en cierta medida todo el proceso, llegando a detectar fallas antes de que ocurra y realizar un mantenimiento previo a dicho evento.

Otro de los usos que ya se ha visto en sectores anteriores es el de los chatbots. Se usan cada vez más para resolver dudas del cliente, haciendo que el tiempo que un empleado invierte en eso se emplee en otras tareas, aumentando la productividad. También, al añadir a dicho chatbot técnicas de Machine Learning, se permite que aprenda y la conversación con las personas sea más fluida y realista, mejorando la experiencia del usuario. Sistemas de este tipo de interacción con el usuario son normalmente formados por software únicamente, aunque hay prototipos que emplean esta tecnología para tener un sistema de atención al público funcional con un hardware con el que el usuario puede interactuar. Es el caso del robot **Mulán**, que trabaja como camarero en España, en la comunidad valenciana, en la Figura 1-15[8]:

Figura 1-15: Robot Mulán trabajando en el restaurante Crensa de Valencia

[8] Fuente:https://www.abc.es/tecnologia/abci-mulan-camarera-robot-valenciana-llama-carino-cliente-restaurante-201908280108_noticia.html

La IA también se puede aplicar en la robótica a los llamados 'cobots', que hacen referencia a los robots colaborativos. Dichos robots pueden ponerse de acuerdo para ayudarse el uno al otro a realizar una tarea concreta. Puede haber diferentes tipos según si la tarea que desempeñan es más o menos pesada de realizar, por lo que serían también muy útiles en las cadenas de montaje para realizar operaciones que requieran gran sincronización por parte de varios robots. Un ejemplo de dicha tecnología es la resolución de un cubo de Rubik por parte de dos brazos robóticos UR3 comunicados con IA, en la Figura 1-16[9]:

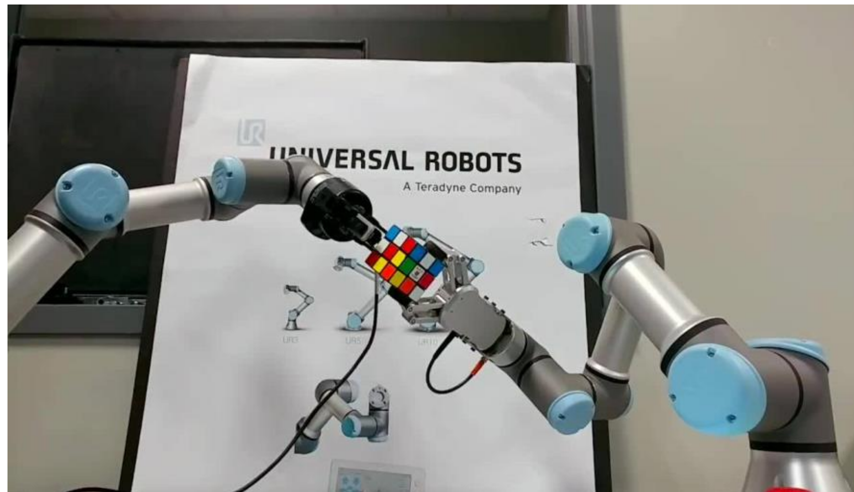


Figura 1-16: Brazos robóticos UR3 resolviendo un cubo de Rubik

Otra de las áreas de investigación relacionadas con lo anterior es la del procesamiento del lenguaje natural, o NLP, que permite que dispositivos diseñados comprendan el lenguaje usado por las personas. Esto haría que la comunicación hombre-máquina fuera muy fluida, hasta el punto de hablar con las máquinas prácticamente como si de humanos se tratase. Se suele usar en el ámbito de procesamiento de textos.

Según la publicación [23], relacionado con lo anterior ha aparecido un modelo de lenguaje conocido como GPT-3, de OpenAI. Básicamente, es como un predictor de texto, pero amplificado mil veces, obteniendo resultados de una magnitud asombrosa. GPT-3 se ha entrenado con prácticamente toda la fuente de conocimiento humano que se encuentra en la red, y tiene un tamaño de 700 GB ubicados en 48 GPU's de 16 GB cada una, resultando en una red de 175.000 millones de parámetros, muy superior al de su predecesor el GPT-2, de 40 GB y 1.500 millones de parámetros.

Se han realizado muchos experimentos, y sus resultados van desde la identificación de una simple búsqueda por internet, hasta la creación de manera automática de una aplicación o una página web con su respectivo código en HTML. Es aquí donde radica su potencia y todas sus posibilidades, ya que está más cerca de comprender a los humanos de una forma casi natural que cualquier otro sistema, llegando incluso a hacer tareas como crear webs que a un programador experto le llevarían algún tiempo, prácticamente de manera automática.

Hay muchísimas otras áreas en las que se emplea esta dupla de robótica e IA, y seguirán apareciendo muchas más, porque es capaz de abarcar la gran mayoría de sectores de la economía mundial. La robótica se está volviendo un imprescindible hoy en día en muchísimas empresas, y más aún al combinarla con la Inteligencia Artificial.

1.5 Comentarios finales

Se han dado unas breves nociones de historia de las redes neuronales para poder saber de dónde viene y desde cuándo se está desarrollando esta tecnología, objeto de estudio en este documento, así como la evolución de los componentes hardware que hacen posible su implementación, enumerando sus diferentes características.

[9] Fuente: https://blog.universal-robots.com/how-to-get-robots-to-talk-to-each-other

También se ha hecho un barrido general de la influencia de las técnicas de inteligencia artificial en el panorama actual de algunos de los sectores más importantes en la actualidad, desde el sector financiero hasta el sector de la robótica que, en conjunto con la Inteligencia Artificial, se puede usar prácticamente en cualquiera de los otros sectores, además de analizar el impacto de las redes neuronales en la Visión Artificial.

En el capítulo 2 se profundizará en el panorama actual de las redes convolucionales, introduciendo también algunos ejemplos de aplicación al campo de detección de señales de tráfico, de modo que se continúe en el capítulo 3 con la metodología usada para elaborar las redes que resuelven el problema de la clasificación de señales de tráfico, presentando los resultados obtenidos en el capítulo 4, para terminar en el capítulo 5 hablando de las conclusiones y posibles proyectos futuros en los que se podrían aplicar estas redes y los conocimientos adquiridos.

2 ESTADO DEL ARTE

Donde no hay imaginación, no hay horror.

- Sir Arthur Conan Doyle -

En este capítulo se profundizará en los diferentes tipos de redes neuronales que existen en la actualidad, usando ejemplos concretos de algunas de las más influyentes en su historia reciente. Servirá para tener una concepción general del panorama actual en el que se encuentra este tipo de tecnología y comprender mejor la red que se usará para resolver el problema concreto que se aborda en este TFG, que se explicará en el próximo capítulo.

2.1 Algunas arquitecturas famosas de redes convolucionales

En esta sección se enunciarán brevemente algunas de las redes neuronales convolucionales más famosas a lo largo de los últimos años. Las redes convolucionales son las que se usarán en la resolución del problema de las señales de tráfico, y son redes que tienen diferentes capas que facilitan el trabajo con imágenes directamente, por lo que son muy usadas en el campo de visión por computador. A las imágenes se le aplica un filtro (kernel) en cada capa, de modo que irá guardando un mapa de características de la imagen. Normalmente, se usan en conjunto con las redes densamente conectadas, o fully-connected.

Para elaborar esta sección, se ha usado como referencia [24].

2.1.1 LeNet-5

Esta red es considerada el primer éxito real en la aplicación de redes convolucionales, ya que logró clasificar con un acierto de un 99.2% los caracteres escritos a mano del conjunto MNIST. Fue desarrollada por Yann LeCun, et al. (1998) en su documento *"Gradient-Based Learning Applied to Document Recognition"* [25], y su arquitectura se muestra en la Figura 2-1:

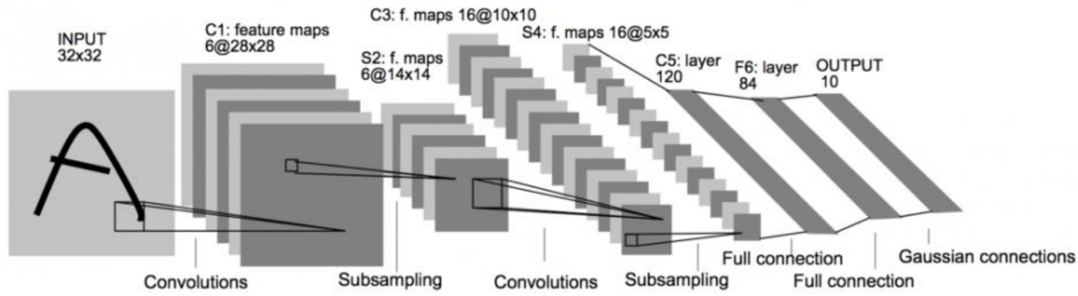


Figura 2-1: Arquitectura de LeNet-5 [25]

Las imágenes que entran a la red son de 32x32 píxeles y en blanco y negro, por lo que los diferentes kernel de las capas convolucionales serán también de las mismas dimensiones, aunque de distinto tamaño. La primera capa es convolucional, de nombre C1 en la imagen, y consta de 6 kernels de 28x28 píxeles, procedentes de realizar una convolución de 5x5 a la imagen de entrada.

La segunda capa de nombre S2 es de submuestreo, o pooling, de modo que reduce el mapa de características de la capa anterior conservando toda la información posible para agilizar la computación de la red. Por ello, el número de kernels es también 6 como el de la capa convolucional a la que se asocia, y ahora el tamaño de dichos kernels se ve reducido a 14x14 al realizar un submuestreo de ventana 2x2 píxeles.

La tercera capa C3 es convolucional, y consta de 16 kernels de 10x10 cada uno al aplicar una convolución de tamaño 5x5 sobre los mapas de características de la capa S2, pero hay una diferencia en estas conexiones respecto a las demás que hay en la red, y es que los mapas de características de S2 y C3 no se obtienen con conexiones de todas sus capas entre sí como el resto. La razón por la que la capa C3 no está conectada con todas las del pooling de la capa S2, se debe a que así se disminuye el número de conexiones a unos valores con límites más razonables para la computación, y además se rompe la simetría de la red, obteniendo diferentes mapas de características en cada kernel al estar conectadas a diferentes entradas. Las conexiones entre dichas capas se pueden ver en la Figura 2-2:

Kernels de C3

Kernels de S2	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	X				X	X	X			X	X	X	X		X	X
1	X	X				X	X	X			X	X	X	X		X
2	X	X	X				X	X	X			X		X	X	X
3		X	X	X			X	X	X	X			X		X	X
4			X	X	X			X	X	X	X		X	X		X
5				X	X	X			X	X	X	X		X	X	X

Conecta con todos los kernels de S2

Conectan con kernels correlativos 3 a 3 | Conectan con kernels correlativos 4 a 4 | Conectan con 4 kernels no correlativos

Figura 2-2: Conexiones entre capas S2 y C3 de LeNet-5 [25]

Tras ésta, la siguiente capa S4 vuelve a ser de pooling, por lo que tendrá también 16 kernels, pero reduciendo el tamaño de 10x10 a 5x5 píxeles al realizar un submuestreo de ventana de 2x2 píxeles. La capa C5 vuelve a ser convolucional, con la particularidad de que ahora el tamaño de sus 120 kernels será de 1x1 píxel. Es decir, esta

capa actúa como "traductor" para poder usar a continuación las capas fully-connected F6 y la capa de salida que dará la probabilidad de clasificación de la imagen en una clase u otra. Esto se hace porque las fully- connected no pueden trabajar con imágenes, es decir, matrices, de la misma manera que las convolucionales, por lo que se traduce el mapa de características a un vector que contiene la información más relevante que han ido sacando todas las capas anteriores, agrupándolas en un vector de características.

Por tanto, lo que se saca en claro es que su estructura se basa en el encadenamiento de bloques de capas convolucionales con capas de pooling para finalizar con el uso de capas fully-connected una vez que termina la etapa convolucional, elemento que sigue siendo muy usado en la actualidad, siendo otra característica el aumento de kernels a medida que se profundiza en la estructura de la red.

2.1.2 AlexNet

Recibe ese nombre en honor a su creador, Alex Krizhevsky, logrando ganar la competición *ImageNet Large Scale Visual Recognition Challenge* del año 2012. El avance que consiguió con este diseño [26] fue muy importante, ya que demostró que las características aprendidas superan a las diseñadas de forma manual como se hacía anteriormente, iniciando una revolución en el campo de visión por computador.

La arquitectura de AlexNet se compone de unas 8 capas en total, siendo las 5 primeras capas convolucionales, las 2 siguientes capas ocultas de una red fully-connected, y la última capa es la capa de salida de la fully-connected. Las capas convolucionales son dispares entre sí, ya que las 2 primeras capas están seguidas de una capa de pooling para minimizar la matriz de características, mientras que las 3 últimas capas convolucionales están una detrás de otra, y es en la última capa cuando se aplica el pooling. En este caso, las etapas de pooling son todas de tipo "maxpooling", que consiste en aplicar una máscara a una sección del mapa de características y quedarse con el mayor de los números de cada celda contenida en dicha ventana, a diferencia que en el modelo anterior LeNet-5, que usaba la media de los valores de los píxeles como pooling.

Hay que comentar también que los datos de entrada, es decir, las imágenes, son en color, luego los kernels de cada capa ya no serán individuales, sino que se compondrán cada uno de 3 matrices de las mismas dimensiones, una correspondiente a cada color, por lo que también puede aprender características de los colores de la imagen. Además, la capa de salida hace uso de la función Softmax, elemento que se ha vuelto casi indispensable de usar en este tipo de redes neuronales que tienen que clasificar varias categorías diferentes

Otra característica de esta red es que usa RELU en lugar de sigmoide como función de activación, a diferencia de la red más parecida a ésta en aquella época, que era LeNet5. El uso de la RELU permite que cada neurona tenga un comportamiento nulo o lineal a partir de cierto valor umbral, de modo que se agregan no linealidades que ayudan al proceso de aprendizaje a retener más conocimientos y a solventar el problema de gradiente que tienen las funciones sigmoides si están mal inicializadas, obteniendo un gradiente casi nulo con salidas de la función cercanas a 0 o 1, a diferencia de la RELU, que en el intervalo de activación positivo tiene gradiente 1. Esto también posibilitó el uso de la técnica del Dropout de neuronas. En la Figura 2-3 se observa la red AlexNet (derecha) respecto a LeNet (izquierda):

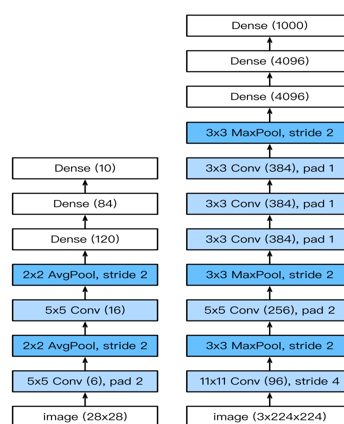


Figura 2-3: Comparación entre arquitectura LeNet (izquierda) y AlexNet (derecha) [24]

Sería interesante comentar que en la época en la que se desarrolló, la capacidad de cómputo de las GPU's no era suficiente para entrenar toda la red a la vez, por lo que se dividió en 2 partes y cada una era procesada por una GPU para su entrenamiento y desarrollo del modelo. Ya que esto no está representado en la imagen anterior donde se compara con LeNet-5, un esquema de esta configuración primigenia, que hoy en día ya no tiene esa limitación debido al avance en el procesamiento de datos, se puede observar en la Figura 2-4:

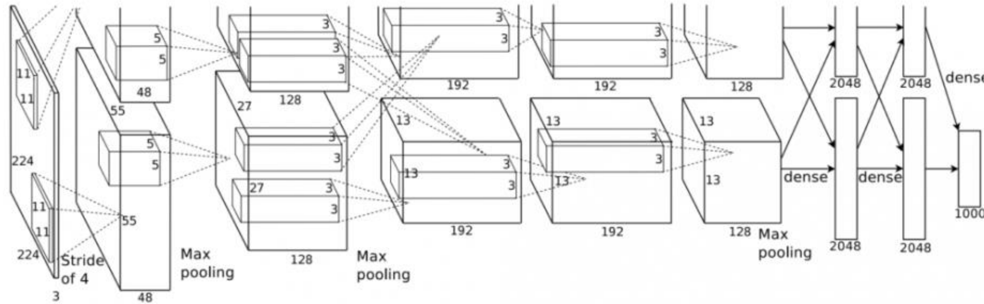


Figura 2-4: Arquitectura AlexNet dividida para poder ser entrenada con GPU en 2012 [26]

Por tanto, esta arquitectura marcó la pauta a seguir para todas las demás, usando estructuras como la de poner varias capas convolucionales de manera consecutiva sin pooling entre ellas, el uso de la ReLU a las salidas de las capas, la función Softmax para la capa de salida o el uso de la técnica de Dropout para desconectar neuronas de manera aleatoria en distintas iteraciones del entrenamiento aportando no linealidades y dando riqueza a los conocimientos obtenidos por la red, además de producirse un sustancial aumento de datos y parámetros a entrenar respecto a la versión LeNet-5, apreciándose en el tamaño de la imagen inicial, que antes era de 32x32 en blanco y negro y en AlexNet es de 224x224 píxeles y en color.

2.1.3 Visual Geometry Group (VGG)

Esta red es posterior a AlexNet y surgió en la Universidad de Oxford en 2014, siendo elaborada por Karen Simonyan y Andrew Zisserman, del Visual Geometry Group, por el que esta red recibe su nombre. Es de las primeras redes en emplear bloques de capas ya preestablecidos, normalmente compuestas de una o más capas convolucionales consecutivas, para finalizar dicho bloque con una etapa de pooling.

Con esta arquitectura hay varias diferentes según el número de capas añadidas, teniendo incluso una de ellas la misma estructura que la red AlexNet anterior, como se ve en la Figura 2-5:

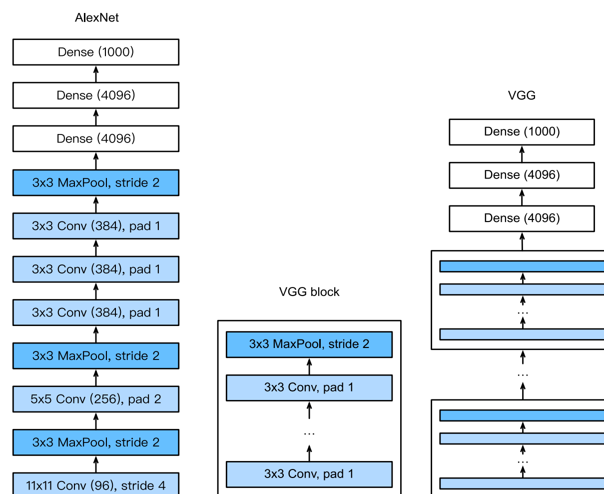


Figura 2-5: AlexNet usando bloques VGG [24]

Las más conocidas son la VGG-16 y la VGG-19, indicando dichos números el número de capas que tiene cada arquitectura. En el documento [27] los creadores de VGG detallan en la Tabla 2-1 la arquitectura de cada una de las redes VGG según el número de capas de cada una:

Tabla 2-1: Redes VGG según el número de capas

ConvNet Configuration					
A	A-LRN	B	C	D	E
11 weight layers	11 weight layers	13 weight layers	16 weight layers	16 weight layers	19 weight layers
input (224 × 224 RGB image)					
conv3-64	conv3-64 **LRN**	conv3-64 **conv3-64**	conv3-64 conv3-64	conv3-64 conv3-64	conv3-64 conv3-64
maxpool					
conv3-128	conv3-128	conv3-128 **conv3-128**	conv3-128 conv3-128	conv3-128 conv3-128	conv3-128 conv3-128
maxpool					
conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256 **conv1-256**	conv3-256 conv3-256 **conv3-256**	conv3-256 conv3-256 conv3-256 **conv3-256**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
FC-4096					
FC-4096					
FC-1000					
soft-max					

De lo anterior se pueden sacar dos conclusiones: el uso de una mayor cantidad de kernel a medida que se recorre la red de inicio a fin, llegando a alcanzar valores de 64,128,256 y 512, y el uso de kernels de menor tamaño en estos bloques VGG respecto a las redes anteriores. Esto último se debe a que al usar un conjunto de varias capas convolucionales consecutivas con filtros de menor tamaño se puede aproximar el resultado al mismo que se podría obtener con una sola capa convolucional con un filtro de mayor tamaño. Por eso era posible desarrollar la red AlexNet con bloques VGG, ya que en la Figura 2-5 se apreciaba que el tamaño de los filtros de las capas convolucionales no corresponde al de la estructura AlexNet, sin embargo, al combinar varias se obtiene el equivalente en VGG.

2.1.4 Network in Network (NiN)

La red conocida como Network in Network [28] fue desarrollada con la idea de usar un perceptrón multicapa para cada píxel de cada canal de forma independiente. La idea es aplicar entonces una especie de "etapa fully-connected" a cada píxel de cada canal. De esta forma, al hacer depender los pesos de la red de cada neurona de un píxel en concreto, se puede ver como que se tiene una capa convolucional de 1x1 por cada píxel que actúa como una fully-connected independiente de los demás píxeles.

El bloque NiN está formado en primer lugar por una capa convolucional de dimensiones variables, normalmente elegida por el diseñador de la red según le convenga en cada caso, y a esta capa le siguen otras dos convolucionales consecutivas, ambas de tamaño 1x1, aportando el efecto de una fully-connected por píxel ya comentado. Para dichas capas la función de activación usada es la ReLU.

El modelo original constaba de 4 bloques NiN con una selección de tamaño de la primera red convolucional de cada bloque de 11x11, 5x5, 3x3 y 3x3 respectivamente. Se nota su inspiración en AlexNet, puesto que tienen similitudes como el número de canales de salida que dicha red, y también se acomoda una capa de pooling de tipo "maxpooling" de 3x3 justo después de cada bloque NiN. Sin embargo, esto no es así en la última etapa de la red, donde no usa las etapas fully-connected de AlexNet, sino que sigue usando bloques NiN seguido ahora de una etapa de polling de tipo "global average polling", es decir, que hace la media en lugar de quedarse con el máximo valor.

Al diseñar con NiN, se eliminan por tanto capas densamente conectadas, reduciendo el número de parámetros y la posibilidad de sobreajuste, aunque en la realidad algunas veces se tarda más en entrenar el modelo usando

esta arquitectura, por lo que es una desventaja bastante importante.

A continuación, se muestra la Figura 2-6 donde se pueden ver todas estas diferencias y similitudes comentadas entre las redes AlexNet, VGG y la red NiN, en su forma de bloque individual y en la aplicación de dichos bloques para formar el modelo clásico de la red NiN:

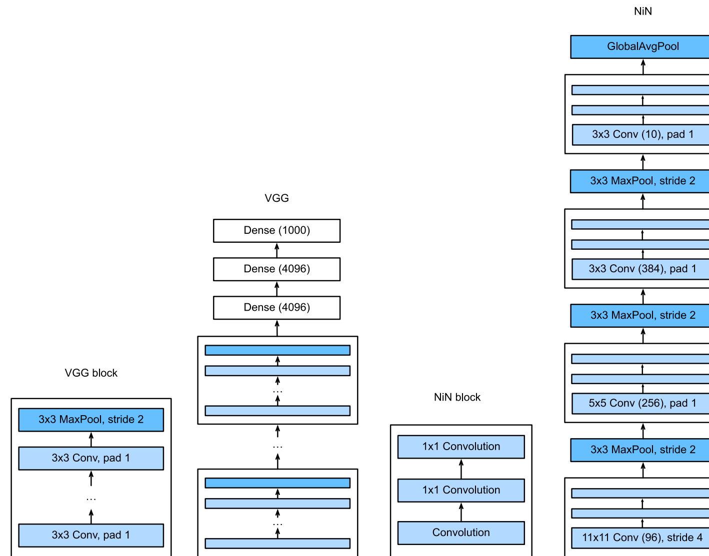


Figura 2-6: Comparación de AlexNet, VGG y NiN [24]

2.1.5 GoogLeNet

Esta red surgió en 2015 [29] y ganó el concurso ImageNet que ya ganó la red Alexnet en 2012. Lo que hace a esta red tan especial es que combina todo lo bueno de las redes anteriores. También se basa en bloques, como la NiN o la VGG, y su bloque principal se llama Inception. La estructura de este bloque es algo peculiar, por lo que se detallará a continuación:

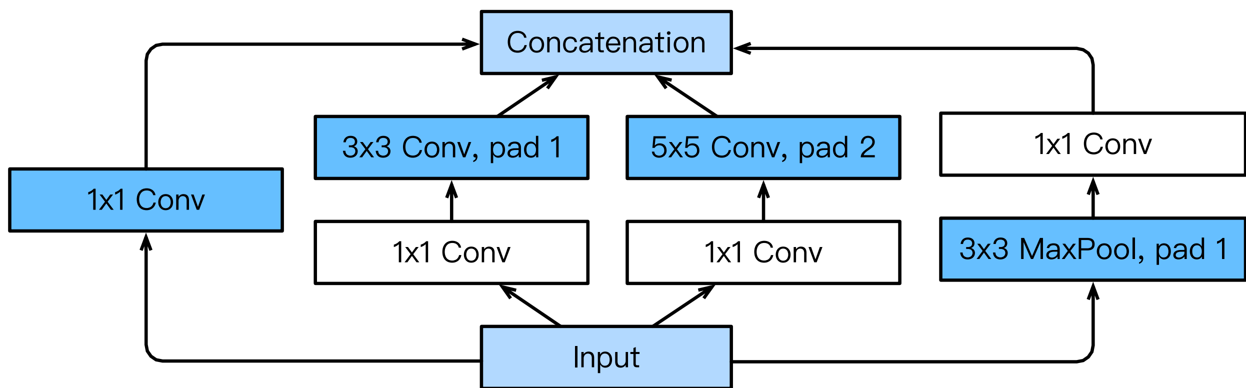


Figura 2-7: Arquitectura del bloque Inception de GoogLeNet [24]

Como se observa en la Figura 2-7, tiene 4 ramas:

- Una rama que hace una convolución de 1x1 reduciendo el número de canales como ya se vio en la NiN.
- Otras dos ramas que hacen primero una convolución de 1x1 para reducir el número de canales, y con ello la complejidad del modelo, y luego una convolución de 3x3 y de 5x5 respectivamente.
- Una última rama que hace primero un pooling de tamaño 3x3 a la entrada del bloque, y luego una

convolución de 1x1 con el mismo fin que en las ramas anteriores.

Una vez que las 4 ramas se han ejecutado de forma paralela y obtenido una salida, se concatenan todas en una única salida. Con esto, lo que se ha conseguido, es explorar las características de la imagen a diferentes escalas, obteniendo un nivel de precisión y detalle en el entrenamiento mucho mejor que en las anteriores, ya que cada rama se encargará de reconocer unas características en concreto.

Por último, en la Figura 2-8 se muestra el modelo de GoogLeNet usando el bloque anteriormente explicado:

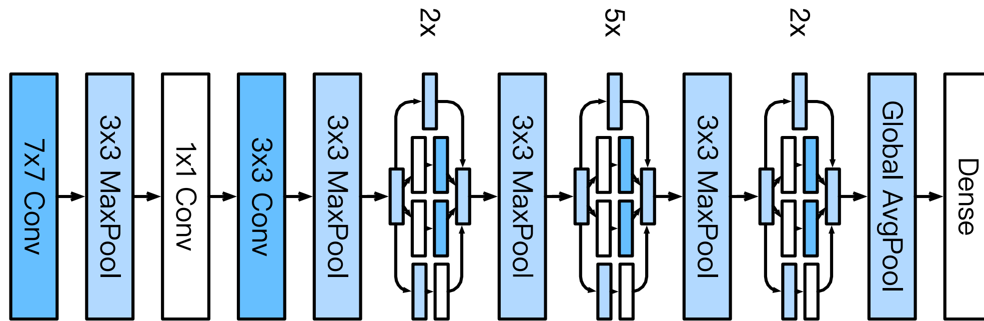


Figura 2-8: Arquitectura de la red GoogLeNet usando el bloque Inception [24]

En primer lugar, se tiene una estructura de capa convolucional de 7x7de 64 canales seguida de una capa pooling de tipo “maxpooling” de 3x3, y terminando con una capa convolucional de 1x1 para reducir la complejidad del modelo al reducir el número de canales. Después, le sigue otra estructura clásica de capa convolucional y pooling de tipo “maxpooling”, ambas de tamaño 3x3, y a continuación se apilan 2 bloques “Inception” con una capa de “maxpooling” entre los bloques 2 y 3, y los bloques 7 y 8. Esto se hace para reducir la dimensionalidad. Finalmente, de manera similar a algunos modelos anteriores, hace uso de una capa de pooling de tipo media global.

Se añade la Tabla 2-2 perteneciente al documento [29] donde viene detallada cada capa de la arquitectura:

Tabla 2-2: Arquitectura GoogleNet con bloques Inception

type	patch size/ stride	output size	depth	#1×1	#3×3 reduce	#3×3	#5×5 reduce	#5×5	pool proj	params	ops
convolution	7×7/2	112×112×64	1							2.7K	34M
max pool	3×3/2	56×56×64	0								
convolution	3×3/1	56×56×192	2		64	192				112K	360M
max pool	3×3/2	28×28×192	0								
inception (3a)		28×28×256	2	64	96	128	16	32	32	159K	128M
inception (3b)		28×28×480	2	128	128	192	32	96	64	380K	304M
max pool	3×3/2	14×14×480	0								
inception (4a)		14×14×512	2	192	96	208	16	48	64	364K	73M
inception (4b)		14×14×512	2	160	112	224	24	64	64	437K	88M
inception (4c)		14×14×512	2	128	128	256	24	64	64	463K	100M
inception (4d)		14×14×528	2	112	144	288	32	64	64	580K	119M
inception (4e)		14×14×832	2	256	160	320	32	128	128	840K	170M
max pool	3×3/2	7×7×832	0								
inception (5a)		7×7×832	2	256	160	320	32	128	128	1072K	54M
inception (5b)		7×7×1024	2	384	192	384	48	128	128	1388K	71M
avg pool	7×7/1	1×1×1024	0								
dropout (40%)		1×1×1024	0								
linear		1×1×1000	1							1000K	1M
softmax		1×1×1000	0								

En el modelo original, se hizo uso de unas redes de salida desde la red principal que pretendían dar solución al problema de desaparición de gradientes, aportando una medida de error adicional, de modo que ya sí que se pudo conectar la salida en diferentes puntos del modelo sin problema. Estas redes se eliminaron del modelo una vez que se entrenó, por eso no aparecen en la imagen anterior. El modelo completo con dichas redes se muestra en la Figura 2-9, teniendo la entrada a la izquierda:

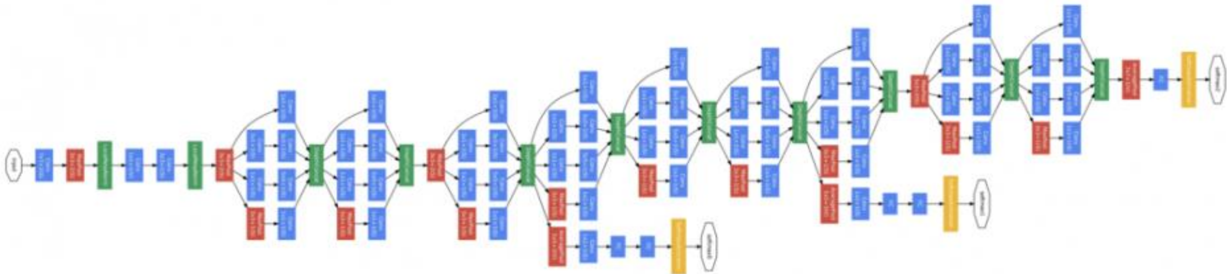


Figura 2-9: Modelo GoogLeNet usado para el entrenamiento en la ILSVRC de 2014 [29]

Como claves de este modelo se pueden destacar el uso de la convolución de 1x1 para reducir el número de canales, la gran profundidad de la red llegando a alcanzar el número de 22 capas, el uso de redes pequeñas en el entrenamiento que le sirven como refuerzo del error, y el desarrollo del módulo Inception, que usa repetidamente y que es capaz de detectar características en diferentes franjas gracias a sus 4 ramas en paralelo.

2.1.6 Residual Networks (ResNet)

Se ha llegado a un punto en el que las redes neuronales pueden ser muy grandes y complejas al añadir capas, pero se aumenta la expresividad de la red. Es decir, al añadir más capas, se mejoran los resultados del entrenamiento. El problema es que, al añadir muchas capas, es posible que se cambie el funcionamiento del modelo de manera imperceptible a simple vista, obteniendo peores resultados y haciendo que la arquitectura de la red no alcance la función matemática deseada. Para que aumente la expresividad añadiendo capas sin cambiar el funcionamiento de la red, al añadir una nueva capa y entrenarla con una función tal que dicha capa tenga como salida la misma entrada, significará que la nueva red formada al añadir dicha capa es tan buena como el modelo anterior, pero mejorando los errores de entrenamiento.

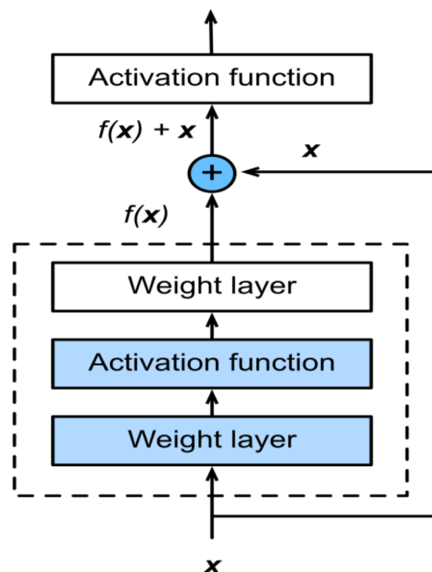


Figura 2-10: Bloque Residual de ResNet [24]

Con lo anterior, se desarrollaron los "Bloques Residuales", que formarían la red ResNet [30], que ganó el concurso de ImageNet ILSVRC en 2015. La idea es conseguir que cada capa, entrene solo la desviación de la identidad del valor de entrada, de modo que sea la suma de la entrada "x" y la salida de dicha capa "f(x)" la que se le pase a la función de activación, por ejemplo, una ReLU. Para sumarlas, la entrada y la salida deben tener las mismas dimensiones. El bloque de ResNet se aprecia en la Figura 2-10.

Una vez explicado lo anterior, el proceso de diseño de ResNet pasó por 2 etapas, ya que primero se realizó una red plana que se inspiró en la red VGG anteriormente comentada, haciendo que la mayoría de las capas convolucionales fueran de 3x3, y el número de kernels aumentara también en las mismas proporciones que VGG a medida que se profundizaba en la red. Esto último, sin embargo, no es del todo cierto ya que se siguieron 2 reglas para elegir el número de kernels o filtros por capas y es que, para el mismo tamaño del mapa de características de salida, el número de kernels se mantiene igual, pero en el momento en que dicho tamaño se reduce a la mitad el número de kernels por capa se duplica, conservando la complejidad por capa de la red. Este modelo acaba con una capa de pooling de tipo "global average pooling" y una fully-connected de 1000 neuronas como capa de salida, usando para ello la función Softmax. Esta red tiene menos filtros y menos complejidad que las redes VGG puesto que, con 34 capas, tiene 3.6 billones de parámetros frente a los 19.6 billones de VGG.

Al diseño anterior se le añadieron entre las capas convolucionales los Bloques Residuales ya explicados y, por tanto, se obtuvo la red ResNet. Más tarde, esta red introdujo un mayor número de capas, habiendo diferentes versiones de la arquitectura ResNet, llegando incluso a tener 152 capas y 11.3 billones de parámetros. En el documento [30], se aporta la comparación de la arquitectura VGG-19, la Plain Network comentada como primera versión de ResNet y la versión ResNet de 34 capas en la Figura 2-11, así como la Tabla 2-3 con la arquitectura detallada de las diferentes versiones de ResNet según el número de capas introducidas:

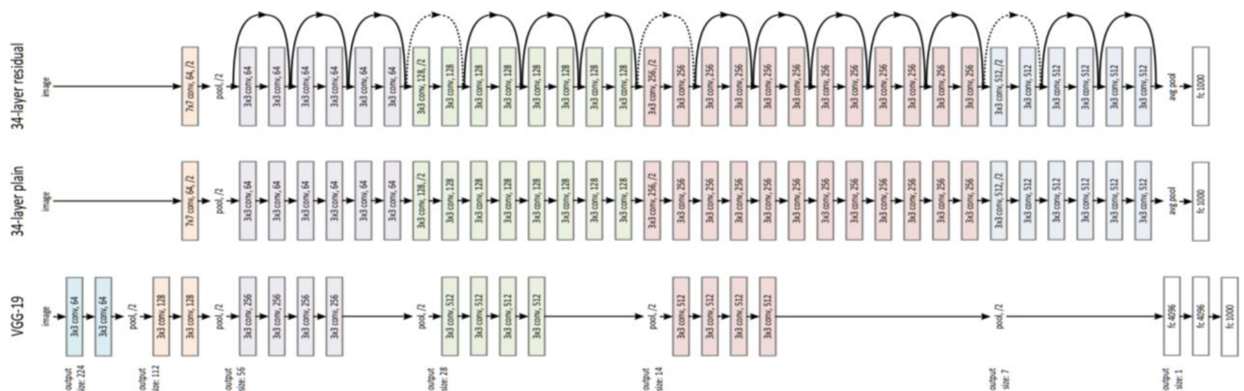


Figura 2-11: Comparación entre VGG-19, PlainNet y ResNet de 34 capas

Tabla 2-3: Arquitecturas de ResNet

layer name	output size	18-layer	34-layer	50-layer	101-layer	152-layer
conv1	112×112	7×7, 64, stride 2				
conv2_x	56×56	3×3 max pool, stride 2				
		[3×3, 64 3×3, 64] ×2	[3×3, 64 3×3, 64] ×3	[1×1, 64 3×3, 64 1×1, 256] ×3	[1×1, 64 3×3, 64 1×1, 256] ×3	[1×1, 64 3×3, 64 1×1, 256] ×3
conv3_x	28×28	[3×3, 128 3×3, 128] ×2	[3×3, 128 3×3, 128] ×4	[1×1, 128 3×3, 128 1×1, 512] ×4	[1×1, 128 3×3, 128 1×1, 512] ×4	[1×1, 128 3×3, 128 1×1, 512] ×8
conv4_x	14×14	[3×3, 256 3×3, 256] ×2	[3×3, 256 3×3, 256] ×6	[1×1, 256 3×3, 256 1×1, 1024] ×6	[1×1, 256 3×3, 256 1×1, 1024] ×23	[1×1, 256 3×3, 256 1×1, 1024] ×36
conv5_x	7×7	[3×3, 512 3×3, 512] ×2	[3×3, 512 3×3, 512] ×3	[1×1, 512 3×3, 512 1×1, 2048] ×3	[1×1, 512 3×3, 512 1×1, 2048] ×3	[1×1, 512 3×3, 512 1×1, 2048] ×3
	1×1	average pool, 1000-d fc, softmax				
FLOPs		1.8×10^9	3.6×10^9	3.8×10^9	7.6×10^9	11.3×10^9

Esta arquitectura ha tenido bastante influencia en las redes que vinieron después, pues el uso de los Residual

Blocks permitió agrandar la profundidad de la red, permitiendo unos mejores resultados sin acarrear los problemas propios de la adición de un mayor número de capas.

2.1.7 DenseNet

Propuesta en 2017 [31], es la evolución natural de ResNet. Consiguió un rendimiento récord en ImageNet y se basa en la idea de concatenar las salidas de cada capa con la entrada, en lugar de hacer la suma y pasarla a la función de activación como con ResNet. Esto permite aplicarle a la entrada cada vez funciones más complejas. Gráficamente, se ve la diferencia en la Figura 2-12:

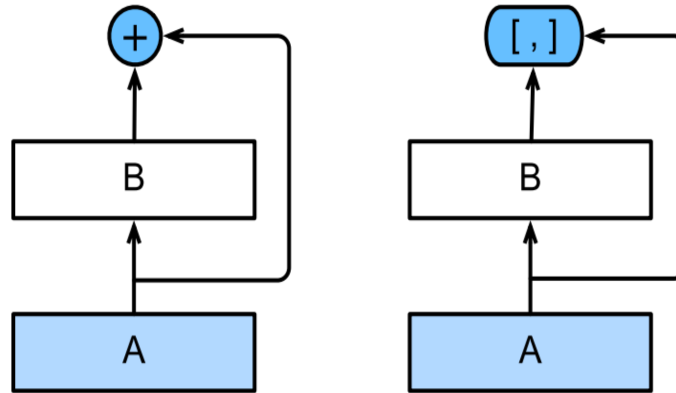


Figura 2-12: Diferencia entre ResNet (izquierda) y DenseNet (derecha) [24]

Una vez que se han concatenado todas esas funciones de capa y se ha llegado a la última, se usa un perceptrón multicapa para unirlas. Se compone de bloques densos que indican la forma de concatenar la entrada y la salida junto con capas de transición que controlan el número de canales para que la salida no sea muy grande. Finalmente, la última capa estará conectada con todas las demás, como se ilustra en la Figura 2-13:

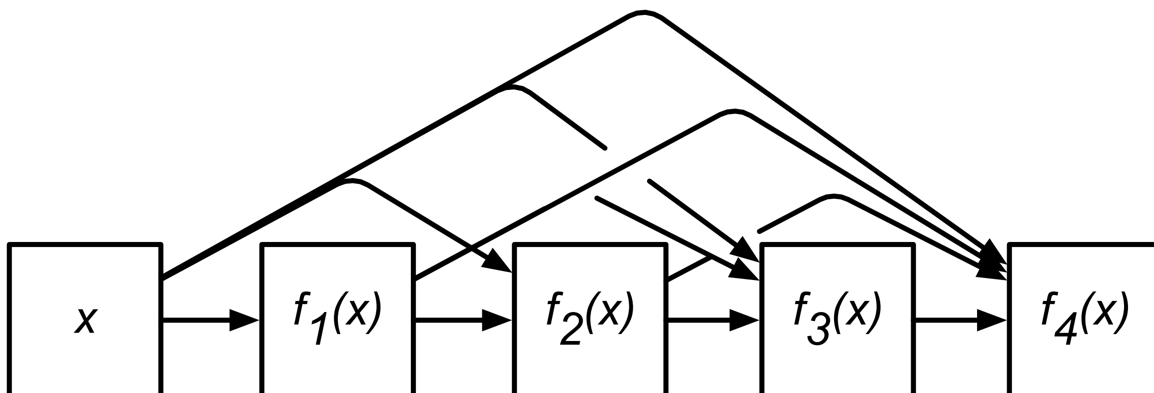


Figura 2-13: Conexiones densas en DenseNet [24]

Fundamentalmente, la DenseNet está formada por unos bloques llamados "Dense Blocks" que realizan la función principal que se acaba de explicar, y por unas capas llamadas capas de transición, que permiten controlar la dimensionalidad de la red, que crece mucho al introducir los Dense Blocks, haciendo así que el número de canales no sea desproporcionadamente grande. Esta característica se representa con un hiperparámetro conocido como tasa de crecimiento *k* (growth rate).

Las capas de transición de los experimentos originales solían ser una consecución de capa de normalización de batch de datos, una capa convolucional 1x1, y una capa de pooling de tipo "average pooling" de 2x2, es decir, la media. Del mismo modo, la función de composición que se usó originalmente se formaba con 3 operaciones consecutivas: una normalización del batch de datos, la aplicación de la función ReLU y, finalmente, una convolución de 3x3. Gráficamente se puede ver en la Figura 2-14 un ejemplo de Dense Block. Además, se aporta la Tabla 2-4 con diferentes arquitecturas DenseNet según el número de capas.

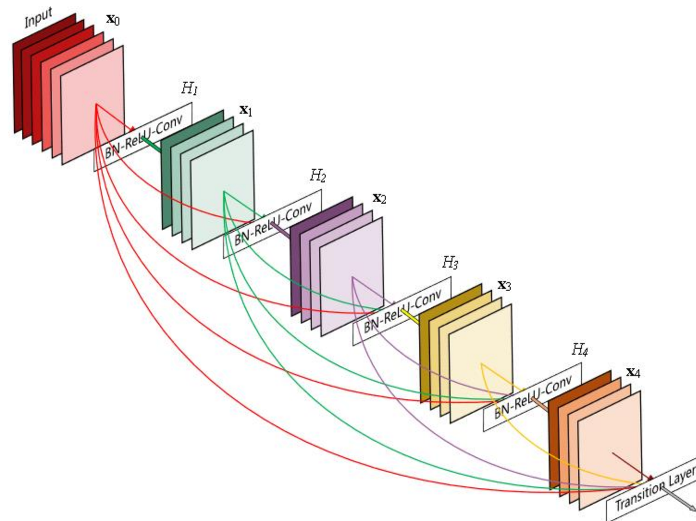


Figura 2-14: Dense Block de 5 capas con tasa de crecimiento *k*=4 [31]

Tabla 2-4: Arquitecturas de DenseNet según el número de capas, indicando para cada una el growth rate correspondiente (*k*) [31]

Layers	Output Size	DenseNet-121($k=32$)	DenseNet-169($k=32$)	DenseNet-201($k=32$)	DenseNet-161($k=48$)
Convolution	112×112	7×7 conv, stride 2			
Pooling	56×56	3×3 max pool, stride 2			
Dense Block (1)	56×56	$\begin{bmatrix}1\times1 \text{ conv}\\3\times3 \text{ conv}\end{bmatrix}\times6$	$\begin{bmatrix}1\times1 \text{ conv}\\3\times3 \text{ conv}\end{bmatrix}\times6$	$\begin{bmatrix}1\times1 \text{ conv}\\3\times3 \text{ conv}\end{bmatrix}\times6$	$\begin{bmatrix}1\times1 \text{ conv}\\3\times3 \text{ conv}\end{bmatrix}\times6$
Transition Layer (1)	56×56	1×1 conv			
	28×28	2×2 average pool, stride 2			
Dense Block (2)	28×28	$\begin{bmatrix}1\times1 \text{ conv}\\3\times3 \text{ conv}\end{bmatrix}\times12$	$\begin{bmatrix}1\times1 \text{ conv}\\3\times3 \text{ conv}\end{bmatrix}\times12$	$\begin{bmatrix}1\times1 \text{ conv}\\3\times3 \text{ conv}\end{bmatrix}\times12$	$\begin{bmatrix}1\times1 \text{ conv}\\3\times3 \text{ conv}\end{bmatrix}\times12$
Transition Layer (2)	28×28	1×1 conv			
	14×14	2×2 average pool, stride 2			
Dense Block (3)	14×14	$\begin{bmatrix}1\times1 \text{ conv}\\3\times3 \text{ conv}\end{bmatrix}\times24$	$\begin{bmatrix}1\times1 \text{ conv}\\3\times3 \text{ conv}\end{bmatrix}\times32$	$\begin{bmatrix}1\times1 \text{ conv}\\3\times3 \text{ conv}\end{bmatrix}\times48$	$\begin{bmatrix}1\times1 \text{ conv}\\3\times3 \text{ conv}\end{bmatrix}\times36$
Transition Layer (3)	14×14	1×1 conv			
	7×7	2×2 average pool, stride 2			
Dense Block (4)	7×7	$\begin{bmatrix}1\times1 \text{ conv}\\3\times3 \text{ conv}\end{bmatrix}\times16$	$\begin{bmatrix}1\times1 \text{ conv}\\3\times3 \text{ conv}\end{bmatrix}\times32$	$\begin{bmatrix}1\times1 \text{ conv}\\3\times3 \text{ conv}\end{bmatrix}\times32$	$\begin{bmatrix}1\times1 \text{ conv}\\3\times3 \text{ conv}\end{bmatrix}\times24$
Classification Layer	1×1	7×7 global average pool			
		1000D fully-connected, softmax			

DenseNet mejora en algunos aspectos a ResNet, pudiendo concatenar la entrada y la salida del bloque en la misma dimensión del canal en lugar de sumarlas, pero teniendo en cuenta que añadir Dense Blocks aumenta considerablemente el número de canales, por lo que se ponen capas de transición para controlarlo.

Estas han sido algunas de las arquitecturas que más impacto han tenido en el panorama actual de las redes convolucionales, aunque no son las únicas. Para terminar con este capítulo, en la próxima sección se hablará sobre la aplicación práctica de redes neuronales y técnicas de visión artificial al objetivo de reconocimiento de señales de tráfico, que es el fin último de este proyecto.

2.2 Aplicación a las señales de tráfico

Tal como se indica en [32], la IA en el tráfico hace uso de modelos de Machine Learning para, a partir de datos anteriores, imagenes del entorno, y demás fuentes de información, elaborar estrategias que permitan solucionar problemas durante el transcurso del tránsito de vehículos. De esta forma, es capaz de detectar mediante técnicas de visión por computador la mayoría de objetos en movimiento que circulan por la vía pública, y clasificarlos en categorías, como "peatones", "camiones", "mascotas", etc. Además, puede llegar a distinguir entre seres vivos o no, pudiendo elaborar medidas de respuesta frente a posibles accidentes, minimizando o incluso evitando daños.

Lo anterior indica que es capaz de detectar cambios en la vía urbana, de modo que puede aprender por ejemplo patrones de las horas de mayor tráfico, puntos de accidentes frecuentes, etc., y actuar en consecuencia redirigiendo el tráfico para agilizarlo, ahorrando tiempo a los usuarios y manteniendo sus vidas más seguras. Además de aprender patrones, también se puede hacer uso de semáforos inteligentes que, mediante imágenes en tiempo real, sea capaz de reajustar el tráfico para ahorrar tiempo. Un ejemplo bastante útil de estas características sería el caso en que hubiera una ambulancia atrapada en un atasco, de modo que se podría rediseñar el tráfico en tiempo real para ayudar a la ambulancia a llegar al hospital en menos tiempo, marcando en muchos casos la diferencia entre la vida y la muerte del paciente a bordo.

En Nueva Delhi, India, ya se están implementando algunos de estos sistemas para mejorar la congestion del tráfico, y es que sus habitantes pasan aproximadamente un 58% más de tiempo en atascos que los conductores del resto del mundo. Esto repercute también en el medio ambiente, haciendo que la contaminación aérea de Nueva Delhi sea preocupante, llegando al punto de que 1 de cada 7 policías sufran problemas respiratorios según un estudio de 2011 [33]. En estas circunstancias, se empezó a implementar lo que se conoce como "sistemas inteligentes de gestión del tráfico" (ITMS), instalándose más de 7,500 cámaras de circuito cerrado, semáforos automáticos, y 1,000 señales LED, todo aportando al conductor datos del tráfico en tiempo real.

La IA analiza los datos obtenidos de la infraestructura anterior y sugiere a los agentes de control de tráfico la posibilidad de tomar decisiciones en tiempo real. Pero no solo eso, sino que al aplicar la IA, la intervención humana de forma manual en el tráfico se reduce al mínimo, de modo que todo se gestiona desde una distancia segura. Sin embargo, la IA puede fallar, por lo que se siguen dejando espacios para la intervención humana en el control del tráfico.

Este caso sería uno que recoge las cualidades más llamativas del panorama actual de la IA en el tráfico, pero este TFG solo se centrará en una pequeña sección de ese apartado, y son las señales de tráfico. La detección de señales de tráfico es uno de los primeros pasos a la hora de realizar la actualización de mapas en línea. Dichos mapas se forman con imágenes de cámaras de muchísimas partes del mundo, pudiendo usar una IA para construir y/o actualizar los mapas en un corto periodo de tiempo.

Esos mapas son usados por los conductores a menudo mediante plataformas del estilo de Google Maps, y son muy útiles siempre que estén actualizados. Por ello, una de las primeras cosas que se hace al elaborar los mapas es detectar las señales de tráfico para ver si han cambiado o han puesto alguna nueva en una zona concreta. Con este objetivo, surgen algunos retos en la detección de las señales, y es que hay un gran número, todas con diferentes formas, tamaños, y colores, y cada una tiene un significado distinto.

Otro problema del reconocimiento de señales es la influencia del ambiente sobre ellas. Al capturar las imágenes, influyen la cámara, la luz y otros factores que no se pueden controlar. Por ejemplo, si la cámara que captura la señal se mueve muy rápido, no la enfoca bien y se verá borrosa. Lo mismo ocurre con las condiciones meteorológicas, pudiendo hacer que la gama de colores que capta la cámara no es la que realmente tienen esas señales, y otras veces simplemente están parcialmente tapadas por otro objeto, ya sea un coche o un árbol. Esto podría hacer por ejemplo que en el entrenamiento no se clasificasen bien las señales y las aprendiera erróneamente, incluso llegando a confundirlas con otras que tengan similitudes con la señal en cuestión.

Tal como se dice en [34], uno de los sistemas de confección de mapas que usa reconocimiento de señales es AMAP, por lo que debe hacer frente a este tipo de problemas. Los requisitos que plantean para un buen funcionamiento son una tasa de recuperación precisa para que a la hora de actualizar el mapa, si se da una recuperación incorrecta, no afecte al funcionamiento general ni a las actualizaciones en tiempo real, que tenga un buen rendimiento, el necesario para procesar cientos de millones de imágenes al día y actualizar los mapas en consecuencia, y por último que tenga una cierta extensibilidad o adaptabilidad al cambio o creación de señales

de tráfico nuevas.

Por todo esto, AMAP divide el problema en 2 partes bien diferenciadas en la Figura 2-15:

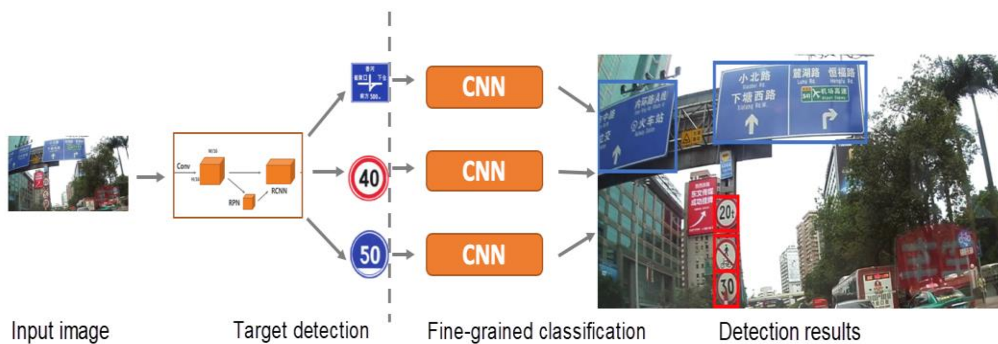


Figura 2-15: Etapas de AMAP para detección de señales de tráfico [34]

1. Detección de señales: Únicamente se encarga de ver cuántas señales de tráfico hay en una imagen y de ubicarlas en dicha imagen. Para ello, usa Faster R-CNN para una mayor velocidad de ejecución y tasa de recuerdo de las señales, configurando una RPN para cada categoría de señales, ya sea círculo, cuadrado, etc., de modo que pueda hacer frente al reconocimiento de imágenes que no tienen la relación de aspecto normal, siendo necesaria únicamente, al menos de manera ideal, una RPN por categoría sin que afecten unas a otras. Las RPN de cada categoría sin embargo sí que comparten la misma capa de convolución, para mejorar el rendimiento y ahorrar tiempo de ejecución.
2. Clasificación de grano fino: En esta etapa ya se analizan y procesan las zonas marcadas en la primera parte para conseguir distinguir de qué tipo de señal de tráfico se trata, usando redes CNN (convolucionales). Para ello, eliminan el ruido de la imagen consiguiendo unas tasas de recuperación y precisión altas, se configura una red de grano fino para cada categoría de modo que sea independiente de las demás categorías y trabajen todas en paralelo para tener mayor eficacia en la búsqueda de la señal. Tampoco estas redes tienen que ser iguales entre sí, siendo algunas más complejas que otras al necesitar una mayor capacidad de identificación de características. Por último, los muestreos se hacen para cada señal independientemente, y también se pueden anotar los resultados obtenidos para unas categorías concretas, favoreciendo la creación de conjuntos de entrenamiento y validación. Para no disparar el uso de la RAM de vídeo, se usó asignación y compartición dinámica de buffers entre los diferentes modelos y se recortó la función de propagación inversa, reduciendo más de un 50% su uso.

¿Qué pasa si el sistema de reconocimiento de señales sufre algún tipo de ataque informático que le impida funcionar de manera adecuada? Pues que habría un peligro considerable, ya que si ese sistema va en un coche con control inteligente de velocidad y se encuentra una señal de limitación mínima a 30 Km/h y la interpreta como limitación mínima a 90 Km/h, la velocidad aumentaría repentinamente hasta llegar a dicho mínimo erróneo, poniendo en peligro al conductor si éste no reacciona a tiempo.

Como solución a este problema, tal como se describe en el portal de prensa de la compañía [35], Bosch propuso en 2019 una cámara con una nueva tecnología, de manera que combinaba el enfoque multitrayecto de Bosch y una IA que permitiría que la detección del entorno fuese más fiable, aumentando la seguridad de la conducción. El nombre de dicha cámara es MPC3, y es capaz de reconocer el entorno de una manera superior a como lo haría un humano, que puede fallar por causas como agotamiento o distracción, cosas de las que la cámara no sufre, y es capaz de detectar objetos y tomar decisiones referentes a la estrategia de conducción con dichas condiciones, todo ello en un instante.

La inteligencia artificial de dicha cámara se integra en el chip V3H y se basa en el principio "know-how" de Bosch. Es capaz de reconocer los bordes transitables de carreteras incluso si están defectuosos o directamente no hay delimitación ninguna, podría mejorar los sistemas de frenado de emergencia automático debido a la habilidad de la cámara a la hora de reconocer patrones incluso estando ocultos de manera parcial, de forma que evitaría choques de coches o atropellos de seres vivos. Pero no se conforma con lo anterior, sino que también es capaz de mejorar la detección de señales de tráfico con un sistema de reconocimiento óptico de caracteres que

puede leer las escrituras de las señales de tráfico y darle la información al conductor. Para solucionar el problema de seguridad si es atacada la IA, lo que se usa entonces es otro sistema separado, que es la visión por computador, ya que el objetivo de los hackers es la IA de reconocimiento de señales de tráfico, según [36]. En la Figura 2-16 se aprecia la cámara MPC3 de Bosch.

Figura 2-16: Cámara MPC3 de Bosch [35]

Siguiendo con otros trabajos, en el documento [37] se detalla el desarrollo de un sistema de detección y clasificación de semáforos, en el que se usa una cámara empotrada en un vehículo como sensor de recepción de datos, con la posibilidad de complementarse dichos sensores con GPS, mapas digitales u otra cámara. Su estructura se divide entonces en 3 partes, como se ve en la Figura 2-17.

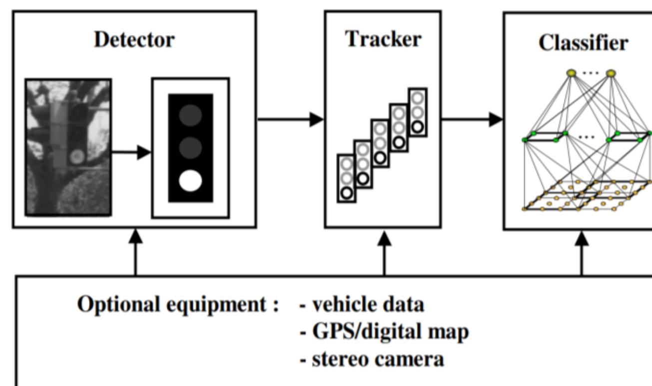


Figura 2-17: Arquitectura de bloques del sistema de detección y clasificación de señales del documento [37]

La primera etapa es un detector de semáforos. Básicamente su función es buscar hipótesis de semáforos en las imágenes capturadas por la cámara, usando para ello características como el color, forma y textura de los semáforos. Es la etapa más crítica porque al ser en tiempo real los datos de entrada le llegan constantemente, por lo que debe ser capaz de manejarlos de manera correcta. Según se dice en el documento en cuestión, en este caso sería mejor emplear operadores locales por su facilidad de implementación en una FPGA en el propio coche. Si hay candidatos falsos detectados en este módulo, se pueden tolerar hasta cierto punto porque en módulos siguientes se descartarán.

La segunda etapa es el módulo de rastreo, o tracker. Su función es recoger los candidatos recogidos en el módulo de detección y agruparlos en un fotograma, de modo que, teniendo en cuenta características como la ubicación relativa, velocidad y aceleración de la cámara respecto al semáforo, permite detectar candidatos a partir de varias imágenes consecutivas. Esto permite que en la primera etapa se puedan aceptar más errores de detección, ya que rellena los huecos que quedan cuando no recibe ninguna imagen. Además, soluciona el problema de detección espontánea que se podría dar en la primera etapa, es decir, si se detectan en la imagen rayos de sol como un semáforo, al pasarla por el tracking esto se descartará porque ese rayo no se va a mantener constante en la imagen

durante varios instantes de captura del mismo escenario, debido al movimiento de la propia cámara, así que lo puede descartar como candidato válido a semáforo en ese caso. El rastreo de los semáforos se debe hacer en cualquier parte de la imagen, no solamente en el centro, ya que, si no se usa GPS o cualquier otro medio de ayuda a los sensores, puede darse el caso en que el semáforo aparezca tapado por un objeto y por eso no lo detecte.

La última etapa es el clasificador de semáforos. El módulo tracker le pasa las imágenes al clasificador, y en cada ristra de imágenes que le pasa, éste las evalúa una a una. Se usa voto de mayoría suave para tomar una decisión. Lo más difícil para este módulo es rechazar los candidatos que no son correctos de la manera más robusta posible, pues tiene el inconveniente de que, dependiendo del tipo de cámara, tendrá que clasificar también el estado de dicho semáforo si la cámara es en escala de grises, o bien dicho estado se conoce directamente porque la cámara tiene visión de color. Su misión entonces es clasificar los candidatos en semáforos o no, y en caso de ser semáforos, dar el estado de dicho semáforo, es decir, si está en verde, rojo, o ámbar. Se emplearon para su diseño redes neuronales de tipo feed-forward con campos receptivos, y se usó aumento de datos en el conjunto de entrenamiento.

Con esta estructura, se desarrollaron diferentes versiones del sistema. Para el diseño con una cámara de color, uno de los problemas fue lidiar con el rango dinámico de color de los sensores, que era muy pequeño. Para detectar un semáforo activo, debía tener una de las luces visibles encendida, y para que pudiera verse tenía que estar expuesta a la cámara un tiempo demasiado largo que hacía que se viera la luz de color blanca por la saturación de la cámara. Por ello, se usaron sensores CMOS de alto rango dinámico, solucionando dicho problema.

Hay 4 clases de píxel en este caso, y son rojo, verde, amarillo, o fondo del escenario. La imagen que le llega al módulo, y se usa una distribución Gaussiana, de manera que la imagen se difumina y se divide en secciones usando análisis de componentes conectados, pudiendo obtener los cuadrantes de la imagen donde se encuentran las luces del semáforo.

El planteamiento cambia para las cámaras en escala de grises, puesto que en realidad el clasificador se adaptará igualmente con una Gaussiana como en el caso a color, ahora la característica funcional para detectar las luces es la incandescencia, es decir, la intensidad lumínica de la luz de cada color que puede detectarse con una cámara de este tipo, ya que no todos los colores tienen la misma emisibilidad. La técnica consiste en hacer un recorte sobre la imagen, centrándolo en la luz detectada y suponiendo que es amarilla, para posteriormente subir y bajar dicho recorte, lo que hace un total de 3 ventanas. Si están por debajo del umbral concreto del color, se rechaza la hipótesis.

Otra característica para detectar semáforos es la forma, por lo que el detector de círculos que se implementa aquí es una buena opción. Usa la transformada de Hough generalizada con máscara de Sobel y basada en la dirección del gradiente de la imagen. Puede detectar círculos de diámetro superior a 6 píxeles de manera eficaz, y gracias a la optimización de la transformada de Hough, funciona a 15 fotogramas por segundo. Para detectar el borde de los semáforos, se optó por usar un filtro de emparejado con plantillas.

Finalmente, se diseñó un clasificador en cascada, de manera que, en cada parte de dicho detector, se van descartando cada vez unos candidatos según diferentes características en cada etapa, usando para ello el algoritmo AdaBoost que combina estudiantes débiles con un clasificador fuerte. Este método da lugar a un menor número de falsos positivos, aunque es algo lento para el tiempo real.

Otro sistema desarrollado en esta línea es el del documento [38]. Aquí se describe un modelo de detección y clasificación de señales de tráfico y semáforos, que emplea un modelo de atención visual que encuentra regiones candidatas en la imagen donde puede haber una señal, pero dicha región no se ajusta al objeto. Este enfoque permite que se puedan detectar mejor los objetos pequeños. Una vez que se explora cada región y el objeto es encontrado, se bordea con una bounding box. En el conjunto de datos LISA, obtuvo una precisión media superior al 90 %.

La estructura del sistema tiene dos partes, de modo que el enfoque va de lo más general a lo particular. El primer bloque es el modelo de atención de propuestas, o *attention proposal modeller* (APM) en inglés, y un localizador y reconocedor de precisión, o *accurate locator and recognizer* (ALR) en inglés. En la Figura 2-18 se puede ver un esquema del modelo del sistema y la función de cada parte.

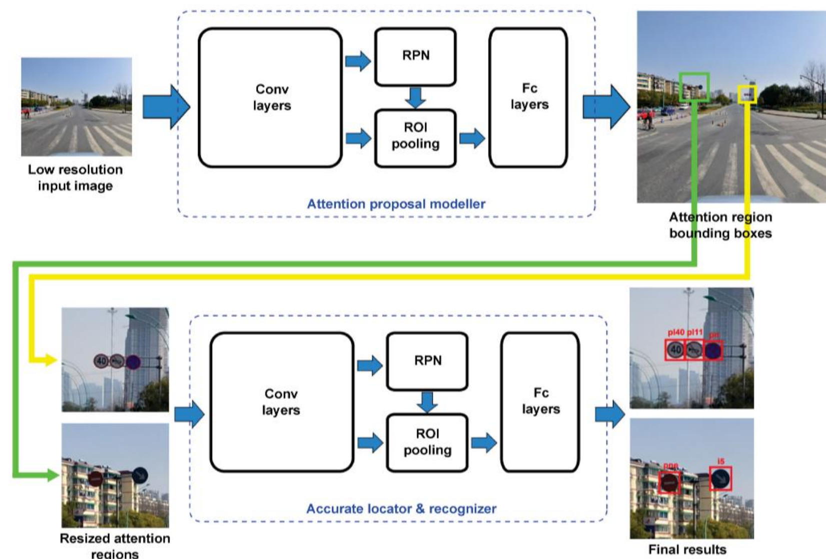


Figura 2-18: Esquema global del sistema [38]

Un modelo de atención de propuestas (APM) tiene como objetivo encontrar con alta confianza y bajo coste computacional una zona candidata en lugar de detectar con precisión el objeto buscado, por lo que la imagen de entrada se baja de resolución para que se centre mejor en las características globales del entorno en lugar de en los detalles. Entonces, encuentra una serie de regiones de atención, y las saca como salidas junto con el nivel de confianza que tiene para cada región detectada.

Se basa en el uso de faster RCNN, con la diferencia de que no está entrenada para detectar bounding boxes de los objetos con precisión, sino que está preparada para encontrar bounding boxes de regiones con posibles candidatos. Siguiendo a la faster RCNN, se encuentran una red RPN (región proposal network) y una red fast RCNN, que comparten una red convolucional que genera mapas de características del tamaño de la imagen de entrada, (W x H). En [38] se indica que el tamaño de las regiones de interés se calcula con centro el mismo de la bounding box que encierra el objeto, y con ancho y alto iguales y de valor el máximo de los 2 que tenga la bounding box del objeto multiplicada por una constante α de valor 5 en este caso. En la Tabla 2-5 se muestra la estructura completa del APM:

Tabla 2-5: Estructura del APM [38]

Layer	Conv1	Conv2	Conv3	Conv4	Conv5	RPN conv	RPN output		Fc6	Fc7	Predictions	
							Scores	Boxes			Scores	Boxes
Channels	64	128	256	256	256	256	18	36	4096	4096	2	8
Kernel size	7	5	3	3	3	3	3	3	—	—	—	—
Stride	2	2	1	1	1	1	1	1	—	—	—	—
Pooling	Max(3,2)	Max(3,2)	—	—	ROI pooling	—	—	—	—	—	—	—

Esas regiones de atención obtenidas por el APM se recortan y se pasan al módulo ALR, que las reescala todas al mismo tamaño según el rendimiento en validación, y permite detectar el objeto con precisión y clasificarlo. Esto ahorra coste computacional porque los cálculos que realiza sobre los cuadros de atención no son los mimos que haría si se usara la imagen entera, de mayor tamaño. Esto se acentúa más para el caso de objetos pequeños, donde su rango pequeño en la imagen permite un aumento de rendimiento en dichos objetos pequeños. Esto va ligado en parte a que al tener una relación mayor el objeto detectado en dicha área respecto al que tiene en la imagen entera se puede localizar y reconocer mejor el objeto porque sus propiedades son más visibles a dicha escala.

En este caso se ha utilizado para el ALR el marco de faster RCNN por proporcionar resultados de última

generación para la mayoría de las tareas de detección, teniendo una arquitectura similar a la del AMP de la Tabla 2-5, pero adaptándola al número de etiquetas de las clases de señales de tráfico y semáforos. Es capaz de localizar y reconocer 45 clases de señales de tráfico, y para el caso de los semáforos se usan 6 clases.

Estos serían algunos de los estudios y trabajos relacionados con el Deep Learning aplicado a Visión Artificial para tareas de control del tráfico.

2.3 Comentarios finales

Se introdujeron varias estructuras conocidas en el mundo de las redes convolucionales, pasando desde LeNet-5, también por la red AlexNet y la revolución que conllevó, hasta llegar a redes basadas en bloques como VGG o DenseNet, y se buscaron algunas aplicaciones prácticas de todos estos conocimientos para el caso que concierne a este trabajo. En este contexto, se introdujo el impacto general de la IA en el control automático del tráfico, aportando el ejemplo concreto de la ciudad de Nueva Delhi, y a continuación se desgranó el sistema de detección de señales de AMAP para desarrollo de mapas en tiempo real, y de algunas innovaciones para incrementar la seguridad de dichos sistemas, como la propuesta por Bosch con la cámara MPC3. También se aportaron otros trabajos que se centraban en la detección y clasificación de señales de tráfico como el que se desarrolló para AMAP, pero además estos tenían la capacidad de detector semáforos.

Una vez alcanzado este punto del proyecto, se tratarán en el siguiente capítulo todas las herramientas necesarias para sacar el proyecto adelante, empezando por una explicación de qué es el Machine Learning, pasando por la explicación de cuál es el problema a resolver en este documento, hasta llegar al diseño de la red propiamente desarrollada para resolverlo.

3 METODOLOGÍA

Nada en la vida es para ser temido, es solo para ser comprendido. Ahora es el momento de entender más, de modo que podamos temer menos.

- Marie Curie -

Una vez que se han tratado en los capítulos los temas de actualidad y la IA en un ámbito general en este capítulo comienza el desarrollo del proyecto, donde se explicará a fondo la metodología usada para crear la red neuronal convolucional que será capaz de clasificar señales de tráfico. Este capítulo se dividirá en 2 partes bien diferenciadas, donde en los primeros 3 capítulos se hablará de qué es el Machine Learning, para continuar con una clasificación de los diferentes tipos de redes neuronales, tanto por su topología como por sus métodos de entrenamiento, y se terminará con una explicación de los diferentes tipos de funciones de activación de las neuronas de las redes y de los diferentes algoritmos de optimización que se pueden dar en el entrenamiento.

Tras lo anterior, la siguiente sección tratará sobre el planteamiento del problema que se desea resolver una vez que se han explicado en anteriores secciones los fundamentos básicos de redes neuronales, y ya en las secciones restantes se abordará la solución diseñada para ese problema, empezando con la obtención de los datos, que son fotografías de señales de tráfico agrupadas en lo que se conoce como "Dataset", detallando todas las capas que se han usado para el diseño de la red, y finalmente, concretando las metodologías de los procesos de entrenamiento, validación y testeo, presentando los resultados en el siguiente capítulo.

3.1 Machine Learning

En la actualidad, la Inteligencia Artificial es una rama de la ciencia que a su vez tiene otras muchas ramas que estudiar, como pueden ser los algoritmos genéticos o el Machine Learning, por ejemplo. Es en esta última en la que se engloba el Deep Learning que se usa en las redes neuronales que se diseñarán en este capítulo. Por ello, es conveniente definir primero brevemente qué es el Machine Learning.

Según [39], Machine Learning es la rama de la Inteligencia Artificial que otorga a los ordenadores y demás máquinas la capacidad de "aprender" por sí mismas sin la necesidad de que la función aprendida sea programada de antemano. Esto lo consigue mediante la aplicación de algoritmos diseñados para cada situación, de modo que, mediante unos datos de entrada, permitirá mediante dichos algoritmos generar unas predicciones de forma que la próxima vez que se encuentre con situaciones de datos similares, podrá predecir y/o reconocer características concretas de dichas situaciones debido al aprendizaje automático previo que ha realizado usando el algoritmo. En la Figura 3-1[10] se muestra dónde se engloba el Deep Learning dentro de la IA:

[10] Fuente: https://medium.com/@experiencIA18/diferencias-entre-la-inteligencia-artificial-y-el-machine-learning-f0448c503cd4

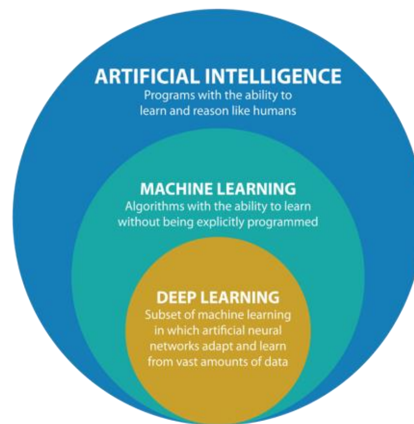


Figura 3-1: Deep Learning en IA

Los algoritmos de Machine Learning son tantos como situaciones posibles en la realidad, por lo que son prácticamente infinitos. Sin embargo, todos estos algoritmos suelen tener unas características que permiten agruparlos en 3 grupos reconocidos:

1- Aprendizaje supervisado: "Supervisado" hace referencia al tipo de datos que se usan para entrenar el algoritmo. Estos conjuntos de datos contienen a su vez la solución a dicho ejercicio, es decir, que contienen una "etiqueta" (label) que indica qué tipo de dato se está evaluando. De esta forma, si se entrena un algoritmo que reconoce fotografías de personas para ver si en dicha foto hay o no una persona, la fotografía que se usa para entrenar contendrá una etiqueta que indicará si hay o no una persona, pudiendo usarse esta información para que el modelo elaborado se corrija a si mismo. El funcionamiento se ilustra en la Figura 3-2[11]. Algunos algoritmos de este tipo son la regresión lineal, random forest y redes neuronales.

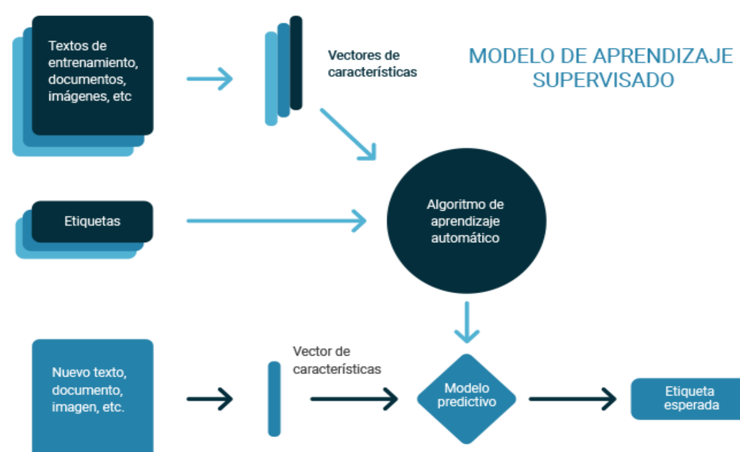


Figura 3-2: Diagrama de funcionamiento de algoritmo aprendizaje supervisado

2- Aprendizaje no supervisado: Es el aprendizaje contrario al anterior. Ahora, los conjuntos de entrenamiento no contendrán ninguna "etiqueta" con la solución al ejercicio, por lo que será el propio algoritmo el que detecte directamente de qué dato se trata. Aplicado al caso anterior, será el algoritmo el que tenga algún tipo de mecanismo para certificar si en la imagen hay una persona o no.

[11] Fuente: https://medium.com/soldai/tipos-de-aprendizaje-autom%C3%A1tico-6413e3c615e2

Funcionamiento en la Figura 3-3[12]. Algunos algoritmos conocidos son clustering (K-means) o principal component analysis (PCA).

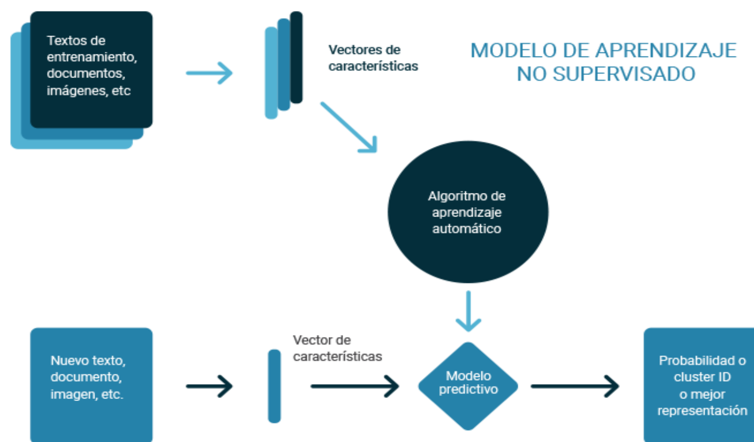


Figura 3-3: Diagrama de funcionamiento de algoritmo aprendizaje no supervisado

3- Reinforcement Learning: Este tipo de algoritmos está muy de moda hoy en día, ya que resuelve situaciones que se dan en la realidad muy a menudo. Básicamente consiste en entrenar un modelo de tal forma que éste tiene un "agente" que explora un espacio desconocido y da recompensas o penalizaciones al algoritmo que se va ejecutando mediante prueba y error. Dicho agente debe encontrar la forma (política) más rápida y óptima de obtener recompensas para llegar antes a una solución válida. El funcionamiento se ilustra en la Figura 3-4[13].

MODELO DE APRENDIZAJE POR REFUERZO

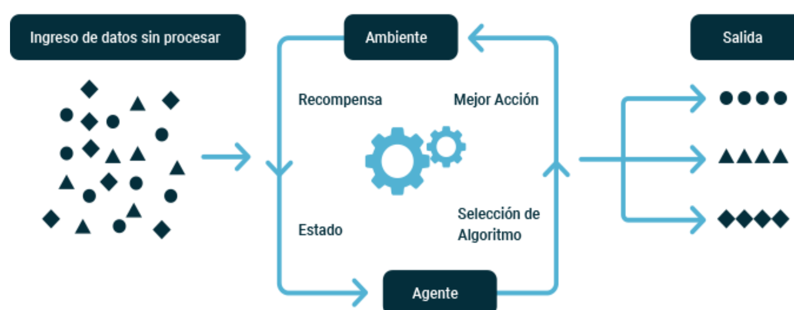


Figura 3-4: Diagrama de funcionamiento de algoritmo Reinforcement Learning

Una vez que se ha visto qué es el Machine Learning, se verá un tipo de algoritmo de Deep Learning, que se engloba dentro del Machine Learning, conocido como redes neuronales. En el problema concreto de este estudio,

[12] Fuente: Como la de la Figura 3-2.
[13] Fuente: La de la Figura 3-2.

se abordará un aprendizaje de tipo supervisado, es decir, se conocerán de antemano de qué tipo son los datos que se le pasan al algoritmo de predicción.

3.2 Tipos de redes neuronales

En esta sección se procederá a explicar los tipos de redes neuronales que existen actualmente atendiendo a criterios como la topología de la red o los algoritmos empleados.

3.2.1 Según la topología

Esta clasificación se centrará en la forma de la red neuronal, atendiendo entre otras características al número y tipo de capas y neuronas que forman dicha red.

3.2.1.1 Perceptrón

El perceptrón es la red neuronal más simple que se despacha en Deep Learning. Tal como se dice en [40], uno de sus usos más básicos es el de separar 2 clases, por lo que hace una clasificación de tipo binario, sacando como salida de dicha neurona el valor '0' o '1' según sea una clase u otra. Sin embargo, el perceptrón también se puede usar para distinguir entre varias clases, y que a la salida muestre una probabilidad de bondad de identificación de dicha clase, obteniendo un porcentaje de posibilidad de que sea o no dicha clase.

Por ello, el modelo general de un perceptrón se observa en la Figura 3-5, donde se multiplica el valor de cada entrada por su peso, se suman todos esos valores, y a ese total se le suma un parámetro de polarización para obtener la salida de dicha neurona:

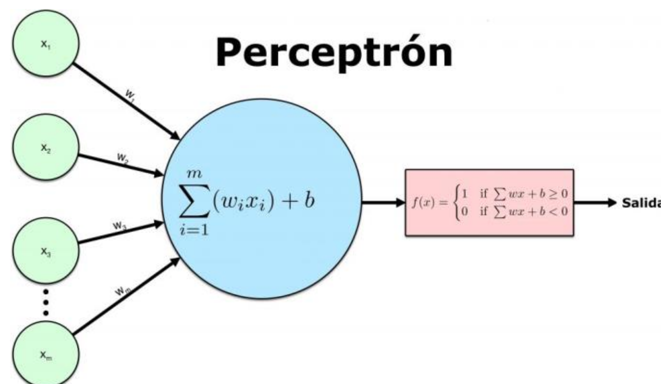


Figura 3-5: Modelo de un perceptrón [40]

Lo que intentaba este modelo era imitar el funcionamiento de una sola neurona. Por ello, va ajustando unos pesos y un parámetro de polarización de forma que los errores se van haciendo más pequeños a medida que se entrena la neurona.

3.2.1.2 Redes monocapa

Es la evolución natural del perceptrón anterior, aunque éste podría ser considerado como el caso concreto de una red monocapa de una sola neurona. En consonancia con lo explicado en [41], estas redes se suelen usar para autoasociación, ya que las neuronas crean conexiones con otras de la misma capa, llamándoselas por ello autoconcurrentes, como la de la Figura 3-6. Por su topología, las monocapa se suelen implementar en circuitos eléctricos con hardware usando diodos para representar las conexiones entre neuronas.

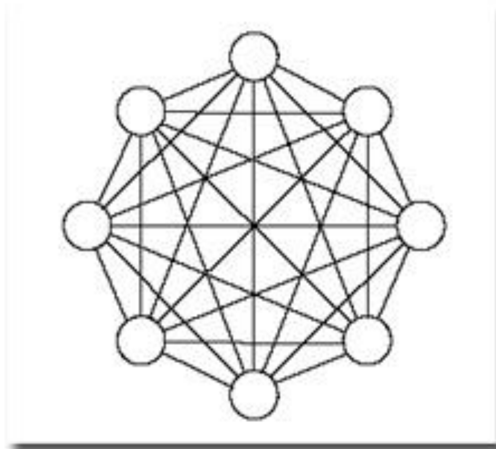


Figura 3-6: Red monocapa autoconcurrente [41]

Algunas redes de este tipo son la red de memoria asociativa ("BRAIN-STATE-IN-A-BOX"), las máquinas de Boltzmann y Cauchy, y la red de Hopfield, que es una de las más típicas.

La red de Hopfield es autoasociativa y no lineal, tiene las neuronas con salidas binarias, y la salida de una neurona se conecta con la entrada de la siguiente, estando una neurona conectada a todas las de la capa excepto con ella misma. En este caso, los pesos que se ajustan en el entrenamiento y que almacenan el conocimiento de la red serán simétricos, obteniendo una matriz de pesos cuadrada y simétrica. Se puede ver gráficamente su topología en la Figura 3-7[14].

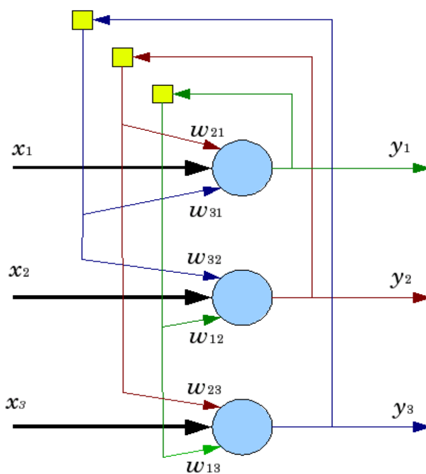


Figura 3-7: Modelo de red de Hopfield con 3 neuronas, siendo x_i las entradas, y_i las salidas, y w_{ij} los pesos de la red

3.2.1.3 Redes multicapa

Este tipo de redes neuronales se caracteriza por el uso de más de una capa de neuronas puestas, generalmente, una continuación de la otra, de manera que se conectan de distintas formas. Si las conexiones entre las capas únicamente se dan hacia capas posteriores a élla, se habla de redes con conexiones hacia delante o **feedforward**, ya que las capas de este tipo de redes no tienen conexiones con capas que reciban las variables de entrada antes

[14] Fuente: https://www.mql5.com/es/articles/1103

que dicha capa. Un ejemplo de estas redes podría ser el perceptrón multicapa [42].

Tal como se cuenta en [43], el perceptrón multicapa es una red neuronal que hace uso de un número N de perceptrones en cada capa y se interconectan entre sí. La entrada se propaga hacia delante por las capas hasta que llega a la última, momento en el que los errores vuelven desde dicha capa hasta las primeras modificando los pesos de las neuronas de las capas de la red para que dicho error sea menor. Esto se conoce como "backpropagation".

Normalmente su estructura es de 3 capas:

- Capa de entrada: Primera capa de la red a la que se le pasan las variables de entrada del exterior, de modo que corresponde cada neurona con un dato de entrada.
- Capas ocultas: Son de diferente cantidad de neurona por capa y de diferente número de capas en total, teniendo que estar todas sus neuronas conectadas con todas las de la capa siguiente, de modo que pueda tener en cuenta las funciones de activación de las capas anteriores y posteriores.
- Capa de salida: Es la que da la clasificación de la clase a la que pertenece la variable de entrada introducida, es decir, saca el resultado de las operaciones que realiza la red en las capas ocultas.

De esta forma, la estructura de la red quedaría de la siguiente forma en la Figura 3-8[15]:

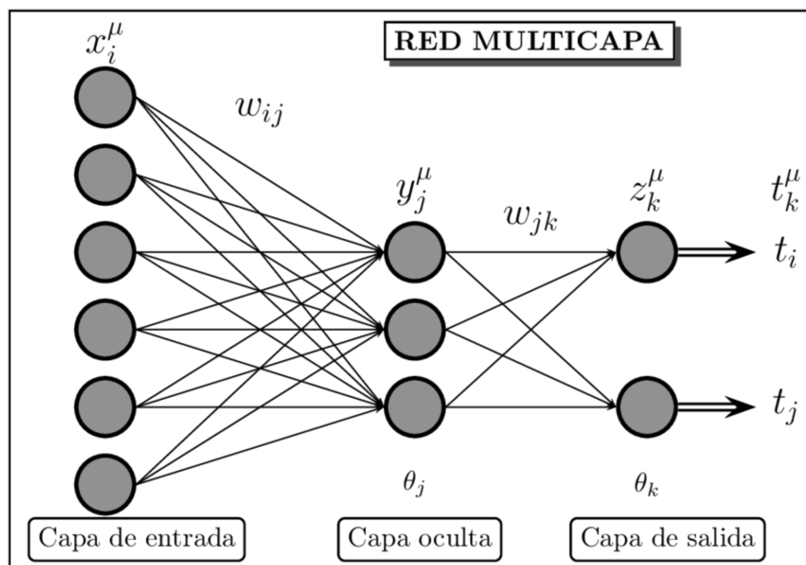


Figura 3-8: Estructura de perceptrón multicapa con 3 capas

Sin embargo, las conexiones entre las capas no son siempre hacia delante, sino que hay veces en que capas posteriores están conectadas a capas anteriores en el recorrido de las variables de entrada, por lo que la información puede regresar a esas capas que están más atrás. Este tipo de redes se conoce por ello como redes con conexiones hacia atrás, feedback, o retroalimentadas. Ejemplos de este tipo de red son las ART, Bidirectional Associative Memory (BAM), o Cognitron, según [42].

Como se dice en [44], la red ART se suele usar para aprendizaje no supervisado y de tipo competitivo, y se basa en la Teoría de Resonancia Adaptativa. Dicha teoría consiste en hacer resonar la información de entrada con los prototipos que tiene la red almacenados como reconocibles hasta ese instante. Esto se hace así porque se soluciona el problema de plasticidad (aprender patrones nuevos) y estabilidad (retener esos nuevos patrones aprendidos) que se produce al modificar los pesos de las neuronas.

Para implementar todo lo anterior, se producen unas conexiones de realimentación entre las capas de salida y entrada de la red, de manera que la información ya aprendida no se pierde, teniéndola en cuenta para modificar

[15] Fuente: https://www.researchgate.net/figure/Figura-113-Perceptron-multicapa_fig16_48932489

el prototipo ya obtenido de una clase si los datos de entrada son parecidos (“resuenan”) en dicha clase, o por el contrario creando una clase nueva para ese prototipo si no se parece a ninguno ya aprendido. Existen 6 tipos de redes ART: ART1, ART2, ART3, Fuzzy ART y ARTMAP. Se puede ver un ejemplo de ART1 en la Figura 3-9[16].

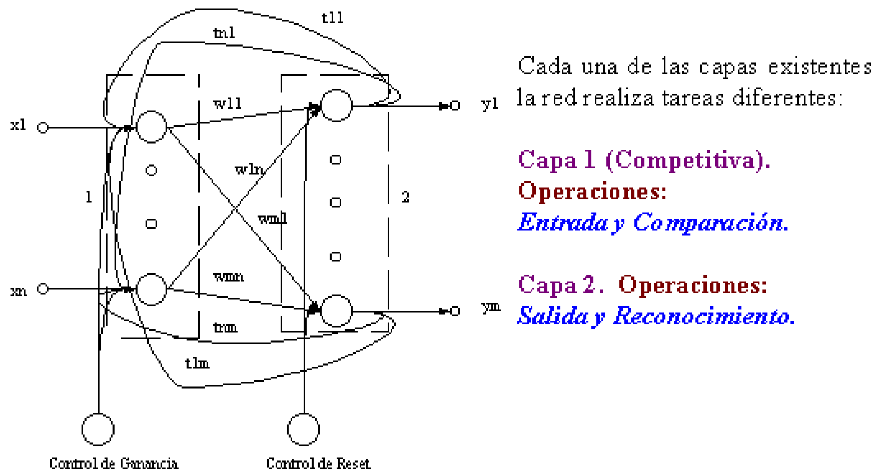


Figura 1:

Figura 3-9: Ejemplo de red tipo ART1

A continuación, y para terminar con esta parte, se muestra en la Figura 3-10 un esquema resumen de las redes neuronales según su topología:

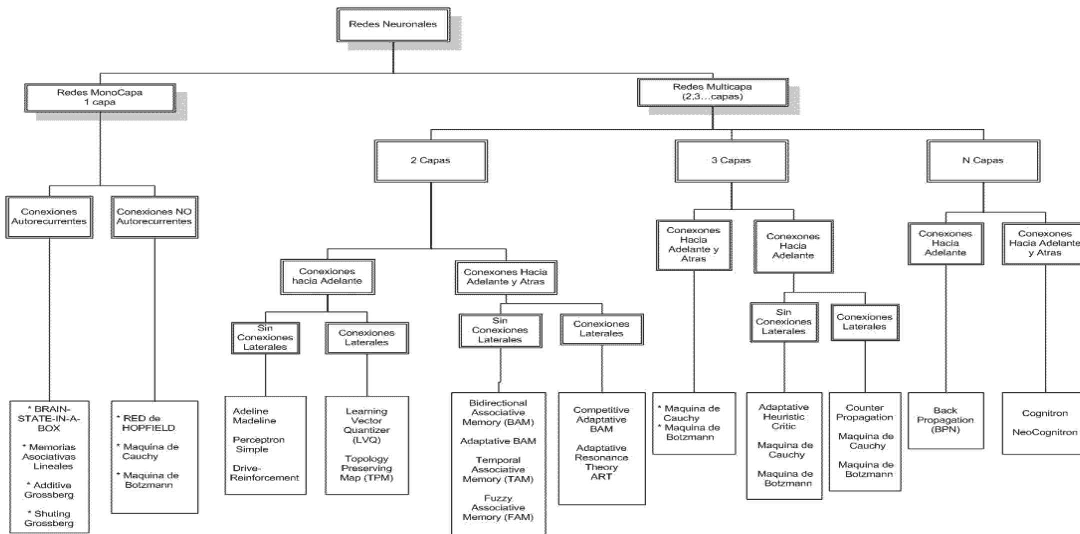


Figura 3-10: Esquema resumen de las topologías de redes neuronales [42]

3.2.2 Según el algoritmo de entrenamiento

En esta clasificación se hará hincapié en el tipo de algoritmo que usa cada red para aprender. Básicamente, puede haber tres clasificaciones: algoritmos de aprendizaje supervisado, algoritmos de aprendizaje no supervisado, y

[16] Fuente: http://www.varpa.org/~mgpenedo/cursos/scx/Tema6/nodo6-2-1.html

algoritmos de aprendizaje por refuerzo, como ya se adelantó cuando se habló anteriormente sobre Machine Learning. A continuación, se explicará con detalle en qué consiste cada una:

3.2.2.1 Redes de aprendizaje supervisado

Tal como se dice en [39] y [42], en estas redes el algoritmo se basa en la suposición de que los datos que el conjunto de datos, o dataset, que se usa para entrenar la red, contiene las características que se desean aprender, o *features*, así como una etiqueta con la clase a la que pertenecen estas características, o *labels*, de modo que usa esta información extra para realimentarse y mejorar los pesos de las neuronas de la red, usando lo que se conoce como "back propagation".

Al inicio, se le pasan unos datos, y la red tiene unos pesos iniciales que pueden estar o no ajustados de alguna forma. Los datos van pasando por todas las capas de la red, y llegan a la última capa, obteniéndose una clasificación/predicción en función de los datos de entrada. A continuación, compara la predicción obtenida con la etiqueta que tenían esos datos de entrada y mira en cuánto se corresponden ambos resultados, obteniendo una especie de medida del error, pudiendo usar funciones como el error cuadrático medio, loss, etc. Si dicho error es muy grande, entonces esta información recorre de nuevo las capas de la red, pero empezando desde la última capa, la de salida, hasta llegar a la primera capa, la de entrada. Esto permitirá que los pesos se reajusten de nuevo para mejorar la predicción de que esos datos o características de entrada se correspondan con la clase concreta que se estaba evaluando, disminuyendo así el error. Esta última acción es lo que se conoce como "back propagation". El funcionamiento de este tipo de entrenamiento se aprecia en la Figura 3-11:

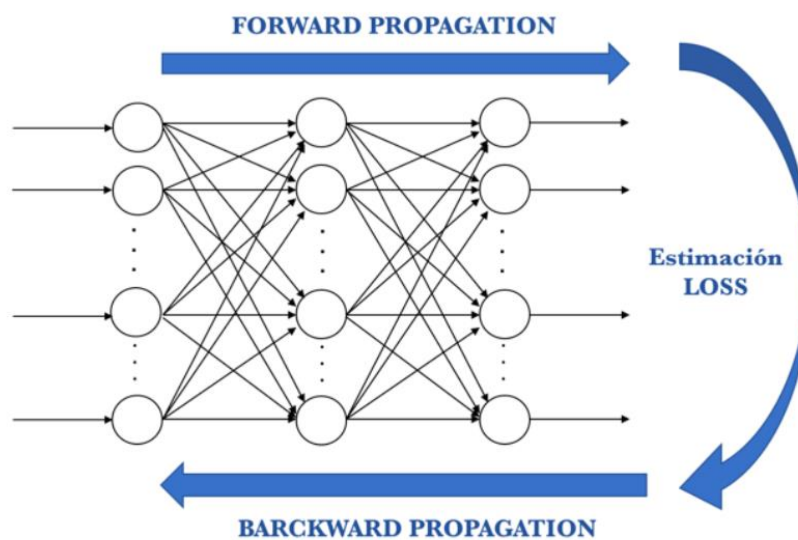


Figura 3-11: Funcionamiento de aprendizaje supervisado y backpropagation [39]

3.2.2.2 Redes de aprendizaje no supervisado

En estas redes el algoritmo no necesita conocer la etiqueta de los datos de entrada porque no hace "back propagation". La idea es hacer que le lleguen a la red un grupo de patrones de datos, de modo que mirará lo parecidas que son sus características entre sí y los clasificará en la misma categoría o no. Cuando le llegue un nuevo dato, se compararán sus características con la de los patrones y se podrá saber a qué clase pertenece. El parecerse se puede cuantificar, ya que una de las características de este tipo de algoritmos es la elección del grado de parecido entre las características, por ejemplo, mediante la desviación típica. Todo esto se encuentra en [42].

La gracia de este tipo de aprendizaje es que la red clasificará y aprenderá todo de manera automática, pero no se sabrán a priori qué categorías usará para clasificar los diferentes tipos de patrones, ni qué características usará para clasificar dichos patrones. Ahora no se está tan interesado en hacer predicciones, como en el caso de aprendizaje supervisado, sino que se desea obtener información sobre el conjunto de datos que se está

analizando.

Una de las técnicas de aprendizaje no supervisado es el de aprendizaje por componentes principales. Básicamente consiste en especializar un grupo de neuronas para que reconozcan las características comunes, o componentes, de los datos que se le vayan proporcionando, pudiendo añadirlos a dichas clases en función de esas características, reduciendo así el número de variables a examinar respecto a las variables originales. Tal como se desarrolla en [45], para calcular estas componentes principales, lo que se hace es estandarizar las características de los datos originales para que tengan **media 0 y desviación estándar de 1** para que las de mayor varianza no dominen a las demás, y a continuación se hace la combinación lineal de dichas características, obteniéndose así las componentes principales como autovectores, ordenándose dichas componentes de mayor a menor varianza.

Otra de las técnicas de aprendizaje no supervisado que se dicen en [42] es el aprendizaje competitivo, donde las neuronas compiten todas para poder representar al patrón de una clase concreta, y las que lo consiguen lo ven reflejados en sus pesos respecto a las neuronas que han perdido la competición. De esta forma, se crearán subconjuntos de clases que son representados por una sola neurona, y cuando a la red se le pase una variable de dicho patrón, únicamente se activará la neurona que represente a esa clase concreta, generalmente poniéndose a '1' mientras que las otras neuronas se ponen a '0'. Su arquitectura es normalmente de dos capas: la primera es la que se conoce como "capa sensor", y es la capa de entrada por la que se le pasan los datos o patrones a la red, y la segunda capa es la capa de salida, que tiene tantas neuronas como clases se necesiten. Todas las neuronas de una capa están conectadas con todas las neuronas de la otra. Así, se aprenden las conexiones sinápticas que hay entre cada neurona correspondiente a cada patrón. Un ejemplo gráfico de su estructura se representa en la Figura 3-12[17]:

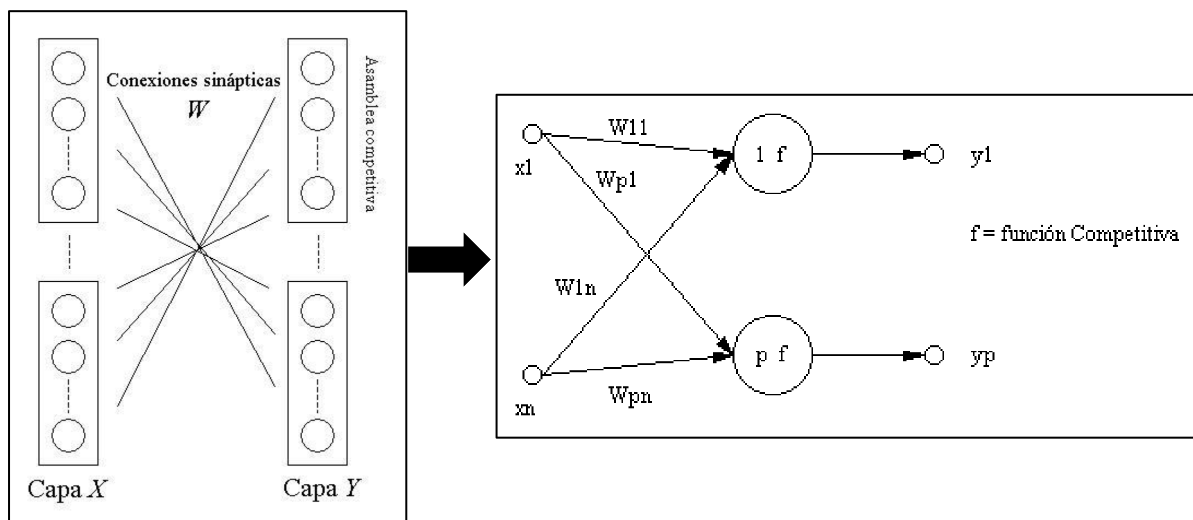


Figura 3-12: Estructura general competitiva (izquierda) y particularización para 2 neuronas (derecha)

3.2.2.3 Redes de aprendizaje por refuerzo

Como se dice en [46], en el aprendizaje por refuerzo, o *reinforcement learning*, las redes implementan un agente que recorre todo el mapa de posibilidades para encontrar las acciones a realizar sobre el medio que permiten alcanzar un objetivo concreto. El agente experimenta por sí mismo para encontrar las acciones con una mayor recompensa. Hay, por tanto, diferencias entre los dos tipos anteriores de aprendizaje y el aprendizaje por refuerzo. En primer lugar, el agente puede aprender de situaciones de su propia experiencia que no estén contenidas en el dataset de entrenamiento, a diferencia del aprendizaje supervisado, ya que en este último las acciones a realizar en cada situación le vienen especificadas en la etiqueta de los datos. Respecto al aprendizaje no supervisado, la diferencia fundamental es que el objetivo será aumentar la recompensa en lugar de encontrar

[17]Fuentes:https://www.researchgate.net/figure/Arquitectura-de-la-red-NELOC_fig1_233987343 y http://www.varpa.org/~mgpenedo/cursos/scx/Tema5/nodo5-3.html

subconjuntos en el dataset.

En este aprendizaje hay varios elementos además del agente y del medio, y son la política, la señal de recompensa y la función de valor. De manera optativa también se puede incluir un modelo del medio ambiente, pero no es estrictamente necesario. La política es lo que determina el comportamiento del agente, es decir, será la acción por realizar cuando se le dé una recompensa. La señal de recompensa entonces será la representación del objetivo alcanzado, devolviendo la red un valor mayor de recompensa cuanto mejor se alcance dicho objetivo. Interesa maximizarla a largo plazo, pero la señal de recompensa es un valor a corto plazo, por eso existe la función de valor, que predice la cantidad de recompensa que el agente obtendrá en un futuro a partir del estado actual. Por ello, lo que en realidad se hará es regir las políticas por el valor obtenido de dicha función, ya que así se obtendrán los mayores valores de recompensa posibles.

Un ejemplo de la aplicación de este tipo de redes es el trazado de caminos en tiempo real por un escenario desconocido, como el desplazamiento de un robot en una habitación con obstáculos que pueden moverse, donde debería ser capaz de parar y corregir la trayectoria actual en tiempo real y aprender sobre la marcha la mejor acción (política) a realizar según la seguridad (recompensa) que tenga para la integridad de los seres vivos y del propio robot.

3.3 Funciones de activación y algoritmos de optimización

En esta sección se hablará de las diferentes funciones de decisión que se suelen usar como función de activación de cada neurona en las redes neuronales, así como los algoritmos de optimización en el entrenamiento que se emplean en dichas redes.

Una función de activación según [47] es la que hay en cada neurona de cada capa, y es la responsable de que las neuronas saquen una salida u otra en función de los datos de entrada que se le administren. Pueden ser lineales o no lineales, y son estas últimas las que más se usan porque permiten a la red adaptarse y aprender mejor gracias a la introducción de no linealidades en el aprendizaje. Pueden ser la función lineal, sigmoide, tangencial, etc.

Por otra parte, en [39] se dice que los algoritmos de optimización no son algo exclusivo de una neurona, sino más bien del proceso de entrenamiento completo. En el entrenamiento se pretende encontrar los pesos que mejor se ajusten a los resultados deseados, y normalmente el resultado óptimo será costoso de alcanzar analíticamente, si es que se puede. Por eso, estos algoritmos intentarán obtener o acercarse a los mejores resultados posibles. Algunos de los algoritmos de optimización más usuales pueden ser el de Descenso por Gradiente (SGD) o el Adam.

3.3.1 Funciones de activación

En este apartado se abordarán las funciones de activación más conocidas, usando como referencia [39] y [47]:

3.3.1.1 Lineal

La función lineal pretende mantener constante el valor de la señal de entrada. Es decir, la salida de la neurona en cuestión le pasará a la siguiente neurona como valores de entrada los mismos datos que tenía la anterior. Es de las más sencillas, y se ilustra en la Figura 3-13. Un ejemplo de uso de esta función sería el de predicción de un valor de ventas, donde se necesita de una regresión lineal para que la red genere un valor único como salida.

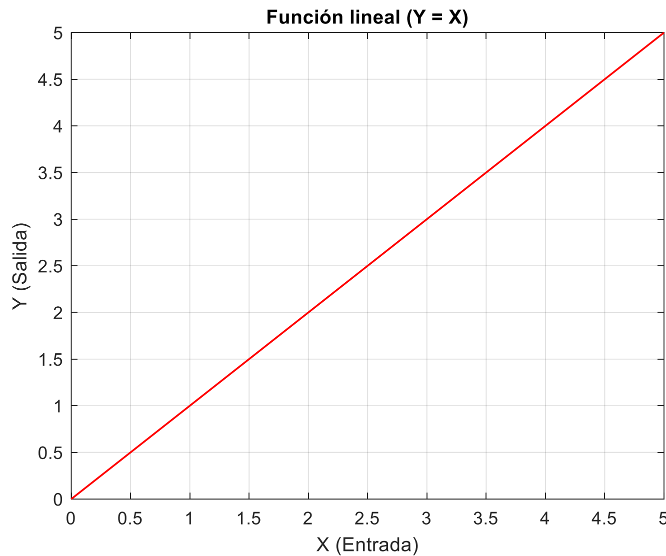


Figura 3-13: Función de activación lineal

3.3.1.2 Umbral

Indica que el valor que se la pasa a la siguiente neurona será '1' si los datos de entrada superan un umbral concreto, o '0' en caso contrario. Es una función escalón básica, y se suele usar para clasificar directamente en 2 clases, o bien cuando las salidas son categóricas. Se ilustrará a continuación en la Figura 3-14 para el caso en que el umbral de entrada es el valor '0':

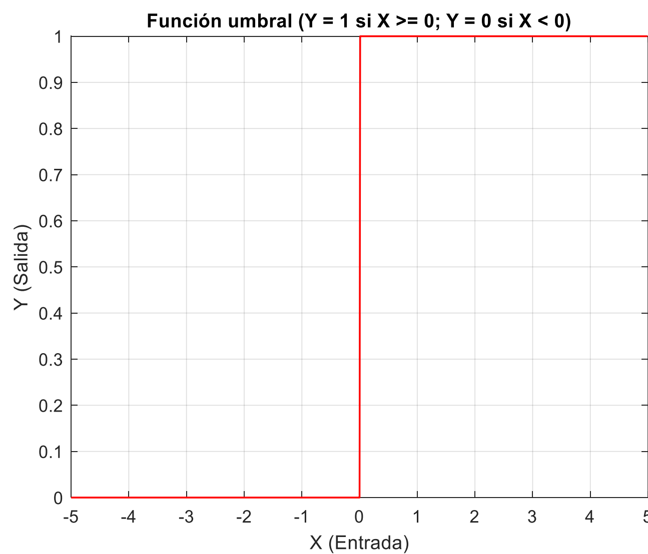


Figura 3-14: Función umbral con umbral = 0

3.3.1.3 Sigmoide

Esta función permite que pequeñas variaciones en la entrada provoquen variaciones muy diferentes en la salida, siendo dichas variaciones poco notables en las zonas extremas de la función, pero muy grandes en comparación en la zona intermedia. La salida está limitada al rango [0,1].

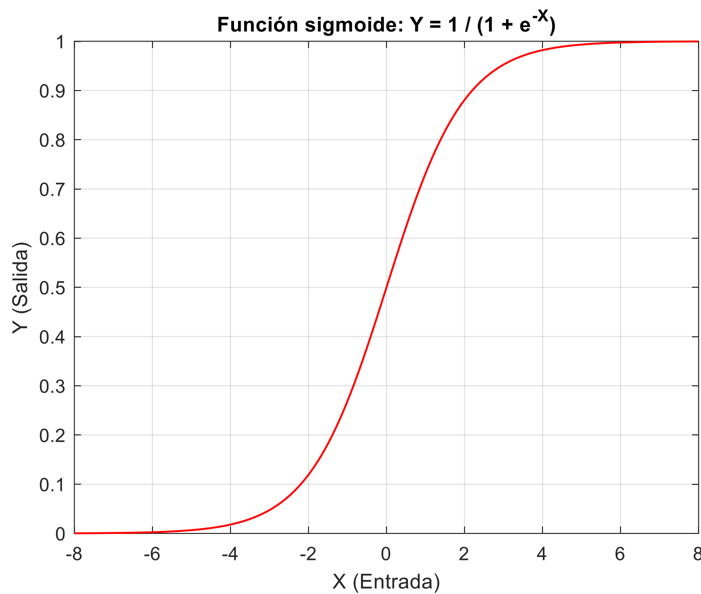


Figura 3-15: Función sigmoide

Se observa en la Figura 3-15 como los valores infinitos tienden a una salida muy cercana a '0' (-∞) o a '1' (+∞), haciendo esos valores tan grandes unos valores posibles de procesar por la red. Esta función de usaba para clasificar en la última capa, ya que se podría interpretar como una probabilidad, pero al no estar centrada eso afecta al aprendizaje, por lo que se ha ido sustituyendo su uso por otras más adecuadas.

3.3.1.4 Tangente hiperbólica

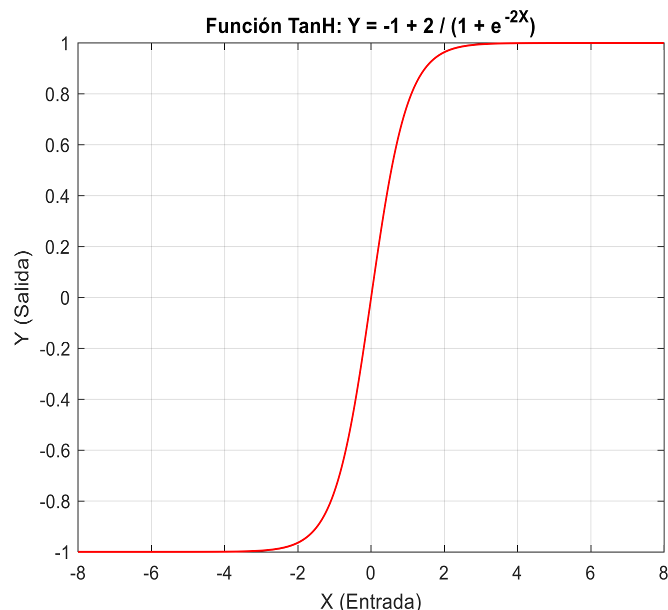


Figura 3-16: Función tangente hiperbólica

Es la relación entre el seno y coseno hiperbólicos, y se parece bastante a la sigmoide, con la diferencia de que, debido a su naturaleza tangencial, su rango de salida está en el intervalo [-1,1], por lo que puede dar valores negativos a la salida que pueden ser bastante útiles en algunos casos. Se puede apreciar en la Figura 3-16.

Esta función está centrada a diferencia de la anterior, pero comparte también el mismo problema, y es que favorece el problema de desaparición del gradiente, si se usan métodos basados en gradiente, haciendo que la red no obtenga un buen aprendizaje por quedar atrapado en un mínimo local, como se verá cuando se expliquen los optimizadores.

3.3.1.5 ReLU

Es una de las más usadas en la actualidad ya que su uso ha resultado muy satisfactorio empíricamente. Lo que pretende es que la salida no cambie de valor hasta que la entrada supere un cierto umbral, pasando entonces como salida el mismo valor que la entrada. Así, podría haber una salida que estuviera a '0' mientras que la entrada cambia, y de repente cuando dicha entrada toma valores por encima de un umbral pasa a tener un comportamiento lineal. Se ilustra en la Figura 3-17.

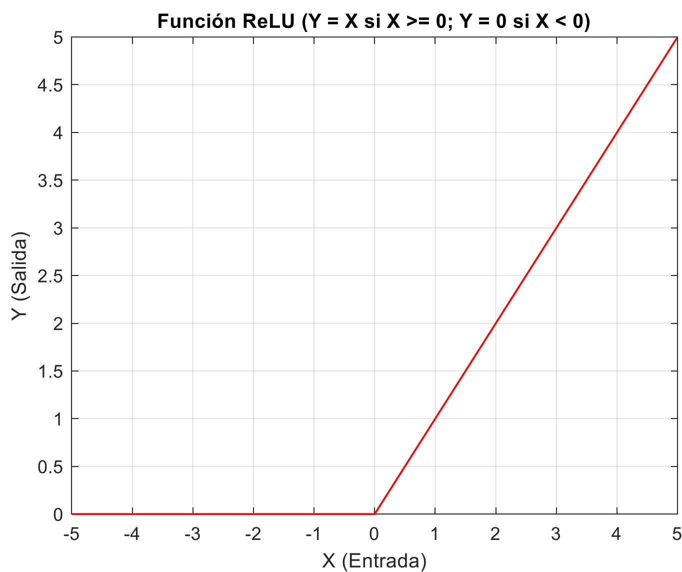


Figura 3-17: Función ReLU con umbral de entrada de valor 0

Si se usan algoritmos de optimización basados en gradiente, puede surgir un problema a la hora de controlar el Dropout (conexión y desconexión aleatorias de neuronas en la fase de aprendizaje para introducir no linealidades en el entrenamiento), si es que se hace uso de esta técnica, ya que se puede dar el caso de que la función valga '0' y que la derivada de dicha función valga también '0', provocando la muerte (desconexión) de neuronas no deseadas. Por eso, hay una variante que en el tramo anterior al lineal introduce una pendiente que es prácticamente '0', pero sin llegar a serlo, evitando así este problema. Dicha variación se aprecia en la Figura 3-18 y es conocida como "Leaky ReLU":

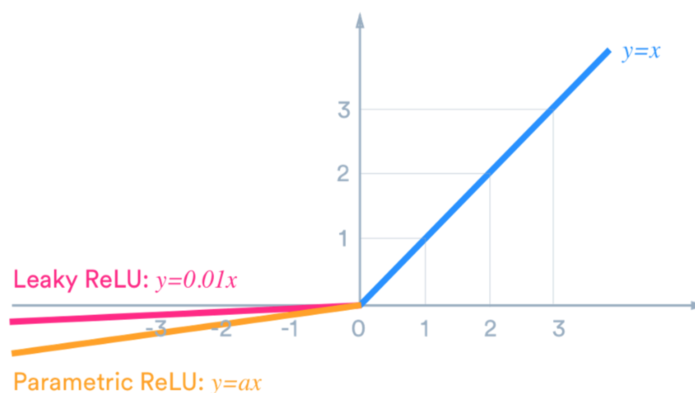


Figura 3-18: Función Leaky ReLU, variación de la ReLU clásica [47]

3.3.1.6 Softmax

Esta función es bastante especial y normalmente irá en la última capa de las redes que se usarán en este proyecto. Es especial porque devuelve el porcentaje de todas las clases posibles, siempre que dichas clases sean mutuamente excluyentes, como un valor entre 0 y 1 según la red crea que los datos de entrada pertenecen a una clase u otra, por lo que se puede usar como capa de decisión de clasificación final. Por tanto, mejora la función sigmoide y tangente hiperbólicas antes comentadas, ya que Softmax introduce varios niveles de decisión, como se ve en la Figura 3-19:

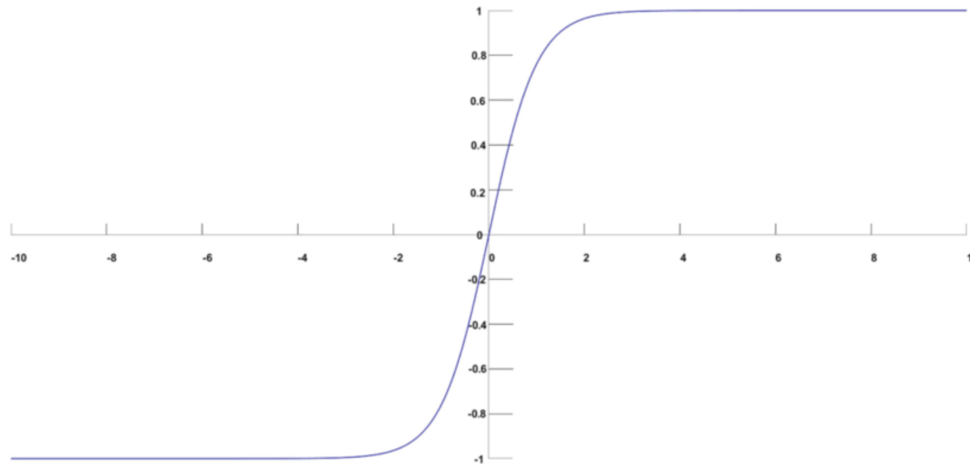


Figura 3-19: Función Softmax [47]

De este modo, si hay una manzana (clase 1), una pera (clase 2) y un plátano (clase 3), es posible que la función Softmax de la última capa devuelva, si la red está bien entrenada, valores como por ejemplo, 0.8 para la clase 1, 0.1 para la clase 2, y 0.1 para la clase 3 si lo que se ha introducido como dato de entrada era una manzana, correspondiente a la clase 1. Lógicamente, la suma de todos los porcentajes de salida debe dar el valor unidad.

3.3.2 Algoritmos de optimización

En este apartado se abordarán los principales algoritmos de optimización, necesarios para que los resultados del entrenamiento sean lo mejores posibles:

3.3.2.1 Gradient Descent

Tal como se cuenta en [39], el algoritmo de gradiente descendente pretende calcular el valor de los pesos de la red para los que el valor de la función de *loss* toma el valor más pequeño posible, porque esto significará que la tasa de fallo de la red será la más pequeña. Básicamente, una función de *loss* es una medida cuantificable del error cometido en el entrenamiento respecto al comportamiento ideal de la red.

Para conseguirlo, hace uso del gradiente (primera derivada) de dicha función para saber dónde se encuentra su valor máximo, de manera que, si el gradiente apunta al valor más alto de la función, el algoritmo ajustará los pesos de modo que se siga el gradiente negativo, es decir, el que apunta al valor de *loss* mínimo, cometiéndose así menor cantidad de error. Este mecanismo se ilustra en la Figura 3-20[18]:

[18] Fuente: http://www.cs.us.es/~fsancho/?e=165

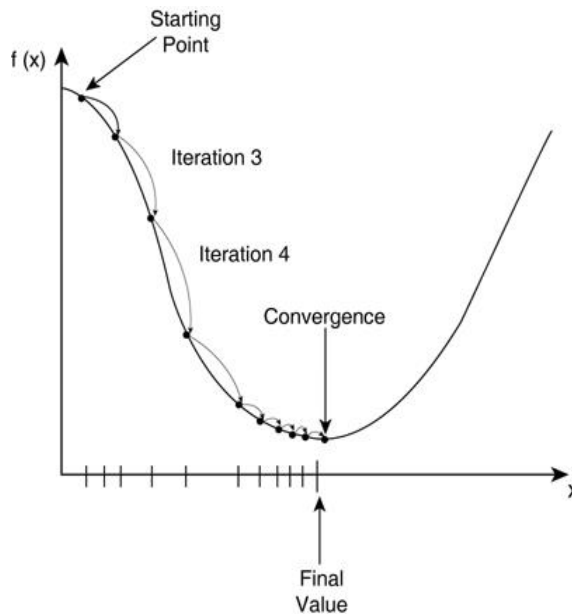


Figura 3-20: Algoritmo de Gradiente Descendente

Se tiene una función de *loss* con un mínimo, y se empieza a iterar desde un punto aleatorio que está en un valor de *loss* alto inicialmente. El gradiente indicará el sentido para el que la función es máxima, por lo que habrá que seguir el sentido contrario, que en este caso es la bajada que hay a la derecha desde el punto inicial hasta el mínimo. Por eso, se cambia el peso en la iteración para que en la siguiente esté más cerca de dicho mínimo. Esto se repetirá hasta que consiga llegar al mínimo o quedarse lo más cerca posible de él.

Aunque esta función es buena, tiene algunos problemas, como que la función de *loss* debe ser derivable para obtener el gradiente, y que puede quedar atrapada en un mínimo local de la función, por lo que es conveniente inicializar los pesos de manera correcta. Otra dificultad añadida es el ajuste del parámetro *learning rate*, que es el paso de avance del algoritmo en cada iteración, y puede calcularse según el gradiente en cada momento, o bien fijarse de antemano. Además de este parámetro, también puede usar el *momentum*, que permite tener en cuenta los gradientes anteriores para darle un pequeño "impulso" al proceso iterativo en caso de que se quede atrapado en un mínimo local (problema de desaparición del gradiente), aunque es una solución drástica por la probable disparidad de los valores de gradientes de anteriores iteraciones.

3.3.2.2 Adaptative Gradient Algorithm (AdaGrad)

Este algoritmo es una modificación del de Gradient Descent clásico, donde la principal novedad, es que adapta el paso, o learning rate, teniendo en cuenta los valores acumulados de los gradientes de iteraciones anteriores para llegar al mínimo más rápido que el algoritmo anterior.

AdaGrad

/adaptive gradient algorithm/

effective learning rate

$$\Delta w_i(t) = -\frac{\eta}{\sqrt{G_i(t)+\epsilon}}\frac{\partial L}{\partial w_i}(t)$$

$$G_i(t) = G_i(t-1) + \left(\frac{\partial L}{\partial w_i}(t)\right)^2$$

with beginning point $G_i(0) = 0$

$G_i\,(0) = 0$

$G_i\,(1) = 0 + \text{non-neg}$

$G_i\,(2) = [0 + \text{non-neg}\,] + \text{non-neg}$

.
.
.

365√DataScience

Figura 3-21: Ecuaciones del algoritmo AdaGrad

Como se aprecia en la Figura 3-21[19], los **pesos de cada neurona** se actualizan en función del gradiente negativo y del learning rate en cada iteración, teniendo el learning rate como denominador el número ∈, valor muy pequeño que asegura que el denominador no sea 0 nunca, y la raíz cuadrada de un parámetro que es la suma del valor en la iteración anterior y el gradiente al cuadrado del instante actual, luego ese valor no será nunca negativo, obteniendo un valor monótonamente creciente. Esto hará que el learning rate sea muy grande al principio y rápidamente se haga muy pequeño, pudiendo quedarse atrapado el algoritmo en un mínimo local.

3.3.2.3 Root Mean Square Propagation (RMSprop)

En base a lo que se dice en [48], este algoritmo es muy parecido al AdaGrad, pero tiene una variación en la estimación del learning rate, y es que ahora no tiene en cuenta el valor del gradiente acumulado, sino que es capaz de variar su percepción para dar más peso a los gradientes de las iteraciones más recientes.

RMSprop

/root mean square propagation/

$$\Delta w_i(t) = -\frac{\eta}{\sqrt{G_i(t)} + \epsilon}\frac{\partial L}{\partial w_i}(t)$$

$$G_i(t) = \beta G_i(t-1) + (1-\beta)\left(\frac{\partial L}{\partial w_i}(t)\right)^2$$

with beginning point $G_i(0) = 0$

Figura 3-22: Ecuaciones del algoritmo RMSprop

En la Figura 3-22[20] se aprecia como las ecuaciones de modificaciones de los pesos de cada neurona son las mismas que en AdaGrad, pero el término del radical en el denominador del learning rate ahora es distinto. Ahora, se multiplica el valor que tenía el término en el estado anterior por el hiperparámetro β, mientras que el cuadrado del gradiente en el estado actual se multiplica por 1-β, siendo β un valor entre 0 y 1.

Es esta adaptación lo que permite que se reduzca la posibilidad de quedar atrapado en un mínimo local, ya que ahora cuando el gradiente es muy pequeño, tiene la posibilidad de aumentarse el escalón para darle un pequeño impulso, solucionando el problema de desaparición del gradiente, tal como explica [49].

3.3.2.4 Gradient Descent with momentum

Se ha visto que el uso del gradiente es bastante útil para realizar la optimización del entrenamiento, pero también acarrea una serie de problemas como la desaparición de dicho gradiente, que produce los encierros en mínimos locales no deseados. Normalmente habrá situaciones donde la función de *loss* sea como las de la Figura 3-23 en lugar de ser como la que se usó en la Figura 3-20 para explicar el Gradient Descent:

[19] https://www.youtube.com/watch?v=gjBOvA5B7ZI

[20] Fuente: Como la de la Figura 3-21

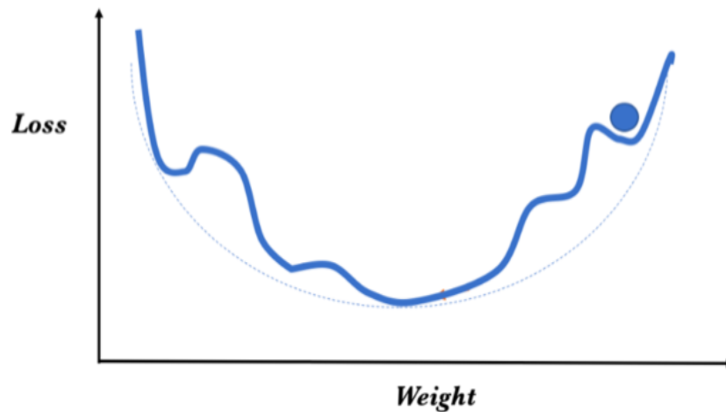


Figura 3-23: Función de *loss* con varios mínimos [39]

En este caso, si solo se usase el gradiente, podría ocurrir que el gradiente se haga '0' al llegar a un mínimo local, de modo que ya no podría salir de ese mínimo y nunca se encontraría el mínimo global, que es el mejor posible. Por ello, se usa el hiperparámetro *momentum*, que permite hacer una media ponderada de los pasos de anteriores iteraciones, de modo que llegado el caso sería posible proporcionar un "empujón" extra que lo saque de dicho mínimo local. El problema de esto es que se les da la misma importancia a todos los gradientes de todas las iteraciones, cuando en realidad hay unos gradientes que tienen mayor relevancia, por lo que se usa el *momentum* para ponderarlos, tomando dicho hiperparámetro un valor entre 0 y 1 en la ponderación. [39]

Es entonces cuando surge el Nesterov Accelerated Gradient (NAG), que permite mejorar el funcionamiento normal del *momentum* permitiendo una combinación lineal entre el gradiente de la iteración anterior y del paso que tuvo en dicha iteración, permitiendo que el paso se reduzca a medida que se acerca al mínimo global deseado, de acuerdo con [48]. Se puede ver una comparación entre ambos métodos de optimización en la Figura 3-24[21]:

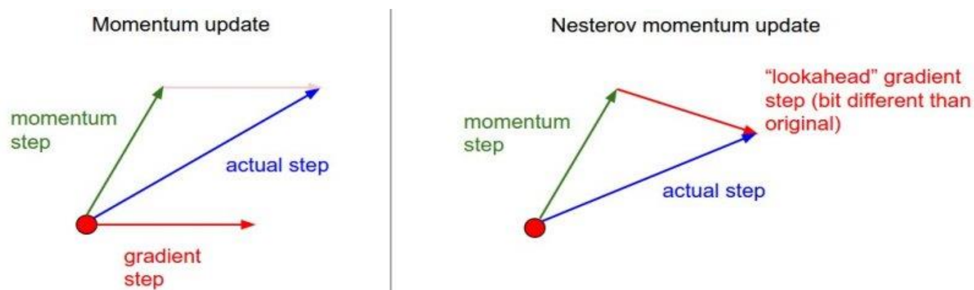


Figura 3-24: Comparación entre la actualización del paso con el momentum clásico y con el Nesterov

3.3.2.5 Adaptative Moment Estimation (Adam)

Este algoritmo mejora al RMSprop anterior, ya que ahora se ha introducido el *momentum* que se acaba de explicar, de modo que usa las 2 fuentes de información disponibles, el gradiente y el *momentum*, para estimar el paso de entrenamiento, asegurando resultados finales de entrenamiento mejores que en los casos anteriores.

[21] Fuente: http://cs231n.github.io/assets/nn3/nesterov.jpeg

Adam

/adaptive moment estimation/

RMSprop

$$\Delta w_i(t) = -\frac{\eta}{\sqrt{G_i(t) + \epsilon}} \frac{\partial L}{\partial w_i}(t)$$

Momentum

$$w \leftarrow w(t) - \eta \frac{\partial L}{\partial w}(t) - \alpha \eta \frac{\partial L}{\partial w}(t-1)$$

$$\Delta w_i(t) = -\frac{\eta}{\sqrt{G_i(t) + \epsilon}} M_i(t)$$

$$M_i(t) = \alpha M_i(t-1) + (1-\alpha) \frac{\partial L}{\partial w_i}(t)$$

$$M_i(0) = 0$$

Figura 3-25: Deducción de la fórmula de Adam a partir del RMSprop y la introducción del hiperparámetro *momentum*

En la Figura 3-25[22] se puede ver que para el paso de cada peso, se obtiene el mismo denominador para el learning rate que se obtenía con el RMSprop, de manera que se sigue distinguiendo el gradiente en cada momento según la relevancia que tenga en el proceso, pero además ahora en lugar de multiplicar el learning rate por el gradiente en el instante actual, se va a multiplicar por otro término que recoge la relevancia de los momentos, de modo que la estructura es igual a la que tenía el término del radical del denominador del RMSprop, pero ahora haciéndolo con los momentos y en lugar de usar el hiperparámetro β se usa el hiperparámetro α, que es el *momentum* y está entre 0 y 1.

Como se especifica en la API de Keras, surgen de éste variaciones como el Adamax, una versión basada en la norma infinita, o el Nadam, que se basa en la visión del algoritmo Adam como un RMSprop usando *Nesterov momentum* en lugar de *momentum* clásico, pero todas se construyen a partir del Adam. Por todo lo anterior, es uno de los algoritmos de optimización más usados hoy en día.

3.4 Planteamiento del problema

Una vez que se conocen todas las herramientas necesarias para empezar a resolver el problema del proyecto, el siguiente paso es explicar en qué consiste exactamente dicho problema. Se quiere diseñar un sistema de clasificación de señales de tráfico para vehículos autónomos que se base en el uso de Visión Artificial implementada con redes neuronales convolucionales de manera que, al proporcionarle una imagen con una señal de tráfico a la red, ésta sea capaz de identificar de qué señal se trata proporcionando al vehículo un estímulo esencial para que pueda actuar en consecuencia con dicha señal. El entorno de programación que se usará será Python con Keras, que usa Tensorflow como back-end.

La red debe ser capaz de distinguir entre 43 tipos de señales de tráfico diferentes, correspondiéndose cada tipo de señal con una clase numérica entre 0 y 42. Las 43 señales a reconocer se pueden ver en la Figura 3-26, donde las clases de 0 a 42 se asignan a cada señal de izquierda a derecha y desde arriba hacia abajo:

[22] Fuente: https://www.youtube.com/watch?v=PXSbJBy52UA

Figura 3-26: Señales de tráfico a clasificar

Cuando esté terminada, la red recibirá una imagen de una de las señales posibles, la irá pasando por diferentes capas, ponderándolas con los parámetros que se hayan obtenido del proceso de entrenamiento, y al final sacará una serie de probabilidades por cada una de las 43 posibilidades, de manera que la que tenga mayor porcentaje será el tipo de señal (o clase) a la que pertenece esa imagen.

En el proceso de entrenamiento es posible adoptar diferentes métricas para saber la bondad del modelo de red neuronal obtenido. En este caso, se eligió la métrica *accuracy* por encima de cualquier otra ya que, aunque en principio podría pensarse que hay algunas señales que tienen más importancia que otras, como la de "STOP" o la de "precaución con los peatones", reflexionando se llegó a la conclusión de que si el vehículo no respetase un giro obligatorio o una rotonda, los daños podrían ser igual de catastróficos, puesto que hay muchas circunstancias que influyen en el tráfico en tiempo real. Por eso, al elegir el *accuracy* se les da la misma importancia a todas las señales de tráfico.

El objetivo de diseño será que, al aplicarle a dicha red el conjunto de test, se obtenga una medida de *accuracy* por encima de un 96 - 97%, garantizando que el margen de error en la identificación de la red será mínimo, y por tanto la seguridad del vehículo autónomo en este apartado será mayor y pondrá en riesgo menor número de vidas. Otra especificación será que la red tenga la rapidez suficiente como para responder a cada imagen. Si se supone un tiempo de llegada por imagen de unos 500 ms, la red debe clasificar la señal de la imagen actual antes de que le llegue otra imagen 500 ms después.

Habiendo explicado el problema y sabiendo qué es lo que hay que solucionar, se pasará a explicar el diseño de la red en cuestión que sea capaz de resolverlo.

3.5 Dataset

Es el conjunto de imágenes elegidas para entrenar, validar y testear la red neuronal convolucional. En este caso, se ha elegido un dataset que ha sido probado y usado antes para entrenar otras redes en la competición IJCNN 2011. Al terminar la competición, se colgó el dataset en Internet y se puede encontrar en la siguiente dirección:

http://benchmark.ini.rub.de/?section=gtsrb&subsection=dataset

Esta batería de datos contiene más de 50,000 imágenes entre todas las imágenes de las 43 señales de tráfico,

siendo cada señal una “categoría” a clasificar por la red.

De acuerdo con el creador de dicho dataset, las imágenes para el entrenamiento se estructuran en una carpeta por cada clase, que contiene a su vez las imágenes de entrenamiento de cada clase y un archivo de extensión .csv como una de tabla de datos donde especifica para cada imagen su nombre, dimensiones, puntos superior izquierdo e inferior derecho de la bounding box que encierra la señal de tráfico de cada imagen, y la clase a la que pertenece cada imagen, o lo que es lo mismo, la señal de tráfico de la que se trata.

Las imágenes se encuentran en formato PPM (Portable Pixmap), varían su tamaño entre un mínimo de 15x15 y un máximo de 250x250 píxeles, sin necesidad de que la imagen sea cuadrada. En general, en estas imágenes no aparece la señal de tráfico únicamente y con enfoque perfecto, sino que se ve el entorno en el que se encuentran y desde orientaciones y enfoques diferentes, pero la señal sí que está siempre centrada en la imagen. Antes de pasárselas a la red será necesario aplicarles un preprocesado.

3.6 Descripción general de las partes de la arquitectura

Una vez que se obtienen las imágenes del dataset anterior, éstas pasan a través de la red para dar finalmente como salida el valor de la clase a la que pertenece la señal que aparece en dicha imagen. La red elaborada consta de 2 partes bien diferenciadas: una etapa de tipo convolucional y otra etapa de tipo densamente conectada, o *Fully Connected*.

3.6.1 Etapa convolucional

Es la etapa por la que esta red es de tipo CNN, y se divide a su vez en capas convolucionales y de pooling, que se explicarán a continuación. También se explicará lo que se obtiene a su salida, que se le pasa a la etapa densamente conectada, y se conocerá como “coding”.

3.6.1.1 Capas convolucionales

Las capas convolucionales son las que conforman la parte principal de este tipo de redes, ya que en cierta manera son las que permiten que se puedan “ver” imágenes. Para conseguirlo, se le pasa como entrada una imagen, y se le aplica un filtro o kernel de manera que se quede con las características más importantes de la imagen, obteniendo a la salida una matriz numérica de menor tamaño que la imagen original. Una imagen no será más que una matriz de números entre 0 y 255 que indican la intensidad del color, teniendo la imagen total una única capa (matriz) que la represente si es una imagen en gama de grises, o 3 matrices si la imagen es en color RGB, una para cada color rojo, verde y azul.

Para que puedan ser interpretadas por la red, las imágenes se pasarán en formato de tensor, de modo que si las imágenes son en color el tensor tendrá dimensiones de ancho y alto y otra de profundidad, donde se pondrá un ‘3’, a diferencia de una imagen en escala de grises, que tendría también esas componentes, pero en la componente de profundidad se pondría un ‘1’ en lugar de un ‘3’.

Todo esto influye a la hora de aplicar los filtros de la capa convolucional ya que, dependiendo del número de canales, en el caso de que fuera en color se deberían aplicar a las 3 matrices de la imagen, por lo que también el filtro sería de profundidad 3. Un filtro o kernel es una máscara que se superpondrá sobre las matrices de la imagen y hará una ponderación con los píxeles que caigan en la máscara, que se moverá de izquierda a derecha y de arriba hacia abajo por la imagen a pasos de un píxel según se ha configurado. Como salida, dará una matriz más pequeña que la original de la imagen como consecuencia de la convolución realizada, donde ya habrá sido capaz de reconocer algunas características.

Cada capa convolucional será capaz de reconocer las características de la imagen a diferentes grados: Una primera capa convolucional podría reconocer líneas rectas verticales, horizontales y oblicuas, la segunda capa líneas curvas, y las capas venideras reconocerán formas más complejas. A las matrices que se van obteniendo en cada convolución se las conoce como mapas de características.

3.6.1.2 Capas de pooling

Las capas pooling en principio podrían no parecer necesarias, ya que en si mismas no poseen la capacidad de "aprender" puesto que no tienen parámetros entrenables, pero se comprueba que resultan de lo más útiles en la práctica y es muy extraño no usarlas en este tipo de redes.

Su funcionamiento pretende básicamente condensar las características reconocidas por las etapas de convolución, reduciendo las dimensiones de su matriz de salida, pero conservando las características más importantes ya detectadas por la etapa de convolución. Esto permite que la matriz a analizar que se le pase a la siguiente etapa de convolución sea menor, disminuyendo el número de parámetros a entrenar de la red, así como su tiempo de entrenamiento, que para redes muy grandes es de agradecer, y todo ello con una pérdida de información mínima.

Hay varios criterios de pooling al pasar la máscara, como por ejemplo hacer la media de los píxeles que caen en dicha máscara, pero en este caso se ha optado por usar el máximo valor del píxel, ya que será la característica que predomine sobre las demás en el tramo de máscara de pooling, y se ha comprobado que da bastantes buenos resultados. Es por todo esto que, tras cada capa de convolución, se pone normalmente una capa de pooling para simplificar los datos obtenidos.

3.6.1.3 Coding

Finalmente, al acabar todas las etapas convolucionales y de pooling que se crean necesarias para el buen aprendizaje de la red, se deben adecuar los datos de la última matriz de características que se haya obtenido para pasarla a la etapa densamente conectada, ya que dicha etapa no es capaz de leer matrices como tal, sino que lee vectores de características. Por tanto, antes de pasar a la siguiente etapa, se convierte la matriz de características a un vector plano que puedan leer las capas siguientes.

3.6.2 Etapa Fully Connected

Esta etapa ya no es específica para imágenes solamente, como lo era la anterior, sino que es la que se usa en otros ámbitos de Deep Learning que no involucran procesamiento y trabajo con imágenes. Se compone de 3 subetapas:

- Una capa de entrada, donde las neuronas reciben el vector de características obtenidos de la capa convolucional, que sería el coding.
- Capas ocultas, donde hay diversas capas de neuronas interconectadas todas unas con otras de modo que van ajustando sus pesos para reconocer dichas características y clasificarlas finalmente.
- Una capa de salida, que en este caso usa la función Softmax para hacer clasificación multiclase, y permite clasificar finalmente a qué clase pertenece el vector de características de la imagen que se le pasó a la entrada.

En esta etapa, las neuronas de una capa están conectadas con todas las neuronas de la capa anterior, y a su vez con todas las de la capa siguiente. El número de neuronas de las capas inicial e intermedia es variable y se deberá encontrar una configuración de neuronas tal que se mantenga la relación cantidad/calidad de estimación, puesto que si se ponen muchas neuronas se clasificará mejor, pero puede tardar muchísimo más el entrenamiento y hay riesgo de overfitting o sobreentrenamiento, y si se eligen pocas neuronas el resultado será deficiente. Igualmente, habría que seguir la misma metodología de elección con la cantidad de capas ocultas.

Finalmente, después de pasar por todas las neuronas de las capas ocultas se llega a la capa de salida. En este momento es realmente cuando se clasifican las características de entrada y se dictamina a qué clase pertenecen. Para conseguirlo, se elabora una capa con la función Softmax. Se compone de un número de neuronas igual al número de clases que se pretende identificar, siendo en este caso de 43 neuronas, ya que hay 43 señales de tráfico que se desean reconocer, de modo que al pasar la información de cada neurona por la función Softmax, dicha capa devuelve como salida de cada neurona el porcentaje de dichas características que la red considera que pertenezcan a la clase correspondiente de cada neurona. El valor de los porcentajes que devuelve está entre 0 y 1, siendo la suma de los porcentajes de las 43 salidas el valor 1, correspondiente al 100%.

De este modo, se puede reconocer qué señal de tráfico es la que hay en la imagen usando la probabilidad más

alta de las que salgan al final, dando como resultado la clase a la que es más probable que pertenezca.

3.6.3 Arquitectura final de la red

Combinando las dos etapas anteriores se ha elaborado una arquitectura que consta de una capa de entrada de 64x64 píxeles, a la que se le aplica una convolución de ventana 3x3, obteniendo una capa convolucional de 32 filtros con dimensiones de 62x62 cada uno, seguido de un maxpooling de 2x2 de tamaño de ventana para obtener una capa con los 32 filtros anteriores disminuyendo sus dimensiones a la mitad, dejándolas en 31x31 píxeles.

A continuación, se hace otra convolución y luego otro maxpooling, ambas operaciones de tamaño de ventana 2x2, y se obtiene primero una capa convolucional de 64 filtros de tamaño 30x30 píxeles, y se reducen las dimensiones de esos 4 filtros de 30x30 a 15x15 en la capa de pooling. Ahora se alternan otras 2 convoluciones de 3x3 píxeles de tamaño de ventana, combinadas cada una con una operación de maxpooling de tamaño 2x2 píxeles como en los casos anteriores, obteniendo una capa convolucional de 128 filtros de 13x13 píxeles seguida de una capa pooling que reduce los filtros a un tamaño de 6x6 píxeles, y finalmente otra etapa convolucional de 256 filtros de 4x4 píxeles seguida de una capa pooling que reduce esos filtros a un tamaño de 2x2 píxeles.

Con lo anterior quedaría completada la etapa convolucional diseñada para la red. Lo siguiente que habría que hacer sería conectar dicha etapa con la etapa densamente conectada, pero para ello habría que pasarle la información recopilada hasta ahora por la etapa convolucional de una forma que la etapa densamente conectada pueda procesarla. Como la última capa de la etapa convolucional tiene tamaño 2x2 y 256 filtros de un canal, la etapa de entrada a la red deberá tener 2x2x256 = 1024 neuronas, luego esos 1024 datos que tiene esa última capa almacenados en matrices se ponen en formato de vector para que puedan ser recogidos por la capa de neuronas densas.

La etapa fully-connected tendrá una capa de entrada de 1024 neuronas como ya se ha comentado, 3 capas ocultas, y una última capa de salida que usará la función Softmax. Las neuronas de las capas ocultas irán decayendo en número a medida que se acerquen a la capa de salida, a modo de embudo, pero no disminuirán de manera aleatoria, sino que se ha diseñado de forma que el número de neuronas de una capa sea la mitad del de la capa anterior. Por eso, si la capa de entrada tiene 1024 neuronas, la primera capa oculta tendrá 512 neuronas, la segunda capa oculta tendrá 256 y la tercera 128. Este ciclo se parte en la última capa, que al usar Softmax como función de activación deberá tener una neurona por cada clase que se desea clasificar, en este caso, tendrá entonces 43 neuronas.

Finalmente, hay que comentar que las funciones de activación usadas en cada capa de la red serán todas del tipo ReLU, ya que es de las que mejores funcionan por introducir la no linealidad en la red, como ya se explicó anteriormente. La estructura aquí comentada se puede apreciar en la Figura 3-27, donde se muestra un esquema del modelo diseñado, y en la Figura 3-28, que muestra el modelo en Keras de dicha red:

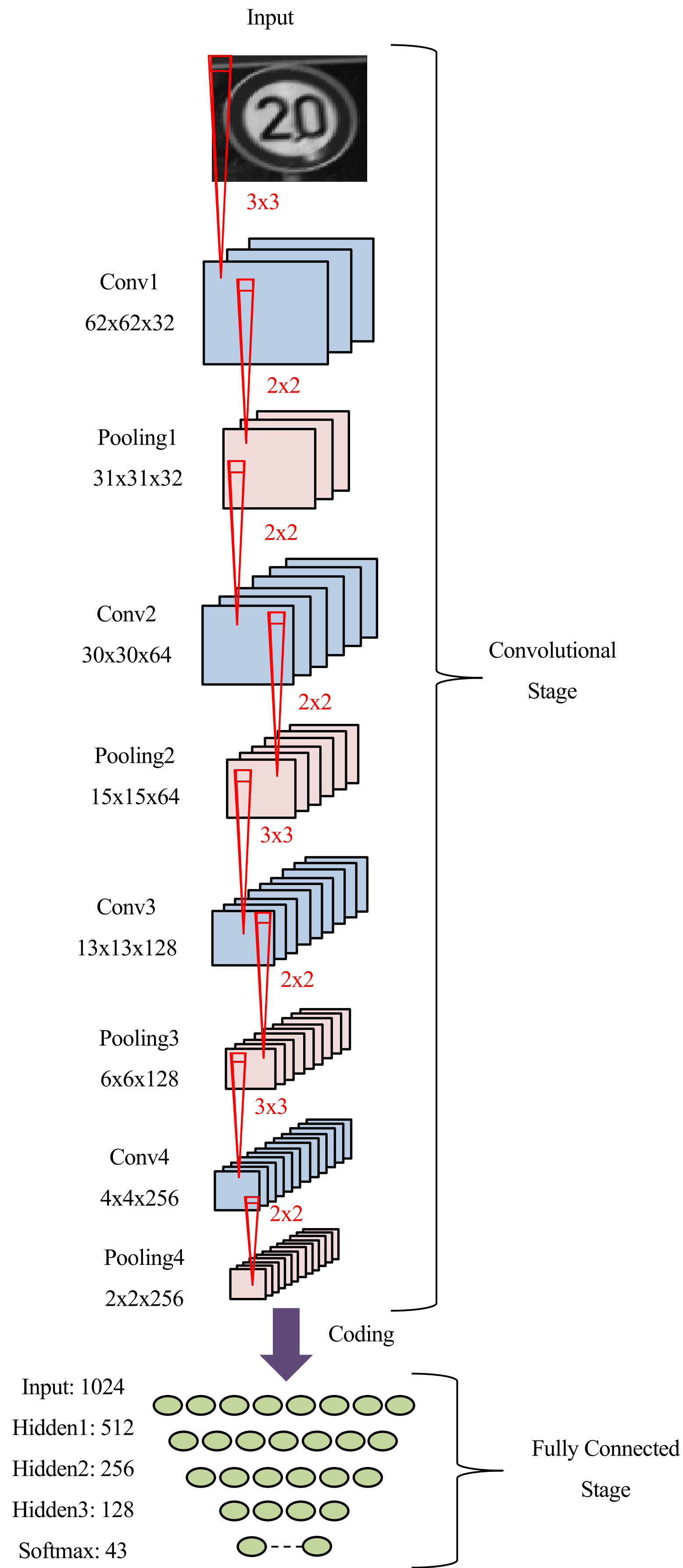


Figura 3-27: Esquema de la red diseñada

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
Conv1_Layer (Conv2D)         (None, 62, 62, 32)        320
_________________________________________________________________
Pooling1_Layer (MaxPooling2D (None, 31, 31, 32)        0
_________________________________________________________________
Conv2_Layer (Conv2D)         (None, 30, 30, 64)        8256
_________________________________________________________________
Pooling2_Layer (MaxPooling2D (None, 15, 15, 64)        0
_________________________________________________________________
Conv3_Layer (Conv2D)         (None, 13, 13, 128)       73856
_________________________________________________________________
Pooling3_Layer (MaxPooling2D (None, 6, 6, 128)         0
_________________________________________________________________
Conv4_Layer (Conv2D)         (None, 4, 4, 256)         295168
_________________________________________________________________
Pooling4_Layer (MaxPooling2D (None, 2, 2, 256)         0
_________________________________________________________________
flatten (Flatten)            (None, 1024)              0
_________________________________________________________________
Input_Dense (Dense)          (None, 1024)              1049600
_________________________________________________________________
Dense_Hidden_Layer1 (Dense)  (None, 512)               524800
_________________________________________________________________
Dense_Hidden_Layer2 (Dense)  (None, 256)               131328
_________________________________________________________________
Dense_Hidden_Layer3 (Dense)  (None, 128)               32896
_________________________________________________________________
Softmax (Dense)              (None, 43)                5547
=================================================================
Total params: 2,121,771
Trainable params: 2,121,771
Non-trainable params: 0
```

Figura 3-28: Modelo de la red desarrollada en Keras

Se observa que el número total de parámetros entrenables de la red es de 2,121,711 parámetros, de los cuales no hay ninguno en las etapas de pooling ni en la etapa de transición desde la etapa convolucional a la densamente conectada, ya que dicha etapa es únicamente una reordenación de los datos ya obtenidos, no se detectan nuevas características, y en las etapas de pooling ocurre lo mismo, es simplemente una reducción del tamaño de la matriz de datos con la mínima pérdida de información posible, pero tampoco son capaces de detectar nuevas características.

3.7 Procesos de Entrenamiento, Validación y Testeo

En esta sección se explicará el proceso de entrenamiento para enseñar a la red neuronal descrita en la sección anterior, así como el proceso de validación para tener una idea anterior al testeo de cuánto más hay que afinar los parámetros obtenidos del entrenamiento, y el proceso de testeo para ver realmente cómo de bien clasifica las señales la red neuronal. Se explicará aquí la metodología seguida, y se presentarán los resultados en el siguiente capítulo.

3.7.1 Entrenamiento

La red elaborada será de aprendizaje supervisado, es decir, en el entrenamiento se sabe a priori de qué clase es la imagen que se usa para el aprendizaje en cada momento. Esto ya está contemplado en el dataset que se definió en el apartado anterior, de modo que se dividirán las imágenes en 2 partes, siendo una para el entrenamiento y otra para test. Los porcentajes para cada parte según la división que se realizó en el experimento original en que se usó el dataset se recogen en la Tabla 3-1:

Tabla 3-1: División de imágenes del dataset para experimentos de training y test

Nº DE IMÁGENES	39,209
ENTRENAMIENTO	26,640 (68 %)
TEST	12,569 (32 %)

Además, antes de pasarle las imágenes a la red, se debe hacer un preprocesado de dicha imagen, ya que:

- La red debe tener un tamaño de imagen de entrada fijo y en este caso las imágenes varían su tamaño.
- Se recomienda escalar los valores de intensidad de las imágenes entre 0 y 1 para que la red procese mejor los datos.

En primer lugar, se definió que el tamaño de todas las imágenes debía ser de 64x64 píxeles, por lo que se reescalaron las imágenes interpolando, de modo que se obtenían todas las imágenes del dataset en dicho tamaño porque fue el elegido como tamaño de la capa de entrada a la red neuronal.

Una vez obtenidas las imágenes redimensionadas, se hizo un escalado dividiendo todos los píxeles de las matrices de la imagen entre 255 para que su valor quedase confinado al intervalo [0,1], y finalmente, una vez que se hizo todo lo anterior ya se obtuvieron las imágenes de una forma adecuada para pasarlas por la red.

El entrenamiento usará "back propagation", es decir, los parámetros de la red toman unos valores al pasarle la imagen, pero esos valores no serían los definitivos, ya que una vez que llega a la última capa y predice qué señal es, comparará si el resultado obtenido concuerdo con la etiqueta que tenía dicha imagen, y recorrerá las capas de toda la red empezando desde la última, en este caso la Softmax, y llegando a la capa de entrada. De esta forma, los parámetros consiguen un mejor ajuste en este último recorrido haciendo la predicción más robusta para ese caso.

Hay varios parámetros en los que fijarse en el proceso de entrenamiento, de los cuales se han elegido en principio dos de ellos:

- *Accuracy*: Este parámetro indica el porcentaje de muestras que han sido clasificadas correctamente respecto a todas las muestras usadas.
- *Loss*: Es un valor que indica el nivel de error de la red a la hora de clasificar las muestras usadas en el entrenamiento.

Lo que se pretende es conseguir el valor más alto de *accuracy*, con el mínimo valor de *loss* posible. Para conseguir esto, el entrenamiento usará un algoritmo de optimización, como el de gradiente descendente (SGD), Adam, Adagrad, RMSProp, etc, que ya fueron explicados. Se probarán algunos de ellos y se verá con cuáles se obtienen mejores resultados.

La función de *loss* se ha elegido que sea categórica, de modo que se codificarán las etiquetas usando "one hot encoding", que consiste en que el valor de cada etiqueta se representará como un número que tiene igual número de cifras que elementos a identificar, en este caso 43 porque hay 43 señales de tráfico correspondiendo cada una a una clase del dataset, y en la posición que le corresponda a dicha etiqueta se pondrá ese valor a '1' y todos los demás a '0'.

Se define también el tamaño del batch de entrenamiento y el número de épocas. El batch es simplemente el

número de imágenes que se le pasan a la red de forma simultánea, y las épocas son el número de repeticiones que realiza del proceso de entrenamiento usando en cada repetición todo el conjunto de datos de entrenamiento. Esto se debe a que es muy complicado que con solo una repetición se obtengan los resultados deseados ya que los pesos de las redes no quedarán suficientemente bien ajustados, obteniendo valores bajos de *accuracy* y altos de *loss*.

Sin embargo, abusar del número de repeticiones para obtener buenos resultados suele traer aparejado un problema que se conoce como overfitting. Cuando se hacen muchas repeticiones, los ajustes de parámetros parecen ser casi perfectos y ajustarse para reconocer todas las imágenes de nuestro conjunto de entrenamiento, pero el problema viene cuando se le pasa una imagen que no conoce, haciendo entonces una predicción peor de lo que debería para todo el entrenamiento usado. Esto se debe a que los pesos se han ajustado exclusivamente para reconocer las imágenes del conjunto de entrenamiento, y es tan perfecto que cuando se le pasa una imagen fuera de ese conjunto no es capaz de reconocer bien lo que se pide porque no es exactamente como el dato de entrenamiento. Esto no es aceptable, por lo que se intenta elegir el número de épocas suficientes para que se ajuste bien al conjunto, pero sin que se aprenda de memoria las imágenes de dicho conjunto. Se explicará como detectar este problema antes del testeo cuando se explique el proceso de validación.

Antes de terminar este bloque, habría que explicar una técnica de mejora del entrenamiento que se ha usado también en los experimentos, y no es otra que el Dropout. Según [50], básicamente consiste en desconectar cierto número de neuronas en las capas de manera aleatoria, de modo que su aprendizaje se ve interrumpido en esa iteración, o época, del proceso de entrenamiento. Lo que se consigue con esto es prevenir al modelo del sobreentrenamiento (*overfitting*), ya que los pesos de las neuronas se actualizan cada vez de manera más independiente de las demás neuronas. Gráficamente, se puede ver en la Figura 3-29.

Para controlar el Dropout, se tiene un parámetro que es una probabilidad que indica el número de neuronas que se mantendrán activas en cada iteración, de modo que las que se desactiven no recibirán los datos de la capa anterior, sino que recibirá un 0, como se aprecia en la Figura 3-30, provocando que las neuronas le den aproximadamente la misma importancia a todas las características de manera independiente en lugar de hacer que una predomine sobre el resto y, por tanto, la sobreaprenda.

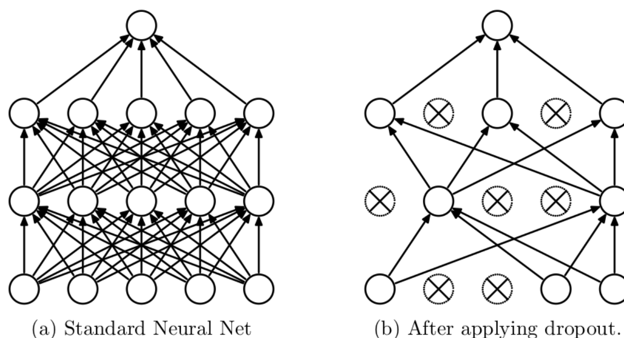


Figura 3-29: Dropout [50]

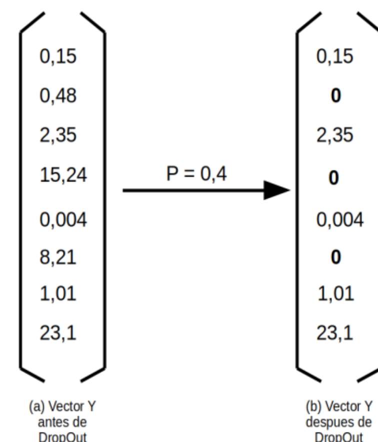


Figura 3-30: Funcionamiento de Dropout [50]

Una vez que se han definido las métricas y mejoras de la metodología de aprendizaje que se van a usar, comienza el proceso de entrenamiento pasándole imágenes a la red de manera que se obtendrá un valor de *accuracy* y un valor de *loss* del proceso de entrenamiento.

3.7.2 Validación

Este proceso actúa como complemento del proceso de entrenamiento, de manera que antes de realizarlo, se

dividirá de nuevo el subconjunto de entrenamiento para sacar también un subconjunto de validación. El subconjunto de test se decidió dejar igual en principio. Ahora la distribución del dataset será como se recoge en la Tabla 3-2:

Tabla 3-2: Distribución final de imágenes del dataset

Nº DE IMÁGENES	39,209
ENTRENAMIENTO	19,536 (50%)
VALIDACIÓN	7,104 (18 %)
TEST	12,569 (32 %)

La validación se usa para conseguir un mejor ajuste de los pesos antes de la etapa de test y es muy útil para controlar el *overfitting* mencionado en el entrenamiento. Para ello, la validación se hará tras cada época del entrenamiento de modo que, al acabar una época, se le pasan los datos de validación obteniendo también un valor de *loss* y *accuracy* para la validación que se pueden usar para depurar el diseño antes de probarlo en el conjunto de test.

Si se produce *overfitting*, al representar los valores de *loss* respecto al número de épocas realizadas, los valores obtenidos para el entrenamiento serán muy pequeños, mientras que los de la validación empezarán a crecer llegados a cierta época. Esto indica que será muy bueno con el conjunto de entrenamiento pero que empezará a predecir mal los datos de validación a partir de dicho valor de época, viéndose así que se produce *overfitting*. Por tanto, lo mejor sería elegir ese valor de número de épocas para el que la *loss* de validación tenga un valor mínimo, obteniendo de esta forma el mejor resultado posible de la red.

También se puede usar para comparar las *accuracy* de ambos procesos, de modo que cuando ambos tengan valores parecidos, se habrá alcanzado el mejor valor posible, y todo lo que se obtenga por encima de dicho valor realizando más épocas puede llegar a producir *overfitting*.

Una vez visto lo importante que es la validación a la hora de conseguir buenos resultados, y teniendo estos procesos ya realizados, la red habrá quedado totalmente entrenada y lista para predecir cualquier valor que se le pase. Para ver la bondad de la red, hace falta una última etapa, conocida como el proceso de test.

3.7.3 Testeo

Es el último paso antes de finalizar el diseño de la red. El proceso de test es el que determinará si lo hecho en los procesos de entrenamiento y validación es aceptable para la aplicación en cuestión o no.

En este caso concreto, es bastante importante obtener una red con un porcentaje de aciertos altísimo, ya que hay en juego vidas humanas, puesto que si se equivoca al detectar una señal y el coche realiza una acción acorde a dicha detección errónea, puede producir un accidente de tráfico poniendo en juego la integridad del coche, así como las vidas humanas del interior de ese vehículo y del resto que circulen por la carretera al mismo tiempo.

Por tanto, si los resultados obtenidos en el test no son satisfactorios, habría que rediseñar la estructura interna de la red, ya sea añadiendo o quitando capas tanto de la parte convolucional como de la parte fully-connected, ajustando hiperparámetros, o cualquier otro método que haga que la red sea tal que cumpla las especificaciones apropiadas.

Se le pasa a la red el subconjunto de test del dataset, y con los resultados de las clasificaciones para cada imagen, se realizará lo que se conoce como matriz de confusión, donde se indica en cada fila y columna el número de imágenes de test bien y mal clasificadas, teniendo como número de filas y columnas el número de señales de tráfico posibles. En las componentes fila y columna que correspondan a una misma señal, aparecerá el número de imágenes bien clasificadas para dicha señal, y en las demás componentes de esa fila, aparecerá el número de imágenes que se sabe que son señales del tipo que indica esa fila, pero la red los ha clasificado como si fueran

del tipo que indica esa columna. Su estructura queda aclarada en la Figura 3-31:

		Predicción	
		Positivos	Negativos
Observación	Positivos	Verdaderos Positivos (VP)	Falsos Negativos (FN)
	Negativos	Falsos Positivos (FP)	Verdaderos Negativos (VN)

Figura 3-31: Matriz de confusión genérica [39]

Una vez obtenida esta matriz, se puede calcular el valor del *accuracy*, *loss*, y precisión del modelo, y ver si cumple especificaciones o hay que rehacer de nuevo el modelo de la red.

3.8 Comentarios finales

Cuando se empezó este capítulo, no se tenía más que una idea de la historia de las redes neuronales, así como del panorama actual en el que se encontraban, tal como se vió en los capítulos 1 y 2. En este capítulo se han proporcionado las herramientas necesarias para el diseño de redes neuronales convolucionales, explicando qué es el Machine Learning, viendo el lugar que ocupa dentro del campo de la Inteligencia Artificial, y a su vez se ha visto el lugar que ocupa el Deep Learning dentro del Machine Learning.

Se han visto también los tipos de redes neuronales que hay, haciendo dos clasificaciones: por topología, atendiendo al número de capas de la red, y por método de aprendizaje, según se usara aprendizaje supervisado, no supervisado o aprendizaje por refuerzo, viendo las características de cada uno de ellos.

También se ha indagado en los diferentes tipos de funciones de activación que hacen posible la obtención de salidas en las neuronas de las capas de la red e ilustrado sus diferentes características, empezando por la función lineal, pasando por la sigmoide, hasta llegar a la función Softmax. Además de esto, se analizaron algunos de los algoritmos de optimización que se usan en el entrenamiento de redes, permitiendo que dicho entrenamiento se lleve a cabo en la menor cantidad de tiempo posible obteniendo los mejores resultados de entrenamiento.

Con todo eso, se diseñó una red que pretendía solucionar el problema abordado, por lo que en el siguiente capítulo se realizarán y detallarán los distintos experimentos realizados con el fin de obtener los mejores resultados posibles.

4 RESULTADOS

¿Heredarán los robots la Tierra? Sí, pero serán nuestros hijos.

- Marvin Minsky -

En este capítulo se presentarán los resultados obtenidos para las redes desarrolladas según la metodología explicada en el anterior capítulo, pudiendo así comparar las diferentes opciones para ver cuál sería la opción con la que se obtienen mejores resultados.

En primer lugar, se comentará la implementación realizada por la red, explicando algunas de las librerías usadas para desarrollar el código en Python, y se dará acceso a dichos códigos para todo aquel que quiera probarlos. Además, se explicarán de manera formal las métricas utilizadas para evaluar la bondad de la red.

Se explicará el preprocesado que ha sufrido el dataset antes de empezar los procesos de entrenamiento de la red, y una vez que se obtienen las imágenes de las señales de tráfico en la forma deseada se empezarán a realizar experimentos para obtener una red que cumpla las especificaciones. De esas especificaciones, la más importante es que la red tenga un porcentaje de *accuracy* en el conjunto de test como mínimo de un 96%, con posibilidad de mejora. Menos de un 96 - 97% no es aceptable porque se ponen en mayor riesgo a las vidas humanas de los usuarios de este sistema de ayuda en la conducción.

Para conseguirlo, se usarán diferentes tipos de optimizadores para ver cuál funciona mejor, y se probarán técnicas como el Dropout para intentar mejorar los porcentajes de acierto de las mejores redes conseguidas en los primeros experimentos. Finalmente, se elegirá una red de las obtenidas por encima de las demás, que será la que solucione el problema de clasificación de señales de tráfico.

4.1 Implementación del diseño

El desarrollo de la red se ha implementado en un código de Python, usando diferentes librerías. La primera de ellas es la librería de *keras*. Aquí se incluyen todos los elementos necesarios para el desarrollo de redes neuronales, como los diferentes tipos de capas, métodos de entrenamiento, validación y test. Contenidas en *keras*, se han usado las librerías *layers* para implementar las capas, y *models* para unir dichas capas en un modelo único, así como para guardar los pesos y estructura de la red diseñada.

También se ha usado la librería *sklearn* para obtener algunos resultados de métricas de test, así como matrices de confusión de cada experimento realizado. Lo anterior se hace respectivamente con *classification_report* Lo anterior se hace respectivamente con *classification_report* y *confusión_matrix*.

Para trabajar con carpetas de directorio se usaron *pathlib* para crear dichos directorios, y *sutil* para copiar carpetas al directorio creado. Se usó a la hora de subdividir el dataset en 3 partes. Los archivos del dataset que contienen las etiquetas de cada imagen y su nombre son de tipo csv, por lo que se ha usado la librería *csv* para poder leer los archivos correspondientes y poder obtener las etiquetas que corresponden a cada imagen.

En cuanto a preprocesado de imagen, se optó por usar la librería OpenCV, declarada como *cv2*, que permitió el cambio de escala de la imagen de escala RGB a escala de grises, y el redimensionado de la imagen a un tamaño de 64x64 píxeles. Para ir visualizando los resultados de los experimentos a medida que se van haciendo se ha usado la librería *matplotlib* que permite sacar gráficas por pantalla, así como las imágenes del dataset.

Por último, para poder trabajar con las imágenes de manera más sencilla y de manera que la red pueda procesarlas, se usó la librería *numpy* que permite trabajar con vectores y matrices, pudiendo expresar dichas imágenes como matrices interpretables por la red.

Estas serían las librerías más importantes que se han usado en el proyecto. Si se desean comprobar los resultados por cuenta propia, o simplemente usar el código o ver su funcionamiento de manera interna, se proporciona el siguiente enlace a GitHub con todos los archivos utilizados: https://github.com/emoscosio/Clasificador-con-redes-convolucionales

En cuanto a las métricas empleadas, la primera de todas fue el *accuracy*. Tal como se dice en [39], dicha métrica se define como la relación existente entre el número de predicciones que el modelo ha hecho correctamente frente al número total de predicciones. Con una matriz de confusión como la de la Figura 3-31, la fórmula que describe esta relación sería la siguiente:

$$\text{accuracy} = \frac{VP + VN}{VP + FP + VN + FN}$$

Donde VP es el número de verdaderos positivos, VN el número de verdaderos negativos, FP el número de falsos positivos, y FN el número de falsos negativos. Sin embargo, el *accuracy* no siempre es suficiente, por lo que hay otro tipo de métricas centradas en otros aspectos, como se indica en [51]. Aquí se detallan algunas métricas, como la *precision*. También se conoce como "Valor Predictivo Positivo", y mide el cociente de los verdaderos positivos que registra el modelo frente a todas las predicciones positivas de dicho modelo. Es una estimación del número de veces que el modelo consigue acertar respecto al número de veces que cree que ha aceptado, como indica la siguiente ecuación:

$$\text{precision} = \frac{VP}{VP + FP}$$

Otra de las métricas más usadas es la sensibilidad o *recall*. Es parecida a la anterior, con la diferencia que ahora da una medida del número de veces que la red detecta positivos reales respecto a todos los positivos que hay realmente. Se calcula con la ecuación siguiente:

$$\text{recall} = \frac{VP}{VP + FN}$$

La diferencia, por tanto, entre *precision* y *recall* sería *precision* se evalúa usando lo que el modelo cree que ha acertado como positivo, mientras que *recall* se evalúa usando realmente lo que es positivo. Por eso, en la primera se usan los falsos positivos (FP) y en la segunda los falsos negativos (FN) en el denominador de las respectivas relaciones métricas. Existe una métrica que engloba las dos anteriores, y se conoce como *F1*. Es el promedio armónico de esas dos, y se representa como el cociente entre *recall* y la suma de *recall* y *precision*:

$$\text{F1} = \frac{recall}{precision + recall}$$

Estas serían algunas de las métricas que existen y las que se emplearan en el proyecto. Para tener una idea de la

bondad general del modelo, en entrenamiento y validación se usará únicamente el *accuracy*, pero una vez obtenidos los modelos, se examinarán en los resultados de test las métricas de *precision* y *recall* para cada clase, además de obtener también el *accuracy* general del modelo en cuestión.

4.2 Preprocesado de las imágenes del dataset

El dataset original ha sufrido ciertas modificaciones. En primer lugar, se crearon 3 carpetas de datos en lugar de 2 como había originalmente. Las originales correspondían una al proceso de entrenamiento y otra al proceso de test. Sin embargo, no había una con imágenes destinadas a la validación, por lo que se dividió de nuevo el número de imágenes de entrenamiento para sacar de ahí un cierto número de imágenes que se usasen en la validación, como ya se comentó en el anterior capítulo.

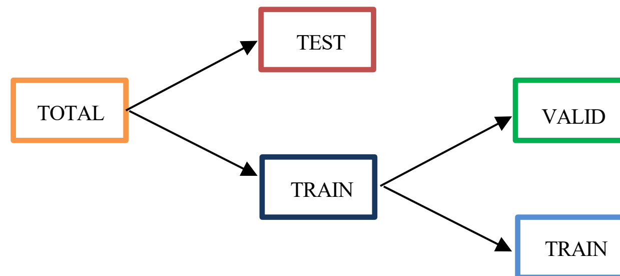


Figura 4-1: Etapas de división de imágenes del dataset

Una vez obtenido el dataset que se va a emplear en cada etapa, hay que tener en cuenta que la capa de entrada a la red debe ser de un tamaño fijo, pero las imágenes del dataset son de diferentes tamaños como ya se comentó en el capítulo anterior. Por ello, hay que redimensionarlas para que todas las imágenes sean del mismo tamaño, y se eligió que la entrada a la red tuviera un tamaño de 64x64, por lo que todas las imágenes se redimensionaron a 64x64 píxeles, como la de la Figura 4-2.

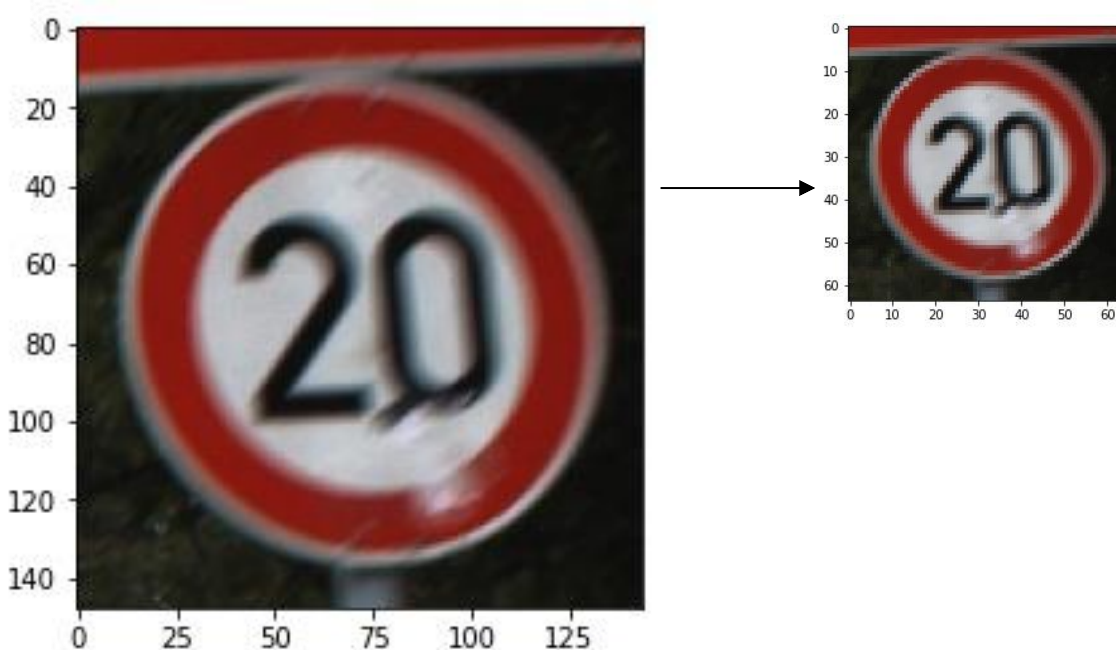


Figura 4-2: Redimensión de imágenes a 64x64 píxeles

Una vez que se redimensionaron las imágenes, se decidió pasarlas también a escala de grises en lugar de dejarlas en RGB como son originalmente, de la forma que indica la Figura 4-3. Esto permite reducir la entrada de la red de tener 3 canales de datos de entrada (R, G y B) a un solo canal, permitiendo que el entrenamiento sea más rápido al tener una menor cantidad de parámetros.

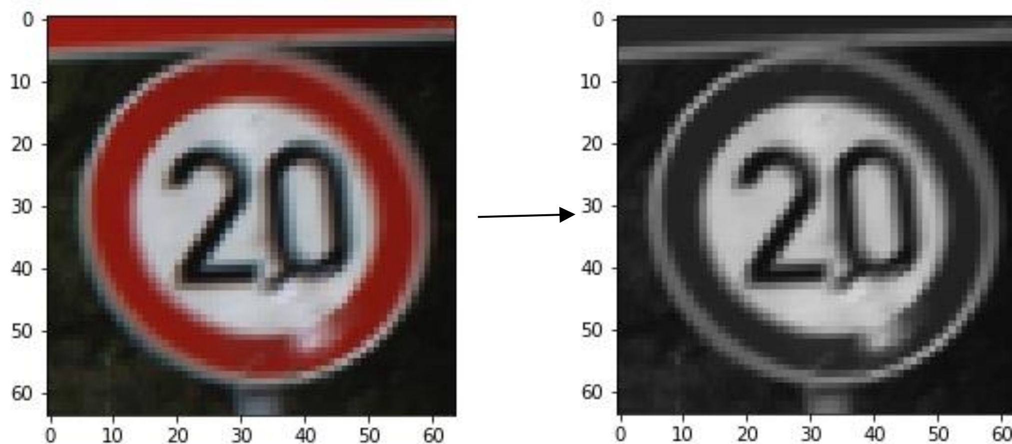


Figura 4-3: Conversión a escala de grises

Cuando se tuvieron las imágenes en el tamaño y formato de color deseados, se normalizaron los valores de los píxeles para que se encontrasen en el rango de 0 a 1 en lugar de estar en el rango de 0 a 255. Esto se hace para que la red no tenga problemas numéricos durante los cálculos matriciales del entrenamiento. Se ve como quedaría la imagen de las anteriores figuras al final en la Figura 4-4.

```
array([[ 37,  38,  39, ...,  38,  32,  32],
       [ 37,  37,  38, ...,  50,  51,  57],
       [ 38,  38,  37, ...,  93,  98, 100],
       ...,
       [ 33,  35,  31, ...,  15,  15,  17],
       [ 38,  39,  35, ...,  15,  16,  17],
       [ 31,  31,  33, ...,  16,  19,  20]], dtype=uint8)
```

```
array([[0.14509805, 0.14901961, 0.15294118, ..., 0.14901961, 0.1254902 ,
        0.1254902 ],
       [0.14509805, 0.14509805, 0.14901961, ..., 0.19607843, 0.2       ,
        0.22352941],
       [0.14901961, 0.14901961, 0.14509805, ..., 0.3647059 , 0.38431373,
        0.39215687],
       ...,
       [0.12941177, 0.13725491, 0.12156863, ..., 0.05882353, 0.05882353,
        0.06666667],
       [0.14901961, 0.15294118, 0.13725491, ..., 0.05882353, 0.0627451 ,
        0.06666667],
       [0.12156863, 0.12156863, 0.12941177, ..., 0.0627451 , 0.07450981,
        0.07843138]], dtype=float32)
```

Figura 4-4: Normalización de píxeles de la imagen en el rango [0,1]

Finalmente, habría que redimensionar el conjunto de datos de cada etapa como un tensor de la forma [-1, 64, 64, 1], de modo que sea un tensor de 4 dimensiones cuyas imágenes son de 64x64 píxeles y tienen un único canal. Con esto, el dataset habría quedado listo para usarse en las diferentes etapas.

4.3 Resultados para primeros experimentos

En primer lugar, se probó a realizar una batería de experimentos con la red descrita sin aplicarle ningún tipo de técnica de mejora del entrenamiento. Para ello, se han usado diferentes algoritmos de optimización de los ya explicados anteriormente, eligiendo de entre todos ellos los optimizadores SGD, RMSprop y Adam.

4.3.1 Entrenamiento y validación

4.3.1.1 SGD

Al realizar experimentos se hicieron con 50 y 100 épocas, y tomando en ambos casos un batch de 200 imágenes. La comparación de *loss* y *accuracy* de entrenamiento y validación en ambos casos son las siguientes:

Para el experimento con 50 épocas, se puede observar en la Figura 4-5 que la función de *loss* en la validación varía constantemente alrededor de la de entrenamiento, suceso que no es deseable. También se observa que aún no se ha alcanzado un valor estable de *accuracy*, por lo que se concluye que serían necesarias más iteraciones, así que habría que aumentar el número de épocas de entrenamiento, por eso se aumentó a 100 épocas y se volvió a realizar el experimento.

Con 100 épocas, a la vista de las gráficas obtenidas en la Figura 4-6, una vez que se queda el *accuracy* en un valor constante, el del entrenamiento es superior en aproximadamente un 3-4 %, tomando un valor de aproximadamente 100%, mientras que en la validación alcanza un 96%. En cuanto a la función de *loss*, se puede observar que a medida que las épocas aumentan, en el entrenamiento va disminuyendo cada vez más, a pesar de algunos picos que se dan en ciertas épocas, mientras que la tendencia que adopta la de la validación es crecer, separándose cada vez más de la del entrenamiento. Lo que está ocurriendo aquí es que se está produciendo overfitting.

A la vista de los resultados, se puede intuir que cuando a estos modelos obtenidos se les pase el conjunto de test, los valores de *accuracy* probablemente incumplirán las especificaciones.

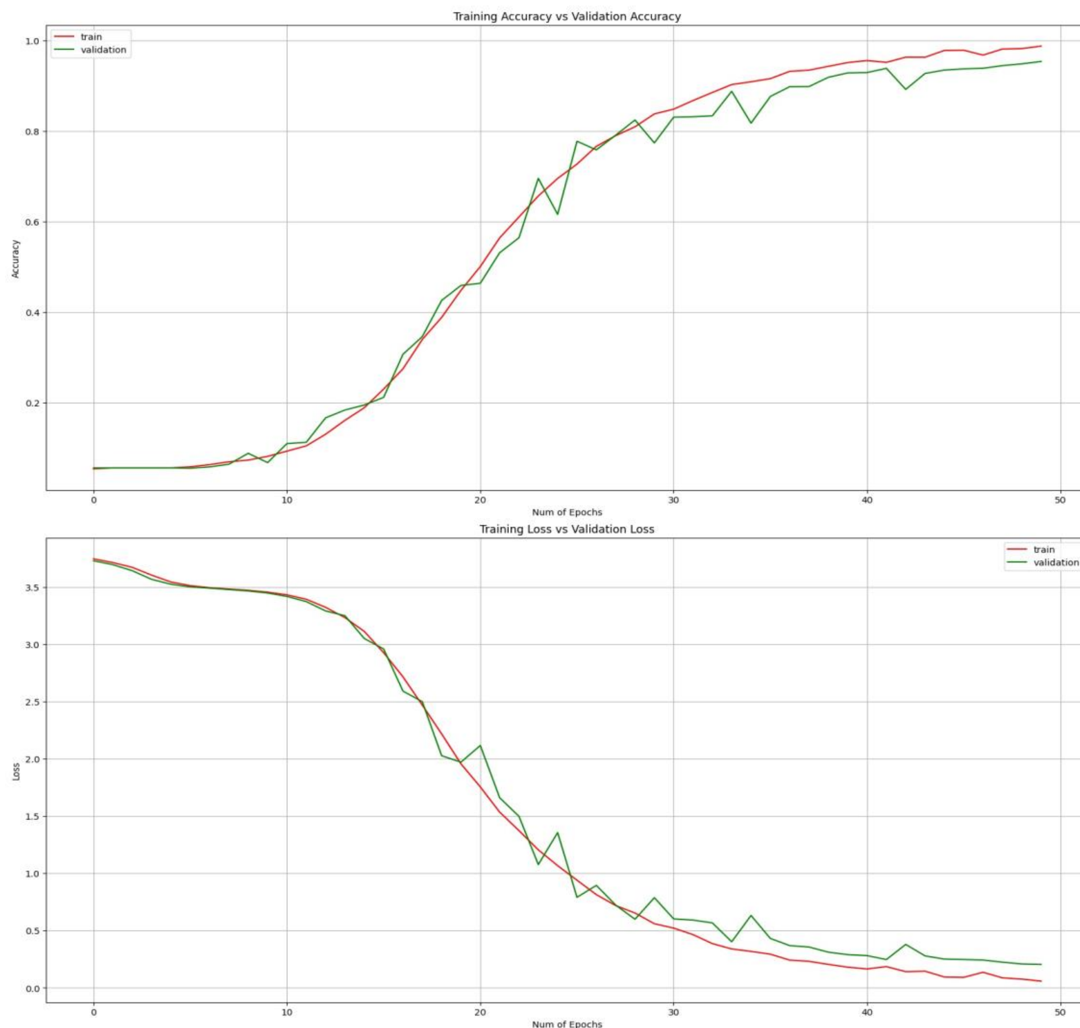


Figura 4-5: Train vs Validation con optimizador SGD y 50 épocas

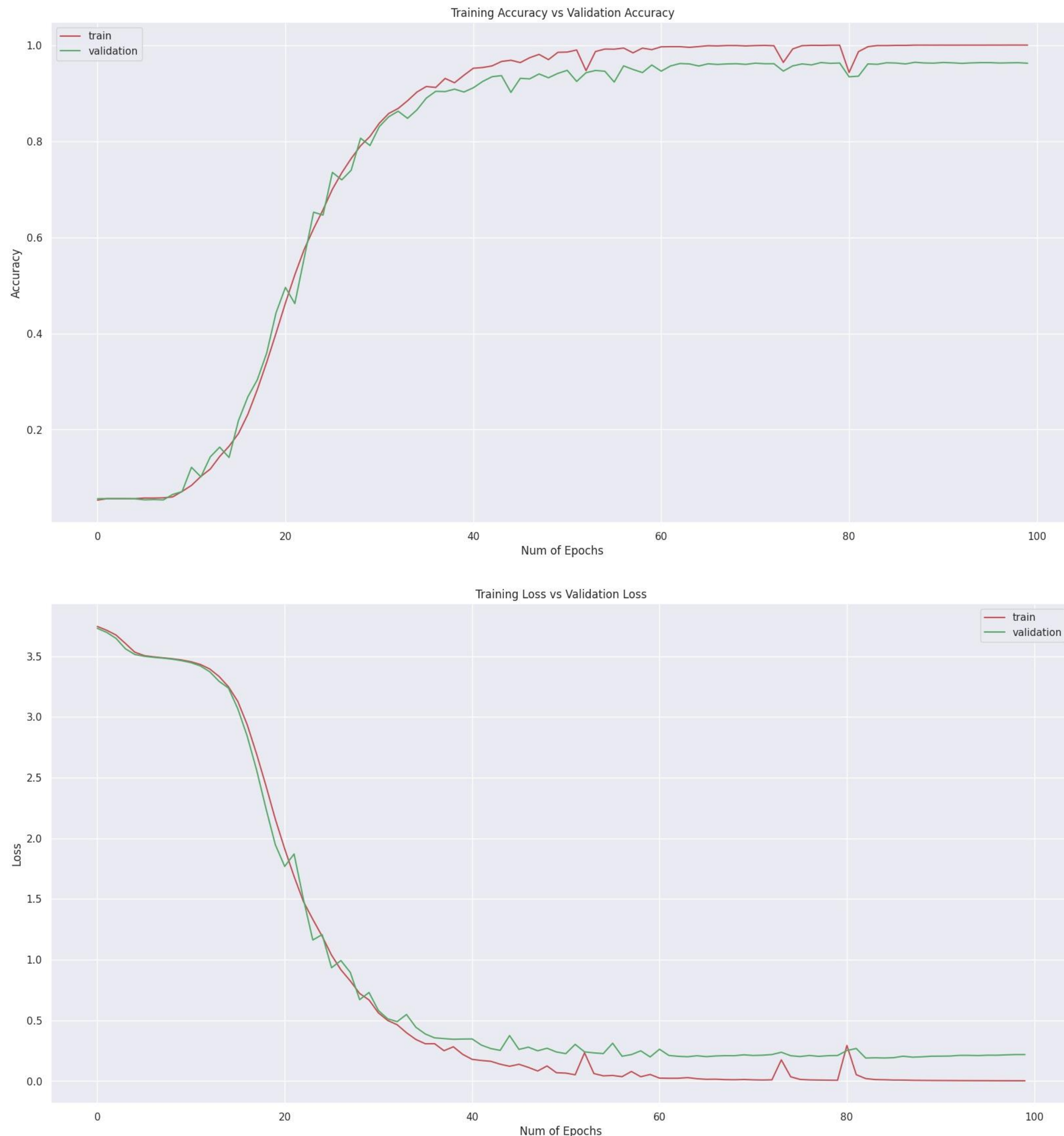


Figura 4-6: Train vs Validation con optimizador SGD y 100 épocas

4.3.1.2 RMSprop

Con este optimizador se ha realizado un experimento de 50 épocas y un batch de 200 imágenes. Se puede observar en la Figura 4-7 que las subidas y bajadas de ambas gráficas son más suaves en comparación con las anteriores, además de llegar a unos valores estables de *accuracy* y *loss* en un número inferior de épocas, que era de esperar por el propio funcionamiento del RMSprop respecto al SGD.

En el *accuracy*, tanto en validación como en el entrenamiento se llegan a alcanzar los mismos valores estables de aproximadamente el 100%, pero la validación sufre algunos picos de bajada de *accuracy* correspondientes con los picos que se producen en la función de *loss*. Dicha función toma valores más altos que los de entrenamiento prácticamente en la totalidad del experimento, produciéndose así overfitting, pero no tanto como en el experimento con el SGD. De hecho, se ve como aproximadamente a partir de la muestra 26 empieza a separarse la de validación de la de entrenamiento, aunque de manera mucho menos agresiva que en el SGD debido a su lenta tendencia de aumento comparada con este último.

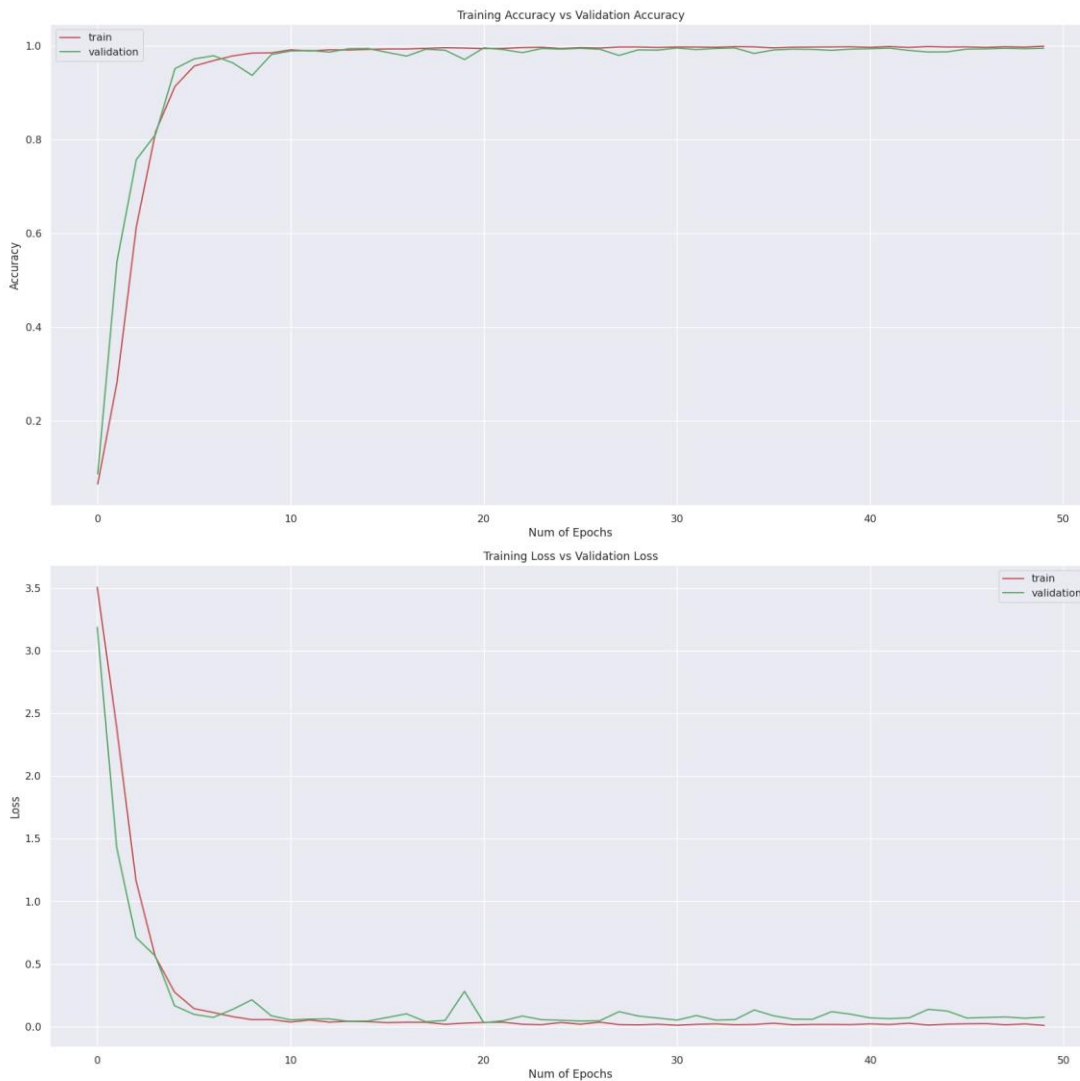


Figura 4-7: Train vs Validation con optimizador RMSprop y 50 épocas

4.3.1.3 Adam

Con este optimizador y el mismo experimento que en los dos anteriores casos, se obtiene una gráfica muchísimo más suave, como se aprecia en la Figura 4-8, donde ya no aparecen los molestos picos que había en la validación, pudiendo obtener una visión más clara de lo que hay que hacer a continuación.

Entorno a la época 25 se alcanza un valor de *accuracy* estable tanto para entrenamiento como para validación, estando este último algo por debajo de la de entrenamiento, que es del 100% mientras que la de validación alcanza un valor de un 99,3 % aproximadamente. El resultado da valores más repetitivos que en el RMSprop, aunque en éste se alcanzaron cotas de validación algo mayores.

Con respecto a la *loss*, una vez que pasa de las 25 épocas parece quedarse constante, siendo de menor valor la de entrenamiento, de modo que se vuelve a producir un overfitting, aunque parece que de menor magnitud que en el RMSprop, ya que en este último había bastantes variaciones.

Una vez realizados todos estos experimentos, llega el momento de ver realmente con cuál se ha obtenido el mejor resultado, y para ello se realiza el proceso de testeo sobre estos modelos.

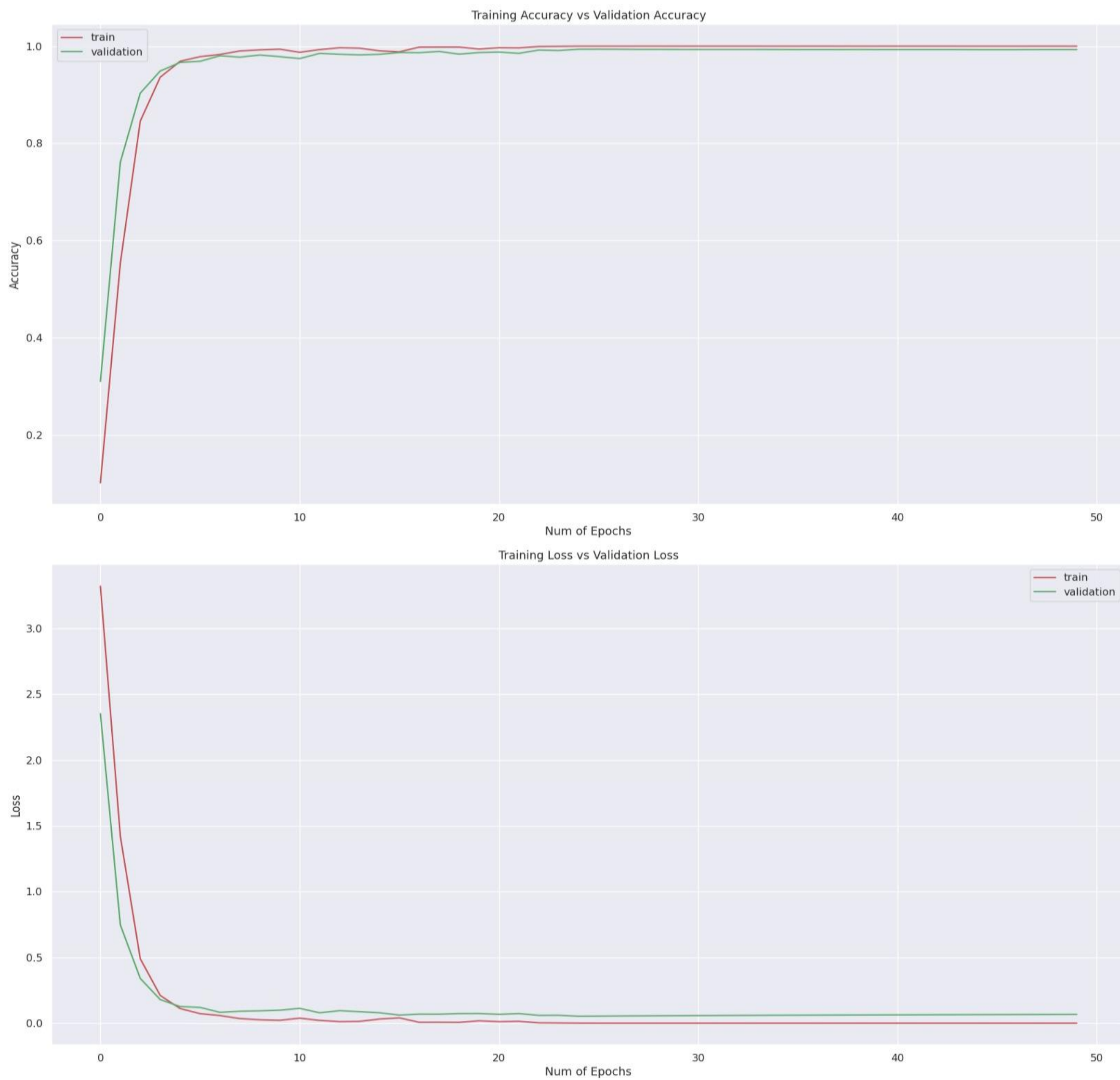


Figura 4-8: Train vs Validation con optimizador Adam y 50 épocas

4.3.2 Resultados de test

Los modelos de las redes anteriores se han evaluado en el conjunto de test de 12,569 imágenes, obteniendo una serie de resultados. En primer lugar, se mostrarán para cada experimento sus resultados de test, incluyendo *accuracy* total y precisión de cada una de las 43 clases de señales (nombradas en los resultados desde la clase 0 a la 42), entre otras métricas. Finalmente, se agruparán en una tabla los resultados más significativos de todos los experimentos en las 3 etapas del proceso y se comentarán.

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40	41	42
0	**9**	38	5	0	5	0	0	0	2	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	8	**528**	77	5	36	5	0	5	8	5	0	0	8	2	0	7	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	4	0	4	0	0	0	0	0	17	0	0
2	0	50	**615**	19	27	6	0	2	9	9	0	0	0	4	0	3	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	4	0	0	0	0
3	1	25	33	**327**	4	26	0	0	2	11	2	4	4	2	0	0	0	0	0	1	0	0	0	0	1	0	0	0	0	0	0	0	0	0	1	1	0	0	3	0	0	0	2
4	0	22	31	4	**556**	14	0	3	12	0	0	0	0	0	0	9	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	7	0	1	0	0	0	1	0	0	0	0
5	0	10	17	39	3	**495**	0	12	17	2	2	0	2	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
6	0	0	0	3	0	1	**112**	0	0	0	0	0	1	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	1	0
7	0	2	7	1	2	40	0	**399**	20	0	5	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	1	0	1
8	0	3	7	27	14	82	0	4	**311**	0	2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
9	0	1	5	10	2	6	0	0	2	**440**	1	0	0	0	0	10	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	1	0	1	0	0	0	0	0
10	0	0	0	10	0	10	0	1	0	10	**623**	0	0	0	0	0	0	0	0	0	0	0	0	4	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	1
11	1	0	1	0	0	1	0	0	0	1	1	**319**	1	1	0	0	0	0	1	6	33	6	0	6	0	3	0	5	4	1	22	0	0	2	0	0	0	1	0	0	3	1	0
12	0	1	1	1	7	0	2	1	0	1	0	0	**675**	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0
13	0	2	3	0	0	0	0	0	0	0	1	0	6	**697**	1	0	0	0	0	0	0	0	1	0	0	2	1	0	0	0	0	0	0	1	0	3	1	0	1	0	0	0	0
14	0	0	0	0	4	0	0	0	0	2	1	0	2	0	**208**	2	0	5	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	10	0	0	0	3	2	0	0	0	0
15	0	9	2	0	0	0	0	0	0	8	0	0	0	15	0	**175**	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0
16	0	0	0	0	0	9	0	27	3	2	0	0	0	0	0	0	**78**	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
17	0	0	0	0	0	0	0	0	0	0	0	0	0	0	12	0	0	**346**	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	1	0	0	0	0	0	0	0	0
18	0	2	0	1	1	1	0	0	1	0	1	8	1	0	0	0	0	0	**289**	0	7	0	2	0	5	4	42	10	1	0	4	0	3	0	1	0	0	0	3	1	0	2	0
19	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	**43**	0	3	0	1	3	0	0	0	1	1	2	5	0	0	0	0	0	0	0	0	0	0	0
20	0	0	0	1	0	0	0	0	0	1	1	2	0	0	0	0	0	0	0	0	**92**	0	0	9	5	1	0	0	6	0	2	0	0	0	0	0	0	0	0	0	0	0	0
21	0	3	2	0	5	4	0	0	0	0	0	1	3	0	0	0	0	0	3	4	2	**1**	2	17	18	5	0	2	0	6	5	7	0	0	0	0	0	0	0	0	0	0	0
22	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	1	0	0	**107**	0	1	3	0	0	0	2	1	4	0	0	0	0	0	0	0	0	0	0	0
23	0	0	0	0	0	0	0	0	0	0	0	3	0	0	0	0	0	0	8	1	41	0	3	**61**	0	1	0	0	16	0	16	0	0	0	0	0	0	0	0	0	0	0	0
24	0	0	0	0	3	0	0	0	0	1	0	1	0	0	0	0	0	0	0	2	0	3	0	1	**48**	0	17	7	4	0	2	0	0	0	0	0	0	1	0	0	0	0	0
25	0	0	0	0	0	1	0	0	0	1	1	0	1	1	0	0	0	0	6	0	15	0	2	1	10	**430**	0	1	1	2	1	4	0	0	0	0	0	0	2	0	0	0	0
26	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	56	0	2	0	4	0	2	1	**106**	3	0	1	0	3	0	1	1	0	0	0	0	0	0	0	0
27	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	7	0	0	0	0	0	14	0	2	**35**	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0
28	0	0	0	0	0	0	0	0	0	0	0	1	2	0	1	0	0	0	0	3	6	0	0	22	5	2	0	0	**124**	9	4	1	0	0	0	0	0	0	0	0	0	0	0
29	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	2	0	29	0	0	3	5	3	0	0	0	**44**	0	1	0	1	0	0	0	0	0	0	0	0	0
30	0	0	0	0	0	0	0	0	0	0	1	13	0	0	0	0	0	0	2	0	12	0	0	1	1	2	0	0	14	0	**104**	0	0	0	0	0	0	0	0	0	0	0	0
31	0	0	0	0	0	0	0	0	1	3	0	0	0	0	0	0	0	0	22	11	10	0	0	0	2	4	0	0	8	0	6	**173**	0	0	0	0	0	0	0	0	0	0	0
32	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	**60**	0	0	0	0	0	0	0	0	0	0
33	0	0	6	0	0	3	0	0	0	7	0	3	0	0	4	0	7	0	0	3	0	0	0	0	0	0	0	0	0	0	18	0	0	**158**	0	0	0	0	0	0	0	0	0
34	0	0	0	4	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	4	0	**101**	11	0	0	0	0	0	0	0
35	0	0	2	4	0	0	0	0	0	0	25	0	0	4	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	6	3	**344**	1	0	1	0	0	0	0
36	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	**119**	0	1	0	0	0	0
37	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	2	0	0	0	3	0	0	0	0	0	0	**54**	0	0	0	0	0
38	1	20	0	2	20	0	0	0	0	1	1	0	10	7	2	9	0	0	0	0	1	0	1	3	6	0	0	0	0	0	0	0	6	0	2	15	3	15	**564**	0	1	0	0
39	0	0	0	0	12	0	0	0	1	0	0	0	0	3	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	2	0	22	0	0	0	4	0	**46**	0	0	0
40	0	1	3	0	8	0	0	12	0	0	0	0	33	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	**60**	1	1
41	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	12	0	0	0	0	0	0	0	0	**48**	0
42	0	0	0	0	0	1	18	1	0	0	0	0	1	0	0	0	2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	**36**

Figura 4-9: Matriz de confusión de SGD con 50 epochs

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40	41	42
0	**17**	19	4	2	10	0	0	1	4	0	0	0	0	0	0	3	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	10	**539**	85	3	50	3	0	2	7	5	0	2	5	1	0	2	0	0	1	0	0	0	1	0	0	0	0	2	0	0	0	0	1	0	0	0	0	0	0	0	1	0	0
2	1	47	**612**	9	27	14	1	1	16	10	1	0	2	2	0	2	0	0	0	0	0	0	0	0	1	2	0	0	0	0	0	0	0	0	0	1	0	1	0	0	0	0	0
3	1	11	30	**333**	5	34	0	2	3	17	2	2	3	1	0	2	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	2
4	0	42	61	3	**506**	15	0	4	17	1	0	0	0	0	0	5	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	5	0	1	0	0	0	0	0	0	0	0
5	0	11	34	25	0	**494**	1	14	6	3	8	0	0	0	0	1	0	0	0	0	2	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
6	0	0	0	0	0	0	**114**	0	0	0	0	1	1	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	3	0	0	0	0	0	0	0	0	0	0
7	0	3	8	1	1	16	0	**430**	17	1	2	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
8	3	4	3	19	4	78	0	9	**328**	1	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
9	0	5	6	8	2	14	0	1	3	**415**	1	0	2	1	0	18	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	2	0	2	0	0	0	0	0	0	0
10	0	0	0	7	0	21	0	2	2	5	**619**	0	0	0	0	0	0	0	0	0	0	0	0	3	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0
11	0	0	0	0	0	3	0	1	0	1	1	**335**	6	0	0	0	0	0	7	1	10	4	3	7	7	5	2	0	4	0	21	2	0	0	0	0	0	0	0	0	0	0	0
12	0	3	1	3	3	0	0	3	0	1	0	0	**675**	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0
13	0	4	0	3	0	1	0	0	1	0	0	1	4	**695**	0	0	0	0	0	0	0	0	2	0	0	2	0	0	0	0	0	0	0	2	3	0	0	0	1	0	1	0	0
14	0	0	0	1	1	0	0	1	0	9	0	0	0	1	**195**	3	0	7	0	0	1	0	0	0	0	12	0	0	0	0	0	0	0	1	1	0	0	2	5	0	0	0	0
15	1	14	4	0	0	0	0	0	0	2	0	0	0	1	0	**184**	0	0	0	0	0	0	0	0	0	0	0	0	0	2	0	0	0	0	0	2	0	0	0	0	0	0	0
16	0	0	0	0	0	0	0	55	0	0	0	0	0	0	0	0	**65**	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
17	0	0	0	0	0	0	0	0	0	0	0	0	0	0	10	0	0	**347**	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	2	0	0	0	0	0	0	0	0
18	1	0	0	0	0	1	0	2	1	1	1	5	1	0	0	0	0	0	**302**	0	6	0	1	1	3	6	41	7	1	0	0	3	1	0	0	0	0	0	5	0	0	0	0
19	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	3	**30**	0	12	0	0	2	0	0	0	2	0	0	10	0	0	0	0	0	0	0	0	0	0	0
20	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	**91**	0	0	5	5	3	1	0	11	0	2	0	0	0	0	0	0	0	1	0	0	0	0
21	0	3	2	1	0	0	0	0	0	0	0	2	6	3	0	0	0	0	4	2	2	**6**	3	10	20	5	3	1	0	3	4	10	0	0	0	0	0	0	0	0	0	0	0
22	0	0	0	0	0	0	0	0	0	0	0	0	0	1	3	0	0	0	1	0	0	0	**109**	0	0	2	3	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0
23	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	4	0	25	0	16	**68**	6	1	0	0	12	2	13	2	0	0	0	0	0	0	0	0	0	0	0
24	0	0	0	0	1	0	0	0	1	2	0	4	0	0	0	0	0	0	4	0	0	2	0	1	**50**	0	13	1	3	0	4	0	0	1	0	0	0	0	0	3	0	0	0
25	0	0	0	0	0	1	0	0	0	0	0	0	0	0	4	0	0	0	12	0	4	0	1	2	8	**435**	5	0	1	3	2	1	0	0	0	0	0	0	1	0	0	0	0
26	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	57	0	1	0	4	0	0	0	**109**	1	0	0	1	5	0	2	0	0	0	0	0	0	0	0	0
27	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	9	0	0	0	0	0	7	0	6	**36**	1	0	0	0	0	0	0	0	0	0	1	0	0	0	0
28	0	0	0	0	0	0	0	0	0	0	0	1	0	0	2	0	0	0	0	3	3	1	1	6	4	2	0	5	**141**	7	2	1	0	0	0	0	0	0	0	0	1	0	0
29	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	1	0	0	6	0	22	0	1	1	4	5	0	0	1	**45**	0	3	0	0	0	0	0	0	0	0	0	0	0
30	0	0	0	0	0	1	0	0	0	0	0	17	0	0	0	0	0	0	8	0	2	0	0	2	0	2	0	0	8	0	**110**	0	0	0	0	0	0	0	0	0	0	0	0
31	0	0	0	0	0	0	0	0	5	0	0	2	0	0	0	0	0	0	10	5	8	1	0	0	1	3	0	2	1	0	4	**191**	0	0	0	0	0	2	0	5	0	0	0
32	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	**60**	0	0	0	0	0	0	0	0	0	0
33	0	0	7	0	0	0	0	0	0	0	0	31	0	0	3	0	0	0	0	0	0	0	0	0	0	5	0	0	0	0	0	0	0	**159**	0	0	0	0	0	0	4	0	0
34	0	0	0	0	0	0	0	0	0	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	**109**	3	0	0	6	0	0	0	0
35	0	0	0	4	0	0	0	0	0	1	13	0	0	5	0	2	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	17	0	**346**	1	0	0	0	0	0	0
36	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	**112**	0	8	0	0	0	0
37	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	**58**	0	0	0	0	0
38	3	17	1	1	15	0	0	0	1	0	2	1	4	2	0	13	0	0	0	0	0	0	6	0	4	2	2	0	0	3	0	1	4	1	1	20	3	3	**576**	0	4	0	0
39	0	0	1	0	24	0	0	0	5	0	0	0	0	0	0	0	0	0	0	0	0	2	0	0	0	0	0	0	0	0	0	0	0	4	0	0	0	4	0	**50**	0	0	0
40	0	8	2	1	4	0	0	6	1	0	0	0	25	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	**72**	0	0
41	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	19	0	0	0	0	0	0	0	0	**41**	0
42	0	0	0	0	0	0	16	4	0	0	0	0	2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	2	**36**

Figura 4-10: Matriz de confusión de SGD con 100 epochs

En la Figura 4-10 se puede ver como en este caso se confunden muchas imágenes, confundiendo por ejemplo 85 imágenes de la clase 1 con la clase 2, 55 de la clase 16 con la clase 7, etc. Estas son unas pocas de las que confunde, ya que en la matriz se ve que todos los valores no están repartidos en la diagonal, teniendo demasiado desacierto. Lo mismo ocurre para la Figura 4-9.

	precision	recall	f1-score	support
0	0.45	0.15	0.23	60
1	0.74	0.73	0.73	720
2	0.75	0.82	0.78	750
3	0.71	0.73	0.72	450
4	0.78	0.84	0.81	660
5	0.70	0.82	0.76	600
6	0.85	0.93	0.89	120
7	0.85	0.83	0.84	480
8	0.80	0.69	0.74	450
9	0.87	0.92	0.89	480
10	0.93	0.94	0.94	660
11	0.90	0.76	0.82	420
12	0.90	0.98	0.94	690
13	0.95	0.97	0.96	720
14	0.91	0.87	0.89	240
15	0.80	0.83	0.82	210
16	0.89	0.65	0.75	120
17	0.99	0.96	0.97	360
18	0.73	0.74	0.73	390
19	0.57	0.72	0.64	60
20	0.37	0.77	0.49	120
21	0.08	0.01	0.02	90
22	0.88	0.89	0.88	120
23	0.47	0.41	0.44	150
24	0.38	0.53	0.44	90
25	0.93	0.90	0.91	480
26	0.62	0.59	0.61	180
27	0.56	0.58	0.57	60
28	0.69	0.69	0.69	180
29	0.67	0.49	0.56	90
30	0.54	0.69	0.61	150
31	0.86	0.72	0.78	240
32	0.59	1.00	0.75	60
33	0.79	0.76	0.77	209
34	0.88	0.84	0.86	120
35	0.91	0.88	0.90	390
36	0.96	0.99	0.98	120
37	0.68	0.90	0.77	60
38	0.97	0.82	0.89	690
39	0.98	0.51	0.67	90
40	0.73	0.50	0.59	120
41	0.89	0.80	0.84	60
42	0.86	0.60	0.71	60
accuracy			0.81	12569
macro avg	0.75	0.74	0.73	12569
weighted avg	0.81	0.81	0.81	12569

	precision	recall	f1-score	support
0	0.46	0.28	0.35	60
1	0.74	0.75	0.74	720
2	0.71	0.82	0.76	750
3	0.79	0.74	0.76	450
4	0.77	0.77	0.77	660
5	0.71	0.82	0.76	600
6	0.86	0.95	0.90	120
7	0.80	0.90	0.84	480
8	0.78	0.73	0.75	450
9	0.87	0.86	0.87	480
10	0.95	0.94	0.94	660
11	0.83	0.80	0.81	420
12	0.91	0.98	0.95	690
13	0.97	0.97	0.97	720
14	0.90	0.81	0.85	240
15	0.78	0.88	0.83	210
16	0.98	0.54	0.70	120
17	0.98	0.96	0.97	360
18	0.71	0.77	0.74	390
19	0.71	0.50	0.59	60
20	0.51	0.76	0.61	120
21	0.21	0.07	0.10	90
22	0.73	0.91	0.81	120
23	0.64	0.45	0.53	150
24	0.41	0.56	0.47	90
25	0.88	0.91	0.89	480
26	0.59	0.61	0.60	180
27	0.65	0.60	0.63	60
28	0.75	0.78	0.77	180
29	0.68	0.50	0.58	90
30	0.67	0.73	0.70	150
31	0.83	0.80	0.81	240
32	0.63	1.00	0.77	60
33	0.84	0.76	0.80	209
34	0.92	0.91	0.92	120
35	0.92	0.89	0.90	390
36	0.97	0.93	0.95	120
37	0.83	0.97	0.89	60
38	0.95	0.83	0.89	690
39	0.86	0.56	0.68	90
40	0.87	0.60	0.71	120
41	0.95	0.68	0.80	60
42	0.95	0.60	0.73	60
accuracy			0.82	12569
macro avg	0.78	0.75	0.75	12569
weighted avg	0.82	0.82	0.81	12569

Figura 4-11: Resultados de test SGD para 50 (izquierda) y 100 epochs (derecha)

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40	41	42
0	57	2	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	718	0	0	1	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
2	0	5	737	1	0	6	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
3	0	0	1	437	0	11	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
4	0	2	3	0	655	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
5	0	6	5	9	0	576	0	3	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
6	0	0	0	0	0	0	120	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
7	0	0	0	0	0	1	0	474	4	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
8	0	0	0	0	3	3	0	0	444	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
9	0	0	0	0	0	0	0	0	0	476	1	0	0	0	0	1	2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
10	0	0	0	0	0	0	0	0	0	4	650	0	0	0	0	0	4	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0
11	0	0	0	0	0	0	0	0	0	0	0	405	7	0	0	0	0	0	0	0	0	0	0	3	0	0	0	0	0	1	4	0	0	0	0	0	0	0	0	0	0	0	0
12	0	0	0	0	0	0	0	0	0	0	0	0	689	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
13	0	0	0	0	0	0	0	0	0	0	0	0	0	718	0	2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
14	0	0	0	0	0	0	0	0	0	0	0	0	0	0	240	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
15	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	208	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	2	0	0	0	0	0	0	0	0	0	0
16	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	120	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
17	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	359	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
18	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	360	0	0	0	1	0	0	1	28	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
19	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	60	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
20	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	119	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
21	0	2	0	0	0	0	0	0	0	0	0	2	7	0	0	0	0	0	0	0	0	65	0	3	0	0	0	0	0	6	0	5	0	0	0	0	0	0	0	0	0	0	0
22	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	120	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
23	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	8	0	0	133	0	0	0	0	0	8	1	0	0	0	0	0	0	0	0	0	0	0	0
24	2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	87	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
25	0	0	0	0	0	0	0	0	0	0	0	0	2	0	0	0	0	0	2	0	1	0	0	0	1	473	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
26	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	2	0	0	0	175	0	0	1	0	0	0	0	1	0	0	0	0	0	0	0	0
27	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	2	0	1	56	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
28	0	0	0	0	0	0	0	0	0	0	0	5	0	0	0	0	0	0	0	0	0	0	1	0	1	1	0	0	172	0	0	0	0	0	0	0	0	0	0	0	0	0	0
29	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	2	0	0	8	0	0	1	79	0	0	0	0	0	0	0	0	0	0	0	0	0
30	0	3	0	0	0	0	0	0	0	0	0	6	0	0	0	0	0	0	0	0	3	0	0	2	1	3	0	0	1	0	131	0	0	0	0	0	0	0	0	0	0	0	0
31	0	0	0	1	0	0	0	0	0	0	0	1	0	0	0	0	2	0	0	0	2	20	0	0	0	0	0	0	0	0	2	209	0	0	0	0	0	0	0	3	0	0	0
32	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	60	0	0	0	0	0	0	0	0	0	0
33	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	178	0	0	0	1	0	9	20	0	0
34	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	120	0	0	0	0	0	0	0	0
35	0	0	0	0	0	0	0	0	0	0	0	0	5	2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	19	0	362	0	2	0	0	0	0	0
36	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	120	0	0	0	0	0	0
37	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	60	0	0	0	0	0
38	4	5	0	1	0	0	0	0	0	0	0	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	3	0	0	0	0	7	2	0	12	615	0	39	0	0
39	0	0	0	4	2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	2	0	0	0	0	70	12	0	0
40	0	1	0	0	0	0	0	0	0	0	0	0	9	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	110	0	0
41	0	0	0	0	0	0	0	0	0	3	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	2	0	0	0	0	0	0	0	0	54	0
42	0	0	0	0	0	0	0	0	0	0	0	0	4	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	56

Figura 4-12: Matriz de confusión RMSprop 50 epochs

	precision	recall	f1-score	support
0	0.90	0.95	0.93	60
1	0.97	1.00	0.98	720
2	0.99	0.98	0.99	750
3	0.96	0.97	0.97	450
4	0.99	0.99	0.99	660
5	0.96	0.96	0.96	600
6	1.00	1.00	1.00	120
7	0.99	0.99	0.99	480
8	0.99	0.99	0.99	450
9	0.99	0.99	0.99	480
10	1.00	0.98	0.99	660
11	0.97	0.96	0.97	420
12	0.95	1.00	0.97	690
13	1.00	1.00	1.00	720
14	1.00	1.00	1.00	240
15	0.98	0.99	0.99	210
16	0.93	1.00	0.96	120
17	1.00	1.00	1.00	360
18	0.99	0.92	0.95	390
19	0.97	1.00	0.98	60
20	0.89	0.99	0.94	120
21	0.76	0.72	0.74	90
22	0.95	1.00	0.98	120
23	0.94	0.89	0.91	150
24	0.95	0.97	0.96	90
25	0.97	0.99	0.98	480
26	0.85	0.97	0.91	180
27	1.00	0.93	0.97	60
28	0.98	0.96	0.97	180
29	0.81	0.88	0.84	90
30	0.95	0.87	0.91	150
31	0.98	0.87	0.92	240
32	0.94	1.00	0.97	60
33	0.90	0.85	0.87	209
34	0.92	1.00	0.96	120
35	0.99	0.93	0.96	390
36	1.00	1.00	1.00	120
37	0.80	1.00	0.89	60
38	1.00	0.89	0.94	690
39	0.85	0.78	0.81	90
40	0.61	0.92	0.73	120
41	1.00	0.90	0.95	60
42	0.98	0.93	0.96	60
accuracy			0.96	12569
macro avg	0.94	0.95	0.95	12569
weighted avg	0.97	0.96	0.97	12569

	precision	recall	f1-score	support
0	0.95	0.88	0.91	60
1	0.98	0.98	0.98	720
2	0.97	0.99	0.98	750
3	0.94	0.98	0.96	450
4	0.98	0.99	0.99	660
5	0.97	0.96	0.96	600
6	0.98	0.99	0.99	120
7	0.98	0.98	0.98	480
8	0.98	0.99	0.99	450
9	0.97	0.99	0.98	480
10	0.99	1.00	0.99	660
11	0.99	0.96	0.97	420
12	0.96	0.99	0.97	690
13	1.00	1.00	1.00	720
14	0.98	0.95	0.97	240
15	0.96	0.89	0.93	210
16	0.99	0.98	0.99	120
17	1.00	1.00	1.00	360
18	0.95	0.97	0.96	390
19	0.92	1.00	0.96	60
20	0.94	0.98	0.96	120
21	0.96	0.54	0.70	90
22	0.92	1.00	0.96	120
23	0.98	0.86	0.92	150
24	0.81	0.87	0.84	90
25	0.99	0.99	0.99	480
26	0.94	0.97	0.95	180
27	0.88	1.00	0.94	60
28	0.94	0.94	0.94	180
29	0.81	0.77	0.79	90
30	0.92	0.96	0.94	150
31	0.90	0.98	0.94	240
32	0.92	1.00	0.96	60
33	0.84	0.85	0.85	209
34	0.98	0.97	0.98	120
35	0.96	0.92	0.94	390
36	0.99	1.00	1.00	120
37	0.80	1.00	0.89	60
38	1.00	0.89	0.94	690
39	0.88	0.78	0.82	90
40	0.67	0.88	0.76	120
41	0.98	0.98	0.98	60
42	1.00	0.98	0.99	60
accuracy			0.96	12569
macro avg	0.94	0.94	0.94	12569
weighted avg	0.96	0.96	0.96	12569

Figura 4-13: Resultados de test de RMSprop (izquierda) y Adam (derecha) con 50 epochs

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40	41	42
0	53	0	0	3	0	0	0	0	4	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	1	704	12	0	2	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
2	0	0	743	0	2	5	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
3	0	0	3	442	0	3	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0
4	2	2	1	0	654	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
5	0	2	7	4	1	576	0	9	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
6	0	0	0	0	0	0	119	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0
7	0	0	0	0	1	6	0	471	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
8	0	0	0	1	0	2	0	3	444	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
9	0	0	0	0	0	0	0	0	0	474	2	0	0	0	4	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
10	0	0	0	0	0	2	0	0	0	1	657	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
11	0	0	0	0	0	0	0	0	0	0	0	402	0	0	0	0	0	0	1	0	0	0	0	0	5	1	0	0	2	0	9	0	0	0	0	0	0	0	0	0	0	0	0
12	0	0	0	0	0	0	0	0	0	0	0	0	681	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	7	0	0
13	0	0	0	0	0	0	0	0	0	0	0	0	0	718	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0
14	0	0	1	0	0	0	0	0	0	0	0	0	0	0	229	6	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	3	0	0	0
15	0	1	2	18	0	0	0	0	0	0	0	0	0	0	0	187	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	2	0	0	0	0	0	0	0
16	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	118	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0
17	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	359	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0
18	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	379	0	0	0	0	0	0	0	7	2	1	0	0	0	1	0	0	0	0	0	0	0	0	0	0
19	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	60	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
20	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	118	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0
21	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	2	2	49	1	1	0	0	3	2	2	1	1	24	0	0	0	0	0	1	0	0	0	0	0
22	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	120	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
23	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	6	0	0	129	0	0	0	0	0	14	0	0	0	0	0	0	0	0	0	0	0	0	0
24	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	12	0	0	0	0	0	78	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
25	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	477	2	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0
26	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	5	0	0	0	174	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0
27	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	60	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
28	0	0	0	0	0	0	1	0	0	0	0	2	0	0	0	0	0	0	0	1	0	2	0	1	2	0	0	0	169	1	0	0	1	0	0	0	0	0	0	0	0	0	0
29	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	3	0	10	2	0	2	1	69	1	1	0	0	0	0	0	0	0	0	0	0	0
30	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	5	0	0	0	0	0	1	0	0	0	0	0	144	0	0	0	0	0	0	0	0	0	0	0	0
31	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	2	0	0	0	0	0	0	0	2	1	0	0	235	0	0	0	0	0	0	0	0	0	0	0
32	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	60	0	0	0	0	0	0	0	0	0	0
33	0	0	0	0	0	0	0	0	0	12	0	0	9	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	178	0	0	0	0	0	6	4	0	0
34	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	117	3	0	0	0	0	0	0	0
35	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	1	0	0	1	0	0	0	0	0	0	0	25	0	357	0	0	0	0	5	0	0
36	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	120	0	0	0	0	0	0
37	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	60	0	0	0	0	0
38	0	0	0	1	0	0	0	0	0	2	0	2	13	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	1	8	0	11	615	0	35	1	0
39	0	0	0	0	7	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	4	0	0	0	0	7	0	0	0	1	0	70	0	0	0
40	0	6	0	0	0	0	0	0	0	0	0	0	6	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	1	106	0	0
41	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	59	0
42	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	59

Figura 4-14: Matriz de confusión Adam 50 epochs

En las Figura 4-12 y Figura 4-14 se aprecia que la diagonal concentra muchísimos más valores que las de las Figura 4-9 y Figura 4-10 por lo que al confundir menos imágenes es de esperar que tenga mejores resultados.

4.3.2.1 Comparación de experimentos

Tabla 4-1: Resultados de los experimentos

	Train Accuracy (%)	Validation Accuracy (%)	Test Accuracy (%)
SGD (50 epochs)	98.72	95.34	80.83
SGD (100 epochs)	99.99	96.21	81.51
RMSprop (50 epochs)	99.91	99.48	96.48
Adam (50 epochs)	100	99.30	96.21

En primer lugar, se aprecia en la Tabla 4-1 como el SGD es el que peor funciona, puesto que, aunque todos tienen overfitting ahora mismo, es el que menores valores de *accuracy* alcanza. En los otros casos, el overfitting hace que el valor de *accuracy* caiga entre un 3 y un 4% respecto del entrenamiento que, aunque no es una gran diferencia, en el ámbito en que se aplicará esta red puede ser crucial. Tanto en el RMSprop como en el Adam se obtienen valores del 96 %.

Si se analiza la precisión obtenida para cada clase, se puede ver en la Figura 4-13 que en el RMSprop se alcanzan los valores más pequeños de precisión en la clase 40, con un 61 %, que es la misma clase para la que el Adam alcanza también su mínimo valor con un 67 %. Por lo demás, en el Adam las precisiones en las otras clases no bajan del 80 %, a diferencia del RMSprop, donde la clase 21 sufre una bajada respecto a las demás clases obteniendo un 76 % de precisión.

En definitiva, el problema claro que se tiene aquí es el overfitting, y aunque ya hay algunos modelos que tienen un mínimo de 96% y cumplirían especificaciones, lo hacen con overfitting y de manera muy ajustada, por lo que para mejorar los resultados de test se podrían probar algunas estrategias como quitar algunas capas, retocar algunos hiperparámetros, o hacer Dropout. Será esta última técnica la que se implementará en los siguientes experimentos. Además, se dejará de hacer uso del optimizador SGD, ya que se ha visto que con los otros dos se obtienen resultados muy superiores a este último.

4.4 Experimentos con Dropout

Con la finalidad de mejorar los problemas de overfitting y elevar el porcentaje de *accuracy* de test en las redes anteriores, se usará la técnica de Dropout. Dicha herramienta permite que un cierto número de neuronas aleatorias de una capa se desactiven durante cada época del entrenamiento. Esto permite que las neuronas de cada capa sean más independientes del resto de neuronas, así como ir olvidando aleatoriamente cierta información para que no se produzca overfitting. Debido a esto, el entrenamiento de la red será más lento que antes.

De manera clásica, la técnica se define con una probabilidad entre 0 y 1, que simboliza el porcentaje de neuronas activas en la capa donde se aplica, pero en el entorno en el que se están desarrollando las redes aquí presentes (Keras), la probabilidad que indica el Dropout simboliza lo contrario, es decir, el porcentaje de neuronas de la capa que permanecen desactivadas.

En las redes desarrolladas, se ha intentado aplicar el Dropout como si fuera una especie de embudo, de manera que en las capas iniciales de la red habrá más neuronas activas que en las capas finales. La razón de hacerlo así es porque al principio la red recoge un mayor número de datos que al final, de manera que si se desactivan muchas neuronas al principio se perderá demasiada información y los resultados serán peores.

Se probaron diferentes probabilidades de Dropout y configuraciones de dichas capas de Dropout, obteniendo 3 tipos diferentes:

- Dropout de tipo 1: Se usó una probabilidad de activación de neuronas del 80% en las capas convolucionales (que se representa aquí con un 20% de neuronas desactivadas), mientras que para las capas densas se usó una probabilidad de activación de solamente el 30% (representado por el 70% en el código en Keras)
- Dropout de tipo 2: Es como el tipo 1 pero se pusieron las capas densas con una probabilidad de activación de neuronas del 50%, mientras que en las capas convolucionales se puso una probabilidad de activación del 75% (25% de desactivación) respecto al tipo 1 donde era del 80%.
- Dropout de tipo 3: Las probabilidades usadas son como las del tipo 2, con la diferencia de que ahora no se hace Dropout entre la última capa oculta de la etapa densamente conectada y la capa de salida Softmax.

Con respecto a los primeros experimentos, se observará en algunos casos un aumento en el número de épocas, ya que el Dropout hace que el entrenamiento sea más lento, de modo que se necesita un mayor número de iteraciones para que la red pueda aprender. A continuación, se muestran los resultados obtenidos con cada Dropout usado y las decisiones tomadas en cada uno.

4.4.1 Dropout 1

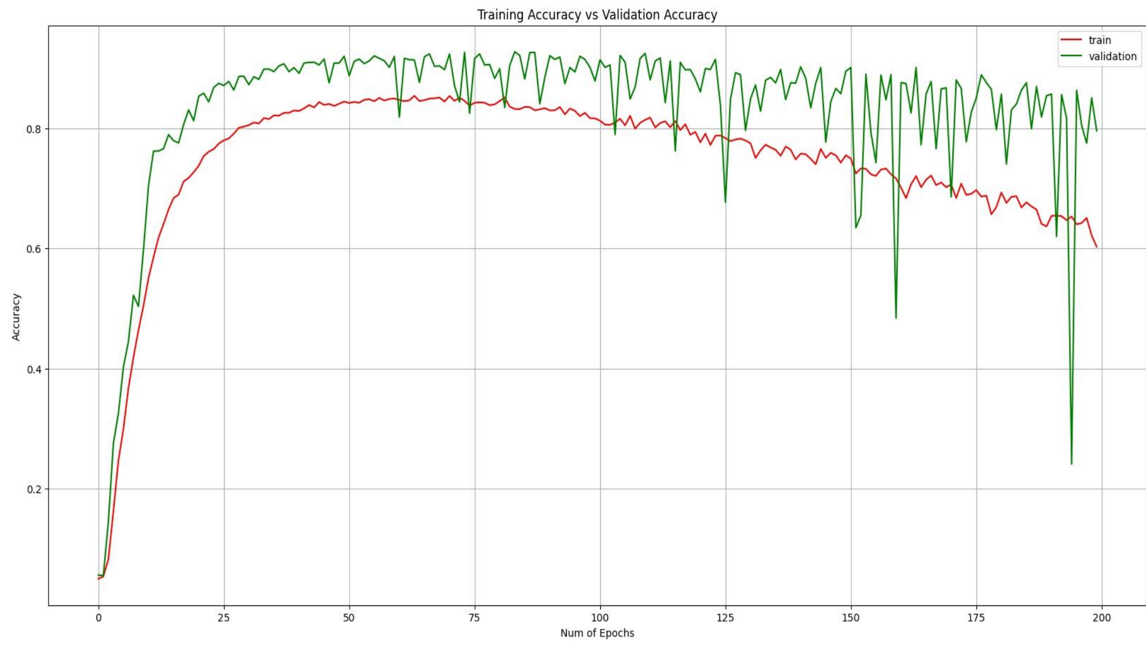
Se presentarán como en los primeros experimentos los resultados de entrenamiento y validación, y después se presentarán los de test.

4.4.1.1 Entrenamiento y validación

Se han usado el optimizador RMSprop y Adam, de los que mejores funcionaron al principio, un batch de 200 imágenes, y un número de épocas de 200.

En la Figura 4-15 se puede apreciar que para el optimizador RMSprop usando este Dropout se estropea totalmente el resultado respecto al anterior, y es que no le permite aprender lo suficiente como para siquiera llegar a cotas de *accuracy* de entrenamiento y validación superiores al 90%, alcanzando su valor máximo entorno a la época 63. A partir de ahí, los valores de *accuracy* bajan con una pendiente de aproximadamente -0.2 %/época y la *loss* aumenta con pendiente de 0.005 ud/época para el entrenamiento. Para la validación también se producen bajadas y subidas en *accuracy* y *loss* respectivamente, pero de forma más lenta y con una considerable variabilidad de los resultados. A la vista de los resultados, se podrá intuir que este modelo no será bueno en el test. El modelo final que se ha obtenido en este caso tiene un 60.29 % *accuracy* de entrenamiento y un 79.59 % de *accuracy* de validación.

En la Figura 4-16, se ve como el entrenamiento del Adam tiene una *accuracy* algo más baja y una *loss* algo mayor que las de validación. Estos valores de entrenamiento menores que los de validación también se pueden aplicar al experimento del RMSprop, y se debe a que al hacer Dropout, éste solo se aplica a la fase de entrenamiento, por tanto, al realizar la validación entran en juego todas las neuronas, pudiendo obtener mayores cotas de *accuracy* y menores de *loss*. Esto ha eliminado el overfitting de los modelos anteriores, pero ha producido un underfitting, o falta de entrenamiento, al no alcanzarse totalmente los valores de validación, dejando un modelo con un *accuracy* de entrenamiento del 96.66% y un *accuracy* de validación del 99.59%, más alto que el de entrenamiento. El número de épocas es de 200 porque se puede apreciar como alcanza un valor estable entorno a las 175 épocas, momento en que su tendencia deja de cambiar y la *accuracy* deja de subir y la *loss* deja de bajar.



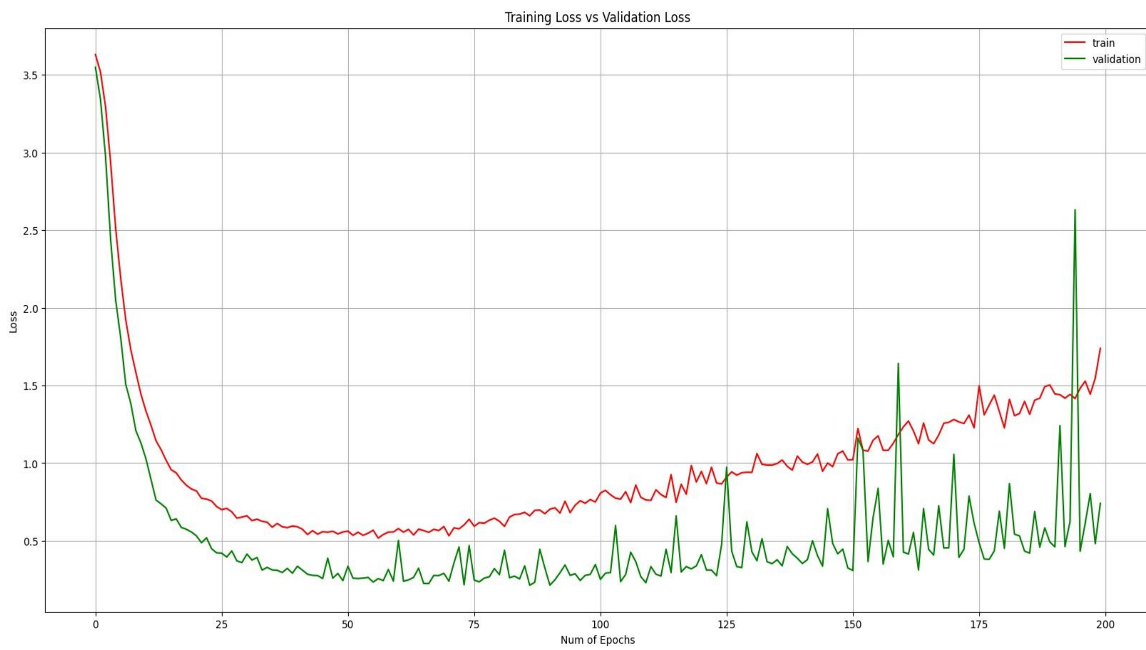


Figura 4-15: Train vs Validation con optimizador RMSprop, 200 épocas, y Dropout 1

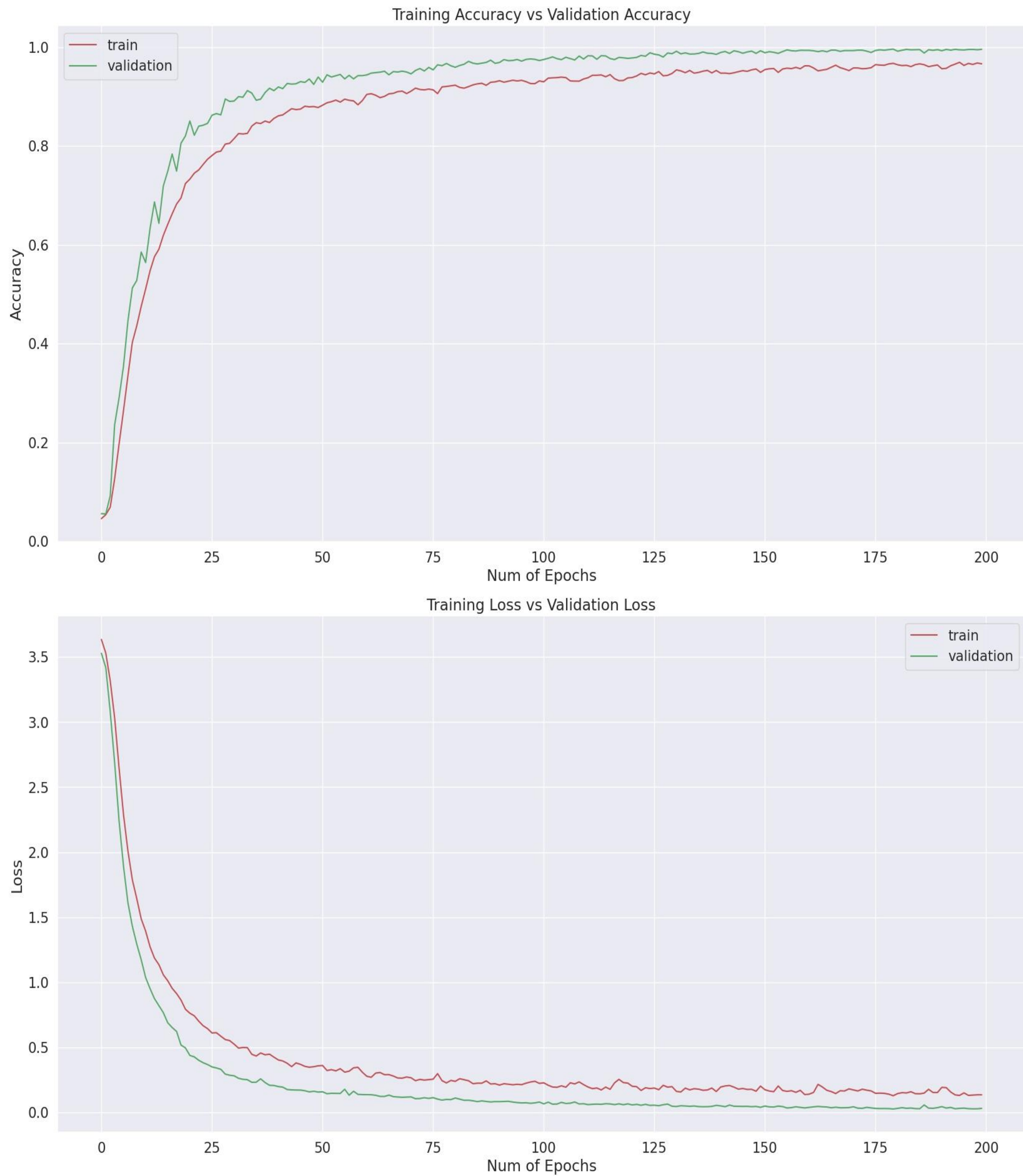


Figura 4-16: Train vs Validation con optimizador Adam, 200 épocas, y Dropout 1

4.4.1.2 Test

Al aplicar el conjunto de test, se ha logrado con el modelo Adam anterior un *accuracy* de test del 97.63%, siendo ya superior al de los modelos de los primeros entrenamientos, mientras que el RMSprop obtuvo un *accuracy* del 76.73 %, empeorando considerablemente los resultados de experimentos anteriores. Por ello, no merece la pena seguir analizando este modelo, así que se muestran a continuación en la figura los resultados de test del modelo Adam:

	precision	recall	f1-score	support
0	1.00	0.97	0.98	60
1	0.99	1.00	0.99	720
2	1.00	0.98	0.99	750
3	1.00	0.99	0.99	450
4	1.00	0.99	0.99	660
5	0.96	0.98	0.97	600
6	1.00	0.99	1.00	120
7	0.98	1.00	0.99	480
8	0.99	0.98	0.98	450
9	1.00	1.00	1.00	480
10	1.00	1.00	1.00	660
11	1.00	0.97	0.99	420
12	1.00	1.00	1.00	690
13	1.00	1.00	1.00	720
14	1.00	1.00	1.00	240
15	1.00	1.00	1.00	210
16	1.00	1.00	1.00	120
17	1.00	1.00	1.00	360
18	1.00	0.98	0.99	390
19	0.70	1.00	0.82	60
20	0.83	0.98	0.90	120
21	0.92	0.73	0.81	90
22	1.00	1.00	1.00	120
23	1.00	0.81	0.89	150
24	0.89	0.99	0.94	90
25	0.93	1.00	0.96	480
26	0.96	0.99	0.98	180
27	1.00	0.87	0.93	60
28	0.84	0.94	0.89	180
29	0.86	0.84	0.85	90
30	1.00	0.89	0.94	150
31	1.00	0.97	0.99	240
32	0.94	1.00	0.97	60
33	0.89	1.00	0.94	209
34	1.00	0.98	0.99	120
35	1.00	0.93	0.97	390
36	1.00	1.00	1.00	120
37	0.90	1.00	0.94	60
38	1.00	0.91	0.95	690
39	1.00	0.93	0.97	90
40	0.73	0.98	0.84	120
41	1.00	0.97	0.98	60
42	0.98	0.98	0.98	60
accuracy			0.98	12569
macro avg	0.96	0.97	0.96	12569
weighted avg	0.98	0.98	0.98	12569

Figura 4-17: Test con optimizador Adam, 200 épocas, y Dropout 1

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40	41	42
0	58	0	0	0	0	0	0	0	2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	717	1	0	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
2	0	1	734	0	0	15	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
3	0	0	0	444	0	6	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
4	0	4	0	0	654	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
5	0	2	2	1	1	591	0	2	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
6	0	0	0	0	0	0	119	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
7	0	0	0	0	0	0	0	479	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
8	0	0	0	0	0	3	0	8	439	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
9	0	0	0	0	0	0	0	0	0	479	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0
10	0	0	0	0	0	0	0	0	0	0	660	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
11	0	0	0	0	0	0	0	0	0	0	0	409	0	0	0	0	0	0	0	0	0	0	0	0	0	9	0	0	2	0	0	0	0	0	0	0	0	0	0	0	0	0	0
12	0	0	0	0	0	0	0	0	0	0	0	0	687	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	3	0	0
13	0	0	0	0	0	0	0	0	0	0	0	0	0	720	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
14	0	0	0	0	0	0	0	0	0	0	0	0	0	0	239	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0
15	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	209	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
16	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	120	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
17	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	359	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
18	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	383	0	0	0	0	0	0	0	7	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
19	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	60	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
20	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	118	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
21	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	4	0	66	0	0	0	14	0	0	1	5	0	0	0	0	0	0	0	0	0	0	0	0	0
22	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	120	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
23	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	17	7	0	0	121	0	0	0	0	1	1	0	1	0	0	0	0	0	0	0	0	1	0	0
24	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	89	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
25	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	479	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0
26	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	178	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
27	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	7	1	0	52	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
28	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	5	0	0	0	1	1	0	0	170	3	0	0	0	0	0	0	0	0	0	0	0	0	0
29	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	11	0	0	2	76	0	0	0	0	0	0	0	0	0	0	0	0	0
30	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	16	0	134	0	0	0	0	0	0	0	0	0	0	0	0
31	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	6	0	0	0	0	0	0	0	0	0	234	0	0	0	0	0	0	0	0	0	0	0
32	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	60	0	0	0	0	0	0	0	0	0	0
33	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	208	0	0	0	1	0	0	0	0	0
34	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	118	0	0	0	0	0	2	0	0
35	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	1	0	0	0	0	0	0	0	21	0	364	0	3	0	0	0	0	0
36	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	120	0	0	0	0	0	0
37	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	60	0	0	0	0	0
38	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	2	12	0	0	0	2	0	0	0	7	3	0	0	0	0	0	0	0	2	625	0	36	0	0
39	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	4	0	0	0	1	0	84	0	0	0
40	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	118	0	0
41	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	2	0	0	0	0	0	0	0	0	58	0
42	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	59

Figura 4-18: Matriz de confusión Adam 200 epochs Dropout 1

4.4.2 Dropout 2

4.4.2.1 Entrenamiento y validación

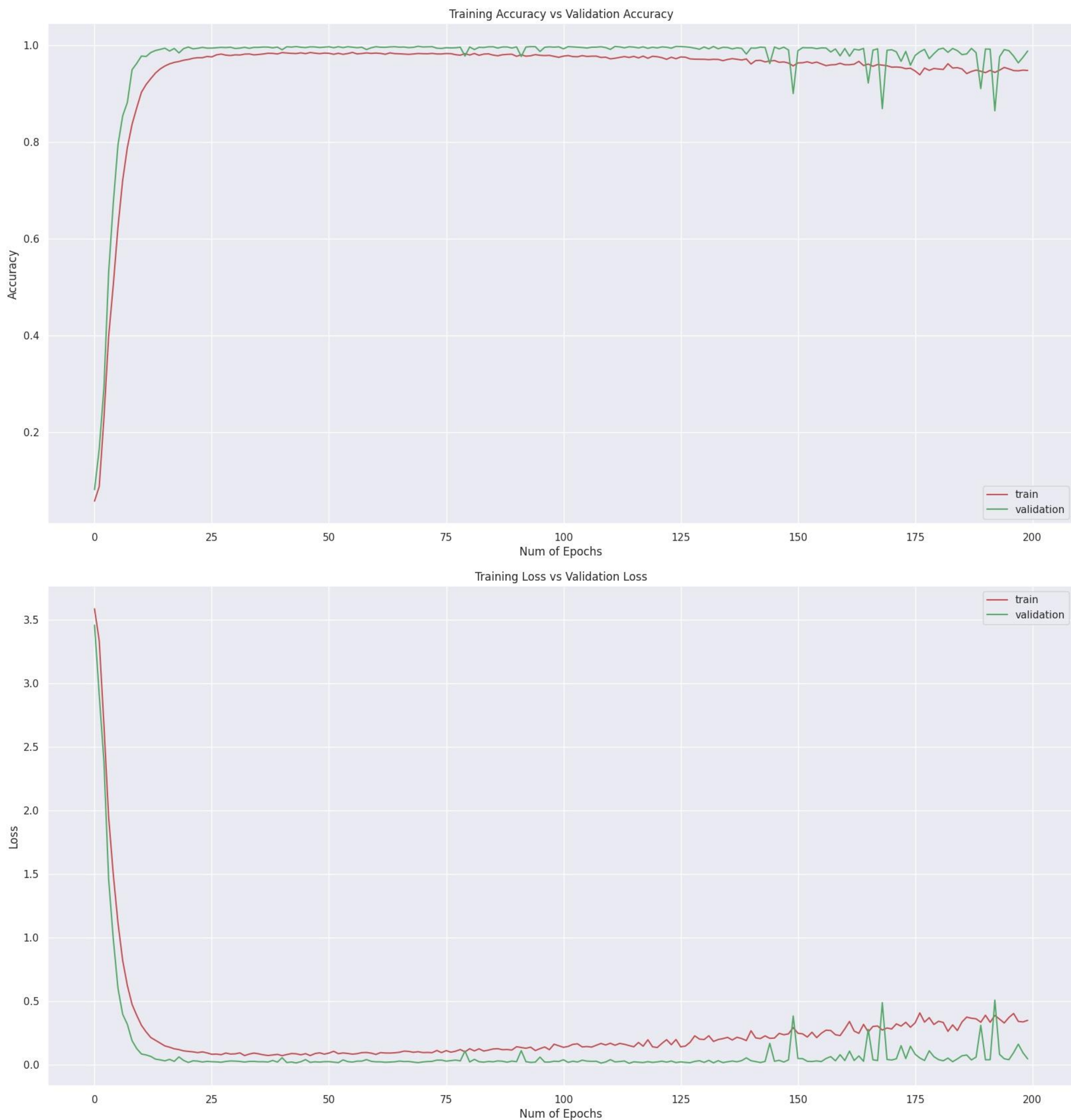


Figura 4-19: Train vs Validation con optimizador RMSprop, 200 épocas, y Dropout 2

En el caso del RMSprop, se ve en la Figura 4-19 que el entrenamiento tiene tendencia a separarse de la validación, pero como no es la validación la que tiene mayor *loss* que el entrenamiento no se ha producido overfitting, sino que es un caso de underfitting otra vez. Este problema se acusa más a partir de la época 75, donde la pendiente de la *loss* de entrenamiento aumenta mientas que la de validación permanece constante, apareciendo en este tramo bastante variabilidad en los resultados, debido a la configuración neuronal, llegando incluso a sobrepasar al entrenamiento en algunos puntos. Esto lleva aparejado un efecto parecido en el *accuracy*, donde el entrenamiento empieza a disminuir y aparece la misma variabilidad en la validación. El modelo que se obtuvo al final tenía un 94.83% de *accuracy* de entrenamiento y un 98.85% de *accuracy* de validación.

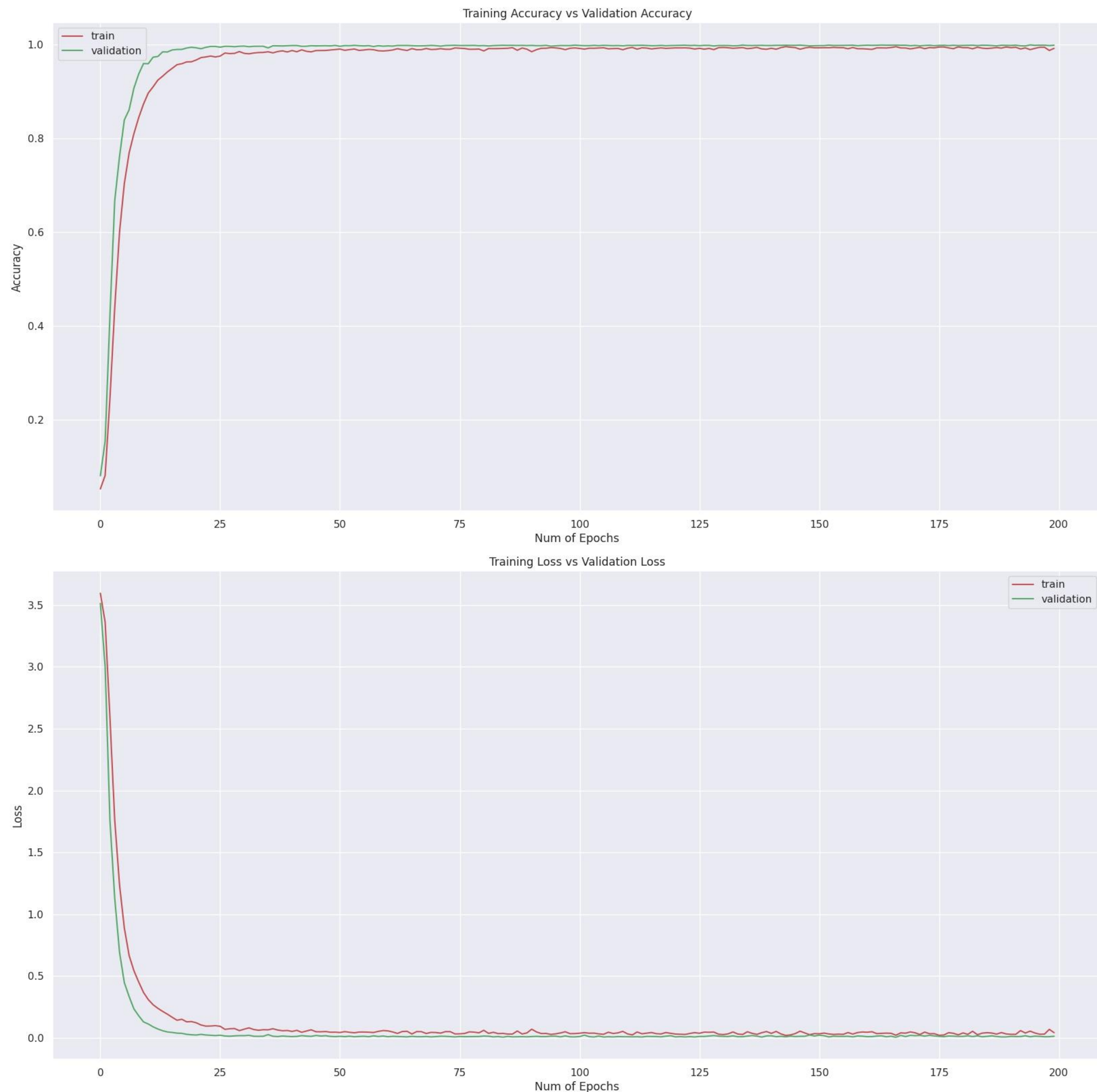


Figura 4-20: Train vs Validation con optimizador Adam, 200 épocas, y Dropout 2

Todo lo anterior no se ve para nada en el caso de usar Adam como algoritmo de optimización (Figura 4-20) y es que las gráficas obtenidas son bastante suaves, con los valores de entrenamiento muy apegados a los de validación, sin overfitting y prácticamente sin underfitting, por lo que en principio este modelo debería ser bastante bueno, ya que los valores de *accuracy* alcanzados son casi del 100% y los valores de *loss* son muy próximos a 0 tanto en entrenamiento como en validación. El modelo obtenido tiene una *accuracy* de entrenamiento del 99.21% y una *accuracy* de validación del 99.89 %.

En ambos casos, se puede apreciar que el número de épocas es excesivo porque se estabiliza en épocas muy iniciales del entrenamiento, por lo que se harán de nuevo los experimentos con 50 épocas en lugar de 200. En la Figura 4-21 se puede ver como el RMSprop ya no sufre de un underfitting tan agresivo, acercándose los valores de entrenamiento a los de validación, por lo que deberían obtenerse mejores resultados que con 200 épocas. El modelo obtenido tiene una *accuracy* de entrenamiento del 98.25% y una *accuracy* de validación del 99.76 %. Para el Adam también se obtuvieron resultados parecidos, mostrados en la Figura 4-22, obteniendo finalmente un modelo con 98.85 % de *accuracy* de entrenamiento y un 99.72 % en la validación.

A priori, estos modelos deberían ser mejores que los que se obtuvieron en la primera aproximación de 200 épocas, por lo que se pasará a comprobar su comportamiento con el conjunto de testeo.

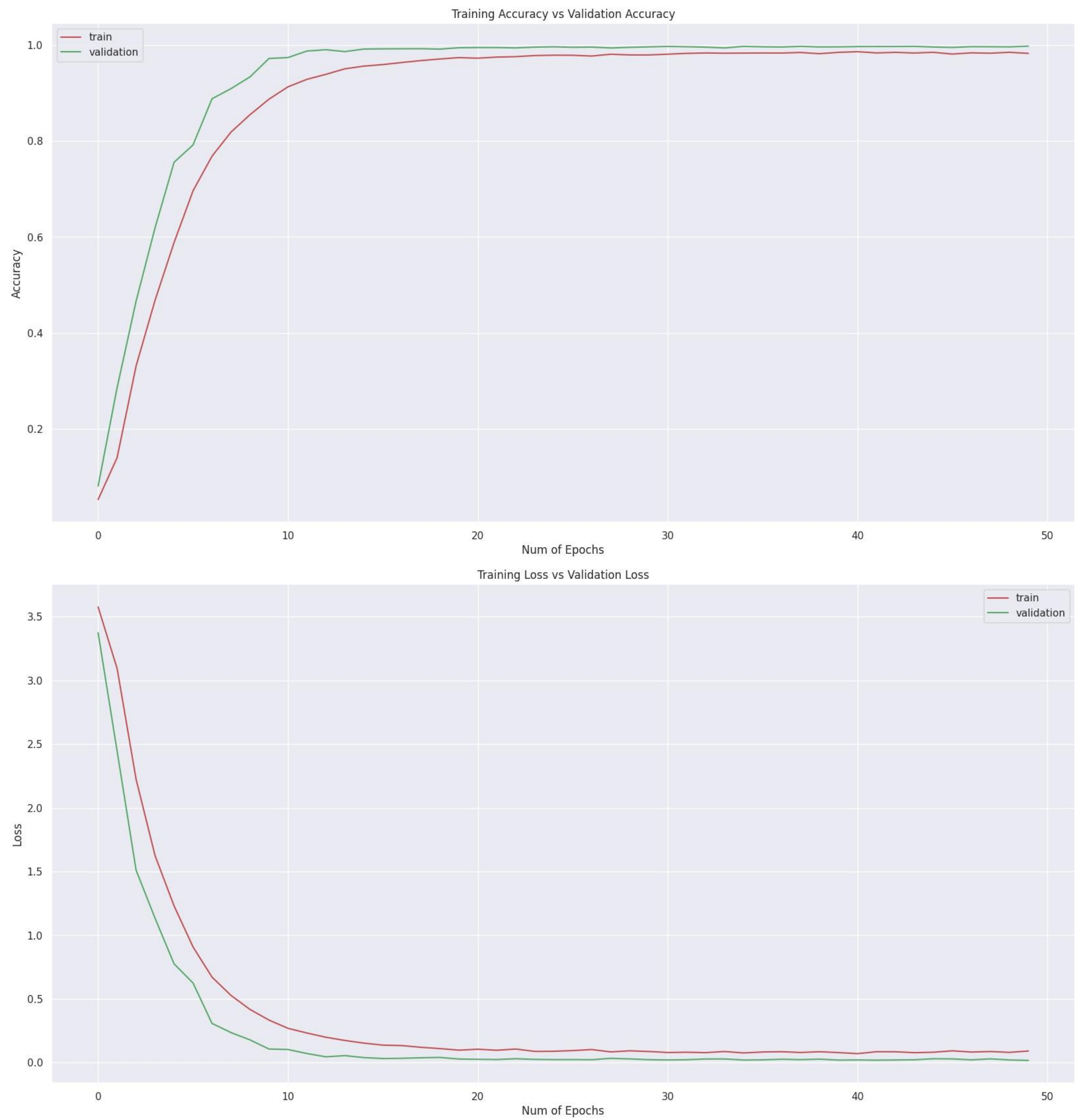


Figura 4-21: Train vs Validation con optimizador RMSprop, 50 épocas, y Dropout 2

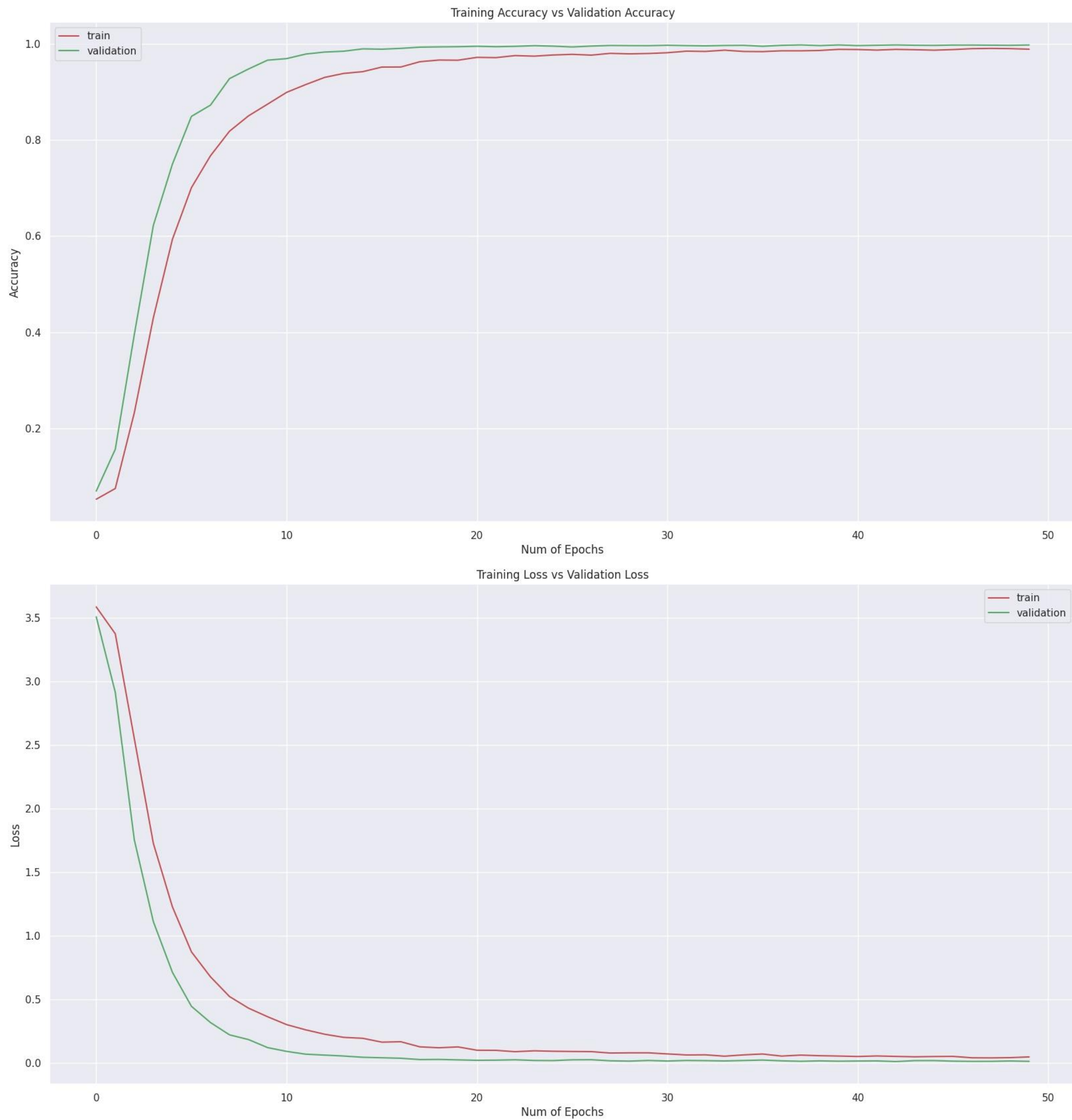


Figura 4-22: Train vs Validation con optimizador Adam, 50 épocas, y Dropout 2

4.4.2.2 Test

Al aplicar el conjunto de test, se ha logrado con el modelo RMSprop de 200 épocas de entrenamiento un valor de *accuracy* de test del 96.24%, prácticamente igualando los obtenidos en los primeros experimentos, mientras que con el modelo con optimizador Adam se ha obtenido un *accuracy* de test del **98.04%**, mejorando el porcentaje más alto alcanzado hasta el momento. Al disminuir el número de épocas a 50, el Adam prácticamente tiene el mismo resultado, con un valor de *accuracy* del **98.03 %**, pero en el caso del optimizador RMSprop se llegó a obtener un valor del **98.21 %**, siendo la mejor de las redes obtenidas en todos los experimentos hasta este punto. Los resultados de test se muestran con más detalle a continuación:

	precision	recall	f1-score	support
0	1.00	0.88	0.94	60
1	0.97	0.98	0.98	720
2	0.99	0.93	0.96	750
3	1.00	0.97	0.98	450
4	1.00	0.99	0.99	660
5	0.69	1.00	0.81	600
6	1.00	1.00	1.00	120
7	0.99	0.98	0.99	480
8	1.00	0.93	0.96	450
9	0.95	1.00	0.97	480
10	1.00	0.98	0.99	660
11	1.00	0.97	0.99	420
12	1.00	0.98	0.99	690
13	1.00	1.00	1.00	720
14	1.00	0.96	0.98	240
15	0.99	0.99	0.99	210
16	0.99	1.00	1.00	120
17	1.00	1.00	1.00	360
18	0.96	0.99	0.97	390
19	1.00	1.00	1.00	60
20	0.99	0.97	0.98	120
21	1.00	0.59	0.74	90
22	1.00	1.00	1.00	120
23	1.00	0.80	0.89	150
24	0.98	0.97	0.97	90
25	1.00	0.97	0.99	480
26	0.97	0.93	0.95	180
27	0.98	0.98	0.98	60
28	1.00	0.92	0.96	180
29	0.71	0.83	0.77	90
30	0.92	0.99	0.95	150
31	0.85	0.99	0.92	240
32	1.00	1.00	1.00	60
33	0.98	1.00	0.99	209
34	1.00	1.00	1.00	120
35	1.00	0.90	0.95	390
36	1.00	0.99	1.00	120
37	1.00	1.00	1.00	60
38	1.00	0.91	0.96	690
39	1.00	0.96	0.98	90
40	0.77	0.62	0.69	120
41	0.98	1.00	0.99	60
42	1.00	0.98	0.99	60
accuracy			0.96	12569
macro avg	0.97	0.95	0.96	12569
weighted avg	0.97	0.96	0.96	12569

	precision	recall	f1-score	support
0	1.00	0.97	0.98	60
1	0.98	0.99	0.98	720
2	0.99	0.98	0.99	750
3	0.96	0.99	0.98	450
4	1.00	1.00	1.00	660
5	0.96	0.97	0.97	600
6	1.00	1.00	1.00	120
7	1.00	1.00	1.00	480
8	0.99	1.00	0.99	450
9	1.00	1.00	1.00	480
10	0.99	1.00	1.00	660
11	1.00	0.99	1.00	420
12	1.00	1.00	1.00	690
13	0.99	1.00	0.99	720
14	1.00	1.00	1.00	240
15	0.98	1.00	0.99	210
16	1.00	1.00	1.00	120
17	1.00	1.00	1.00	360
18	1.00	0.97	0.98	390
19	0.97	1.00	0.98	60
20	0.92	1.00	0.96	120
21	0.99	0.78	0.87	90
22	0.97	1.00	0.98	120
23	0.97	0.94	0.96	150
24	0.96	0.88	0.92	90
25	0.97	1.00	0.98	480
26	0.90	0.96	0.93	180
27	0.88	0.98	0.93	60
28	0.99	0.97	0.98	180
29	0.94	0.86	0.90	90
30	0.97	0.94	0.96	150
31	0.93	1.00	0.96	240
32	0.98	1.00	0.99	60
33	0.99	1.00	0.99	209
34	0.99	1.00	1.00	120
35	0.94	1.00	0.97	390
36	0.99	1.00	1.00	120
37	0.82	1.00	0.90	60
38	1.00	0.91	0.95	690
39	0.99	0.82	0.90	90
40	0.90	0.78	0.84	120
41	1.00	1.00	1.00	60
42	1.00	1.00	1.00	60
accuracy			0.98	12569
macro avg	0.97	0.97	0.97	12569
weighted avg	0.98	0.98	0.98	12569

Figura 4-23: Test RMSprop (izquierda) y Adam (derecha) para Dropout 2 con 200 epochs

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40	41	42
0	53	5	0	0	0	1	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	709	0	0	1	9	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
2	0	4	697	0	0	49	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
3	0	0	0	435	0	15	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
4	0	3	0	0	652	5	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
5	0	0	0	1	0	598	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
6	0	0	0	0	0	0	120	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
7	0	0	0	0	0	11	0	469	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
8	0	0	0	0	0	27	0	3	420	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
9	0	0	0	0	0	2	0	0	0	478	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
10	0	0	0	0	0	3	0	0	0	8	649	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
11	0	0	0	0	0	3	0	0	0	0	0	408	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	8	0	0	0	0	0	0	0	0	0	0	0	0
12	0	0	0	0	0	0	0	0	0	15	0	0	675	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
13	0	0	0	0	0	0	0	0	0	0	0	0	0	720	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
14	0	0	1	0	0	9	0	0	0	0	0	0	0	0	230	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
15	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	208	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
16	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	120	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
17	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	360	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
18	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	387	0	0	0	0	0	0	0	2	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0
19	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	60	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
20	0	0	0	0	0	2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	117	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0
21	0	0	0	0	0	5	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	53	0	0	0	0	0	0	0	2	0	30	0	0	0	0	0	0	0	0	0	0	0
22	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	120	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
23	0	0	0	0	0	2	0	0	0	0	0	0	0	0	0	0	0	0	2	0	0	0	0	120	0	0	0	0	0	25	0	1	0	0	0	0	0	0	0	0	0	0	0
24	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	87	0	2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
25	0	0	0	0	0	7	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	467	0	0	0	2	2	1	0	0	0	0	0	0	0	0	0	0	0
26	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	11	0	0	0	0	0	0	0	168	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
27	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	59	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
28	0	1	0	0	0	8	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	2	0	0	0	166	0	3	0	0	0	0	0	0	0	0	0	0	0	0
29	0	0	0	0	0	8	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	75	0	6	0	0	0	0	0	0	0	0	0	0	0
30	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	148	1	0	0	0	0	0	0	0	0	0	0	0
31	0	0	0	0	0	2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	238	0	0	0	0	0	0	0	0	0	0	0
32	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	60	0	0	0	0	0	0	0	0	0	0
33	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	209	0	0	0	0	0	0	0	0	0
34	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	120	0	0	0	0	0	0	0	0
35	0	0	5	0	0	31	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	2	0	351	0	0	0	0	0	0	0
36	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	119	0	0	0	0	0	0
37	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	60	0	0	0	0	0
38	0	11	0	0	0	23	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	631	0	23	0	0
39	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	2	0	0	0	0	0	86	0	0	0
40	0	0	0	0	0	45	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	75	0	0
41	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	60	0
42	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	59

Figura 4-24: Matriz de confusión RMSprop 200 epochs Dropout 2

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40	41	42
0	58	0	0	0	0	0	0	0	2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	715	3	0	0	1	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
2	0	0	738	0	0	12	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
3	0	0	0	446	0	4	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
4	0	0	0	0	658	0	0	0	1	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
5	0	0	0	14	0	583	0	0	1	0	1	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
6	0	0	0	0	0	0	120	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
7	0	0	0	0	0	0	0	480	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
8	0	0	0	0	0	0	0	1	449	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
9	0	0	0	0	0	0	0	0	0	480	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
10	0	0	0	0	0	0	0	0	0	0	660	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
11	0	0	0	0	0	0	0	0	0	0	0	416	0	0	0	0	0	0	0	0	0	0	0	1	0	1	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0
12	0	0	0	0	0	0	0	0	0	0	0	0	690	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
13	0	0	0	0	0	0	0	0	0	0	0	0	0	720	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
14	0	0	0	0	0	0	0	0	0	0	0	0	0	0	239	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0
15	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	210	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
16	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	120	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
17	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	360	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
18	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	377	0	0	0	0	0	0	0	13	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
19	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	60	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
20	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	120	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
21	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	2	0	70	0	0	0	0	1	0	0	4	0	13	0	0	0	0	0	0	0	0	0	0	0
22	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	120	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
23	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	9	0	0	141	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
24	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	79	0	4	6	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
25	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	478	0	1	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0
26	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	4	0	0	3	173	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
27	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	59	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
28	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	1	1	174	0	2	0	0	0	0	0	0	0	0	0	0	0	0
29	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	3	0	6	1	0	0	77	0	1	0	0	0	0	0	1	0	0	0	0	0
30	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	4	0	0	0	0	141	4	0	0	0	0	0	0	0	0	0	0	0
31	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	240	0	0	0	0	0	0	0	0	0	0	0
32	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	60	0	0	0	0	0	0	0	0	0	0
33	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	209	0	0	0	0	0	0	0	0	0
34	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	120	0	0	0	0	0	0	0	0
35	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	389	0	0	0	0	0	0	0
36	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	120	0	0	0	0	0	0
37	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	60	0	0	0	0	0
38	0	15	0	1	0	0	0	0	0	0	1	0	0	4	0	1	0	0	0	0	1	0	0	0	0	0	0	0	0	0	1	0	1	0	0	19	0	8	626	1	11	0	0
39	0	0	1	1	0	1	0	0	1	0	0	0	0	3	0	2	0	0	0	0	0	0	0	0	1	2	0	0	1	0	0	0	0	3	0	0	0	0	0	74	0	0	0
40	0	3	4	0	0	6	0	0	0	0	2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	7	0	4	0	0	94	0	0
41	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	60	0
42	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	60

Figura 4-25: Matriz de confusión Adam 200 epochs Dropout 2

	precision	recall	f1-score	support
0	0.98	0.93	0.96	60
1	0.99	0.99	0.99	720
2	0.96	0.98	0.97	750
3	0.98	1.00	0.99	450
4	1.00	0.99	1.00	660
5	0.96	0.99	0.98	600
6	0.98	1.00	0.99	120
7	1.00	0.99	0.99	480
8	0.98	0.99	0.99	450
9	0.97	1.00	0.98	480
10	1.00	1.00	1.00	660
11	1.00	1.00	1.00	420
12	0.99	1.00	0.99	690
13	1.00	1.00	1.00	720
14	1.00	1.00	1.00	240
15	0.95	1.00	0.97	210
16	1.00	1.00	1.00	120
17	1.00	1.00	1.00	360
18	1.00	0.99	0.99	390
19	0.97	1.00	0.98	60
20	0.94	1.00	0.97	120
21	1.00	0.84	0.92	90
22	0.95	1.00	0.98	120
23	0.99	0.93	0.96	150
24	0.96	0.81	0.88	90
25	1.00	1.00	1.00	480
26	0.98	0.99	0.98	180
27	0.77	1.00	0.87	60
28	0.99	1.00	0.99	180
29	0.94	0.93	0.94	90
30	0.99	0.99	0.99	150
31	0.98	0.99	0.98	240
32	0.82	1.00	0.90	60
33	0.98	1.00	0.99	209
34	0.95	1.00	0.98	120
35	1.00	0.98	0.99	390
36	0.99	1.00	1.00	120
37	0.98	1.00	0.99	60
38	1.00	0.90	0.95	690
39	1.00	0.82	0.90	90
40	0.82	0.82	0.82	120
41	0.98	0.95	0.97	60
42	1.00	1.00	1.00	60
accuracy			0.98	12569
macro avg	0.97	0.97	0.97	12569
weighted avg	0.98	0.98	0.98	12569

	precision	recall	f1-score	support
0	1.00	0.93	0.97	60
1	1.00	0.99	0.99	720
2	0.99	0.99	0.99	750
3	0.99	0.98	0.99	450
4	1.00	0.99	1.00	660
5	0.97	0.98	0.97	600
6	0.99	1.00	1.00	120
7	0.98	1.00	0.99	480
8	0.99	0.98	0.98	450
9	1.00	1.00	1.00	480
10	1.00	1.00	1.00	660
11	0.99	0.99	0.99	420
12	1.00	1.00	1.00	690
13	1.00	1.00	1.00	720
14	1.00	0.99	1.00	240
15	0.99	1.00	0.99	210
16	1.00	1.00	1.00	120
17	1.00	1.00	1.00	360
18	0.97	0.99	0.98	390
19	0.97	1.00	0.98	60
20	0.87	0.99	0.93	120
21	1.00	0.69	0.82	90
22	0.94	1.00	0.97	120
23	0.88	0.92	0.90	150
24	0.99	0.94	0.97	90
25	1.00	0.99	0.99	480
26	0.97	0.95	0.96	180
27	0.94	1.00	0.97	60
28	0.99	0.96	0.97	180
29	0.86	0.78	0.82	90
30	0.97	0.96	0.97	150
31	0.94	1.00	0.97	240
32	0.81	1.00	0.90	60
33	0.97	1.00	0.98	209
34	1.00	0.96	0.98	120
35	1.00	0.99	0.99	390
36	0.99	1.00	1.00	120
37	0.87	1.00	0.93	60
38	1.00	0.91	0.95	690
39	0.97	0.94	0.96	90
40	0.74	0.96	0.83	120
41	1.00	1.00	1.00	60
42	0.98	1.00	0.99	60
accuracy			0.98	12569
macro avg	0.96	0.97	0.97	12569
weighted avg	0.98	0.98	0.98	12569

Figura 4-26: Test RMSprop (izquierda) y Adam (derecha) para Dropout 2 con 50 epochs

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40	41	42
0	56	1	0	0	0	0	0	0	3	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	1	715	2	0	0	0	0	0	1	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
2	0	0	738	2	0	9	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
3	0	0	0	448	0	2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
4	0	1	2	1	655	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
5	0	0	3	3	0	592	0	1	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
6	0	0	0	0	0	0	120	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
7	0	0	0	0	0	1	0	477	2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
8	0	0	0	0	0	2	0	1	447	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
9	0	0	0	0	0	0	0	0	0	479	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
10	0	0	0	0	0	0	0	0	0	1	659	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
11	0	0	0	0	0	0	0	0	0	0	0	418	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0
12	0	0	0	0	0	0	0	0	0	2	0	0	688	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
13	0	0	0	0	0	0	0	0	0	0	0	0	0	720	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
14	0	0	0	0	0	0	0	0	0	0	0	0	0	0	239	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0
15	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	210	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
16	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	120	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
17	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	360	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
18	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	387	0	0	0	1	0	0	0	2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
19	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	60	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
20	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	120	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
21	0	0	0	0	0	3	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	76	1	0	3	0	0	1	0	1	0	5	0	0	0	0	0	0	0	0	0	0	0
22	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	120	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
23	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	1	4	0	2	140	0	0	0	0	0	2	0	0	0	0	0	0	0	0	0	0	0	0	0
24	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	73	0	2	15	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
25	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	478	0	1	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0
26	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	178	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0
27	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	60	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
28	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	180	0	0	0	0	0	0	0	0	0	0	0	0	0	0
29	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	2	0	0	0	0	0	2	84	0	1	0	0	0	0	0	0	0	0	0	0	0
30	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	2	0	0	0	0	148	0	0	0	0	0	0	0	0	0	0	0	0
31	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	2	0	0	0	0	0	0	0	0	0	0	237	0	0	0	0	0	0	0	0	0	0	0
32	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	60	0	0	0	0	0	0	0	0	0	0
33	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	209	0	0	0	0	0	0	0	0	0
34	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	120	0	0	0	0	0	0	0	0
35	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	2	4	0	382	0	1	0	0	0	0	0
36	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	120	0	0	0	0	0	0
37	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	60	0	0	0	0	0
38	0	1	11	2	0	1	2	0	0	10	0	0	5	0	0	3	0	0	0	0	1	0	0	1	0	0	0	0	0	1	0	0	1	0	6	0	1	0	621	0	22	1	0
39	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	7	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	9	0	0	0	0	0	0	74	0	0	0
40	0	1	11	0	0	3	0	0	0	1	0	0	4	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	99	0	0
41	0	0	0	0	0	0	0	0	0	3	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	57	0
42	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	60

Figura 4-27: Matriz de confusión RMSprop 50 epochs Dropout 2

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40	41	42
0	56	1	0	0	1	0	0	0	2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	714	2	0	0	2	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
2	0	0	742	0	0	7	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
3	0	0	1	443	0	5	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0
4	0	0	2	0	656	0	0	0	2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
5	0	0	2	4	0	588	0	3	0	0	2	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
6	0	0	0	0	0	0	120	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
7	0	0	0	0	0	0	0	480	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
8	0	0	0	0	0	5	0	6	439	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
9	0	0	0	0	0	0	0	0	0	479	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
10	0	0	0	0	0	0	0	0	0	2	658	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
11	0	0	0	0	0	0	0	0	0	0	0	415	0	0	0	0	0	0	0	0	2	0	0	0	0	0	2	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0
12	0	0	0	0	0	0	0	0	0	0	0	0	688	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	1	0	0
13	0	0	0	0	0	0	0	0	0	0	0	0	0	720	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
14	0	0	0	0	0	0	0	0	0	0	0	0	0	0	238	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0
15	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	210	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
16	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	120	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
17	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	360	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
18	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	387	0	0	0	0	0	1	0	2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
19	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	60	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
20	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	119	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
21	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	2	0	62	0	3	0	0	1	0	0	6	1	14	0	0	0	0	0	0	0	0	0	0	0
22	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	120	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
23	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	11	0	0	138	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
24	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	3	0	0	0	0	0	85	0	0	2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
25	0	0	0	0	0	0	0	0	0	0	0	2	1	0	0	0	0	0	0	0	0	0	0	0	0	476	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0
26	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	9	0	0	0	0	0	0	0	171	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
27	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	60	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
28	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	2	0	1	172	3	1	0	0	0	0	0	0	1	0	0	0	0	0
29	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	6	13	0	0	0	0	0	70	0	0	0	0	0	0	0	0	0	0	0	0	0
30	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	2	0	0	0	0	0	0	0	0	2	144	1	0	0	0	0	0	0	0	0	0	0	0
31	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	239	0	0	0	0	0	0	0	0	0	0	0
32	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	60	0	0	0	0	0	0	0	0	0	0
33	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	208	0	0	0	0	0	1	0	0	0
34	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	5	0	115	0	0	0	0	0	0	0	0
35	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	4	0	385	1	0	0	0	0	0	0
36	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	120	0	0	0	0	0	0
37	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	60	0	0	0	0	0
38	0	1	0	1	0	0	1	0	0	0	0	0	0	0	0	2	0	0	0	0	2	0	2	0	0	0	0	0	1	0	0	0	6	1	0	0	0	8	625	0	40	0	0
39	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	2	1	0	0	0	0	0	85	0	0	0
40	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	2	1	115	0	1
41	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	60	0
42	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	60

Figura 4-28: Matriz de confusión Adam 50 epochs Dropout 2

4.4.3 Dropout 3

4.4.3.1 Entrenamiento y validación

Se empezó de nuevo con experimentos de 200 épocas para ver el comportamiento del modelo con este tipo de Dropout.

En la Figura 4-30 se aprecia que el RMSprop sufre underfitting, aumentando aproximadamente a partir de la época 60. El modelo obtenido al final tiene un valor de *accuracy* del 97.87 % en el entrenamiento y un 99.70 % en la validación. A pesar de haber underfitting, al ser de valor tan reducido los resultados de test deberían ser aceptables.

En la Figura 4-29 que recoge los resultados con el optimizador Adam, sin embargo, se aprecia que, los valores de *accuracy* de entrenamiento y validación prácticamente son iguales una vez que se estabiliza, con valores cercanos al 100% y con valores de *loss* aún más cercanos a 0 que en el RMSprop, por lo que también parece un modelo razonablemente bueno a priori, incluso mejor que en el RMSprop, ya que el underfitting está menos acusado. El valor de *accuracy* de entrenamiento obtenido para el modelo final fue del 99.55%, mientras que para la de validación se obtuvo un 99.86%.

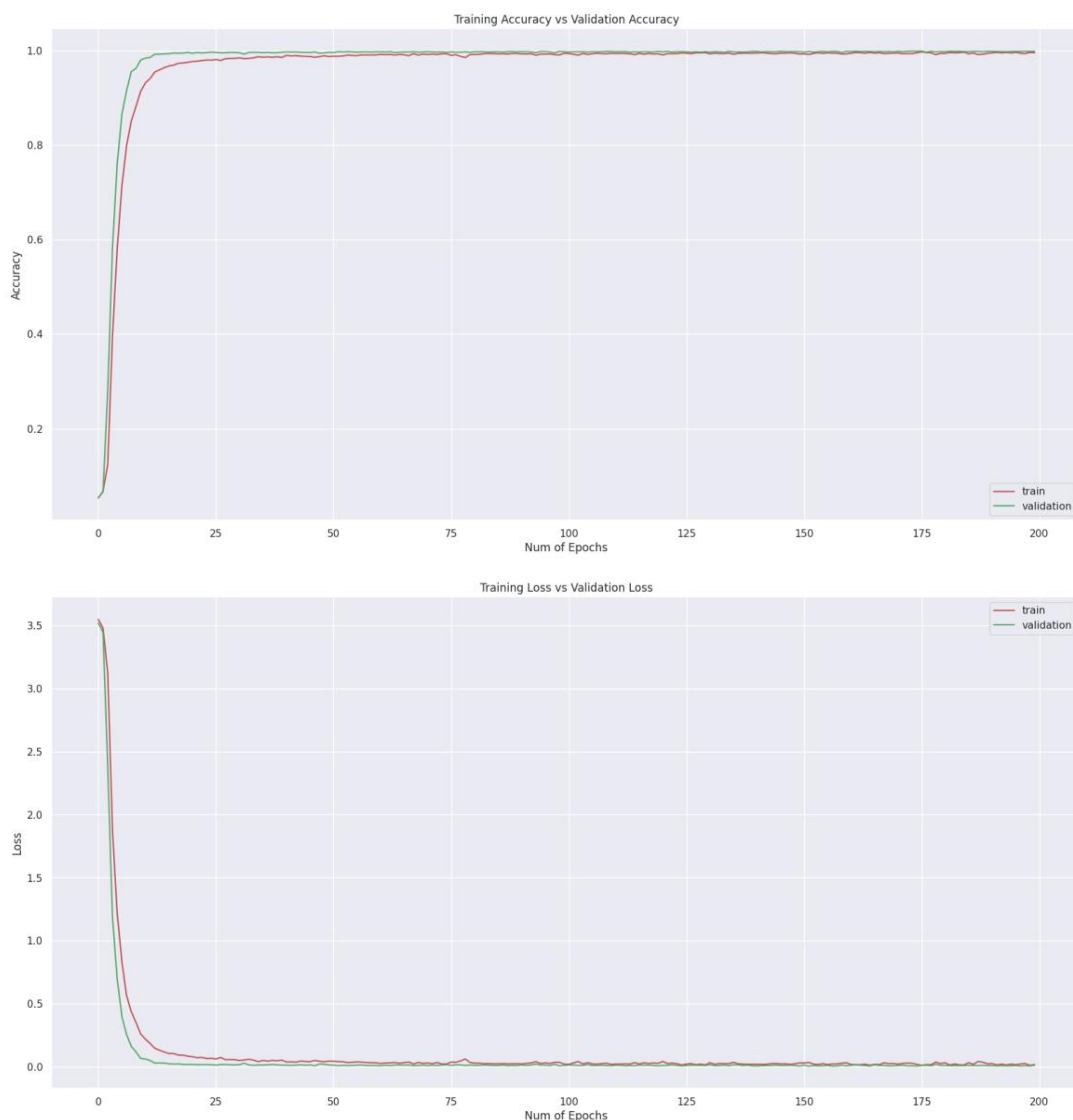


Figura 4-29: Train vs Validation con optimizador Adam, 200 épocas, y Dropout 3

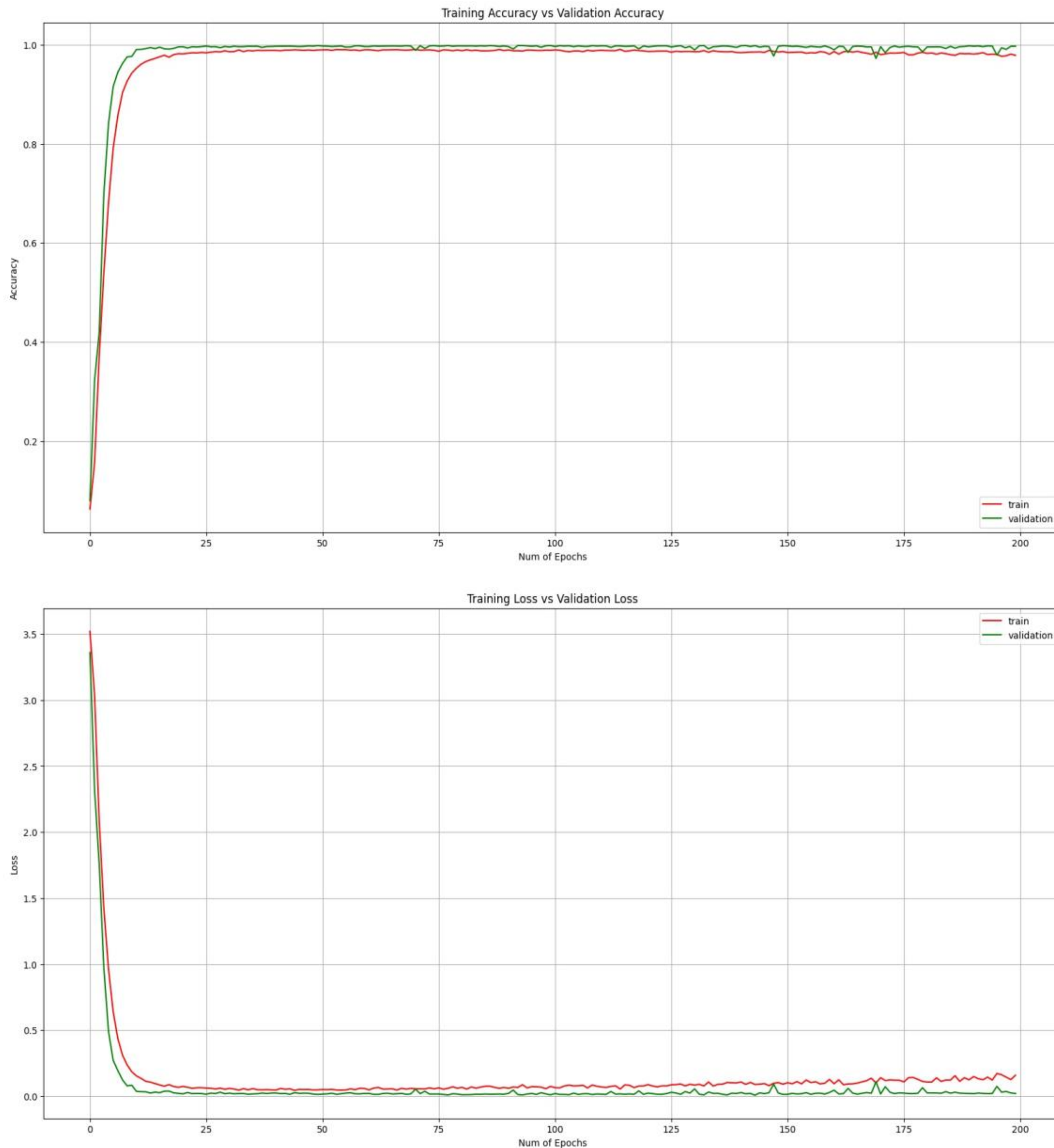


Figura 4-30. Train vs Validation con optimizador RMSprop, 200 épocas, y Dropout 3

En ambos se aprecia que los valores se estabilizan en un número de épocas anterior, por tanto, se harán experimentos con un menor número de épocas, para ver mejor la variación del entrenamiento respecto de la validación. El número de épocas elegido para ambos experimentos es de 50 épocas.

A la vista de los resultados obtenidos en la Figura 4-31 y Figura 4-32, se elimina en gran parte el underfitting que sufría el RMSprop, y ambos experimentos obtienen resultados bastante parecidos entre sí, de nuevo con valores *loss* muy próximos a 0 y con valores de *accuracy* próximos al 100 %, aunque estando siempre el entrenamiento por debajo de la validación. Para el RMSprop se obtuvo un valor de *accuracy* de test del 98.88% mientras que en validación fue del 99.75 %. Para el Adam se obtuvo un 98.96% en entrenamiento y un 99.77% en validación.

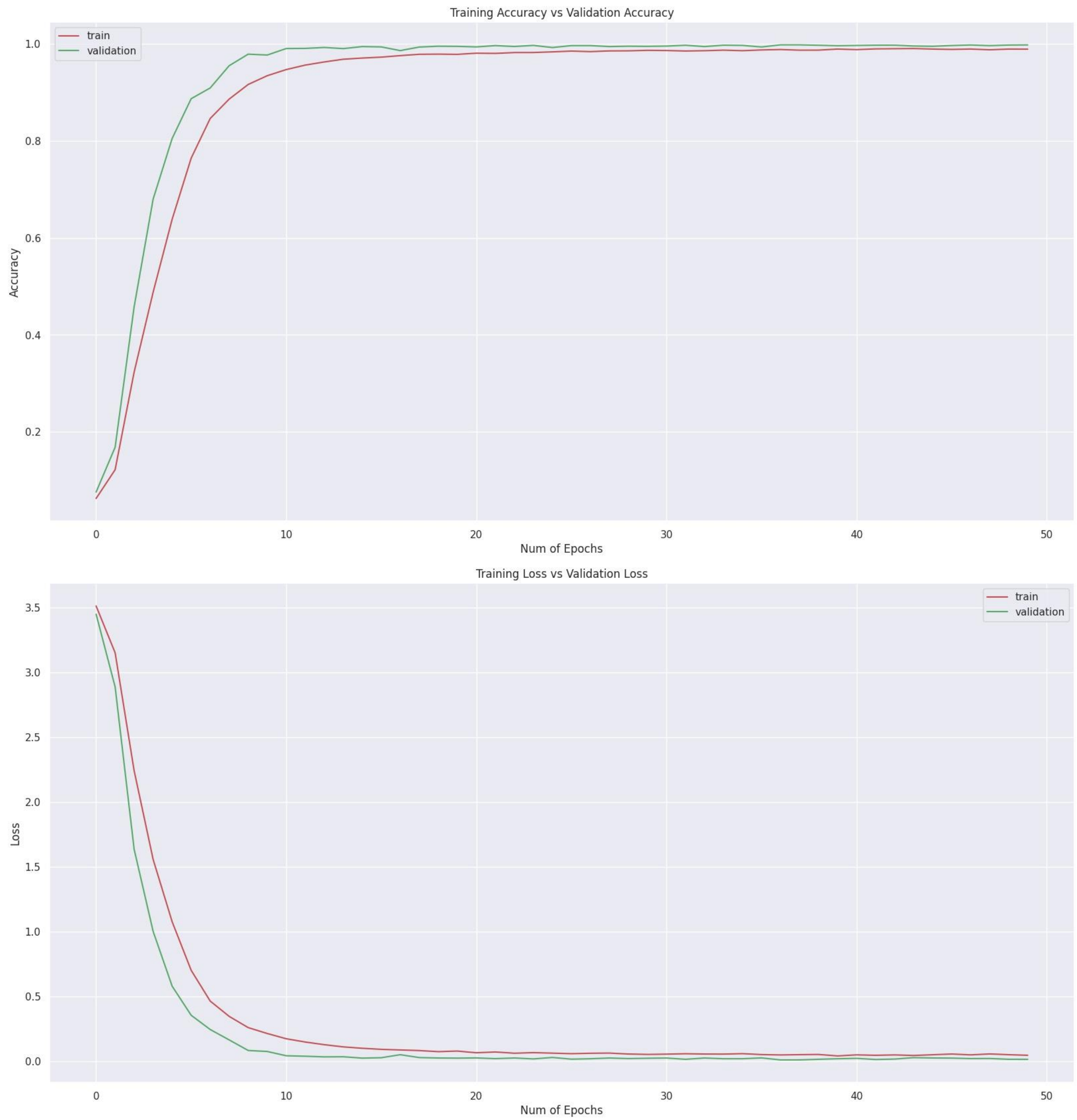


Figura 4-31: Train vs Validation RMSprop 50 epochs y Dropout 3

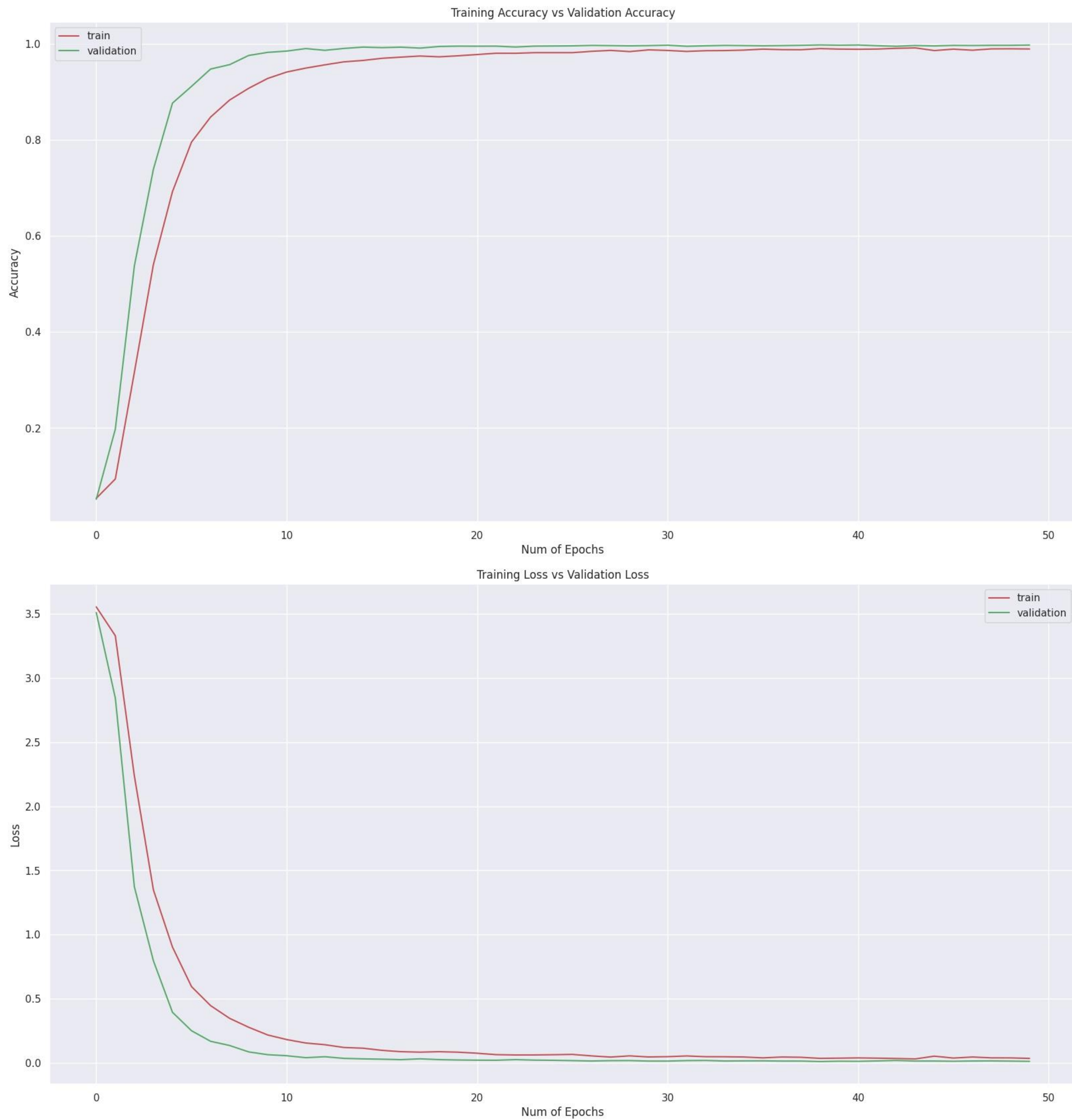


Figura 4-32: Train vs Validation Adam 50 epochs y Dropout 3

4.4.3.2 Test

Al aplicar el conjunto de test, en los experimentos con 50 epochs se logró un *accuracy* de test del 97.70 % para el RMSprop, un valor algo inferior al obtenido en el Dropout 2, que fue de un 98.21 %, mientras que en el Adam prácticamente no varía, obteniendo un valor del 98.01 %. En los experimentos de 200 epochs, a pesar de lo comentado anteriormente, se han obtenido unos modelos con *accuracy* **98.22 %** para el RMSprop, muy parecido al que se obtuvo en el Dropout 2, mientras que para el optimizador Adam se ha obtenido un **98.51 %**, siendo ambos valores los más altos alcanzados.

	precision	recall	f1-score	support
0	0.81	0.93	0.87	60
1	0.98	1.00	0.99	720
2	0.99	0.97	0.98	750
3	0.99	1.00	0.99	450
4	0.99	1.00	0.99	660
5	0.98	0.98	0.98	600
6	1.00	1.00	1.00	120
7	1.00	1.00	1.00	480
8	0.99	0.99	0.99	450
9	0.99	1.00	1.00	480
10	1.00	1.00	1.00	660
11	1.00	1.00	1.00	420
12	0.98	0.99	0.98	690
13	1.00	1.00	1.00	720
14	1.00	0.98	0.99	240
15	0.98	1.00	0.99	210
16	0.93	1.00	0.96	120
17	1.00	1.00	1.00	360
18	0.99	0.94	0.97	390
19	0.97	1.00	0.98	60
20	0.83	1.00	0.91	120
21	0.99	0.76	0.86	90
22	0.98	1.00	0.99	120
23	0.98	0.87	0.92	150
24	0.96	0.73	0.83	90
25	1.00	1.00	1.00	480
26	0.82	0.98	0.89	180
27	0.86	1.00	0.92	60
28	0.95	0.99	0.97	180
29	0.92	0.97	0.94	90
30	0.99	0.97	0.98	150
31	0.95	0.99	0.97	240
32	1.00	1.00	1.00	60
33	0.91	1.00	0.95	209
34	1.00	0.99	1.00	120
35	1.00	0.93	0.96	390
36	1.00	1.00	1.00	120
37	0.87	1.00	0.93	60
38	1.00	0.90	0.95	690
39	1.00	0.93	0.97	90
40	0.71	0.85	0.77	120
41	0.98	1.00	0.99	60
42	1.00	1.00	1.00	60
accuracy			0.98	12569
macro avg	0.96	0.97	0.96	12569
weighted avg	0.98	0.98	0.98	12569

	precision	recall	f1-score	support
0	1.00	0.98	0.99	60
1	0.99	0.99	0.99	720
2	0.98	0.98	0.98	750
3	0.96	0.99	0.98	450
4	1.00	0.99	1.00	660
5	0.98	0.96	0.97	600
6	1.00	1.00	1.00	120
7	0.99	1.00	0.99	480
8	0.99	1.00	0.99	450
9	1.00	0.98	0.99	480
10	1.00	1.00	1.00	660
11	0.99	0.99	0.99	420
12	1.00	1.00	1.00	690
13	1.00	1.00	1.00	720
14	1.00	1.00	1.00	240
15	1.00	0.99	1.00	210
16	1.00	1.00	1.00	120
17	1.00	0.99	1.00	360
18	1.00	0.97	0.98	390
19	0.94	1.00	0.97	60
20	0.92	0.99	0.95	120
21	0.97	0.72	0.83	90
22	0.96	1.00	0.98	120
23	0.95	0.97	0.96	150
24	1.00	0.93	0.97	90
25	1.00	0.99	0.99	480
26	0.92	0.98	0.95	180
27	0.92	1.00	0.96	60
28	0.91	0.99	0.95	180
29	0.95	0.96	0.95	90
30	0.95	0.95	0.95	150
31	0.94	0.99	0.97	240
32	0.97	1.00	0.98	60
33	0.94	1.00	0.97	209
34	1.00	0.97	0.99	120
35	1.00	0.92	0.96	390
36	1.00	1.00	1.00	120
37	0.97	1.00	0.98	60
38	1.00	0.90	0.95	690
39	0.91	0.94	0.93	90
40	0.71	0.98	0.82	120
41	0.92	1.00	0.96	60
42	0.98	1.00	0.99	60
accuracy			0.98	12569
macro avg	0.97	0.98	0.97	12569
weighted avg	0.98	0.98	0.98	12569

Figura 4-33: Test con optimizador RMSprop (izquierda) y Adam (derecha), 50 épocas, y Dropout 3

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40	41	42
0	56	0	0	0	0	0	0	0	4	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	717	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0
2	0	8	731	0	1	10	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
3	0	0	0	448	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
4	0	2	0	0	658	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
5	0	0	0	5	1	591	0	2	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
6	0	0	0	0	0	0	120	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
7	0	0	0	0	0	0	0	479	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
8	0	0	0	0	1	2	0	0	447	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
9	0	0	0	0	0	0	0	0	0	480	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
10	0	0	0	0	0	0	0	0	0	2	658	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
11	0	0	0	0	0	0	0	0	0	0	0	420	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
12	0	0	0	0	0	0	0	0	0	0	0	0	681	0	0	0	8	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0
13	0	0	0	0	0	0	0	0	0	0	0	0	1	719	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
14	0	0	0	0	0	0	0	0	0	0	0	0	0	0	235	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	4	0	0	0	0	0	0	0	0	0
15	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	210	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
16	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	120	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
17	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	360	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
18	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	366	0	0	0	0	0	0	0	23	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
19	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	60	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
20	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	120	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
21	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	68	0	2	3	0	0	0	0	5	0	10	0	0	0	0	0	0	0	0	0	0	0
22	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	120	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
23	0	0	0	0	0	0	0	0	0	0	2	0	0	0	0	0	0	0	0	0	17	0	0	130	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0
24	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	66	0	13	10	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
25	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	478	2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
26	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	2	0	0	0	1	0	0	0	176	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0
27	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	60	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
28	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	179	0	0	0	0	0	0	0	0	0	0	0	0	0	0
29	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	87	0	2	0	0	0	0	0	0	0	0	0	0	0
30	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	5	0	0	0	0	0	0	0	0	0	145	0	0	0	0	0	0	0	0	0	0	0	0
31	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0	0	0	0	0	0	0	0	238	0	0	0	0	0	0	0	0	0	0	0
32	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	60	0	0	0	0	0	0	0	0	0	0
33	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	209	0	0	0	0	0	0	0	0	0
34	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	119	0	0	0	0	0	1	0	0
35	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	16	0	363	0	8	0	0	1	0	0
36	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	120	0	0	0	0	0	0
37	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	60	0	0	0	0	0
38	13	1	4	1	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	7	1	1	0	0	0	0	0	0	1	620	0	39	0	0
39	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	3	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0	0	1	0	0	0	0	0	84	0	0	0
40	0	1	0	0	0	0	0	0	0	0	0	0	13	0	0	0	1	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	102	1	0
41	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	60	0
42	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	60

Figura 4-34: Matriz de confusión RMSprop 50 epochs Dropout 3

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40	41	42
0	59	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	716	3	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
2	0	1	738	2	0	8	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
3	0	0	0	447	0	2	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
4	0	1	1	0	656	0	0	0	2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
5	0	1	7	11	0	576	0	4	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
6	0	0	0	0	0	0	120	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
7	0	0	0	0	0	0	0	480	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
8	0	0	0	0	0	2	0	0	448	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
9	0	0	1	0	0	0	0	0	0	472	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	5	0
10	0	0	0	0	0	0	0	0	0	1	659	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
11	0	0	0	0	0	0	0	0	0	0	0	417	0	0	0	0	0	0	0	0	0	0	0	0	0	0	2	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0
12	0	0	0	0	0	0	0	0	0	0	0	0	690	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
13	0	0	0	0	0	0	0	0	0	0	0	0	0	720	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
14	0	0	0	0	0	0	0	0	0	0	0	0	0	0	240	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
15	0	0	2	0	0	0	0	0	0	0	0	0	0	0	0	208	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
16	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	120	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
17	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	358	0	0	0	2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
18	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	377	0	0	0	1	0	0	0	12	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
19	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	60	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
20	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	119	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
21	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	3	0	65	1	3	0	0	1	0	0	0	3	13	0	0	0	0	0	0	0	0	0	0	0
22	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	120	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
23	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	4	0	0	146	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
24	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	84	0	0	5	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
25	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	2	0	0	0	0	477	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0
26	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	3	0	0	0	176	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0
27	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	60	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
28	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	178	0	2	0	0	0	0	0	0	0	0	0	0	0	0
29	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	3	0	1	0	0	0	86	0	0	0	0	0	0	0	0	0	0	0	0	0
30	0	0	0	0	0	0	0	0	0	0	0	2	0	0	0	0	0	0	0	0	4	0	0	0	0	0	0	0	0	0	143	1	0	0	0	0	0	0	0	0	0	0	0
31	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	1	0	238	0	0	0	0	0	0	0	0	0	0	0
32	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	60	0	0	0	0	0	0	0	0	0	0
33	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	209	0	0	0	0	0	0	0	0	0
34	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	117	0	0	0	0	0	3	0	0
35	0	1	0	4	0	0	0	0	0	0	0	0	2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	6	0	0	0	1	14	0	358	0	1	0	0	3	0	0
36	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	120	0	0	0	0	0	0
37	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	60	0	0	0	0	0
38	0	2	1	0	0	0	0	1	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	10	3	2	0	0	0	0	0	0	0	619	8	43	0	0
39	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	2	0	0	0	0	0	0	0	0	0	0	85	0	0	0
40	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	118	0	1
41	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	60	0
42	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	60

Figura 4-35: Matriz de confusión Adam 50 epochs Dropout 3

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40	41	42
0	57	0	0	0	0	0	0	0	3	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	716	1	0	0	3	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
2	0	1	729	0	1	18	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
3	0	0	0	444	0	6	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
4	0	0	1	0	657	2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
5	0	0	0	0	0	599	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
6	0	0	0	0	0	0	120	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
7	0	0	0	0	0	2	0	478	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
8	0	0	0	0	0	2	0	3	445	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
9	0	0	0	0	0	0	0	0	0	479	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0
10	0	0	0	0	0	0	0	0	0	6	654	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
11	0	0	0	0	0	0	0	0	0	0	0	418	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	2	0	0	0	0	0	0	0	0	0	0	0	0
12	0	0	0	0	0	0	0	0	0	0	0	2	671	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	17	0
13	0	0	0	0	0	0	0	0	0	0	0	0	0	720	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
14	0	0	0	0	0	0	0	0	0	0	0	0	0	0	238	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	1	0	0
15	0	0	2	0	0	0	0	0	0	1	0	0	0	0	0	207	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
16	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	120	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
17	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	360	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
18	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	388	0	0	0	1	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
19	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	60	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
20	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	119	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
21	0	0	5	0	0	0	0	0	0	0	0	2	0	0	0	0	0	0	0	1	1	69	0	2	0	0	1	0	0	0	8	1	0	0	0	0	0	0	0	0	0	0	0
22	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	120	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
23	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	145	0	0	0	0	0	1	3	0	0	0	0	0	0	0	0	0	0	0	0
24	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	87	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
25	0	0	0	0	0	2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	477	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0
26	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	3	0	0	0	0	0	0	0	174	1	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0
27	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	60	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
28	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	174	2	4	0	0	0	0	0	0	0	0	0	0	0	0
29	0	0	2	0	0	3	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	1	0	0	0	81	1	0	0	0	0	0	0	0	1	0	0	0	0
30	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	150	0	0	0	0	0	0	0	0	0	0	0	0
31	0	0	2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	237	0	0	0	0	0	0	0	0	0	0	0
32	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	60	0	0	0	0	0	0	0	0	0	0
33	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	209	0	0	0	0	0	0	0	0	0
34	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	120	0	0	0	0	0	0	0	0
35	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	390	0	0	0	0	0	0	0
36	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	120	0	0	0	0	0	0
37	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	60	0	0	0	0	0
38	1	2	12	0	3	5	0	0	0	2	0	1	0	0	0	0	0	0	0	0	0	0	0	10	0	0	0	0	0	0	1	0	0	0	0	0	0	0	627	0	26	0	0
39	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	5	0	0	0	0	0	84	0	0	0
40	0	0	4	0	1	11	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	103	0	0
41	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	60	0
42	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	59

Figura 4-36: Matriz de confusión RMSprop 200 epochs Dropout 3

	precision	recall	f1-score	support
0	0.98	0.95	0.97	60
1	1.00	0.99	1.00	720
2	0.96	0.97	0.97	750
3	1.00	0.99	0.99	450
4	0.99	1.00	0.99	660
5	0.92	1.00	0.96	600
6	1.00	1.00	1.00	120
7	0.99	1.00	0.99	480
8	0.99	0.99	0.99	450
9	0.98	1.00	0.99	480
10	1.00	0.99	0.99	660
11	0.99	1.00	0.99	420
12	1.00	0.97	0.99	690
13	1.00	1.00	1.00	720
14	1.00	0.99	1.00	240
15	1.00	0.99	0.99	210
16	1.00	1.00	1.00	120
17	1.00	1.00	1.00	360
18	0.99	0.99	0.99	390
19	0.98	1.00	0.99	60
20	0.99	0.99	0.99	120
21	1.00	0.77	0.87	90
22	0.99	1.00	1.00	120
23	0.91	0.97	0.94	150
24	1.00	0.97	0.98	90
25	1.00	0.99	0.99	480
26	0.98	0.97	0.97	180
27	0.97	1.00	0.98	60
28	1.00	0.97	0.98	180
29	0.96	0.90	0.93	90
30	0.89	1.00	0.94	150
31	1.00	0.99	0.99	240
32	1.00	1.00	1.00	60
33	0.97	1.00	0.99	209
34	0.99	1.00	1.00	120
35	1.00	1.00	1.00	390
36	0.99	1.00	1.00	120
37	1.00	1.00	1.00	60
38	1.00	0.91	0.95	690
39	1.00	0.93	0.97	90
40	0.79	0.86	0.82	120
41	0.76	1.00	0.86	60
42	1.00	0.98	0.99	60
accuracy			0.98	12569
macro avg	0.98	0.98	0.98	12569
weighted avg	0.98	0.98	0.98	12569

	precision	recall	f1-score	support
0	1.00	0.93	0.97	60
1	0.98	0.99	0.99	720
2	0.99	1.00	0.99	750
3	0.97	1.00	0.98	450
4	1.00	1.00	1.00	660
5	0.99	0.97	0.98	600
6	1.00	1.00	1.00	120
7	0.98	1.00	0.99	480
8	0.99	0.99	0.99	450
9	0.99	1.00	1.00	480
10	1.00	1.00	1.00	660
11	1.00	1.00	1.00	420
12	1.00	1.00	1.00	690
13	1.00	1.00	1.00	720
14	1.00	1.00	1.00	240
15	1.00	1.00	1.00	210
16	1.00	1.00	1.00	120
17	1.00	1.00	1.00	360
18	0.98	0.99	0.99	390
19	0.94	1.00	0.97	60
20	0.83	1.00	0.91	120
21	1.00	0.90	0.95	90
22	0.98	1.00	0.99	120
23	0.96	0.85	0.90	150
24	1.00	0.93	0.97	90
25	0.99	0.99	0.99	480
26	0.97	0.97	0.97	180
27	0.98	0.98	0.98	60
28	0.98	0.96	0.97	180
29	0.95	0.92	0.94	90
30	0.97	0.96	0.97	150
31	0.98	1.00	0.99	240
32	0.98	1.00	0.99	60
33	0.99	1.00	0.99	209
34	1.00	1.00	1.00	120
35	1.00	0.99	1.00	390
36	0.99	1.00	1.00	120
37	0.87	1.00	0.93	60
38	1.00	0.91	0.95	690
39	1.00	0.94	0.97	90
40	0.77	0.97	0.86	120
41	1.00	0.95	0.97	60
42	1.00	1.00	1.00	60
accuracy			0.99	12569
macro avg	0.98	0.98	0.98	12569
weighted avg	0.99	0.99	0.99	12569

Figura 4-37: Test con optimizador RMSprop (izquierda) y Adam (derecha), 200 épocas, y Dropout 3

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40	41	42
0	56	0	0	0	0	0	0	0	4	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	714	4	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
2	0	0	747	0	0	3	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
3	0	0	0	449	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
4	0	0	2	0	658	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
5	0	0	1	9	1	582	0	6	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
6	0	0	0	0	0	0	120	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
7	0	0	0	0	0	0	0	480	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
8	0	0	0	0	0	2	0	4	444	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
9	0	0	0	0	0	0	0	0	0	480	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
10	0	0	0	0	0	0	0	0	0	0	660	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
11	0	0	0	0	0	0	0	0	0	0	0	418	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0
12	0	0	0	0	0	0	0	0	0	0	0	0	690	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
13	0	0	0	0	0	0	0	0	0	0	0	0	0	720	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
14	0	0	0	0	0	0	0	0	0	0	0	0	0	0	240	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
15	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	210	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
16	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	120	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
17	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	360	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
18	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	388	0	0	0	0	0	0	0	2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
19	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	60	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
20	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	120	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
21	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	81	0	2	0	0	1	0	0	1	0	4	0	0	0	0	0	0	0	0	0	0	0
22	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	120	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
23	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	1	21	0	0	127	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
24	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	3	0	0	0	0	0	84	0	2	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
25	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	476	0	0	1	0	1	0	0	0	0	0	1	0	0	0	0	0	0
26	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	2	0	0	0	2	0	0	0	175	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
27	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	59	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
28	0	1	0	3	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	2	1	0	173	0	0	0	0	0	0	0	0	0	0	0	0	0	0
29	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	2	0	0	3	0	1	0	0	0	83	1	0	0	0	0	0	0	0	0	0	0	0	0
30	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	2	0	2	0	0	0	0	0	0	0	0	0	144	2	0	0	0	0	0	0	0	0	0	0	0
31	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	239	0	0	0	0	0	0	0	0	0	0	0
32	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	60	0	0	0	0	0	0	0	0	0	0
33	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	209	0	0	0	0	0	0	0	0	0
34	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	120	0	0	0	0	0	0	0	0
35	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	388	0	0	0	0	1	0	0
36	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	120	0	0	0	0	0	0
37	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	60	0	0	0	0	0
38	0	12	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	3	1	0	0	0	0	0	0	9	629	0	34	0	0
39	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	1	3	0	0	0	0	0	85	0	0	0
40	0	0	0	1	0	0	0	0	0	0	0	0	2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	117	0	0
41	0	0	0	0	0	0	0	0	0	3	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	57	0
42	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	60

Figura 4-38: Matriz de confusión Adam 200 epochs Dropout 3

4.5 Experimentos de ajuste de hiperparámetros

En este apartado se seguirá intentando mejorar la red usando de nuevo la validación para ajustar más los hiperparámetros. Para ello se ajustarán en función de los resultados de validación algunos parámetros como el número de capas o el número de neuronas por capa respecto de la red original.

Se empezará con el modelo usando Adam con 50 épocas y un batch de 200 imágenes, tal como se hizo en el primer experimento, de manera que se incrementará el número de capas densas en 2, y se aumentará también el número de neuronas, tanto en las capas densas, como en las capas convolucionales añadiendo un mayor número de filtros por capa. La red quedaría como en la Figura 4-39:

Layer (type)	Output Shape	Param #
Conv1_Layer (Conv2D)	(None, 62, 62, 64)	640
Pooling1_Layer (MaxPooling2D	(None, 31, 31, 64)	0
Conv2_Layer (Conv2D)	(None, 30, 30, 128)	32896
Pooling2_Layer (MaxPooling2D	(None, 15, 15, 128)	0
Conv3_Layer (Conv2D)	(None, 13, 13, 256)	295168
Pooling3_Layer (MaxPooling2D	(None, 6, 6, 256)	0
Conv4_Layer (Conv2D)	(None, 4, 4, 512)	1180160
Pooling4_Layer (MaxPooling2D	(None, 2, 2, 512)	0
flatten_6 (Flatten)	(None, 2048)	0
dense_12 (Dense)	(None, 2048)	4196352
Dense_Hidden_Layer1 (Dense)	(None, 1024)	2098176
Dense_Hidden_Layer2 (Dense)	(None, 512)	524800
Dense_Hidden_Layer3 (Dense)	(None, 256)	131328
Dense_Hidden_Layer4 (Dense)	(None, 128)	32896
Dense_Hidden_Layer5 (Dense)	(None, 64)	8256
dense_13 (Dense)	(None, 43)	2795

Total params: 8,503,467
Trainable params: 8,503,467
Non-trainable params: 0

Figura 4-39: Modelo con Adam 50 epochs modificación 1

El número de parámetros de la red ha aumentado a 8,503,467, siendo aproximadamente 4 veces superior al original. Los resultados se muestran en la Figura 4-40:

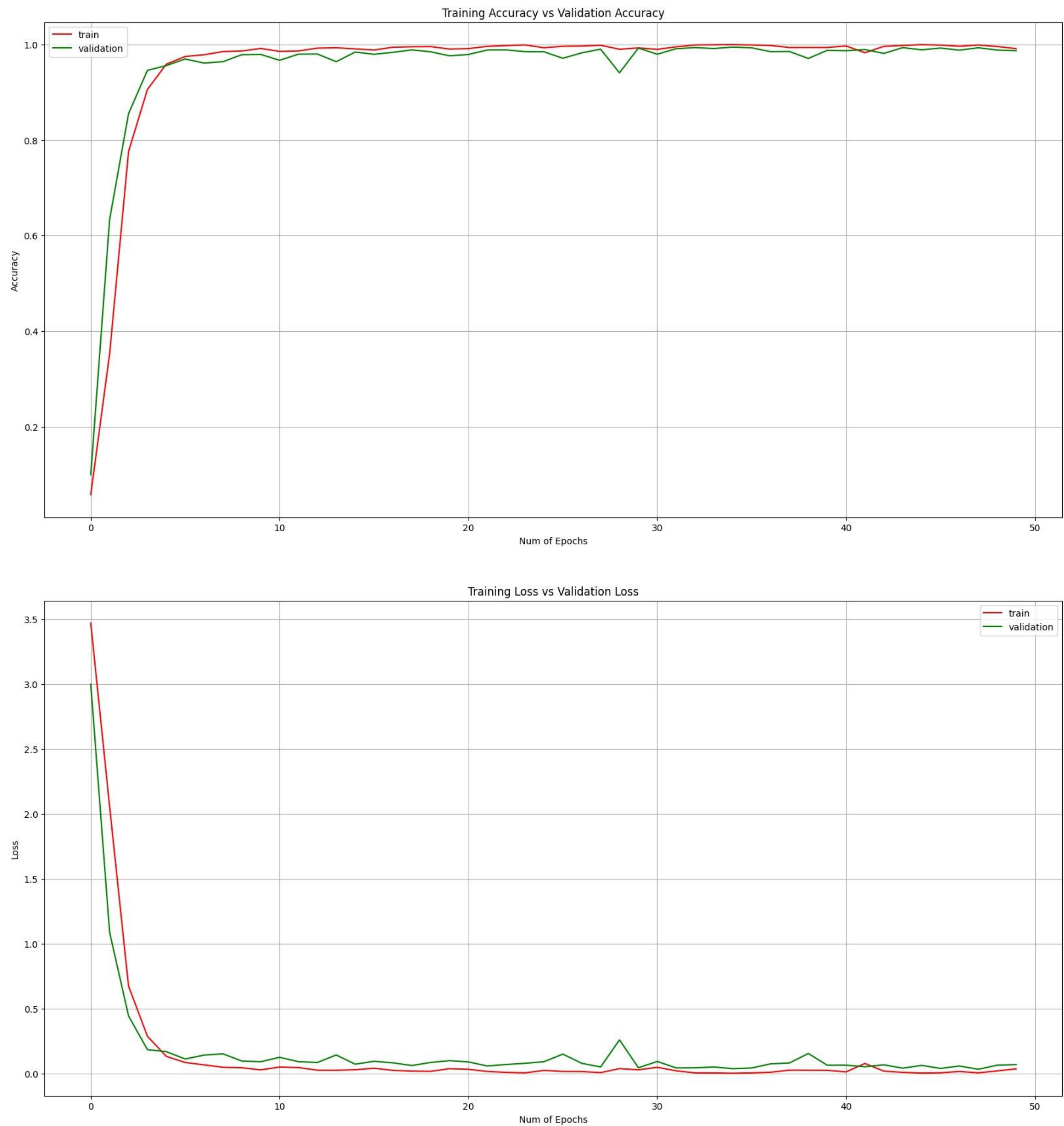


Figura 4-40: Train vs Validation Adam 50 epochs modificación 1

Los resultados obtenidos empeoran respecto al caso de los primeros experimentos, y es que al aumentar el número de parámetros de la red el overfitting se hace más acusado, obteniendo mayor variabilidad en los resultados en la validación respecto a la Figura 4-8. Los resultados del modelo obtenido son 99.12 % *accuracy* de entrenamiento, 98.69 % de validación, y un 95.84 % de test. Estos resultados indican que, al aumentar el número de parámetros de la red, el riesgo de correr overfitting aumenta considerablemente.

Para ver que efecto tiene disminuir estos hiperparámetros, se ha realizado otro experimento en el que se vuelven a disminuir las neuronas de las capas convolucionales y de las capas densas, eliminando también algunas capas densas ocultas, quedando el modelo de la Figura 4-42:

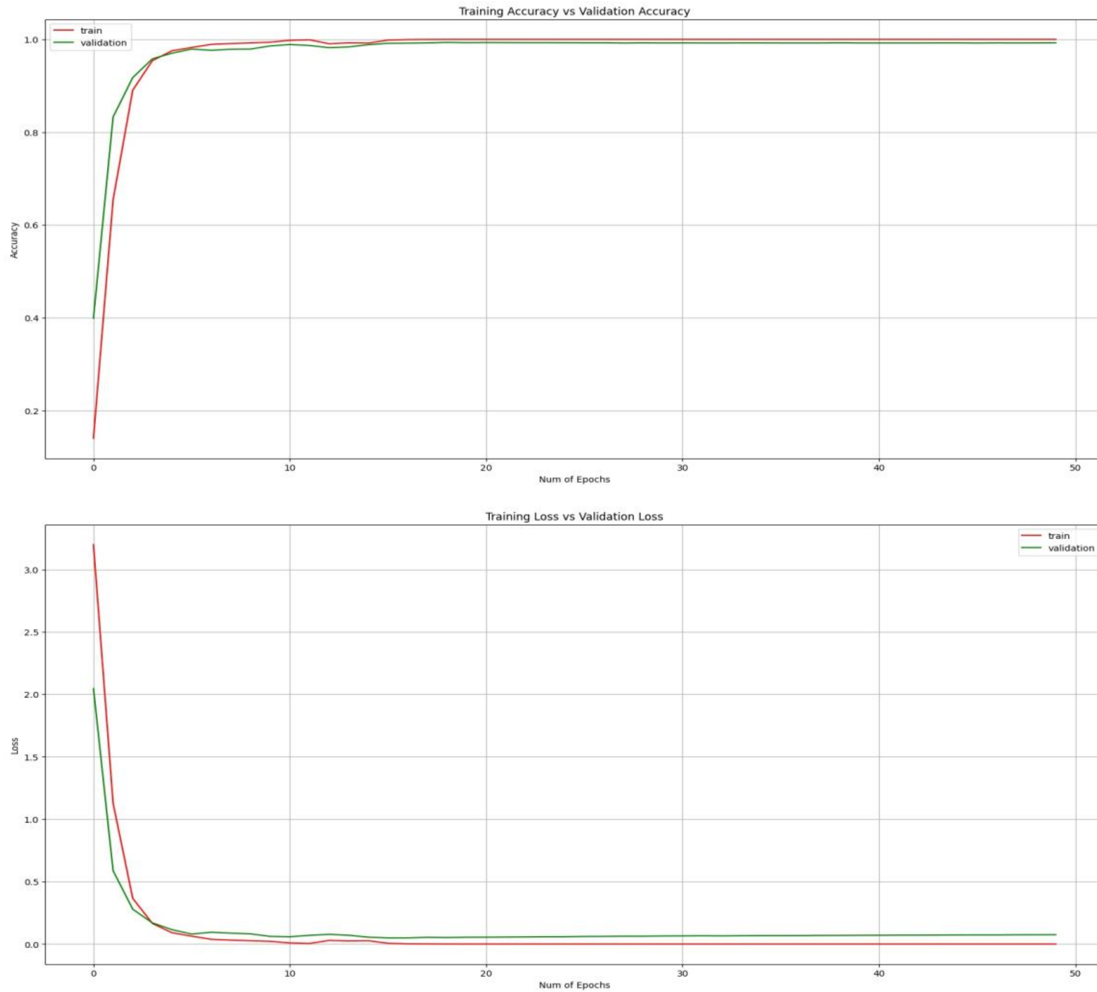


Figura 4-41: Train vs Validation Adam 50 epochs modificación 2

```
Layer (type)                 Output Shape              Param #
=================================================================
Conv1_Layer (Conv2D)         (None, 62, 62, 16)        160

Pooling1_Layer (MaxPooling2D (None, 31, 31, 16)        0

Conv2_Layer (Conv2D)         (None, 30, 30, 32)        2080

Pooling2_Layer (MaxPooling2D (None, 15, 15, 32)        0

Conv3_Layer (Conv2D)         (None, 13, 13, 64)        18496

Pooling3_Layer (MaxPooling2D (None, 6, 6, 64)          0

Conv4_Layer (Conv2D)         (None, 4, 4, 128)         73856

Pooling4_Layer (MaxPooling2D (None, 2, 2, 128)         0

flatten_1 (Flatten)          (None, 512)               0

dense_2 (Dense)              (None, 512)               262656

Dense_Hidden_Layer1 (Dense)  (None, 256)               131328

Dense_Hidden_Layer2 (Dense)  (None, 128)               32896

dense_3 (Dense)              (None, 43)                5547
=================================================================
Total params: 527,019
Trainable params: 527,019
Non-trainable params: 0
```

Figura 4-42: Modelo con Adam 50 epochs modificación 2

Ahora los parámetros han disminuido a 527,019, que sería aproximadamente 4 veces inferior al número de parámetros originales, y los resultados se aprecian en la Figura 4-41. Respecto al experimento anterior, se puede apreciar una disminución de la variabilidad de los resultados en la validación, pero aún así el overfitting no ha desaparecido. Ahora el valor de *accuracy* del modelo final obtenido es del 100 % en entrenamiento, 99.27 % en validación, y 95.16 % en el test.

Para disminuirlo habría que seguir disminuyendo el número de neuronas y capas de la red, pero al tener menor número de parámetros se perderá poder predictivo, cosa que no interesa. Por eso, se realizarán experimentos similares, pero usando el Dropout anterior en todas las capas excepto entre la última capa oculta y la capa de salida de la etapa densamente conectada y teniendo en todos los casos siempre una probabilidad de activación del 75 % (25 % en el código Keras, como ya se explicó).

Si se parte del experimento ya realizado para el optimizador Adam con Dropout 3, se observaba en los resultados que aparecía underfitting, por lo que según indica la validación, habría que justar los parámetros del modelo para disminuirla, y para ello habría que aumentar el número de parámetros. Si se prueba con la primera modificación que se hizo antes añadiendo el Dropout, el modelo queda como en la Figura 4-43. Los resultados se muestran en la Figura 4-44.

Layer (type)	Output Shape	Param #
Conv1_Layer (Conv2D)	(None, 62, 62, 64)	640
Pooling1_Layer (MaxPooling2D	(None, 31, 31, 64)	0
Dropout1_CNN_Layer (Dropout)	(None, 31, 31, 64)	0
Conv2_Layer (Conv2D)	(None, 30, 30, 128)	32896
Pooling2_Layer (MaxPooling2D	(None, 15, 15, 128)	0
Dropout2_CNN_Layer (Dropout)	(None, 15, 15, 128)	0
Conv3_Layer (Conv2D)	(None, 13, 13, 256)	295168
Pooling3_Layer (MaxPooling2D	(None, 6, 6, 256)	0
Dropout3_CNN_Layer (Dropout)	(None, 6, 6, 256)	0
Conv4_Layer (Conv2D)	(None, 4, 4, 512)	1180160
Pooling4_Layer (MaxPooling2D	(None, 2, 2, 512)	0
Dropout4_CNN_Layer (Dropout)	(None, 2, 2, 512)	0
flatten (Flatten)	(None, 2048)	0
dense (Dense)	(None, 2048)	4196352
Dense_Hidden_Layer1 (Dense)	(None, 4096)	8392704
Dropout1_Dense_Layer (Dropou	(None, 4096)	0
Dense_Hidden_Layer2 (Dense)	(None, 2048)	8390656
Dropout2_Dense_Layer (Dropou	(None, 2048)	0
Dense_Hidden_Layer3 (Dense)	(None, 1024)	2098176
Dropout3_Dense_Layer (Dropou	(None, 1024)	0
Dense_Hidden_Layer4 (Dense)	(None, 512)	524800
Dropout4_Dense_Layer (Dropou	(None, 512)	0
Dense_Hidden_Layer5 (Dense)	(None, 256)	131328
dense_1 (Dense)	(None, 43)	11051

Total params: 25,253,931
Trainable params: 25,253,931

Figura 4-43: Modelo Dropout con Adam 50 epochs modificación 3

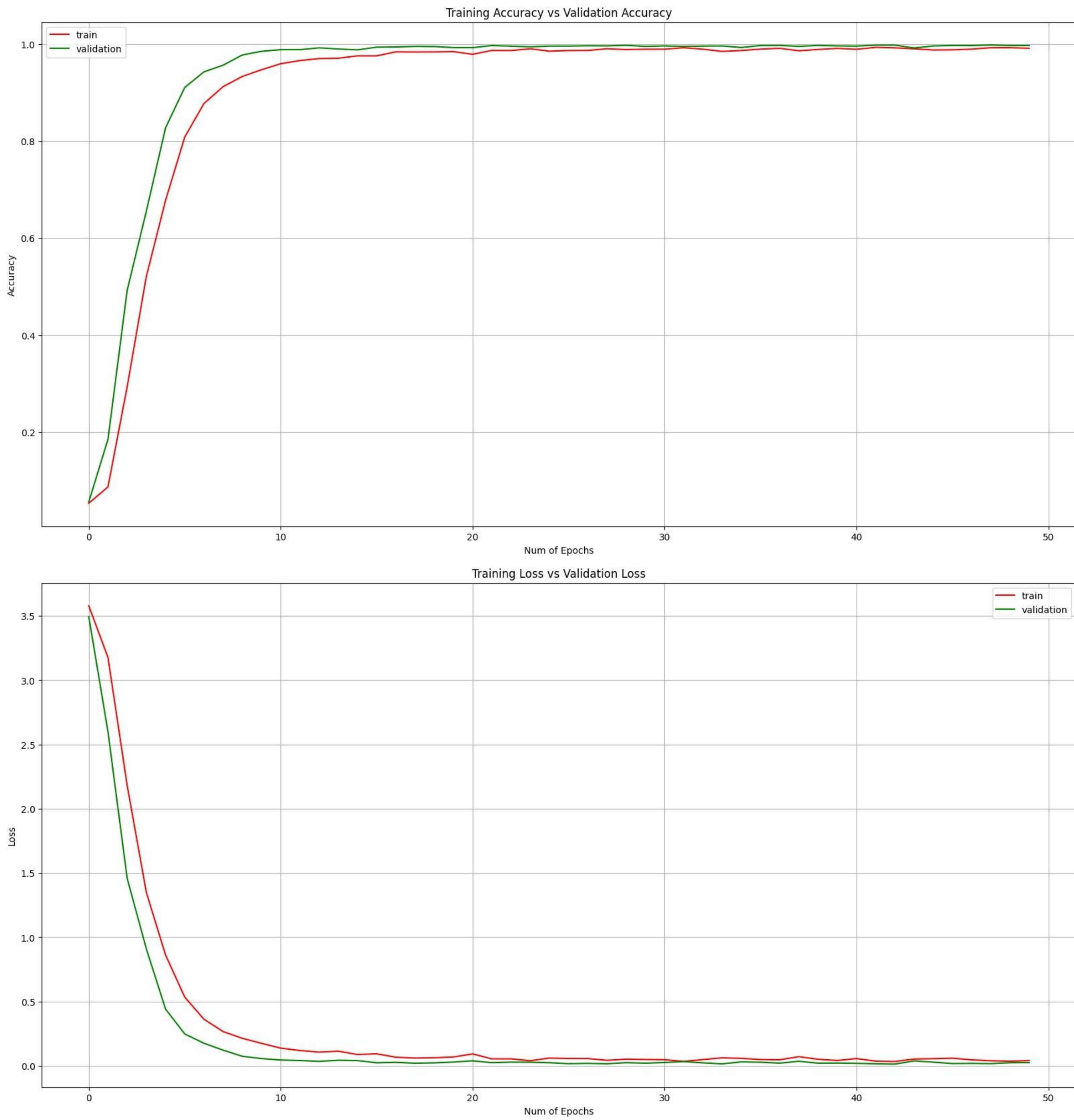


Figura 4-44: Train vs Validation Adam 50 epochs Dropout modificación 3

En primer lugar, hay que destacar el enorme aumento del número de parámetros respecto a los demás casos y es que ahora adquiere la cantidad de 25,253,931 parámetros. Esto hace que respecto a la Figura 4-32, la validación y el entrenamiento estén en apariencia más cercanos entre sí, lo que disminuiría el underfitting. Sin embargo, el resultado final es bastante parecido al que se obtuvo, con un *accuracy* de test del 97.30 %. Si se le bajase más el Dropout se daría el caso de los 2 experimentos anteriores, donde aparecería overfitting, y si se siguen aumentando los parámetros, también aparecerá tarde o temprano, empeorando el funcionamiento del sistema. Los resultados de test de estos experimentos se muestran con mayor detalle a continuación:

	precision	recall	f1-score	support
0	1.00	0.47	0.64	60
1	0.99	0.98	0.98	720
2	0.96	0.99	0.98	750
3	0.95	0.98	0.97	450
4	0.94	1.00	0.97	660
5	0.96	0.94	0.95	600
6	0.98	1.00	0.99	120
7	0.99	0.96	0.98	480
8	0.97	0.98	0.98	450
9	0.95	1.00	0.97	480
10	0.99	1.00	1.00	660
11	0.96	0.98	0.97	420
12	0.94	0.96	0.95	690
13	1.00	1.00	1.00	720
14	0.98	0.97	0.98	240
15	0.98	1.00	0.99	210
16	1.00	0.91	0.95	120
17	0.99	1.00	1.00	360
18	0.97	0.98	0.97	390
19	0.98	1.00	0.99	60
20	0.83	0.98	0.90	120
21	0.90	0.68	0.77	90
22	0.95	1.00	0.98	120
23	0.97	0.75	0.85	150
24	0.92	0.84	0.88	90
25	0.97	0.97	0.97	480
26	0.87	0.97	0.92	180
27	0.95	0.95	0.95	60
28	0.96	0.86	0.91	180
29	0.80	0.74	0.77	90
30	0.72	0.99	0.83	150
31	1.00	0.92	0.95	240
32	0.98	1.00	0.99	60
33	0.88	0.97	0.92	209
34	0.97	0.93	0.95	120
35	0.98	0.98	0.98	390
36	0.94	0.99	0.97	120
37	0.92	1.00	0.96	60
38	1.00	0.89	0.94	690
39	0.92	0.74	0.82	90
40	0.75	0.77	0.76	120
41	1.00	0.98	0.99	60
42	1.00	1.00	1.00	60
accuracy			0.96	12569
macro avg	0.95	0.93	0.93	12569
weighted avg	0.96	0.96	0.96	12569

	precision	recall	f1-score	support
0	0.96	0.75	0.84	60
1	0.96	0.97	0.97	720
2	0.95	0.96	0.96	750
3	0.90	0.94	0.92	450
4	0.95	0.97	0.96	660
5	0.89	0.91	0.90	600
6	0.91	0.97	0.94	120
7	0.96	0.95	0.95	480
8	0.95	0.93	0.94	450
9	0.94	0.97	0.95	480
10	0.98	1.00	0.99	660
11	0.97	0.92	0.94	420
12	0.94	1.00	0.97	690
13	0.99	0.97	0.98	720
14	0.98	0.93	0.95	240
15	0.96	0.97	0.96	210
16	0.99	0.72	0.83	120
17	1.00	0.98	0.99	360
18	0.91	0.85	0.88	390
19	0.51	0.93	0.66	60
20	0.74	0.99	0.85	120
21	0.67	0.18	0.28	90
22	0.97	1.00	0.98	120
23	0.88	0.61	0.72	150
24	0.98	0.89	0.93	90
25	0.96	0.99	0.97	480
26	0.75	0.91	0.82	180
27	0.85	0.92	0.88	60
28	0.85	0.92	0.88	180
29	0.90	0.78	0.83	90
30	0.79	0.89	0.84	150
31	0.89	0.87	0.88	240
32	0.68	1.00	0.81	60
33	0.88	0.86	0.87	209
34	0.98	1.00	0.99	120
35	0.96	0.94	0.95	390
36	0.97	1.00	0.98	120
37	0.75	1.00	0.86	60
38	0.99	0.87	0.93	690
39	0.93	0.71	0.81	90
40	0.61	0.86	0.71	120
41	0.94	0.83	0.88	60
42	1.00	0.77	0.87	60
accuracy			0.93	12569
macro avg	0.90	0.89	0.88	12569
weighted avg	0.93	0.93	0.93	12569

	precision	recall	f1-score	support
0	1.00	0.98	0.99	60
1	0.98	1.00	0.99	720
2	0.99	0.99	0.99	750
3	0.97	1.00	0.98	450
4	1.00	1.00	1.00	660
5	0.99	0.96	0.98	600
6	1.00	0.99	1.00	120
7	1.00	0.97	0.99	480
8	0.95	1.00	0.97	450
9	1.00	1.00	1.00	480
10	0.99	1.00	1.00	660
11	0.98	0.99	0.98	420
12	1.00	1.00	1.00	690
13	1.00	1.00	1.00	720
14	0.99	0.99	0.99	240
15	0.98	1.00	0.99	210
16	1.00	1.00	1.00	120
17	1.00	0.99	1.00	360
18	0.99	0.91	0.95	390
19	0.94	1.00	0.97	60
20	0.84	1.00	0.91	120
21	1.00	0.69	0.82	90
22	0.98	1.00	0.99	120
23	1.00	0.81	0.89	150
24	1.00	0.92	0.96	90
25	0.97	1.00	0.98	480
26	0.84	0.98	0.90	180
27	0.94	1.00	0.97	60
28	0.99	0.89	0.94	180
29	0.89	0.82	0.86	90
30	0.78	0.94	0.85	150
31	0.96	1.00	0.98	240
32	0.88	1.00	0.94	60
33	0.86	1.00	0.92	209
34	0.99	0.97	0.98	120
35	1.00	0.93	0.96	390
36	0.98	1.00	0.99	120
37	0.87	1.00	0.93	60
38	1.00	0.89	0.94	690
39	1.00	0.93	0.97	90
40	0.70	0.93	0.80	120
41	1.00	0.98	0.99	60
42	0.98	1.00	0.99	60
accuracy			0.97	12569
macro avg	0.96	0.96	0.96	12569
weighted avg	0.98	0.97	0.97	12569

Figura 4-45: Resultados de test de la modificación de red 1, 2 y 3 de izquierda a derecha

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40	41	42
0	28	3	0	0	26	0	0	0	3	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	704	6	0	7	2	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
2	0	2	742	0	0	5	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
3	0	0	1	442	0	3	0	0	0	1	0	0	0	0	0	2	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
4	0	0	1	0	658	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
5	0	0	15	17	3	561	0	2	0	0	1	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
6	0	0	0	0	0	0	120	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
7	0	0	1	0	0	9	0	461	8	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
8	0	0	0	0	2	5	0	1	440	0	0	0	0	0	0	2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
9	0	0	0	1	0	0	0	0	0	478	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0
10	0	0	0	0	0	0	0	0	0	1	659	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
11	0	0	0	0	0	0	0	0	0	0	0	410	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	9	0	0	0	0	0	0	0	0	0	0	0	0
12	0	0	0	0	0	0	1	0	0	9	0	0	663	0	0	0	0	2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	14	0	0
13	0	0	0	0	0	0	0	0	0	0	0	0	0	720	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
14	0	0	3	0	1	0	0	0	0	0	0	0	0	0	234	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	2	0	0	0	0	0	0	0	0	0
15	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	209	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
16	0	0	0	0	0	0	0	0	0	11	0	0	0	0	0	0	109	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
17	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	360	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
18	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	381	0	0	0	1	0	0	0	7	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
19	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	60	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
20	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	118	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0
21	0	0	0	0	0	0	0	0	1	0	0	6	0	0	0	0	0	0	1	1	0	61	3	0	1	1	4	0	3	3	3	1	0	0	0	0	0	0	0	1	0	0	0
22	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	120	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
23	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	18	0	0	113	0	0	0	0	0	1	17	0	0	0	0	0	0	0	0	0	0	0	0
24	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	76	0	13	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
25	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	9	0	1	0	0	0	0	468	1	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0
26	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	3	0	0	0	1	0	0	0	174	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0
27	0	0	0	0	0	0	0	0	0	0	0	2	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	57	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
28	0	0	0	0	0	0	0	0	0	0	0	2	0	0	0	0	0	0	0	0	0	0	1	0	2	0	0	0	155	3	17	0	0	0	0	0	0	0	0	0	0	0	0
29	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	2	8	1	1	1	67	10	0	0	0	0	0	0	0	0	0	0	0	0
30	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	149	0	0	0	0	0	0	0	0	0	0	0	0
31	0	0	0	0	0	0	0	0	0	0	0	7	0	0	0	0	0	0	0	0	0	7	0	2	0	0	0	0	1	2	1	220	0	0	0	0	0	0	0	0	0	0	0
32	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	60	0	0	0	0	0	0	0	0	0	0
33	0	3	0	2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	202	0	0	1	0	0	0	1	0	0
34	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	112	4	3	1	0	0	0	0	0
35	0	0	0	3	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	2	383	0	0	0	0	0	0	0
36	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	119	0	0	0	0	0	0
37	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	60	0	0	0	0	0
38	0	1	1	1	1	0	0	0	0	0	0	0	22	1	1	0	0	0	0	0	5	0	0	1	1	2	0	0	0	7	0	0	0	4	2	0	1	4	615	4	16	0	0
39	0	0	0	0	2	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	19	0	0	0	0	0	67	0	0	0
40	0	0	0	0	0	0	0	0	0	0	2	0	21	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	1	0	1	1	92	0	0
41	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	59	0
42	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	60

Figura 4-46: Matriz de confusión de modificación 1

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40	41	42
0	**45**	6	0	1	6	0	0	0	2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	2	**700**	11	0	1	0	0	0	1	1	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	2	0	0
2	0	12	**720**	4	1	9	0	1	2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0
3	0	1	4	**421**	0	18	0	1	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	2	0	0	0	0	0	0	0	0	0	1	0	0
4	0	2	3	1	**638**	1	0	0	9	0	0	0	0	0	0	2	0	0	0	4	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
5	0	2	14	30	0	**547**	0	5	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
6	0	0	0	0	0	1	**117**	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	1	0	0
7	0	0	2	0	0	22	0	**455**	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
8	0	0	0	11	2	9	0	7	**419**	2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
9	0	0	0	0	0	1	0	0	0	**464**	6	0	0	3	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	3	0	0	0	0	0	0	0
10	0	0	0	0	0	2	0	0	0	0	**658**	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
11	0	0	0	0	0	0	0	0	0	0	0	**385**	0	0	0	0	0	0	4	0	4	0	0	2	0	3	4	1	0	0	16	1	0	0	0	0	0	0	0	0	0	0	0
12	0	0	0	0	0	0	0	0	0	0	0	0	**688**	2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
13	0	1	0	0	0	0	0	0	0	0	0	0	12	**698**	0	4	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	5	0	0	0	0
14	0	0	0	1	0	0	0	0	0	0	0	0	0	0	**222**	0	0	1	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	9	0	0	0	0	0	4	2	0	0
15	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	**204**	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	5	0	0	1	0	0	0	0	0	0	0
16	0	0	0	0	0	0	0	4	3	27	0	0	0	0	0	0	**86**	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
17	0	0	0	0	0	0	0	0	0	0	0	0	3	0	1	1	0	**354**	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0
18	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	**332**	10	0	0	0	0	0	3	36	6	1	0	0	0	1	0	0	0	0	0	0	0	0	0	0
19	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	**56**	0	3	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
20	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	**119**	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
21	0	0	0	0	0	0	0	0	0	0	0	6	0	0	0	0	0	0	6	20	0	**16**	1	1	0	0	5	0	0	1	0	25	1	0	0	0	0	1	0	0	6	1	0
22	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	**120**	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
23	0	0	0	0	0	0	1	0	0	0	1	0	0	0	0	0	0	0	0	0	19	0	0	**91**	0	0	0	0	20	2	15	1	0	0	0	0	0	0	0	0	0	0	0
24	0	0	0	0	0	0	0	0	1	0	0	3	0	0	0	0	0	0	0	1	0	0	0	0	**80**	0	2	1	2	0	0	0	0	0	0	0	0	0	0	0	0	0	0
25	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	1	0	2	1	1	**473**	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0
26	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	13	0	0	0	0	0	0	0	**164**	1	0	0	0	0	0	0	2	0	0	0	0	0	0	0	0
27	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	2	0	0	0	0	0	0	0	3	**55**	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
28	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	2	0	0	1	1	7	0	1	**165**	2	0	0	0	0	0	0	0	0	0	0	0	0	0
29	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	13	0	1	2	0	3	0	0	0	**70**	1	0	0	0	0	0	0	0	0	0	0	0	0
30	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	5	0	1	0	0	2	0	1	0	0	6	2	**133**	0	0	0	0	0	0	0	0	0	0	0	0
31	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	1	18	0	5	0	2	0	2	2	0	0	0	0	**209**	0	0	0	0	0	0	0	0	0	0	0
32	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	**60**	0	0	0	0	0	0	0	0	0	0
33	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	**180**	0	0	0	0	0	0	28	0	0
34	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	**120**	0	0	0	0	0	0	0	0
35	0	0	0	1	0	0	0	0	0	0	1	1	3	1	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	10	0	**367**	4	0	0	0	1	0	0
36	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	**120**	0	0	0	0	0	0
37	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	**60**	0	0	0	0	0
38	0	2	0	0	0	3	0	0	0	0	0	0	9	1	4	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	12	4	0	11	0	19	**599**	1	24	0	0
39	0	1	0	0	22	0	0	0	1	0	0	0	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	**64**	0	0	0
40	0	0	0	0	0	0	0	0	0	0	0	0	17	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	**103**	0	0
41	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	9	0	0	0	0	0	0	0	0	**50**	0
42	0	0	0	0	0	1	11	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	2	**46**

Figura 4-47: Matriz de confusión de modificación 2

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40	41	42
0	**59**	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	**717**	1	0	1	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
2	0	2	**739**	2	0	4	0	0	3	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
3	0	0	0	**449**	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
4	0	0	1	0	**657**	0	0	0	1	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
5	0	4	2	11	1	**578**	0	0	3	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
6	0	0	0	0	0	0	**119**	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0
7	0	0	0	0	0	0	0	**467**	13	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
8	0	0	1	0	0	1	0	0	**448**	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
9	0	0	0	0	0	0	0	0	0	**479**	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
10	0	0	0	0	0	0	0	0	0	1	**659**	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
11	0	0	0	0	0	0	0	0	0	0	0	**415**	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	5	0	0	0	0	0	0	0	0	0	0	0	0
12	0	0	0	0	0	0	0	0	0	0	0	0	**689**	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
13	0	0	0	0	0	0	0	0	0	0	0	0	0	**720**	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
14	0	0	0	0	0	0	0	0	0	0	0	0	0	0	**238**	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	2	0	0	0	0	0	0	0	0	0
15	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	**209**	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0
16	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	**120**	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
17	0	0	0	0	0	0	0	0	0	0	0	0	0	0	3	0	0	**357**	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
18	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	**356**	0	0	0	0	0	0	0	33	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
19	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	**60**	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
20	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	**120**	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
21	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	4	0	**62**	0	0	0	7	1	0	0	7	2	7	0	0	0	0	0	0	0	0	0	0	0
22	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	**120**	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
23	0	0	0	0	0	0	0	0	0	0	1	3	0	0	0	0	0	0	0	0	3	0	1	**121**	0	0	0	0	0	1	19	0	0	0	0	0	0	0	0	0	0	0	1
24	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	1	0	0	0	0	0	**83**	0	0	3	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0
25	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	**479**	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0
26	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	2	0	0	0	**176**	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0
27	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	**60**	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
28	0	0	0	0	0	0	0	0	0	0	0	4	0	0	0	0	0	0	0	0	5	0	0	0	0	0	0	0	**161**	0	10	0	0	0	0	0	0	0	0	0	0	0	0
29	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	7	0	0	0	0	6	0	0	0	**74**	1	2	0	0	0	0	0	0	0	0	0	0	0
30	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	6	0	0	0	0	0	0	0	0	1	**141**	1	0	0	0	0	0	0	0	0	0	0	0
31	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	**239**	0	0	0	0	0	0	0	0	0	0	0
32	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	**60**	0	0	0	0	0	0	0	0	0	0
33	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	**209**	0	0	0	0	0	0	0	0	0
34	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	**117**	0	0	0	0	0	3	0	0
35	0	0	0	0	0	0	0	0	0	0	2	0	1	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	24	0	**361**	1	0	0	0	0	0	0
36	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	**120**	0	0	0	0	0	0
37	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	**60**	0	0	0	0	0
38	0	11	0	2	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	1	0	7	0	0	0	0	9	**616**	0	42	0	0
39	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	5	0	0	0	0	0	**84**	0	0	0
40	0	0	0	0	0	0	0	0	0	1	1	0	1	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	2	1	0	1	0	1	0	**111**	0	0
41	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	**59**	0
42	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	**60**

Figura 4-48: Matriz de confusión de modificación 3

4.6 Discusión de los resultados

Se recopilarán los resultados obtenidos en la Tabla 4-2:

Tabla 4-2: Resultados experimentales

	Adam's Test Accuracy	RMSprop's Test Accuracy	SGD's Test Accuracy
Primeros experimentos con 50 epochs	96.21	96.48	80.83
Dropout 1 con 200 epochs	97.63	76.73	
Dropout 2 con 200 epochs	98.04	96.24	
Dropout 2 con 50 epochs	98.03	98.21	
Dropout 3 con 200 epochs	98.51	98.22	
Dropout 3 con 50 epochs	98.01	97.70	
Ajuste de hiperparámetros modificación 1 con 50 epochs	95.84		
Ajuste de hiperparámetros modificación 2 con 50 epochs	95.16		
Ajuste de hiperparámetros modificación 3 con 50 epochs	97.30		

En primer lugar, se ha observado que, de los optimizadores empleados, el que ha dado mejores resultados en la mayoría de los experimentos ha sido el Adam. Hay veces en las que el RMSprop se compara al funcionamiento del Adam, pero empíricamente se ha visto que las gráficas obtenidas con Adam tanto en validación como en entrenamiento son más suaves y con menor variabilidad de los resultados que en el RMSprop. Esto se debe a que, como se comentó en la explicación de dichos optimizadores, el Adam permite hacer uso del *momentum*, convirtiéndose en una especie de RMSprop con *momentum*, por lo que a priori era de esperar que los resultados alcanzados fueran mejores, entre otras cosas, por su capacidad de no quedarse atrapado en mínimos locales de la función de *loss*.

En los primeros experimentos, se vio que aparecía overfitting en todos los resultados. Esto no era deseable porque la red no se adaptaría bien a imágenes que no hubiera visto nunca. Como medida para disminuir este efecto, se empleó el Dropout. Se variaron los porcentajes de Dropout para ver cómo afectaba esto a la red. Lo primero que se notó, fue un leve aumento en el tiempo de entrenamiento, debido a la desconexión de las neuronas de manera aleatoria, que hacía que el ajuste de parámetros se tornara más lento por no tener activa la totalidad de las neuronas en cada época.

Precisamente, es esto lo que luego permite que el overfitting disminuya en consecuencia, no observándose en ninguno de estos experimentos. Sin embargo, como la validación tomaba valores de *loss* inferiores a los de entrenamiento en todo momento, se puede decir que ahora apareció un problema de underfitting, pero que era

mucho menos agresivo que el overfitting de los primeros experimentos, y en algunos casos era prácticamente nulo, como en el caso de la Figura 4-29.

En el Dropout 1 se obtienen los peores resultados de todos los experimentos de Dropout, pero son mejores que los que se dan en los primeros experimentos. Es en los experimentos de Dropout 2 y Dropout 3 donde realmente los resultados empiezan a mejorar disminuyendo el underfitting a cantidades muy pequeñas, como ya se ha visto. Por tanto, lo que se puede concluir del uso del Dropout en esta red es que cuando es muy pequeño, es decir, cuando prácticamente la totalidad de las neuronas está activa, como en el primer caso, los resultados mejoran respecto a la red sin Dropout. Si el Dropout se hace más agresivo, y de manera que al principio sea más pequeño que al final de la red, los resultados mejoran considerablemente respecto al caso anterior. Y, finalmente, si entre la última capa oculta y la capa de salida no se hace Dropout, como el caso de Dropout 3, se consiguen como mínimo los mismos resultados que en la situación anterior, e incluso mejorándolos.

El mejor Dropout entonces será el Dropout 3, ya que, al principio, cuando hay más información que tomar, es menos agresivo, dejando un mayor número de neuronas activas para la toma de datos, mientras que al final de la red hay más neuronas desactivadas, donde la información ya está condensada en la red y hay que evitar el overfitting. Entre la capa de salida y la última capa oculta no se añade Dropout para que se le proporcione más información a la capa de salida Softmax que hace las predicciones. Este es el razonamiento de uso de este Dropout.

Finalmente, se hicieron varios experimentos que tenían como objetivo ver cómo influía en la red el número de neuronas en las capas y el número de capas, de modo que se hicieron algunas modificaciones en función de lo que indicase la validación en cada momento. Se partió del modelo original, sin aplicar Dropout, y utilizando únicamente el optimizador Adam por lo comentado anteriormente.

Primero se cambió la estructura con tendencia a **aumentar** el número de neuronas y el número de capas. Concretamente, se aumentaron el número de kernels por capa convolucional sumando una potencia de 2 a cada capa. Es decir, originalmente la primera capa tenía 32 kernels, por lo que ahora tendrá 64, la siguiente tendrá 128, y así hasta la última capa convolucional. Lo mismo ocurrió con la etapa densamente conectada, teniendo ahora 2048 neuronas en su capa de entrada en lugar de 1024, y en cada capa sucesiva se disminuyó el número de neuronas en potencia de dos. Antes de la capa de salida, se incluyó otra capa oculta con 64 neuronas.

Esto llevó aparejado un aumento de variabilidad de los resultados en la validación, debido probablemente al overfitting que se produce por la cantidad de nuevos parámetros entrenables, siendo de unos 8.5 millones frente a los 2.1 millones que tenía el modelo original. Por tanto, se dedujo que un aumento de parámetros del modelo de manera exagerada y sin ningún método de contención de overfitting, producía dicho problema y empeoraba los resultados originales.

El siguiente paso fue realizar lo contrario, **disminuir** el número de parámetros. Para ello, se volvió a partir de la red original y se hizo lo contrario del experimento anterior, es decir, se disminuyó el número de kernels de la etapa convolucional en una potencia de 2 cada una, y lo mismo para las capas de la etapa densamente conectadas. Además, si antes se añadió una capa más en la etapa convolucional, ahora se va a eliminar una capa, teniendo solo 2 capas ocultas. El número de parámetros pasó de 2.1 millones a 0.53 millones aproximadamente, unas 4 veces menos. El efecto que tuvo fue una suavización de las gráficas de validación respecto al aumento de los parámetros. Esta falta de variación en la validación respecto al anterior se puede deber a una menor sensibilidad en de la red a la detección de clases, porque si se observan los experimentos anteriores, rara vez entrenamiento y validación se quedan en un valor de *accuracy* estable al 100 %, ya que sufren pequeñas variaciones en cada época. Sin embargo, aquí parecen quedarse constantes, indicando una falta de entrenamiento, o underfitting.

Para terminar, se hizo otro experimento sobre una red modificada para aumentar el número de parámetros parecida a la que se acababa de hacer, con el fin de reducir el underfitting que se producía en los experimentos

de Dropout. Lo que se obtuvo como resultado fue un entrenamiento y validación casi iguales a los obtenidos con el Dropout 2 o Dropout 3, por lo que, a pesar de lo pesado que se hizo el modelo con unos 25.3 millones de parámetros, no se consiguió una aparente mejoría.

De los experimentos realizados también se puede sacar como conclusión que el número de épocas puede estropear los resultados si no se elige de manera correcta, ya que han habido experimentos donde un mayor número de épocas no ha influido en exceso, como en el caso de Dropout 3, pero en el caso de Dropout 2 con RMSprop se pudo apreciar claramente como el obtenido cuando se eligieron 50 épocas (Figura 4-21) de entrenamiento era mucho mejor que el obtenido con 200 épocas (Figura 4-19), donde los porcentajes de *accuracy* de test cayeron un 2 %.

Por tanto, una vez que se ha analizado todo lo anterior, las mejores redes obtenidas de todos los experimentos realizados son la RMSprop de 50 épocas con el Dropout 2 con un **98.21 %** de *accuracy*, RMSprop de 200 épocas con Dropout 3 con un **98.22 %**, y finalmente la red Adam con 200 épocas y Dropout 3 con un **98.51 %**.

Examinándolas con mayor detalle, se puede ver que la diferencia entre unas y otras es que para algunas clases tiene mayor precisión, pero se compensa con la bajada de precisión en otras clases que en cualquiera de los otros 2 modelos tiene un valor mayor de precisión. Lo mismo ocurriría con la métrica recall, por lo que en definitiva, si hay que elegir uno de estos modelos como el mejor en este dataset, y dadas sus similitudes, se elegiría el Adam con Dropout 3 por tener un porcentaje del 98.51% de *accuracy* respecto al 98.2 % de las otras 2 redes, ya que esto indica que dicho modelo será mejor en general con todas las señales, siendo ese el objetivo, encontrar un modelo que sea el que mejor se adapte a todas las clases de una forma general, que es lo que indica la métrica *accuracy*, siendo esa la razón de su elección sobre cualquier otra métrica.

Una vez elegida la red final de diseño, se comentarán con más detalles sus resultados de test:

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40	41	42
0	56	0	0	0	0	0	0	0	4	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	714	4	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
2	0	0	747	0	0	3	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
3	0	0	0	449	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
4	0	0	2	0	658	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
5	0	0	1	9	1	582	0	6	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
6	0	0	0	0	0	0	120	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
7	0	0	0	0	0	0	0	480	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
8	0	0	0	0	0	2	0	4	444	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
9	0	0	0	0	0	0	0	0	0	480	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
10	0	0	0	0	0	0	0	0	0	0	660	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
11	0	0	0	0	0	0	0	0	0	0	0	418	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0
12	0	0	0	0	0	0	0	0	0	0	0	0	690	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
13	0	0	0	0	0	0	0	0	0	0	0	0	0	720	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
14	0	0	0	0	0	0	0	0	0	0	0	0	0	0	240	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
15	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	210	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
16	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	120	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
17	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	360	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
18	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	388	0	0	0	0	0	0	0	2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
19	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	60	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
20	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	120	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
21	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	81	0	2	0	0	1	0	0	1	0	4	0	0	0	0	0	0	0	0	0	0	0
22	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	120	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
23	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	1	21	0	0	127	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
24	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	3	0	0	0	0	0	84	0	2	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
25	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	476	0	0	1	0	1	0	0	0	0	0	1	0	0	0	0	0	0
26	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	2	0	0	0	2	0	0	0	175	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
27	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	59	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
28	0	1	0	3	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	2	1	0	173	0	0	0	0	0	0	0	0	0	0	0	0	0	0
29	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	2	0	0	3	0	1	0	0	0	83	1	0	0	0	0	0	0	0	0	0	0	0	0
30	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	2	0	2	0	0	0	0	0	0	0	0	0	144	2	0	0	0	0	0	0	0	0	0	0	0
31	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	239	0	0	0	0	0	0	0	0	0	0	0
32	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	60	0	0	0	0	0	0	0	0	0	0
33	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	209	0	0	0	0	0	0	0	0	0
34	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	120	0	0	0	0	0	0	0	0
35	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	388	0	0	0	0	1	0	0
36	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	120	0	0	0	0	0	0
37	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	60	0	0	0	0	0
38	0	12	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	3	1	0	0	0	0	0	0	9	629	0	34	0	0
39	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	1	3	0	0	0	0	0	85	0	0	0
40	0	0	0	1	0	0	0	0	0	0	0	0	2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	117	0	0
41	0	0	0	0	0	0	0	0	0	3	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	57	0
42	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	60

Figura 4-49: Matriz de confusión del modelo final

	precision	recall	f1-score	support
0	1.00	0.93	0.97	60
1	0.98	0.99	0.99	720
2	0.99	1.00	0.99	750
3	0.97	1.00	0.98	450
4	1.00	1.00	1.00	660
5	0.99	0.97	0.98	600
6	1.00	1.00	1.00	120
7	0.98	1.00	0.99	480
8	0.99	0.99	0.99	450
9	0.99	1.00	1.00	480
10	1.00	1.00	1.00	660
11	1.00	1.00	1.00	420
12	1.00	1.00	1.00	690
13	1.00	1.00	1.00	720
14	1.00	1.00	1.00	240
15	1.00	1.00	1.00	210
16	1.00	1.00	1.00	120
17	1.00	1.00	1.00	360
18	0.98	0.99	0.99	390
19	0.94	1.00	0.97	60
20	0.83	1.00	0.91	120
21	1.00	0.90	0.95	90
22	0.98	1.00	0.99	120
23	0.96	0.85	0.90	150
24	1.00	0.93	0.97	90
25	0.99	0.99	0.99	480
26	0.97	0.97	0.97	180
27	0.98	0.98	0.98	60
28	0.98	0.96	0.97	180
29	0.95	0.92	0.94	90
30	0.97	0.96	0.97	150
31	0.98	1.00	0.99	240
32	0.98	1.00	0.99	60
33	0.99	1.00	0.99	209
34	1.00	1.00	1.00	120
35	1.00	0.99	1.00	390
36	0.99	1.00	1.00	120
37	0.87	1.00	0.93	60
38	1.00	0.91	0.95	690
39	1.00	0.94	0.97	90
40	0.77	0.97	0.86	120
41	1.00	0.95	0.97	60
42	1.00	1.00	1.00	60
accuracy			0.99	12569
macro avg	0.98	0.98	0.98	12569
weighted avg	0.99	0.99	0.99	12569

Figura 4-50: Resultados de test de modelo final

En cuanto a la métrica *precision*, es superior al 90 % en la mayoría de las categorías, por lo que en esas clases de todos los positivos que la red ha detectado como positivos, el 90 % como mínimo serán verdaderos positivos. Este porcentaje de acierto decae en las clases 20, 37 y 40, donde tienen un porcentaje del 83 %, 87 % y 77 % respectivamente.

En el caso del *recall* también tiene la mayoría un 90 %, por lo que de todos los positivos detectados y que sean realmente positivos, el 90 % como mínimo serán verdaderos positivos. Solamente hay una clase que baja del 90 % es la clase 23, y se queda en un 85 %.

Clases como la 40 que tenían una *precision* del 77 %, ahora tienen un *recall* del 97 %. Con esto se puede deducir que, aunque se tenga un resultado bajo en verdaderos positivos según los positivos que la red haya detectado, en realidad su acierto de verdaderos positivos respecto a los que realmente son verdaderos positivos es más alto de lo que se creía, siendo en este caso concreto de un 20 % más alto. Por tanto, se puede concluir que la red ofrecerá resultados muy robustos y fiables.

En cuanto a la matriz de confusión en la Figura 4-49, se puede observar que no hay mucha confusión de unas clases con otras. Los fallos más relevantes se dan en la clase 23, que confunde 21 imágenes con la clase 20, y la clase 38, que confunde 12 imágenes con la clase 1 y 34 imágenes con la clase 40. Hay que destacar que las

imágenes de test para este último caso son 690, por lo que esas 46 imágenes tampoco son tantas comparadas con esa cantidad.

El resto de las imágenes están bastante centradas en sus filas y columnas respectivas, indicando los aciertos correctos que hace la red de cada imagen. Algunas de esas imágenes confundidas se ven en las Figura 4-51 y Figura 4-52, indicando el número de la imagen del conjunto de test que ha sido confundida.

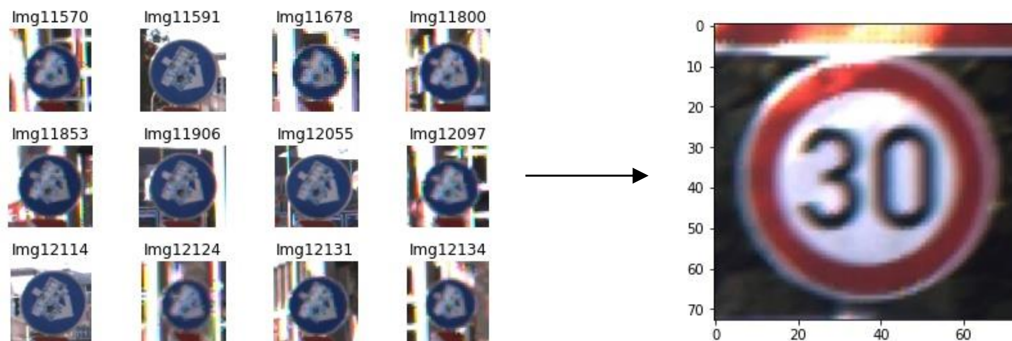


Figura 4-51: Imágenes de la clase 38 confundidas con la clase 1

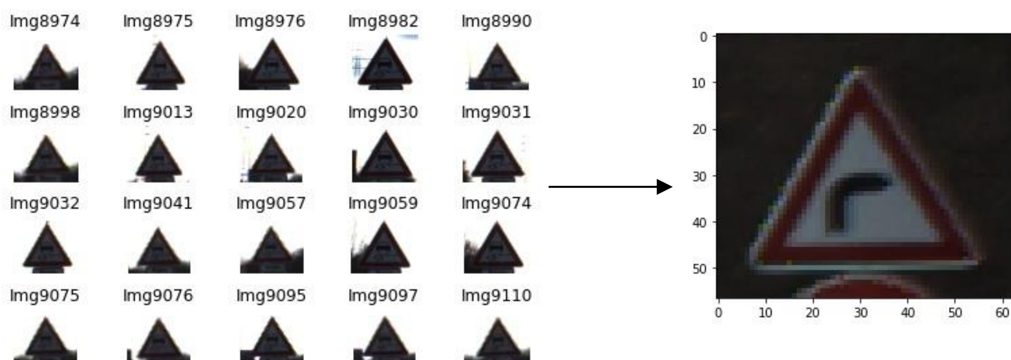


Figura 4-52: Imágenes de la clase 23 confundidas con la clase 20

Por último, se calculó el tiempo medio que tardaba en clasificar una señal. Para ello, se tomaron 10 imágenes y se calculó el tiempo que tardó en cada una, haciendo al final la media. El resultado obtenido fue de 0.02494 segundos, por lo que de media tarda del orden de 30 ms por cada imagen que le llegue, incluyendo preprocesado de dicha imagen. Cumple con el requisito de los 500 ms de margen que se impuso al principio, estando del orden de 17 veces por debajo de ese tiempo límite. Este experimento se realizó con una GPU proporcionada por Google Colab.

4.7 Implementación en sistema embebido

En este apartado se trató de implementar la red conseguida, y cuyos resultados se acaban de comentar, en un sistema embebido real para poner a prueba su funcionamiento y ver el tiempo de respuesta que tiene en comparación con los experimentos realizados anteriormente.

Por disponibilidad de materiales en el departamento, se ha usado una Raspberry Pi 3 modelo B, con sistema

operativo Raspbian buster, y una cámara para Raspberry conocida como Raspberry Pi Camera V2. Las especificaciones de dichos dispositivos se muestran en la Tabla 4-3:

Tabla 4-3: Especificaciones de sistema embebido

Raspberry Pi 3	**Raspberry Pi Camera**
Quad Core 1.2GHz Broadcom BCM2837 64bit CPU	8 megapixel native resolution sensor-capable of 3280 x 2464 pixel static images
1GB RAM	Supports 1080p30, 720p60 and 640x480p90 video
BCM43438 wireless LAN and Bluetooth Low Energy (BLE) on board	Size: 25mm x 23mm x 9mm
100 Base Ethernet	Sensor: Sony IMX219
40-pin extended GPIO	
4 USB 2 ports	
4 Pole stereo output and composite video port	
Full size HDMI	
CSI camera port for connecting a Raspberry Pi camera	
DSI display port for connecting a Raspberry Pi touchscreen display	
Micro SD port for loading your operating system and storing data	
Upgraded switched Micro USB power source up to 2.5A	

Para visualizar gráficamente el SO se ha usado una aplicación llamada VNC Viewer, que permite crear un servidor local o en red para ver la salida de la Raspberry en la pantalla del portátil, haciendo el manejo algo más cómodo que al usar 2 pantallas con 2 teclados, usando para ello una conexión local configurada en el puerto Ethernet. El montaje completo se muestra en la Figura 4-53.

Para utilizar la cámara, se usó la librería “picamera”, que contiene funciones y opciones de configuración de la cámara tales como resolución de captura de imagen, filtros aplicables, etc. Se cargó el modelo de red obtenido y se realizaron varias detecciones de señales, y se cronometró el tiempo que tardaba en clasificar la señal una vez capturada. El código empleado se encuentra en el enlace de GitHub proporcionado al principio del capítulo.

Figura 4-53: Montaje de Raspberry Pi 3 con cámara

Para emular la llegada de las señales de tráfico a la cámara, se utilizaron fotografías de dichas señales impresas en papel, de modo que se enseñaba el papel con la señal a la cámara, y una vez que estuviera en el encuadre, se indica por teclado que tome una fotografía. Dicha imagen se guardará directamente como una matriz numpy en lugar de como un archivo tipo jpg o png por comodidad de trabajo con la red.

A partir de que la cámara tome la fotografía, el proceso es automático y empieza a identificar la señal correspondiente. Se han usado señales de límites máximos de velocidad, ceda el paso y entrada prohibida, entre otras. Los resultados obtenidos del experimento se muestran a continuación:

```
Numero de imagenes clasificadas:  25
Tiempo medio que tarda en segundos:  0.3771673583984375
Historial de predicciones:
[7]   ==>  ['Velocidad máxima 100 Km/h']
[17]  ==>  ['Entrada prohibida']
[37]  ==>  ['Recto e izquierda únicas direcciones permitidas']
[0]   ==>  ['Velocidad máxima 20 Km/h']
[1]   ==>  ['Velocidad máxima 30 Km/h']
[4]   ==>  ['Velocidad máxima 70 Km/h']
[1]   ==>  ['Velocidad máxima 30 Km/h']
[5]   ==>  ['Fin de limitación de velocidad máxima 80 Km/h']
[1]   ==>  ['Velocidad máxima 30 Km/h']
[4]   ==>  ['Velocidad máxima 70 Km/h']
[13]  ==>  ['Ceda el paso']
[22]  ==>  ['Perfil irregular']
[12]  ==>  ['Calzada con prioridad']
[40]  ==>  ['Intersección de sentido giratorio-obligatorio']
[26]  ==>  ['Proximidad de semáforo']
[13]  ==>  ['Ceda el paso']
[14]  ==>  ['STOP']
[40]  ==>  ['Intersección de sentido giratorio-obligatorio']
[38]  ==>  ['Paso obligatorio derecha']
[39]  ==>  ['Paso obligatorio izquierda']
[35]  ==>  ['Sentido obligatorio recto']
[14]  ==>  ['STOP']
[2]   ==>  ['Velocidad máxima 50 Km/h']
[12]  ==>  ['Calzada con prioridad']
[35]  ==>  ['Sentido obligatorio recto']
```

En primer lugar, se indican todas las imágenes clasificadas correctamente por la red en el sistema embebido, y el tiempo medio que tarda en clasificar una señal, que en este caso ha sido de 0.37717 segundos. Se puede considerar entonces que el tiempo medio será del orden de 0.4 segundos, o 400 ms. En relación con los experimentos de simulación, donde se obtuvo un tiempo de 30 ms, se ha incrementado en una cantidad de 13.3 veces el tiempo de simulación, pero aún está por debajo del límite de 500 ms que se estableció como especificación al principio del proyecto.

En el historial de señales clasificadas, se puede ver el número de la clase a la que pertenece la señal, con el nombre de dicha señal que se asocia con esa clase. El funcionamiento en general es similar al de la simulación. Es decir, reconoce la mayoría de las señales de tráfico adquiridas por la red, pero se ha observado un funcionamiento peor para el caso de señales de velocidad máxima, en concreto para el caso de las señales de 90 km/h y 100 km/h, ya que en ocasiones las confunde con la señal de limitación máxima de 50 Km/h y con la

señal de fin de prohibición de limitación máxima de 80 Km/h. Una explicación sería que las imágenes de esas señales son algo diferentes de las que la red usó para entrenarse, provocando el conflicto. Por lo demás, no se ha observado otro comportamiento anómalo.

Hay diferentes maneras de mejorar el funcionamiento de la red en el sistema embebido, siendo una de ellas el uso de hardware específico para el desempeño de Deep Learning en este tipo de sistemas. En el siguiente capítulo se comentará alguno de estos dispositivos y cómo pueden mejorar el funcionamiento de la red.

5 CONCLUSIONES Y FUTUROS TRABAJOS

Locura es hacer la misma cosa una y otra vez
esperando obtener diferentes resultados.

- Albert Einstein -

Como último capítulo del proyecto, se sacarán las conclusiones pertinentes de todo lo realizado en anteriores capítulos, y se desarrollarán las posibles mejoras y proyectos futuros que se podrían realizar con la red diseñada. En el apartado de futuros proyectos se abordarán algunas posibles mejoras que se podrían hacer a la red actual, como un aumento de señales o su implementación en un sistema físico.

5.1 Conclusiones del proyecto

A lo largo de todo el proyecto se ha ido desde lo más general a lo más concreto. Se empezó con un análisis generalizado del panorama actual del Deep Learning en el mundo laboral, seguido de un examen de las diferentes estructuras que han ido rompiendo barreras durante años para conseguir los mejores avances en este campo. Después, se aportaron algunas aplicaciones prácticas de Deep Learning al área de control de tráfico usando Visión Artificial, permitiendo conocer algunos trabajos previos que sirvieran como guía para el realizado en este documento.

Con todo este análisis previo realizado, se comenzaron a explicar las arquitecturas de las redes que se usarían en este proyecto, que son las redes densamente conectadas y las redes convolucionales, siendo estas últimas las que más se usan en el trabajo con imágenes, como se sacó en claro de la investigación anterior. Se analizaron las diferentes estructuras de redes neuronales, las diferentes funciones de decisión que dominan dichas neuronas, y los algoritmos de optimización de entrenamiento más usuales, que son fundamentales para obtener unos resultados finales satisfactorios.

Una vez llegados a este, se estaba listo para realizar un diseño propio que resolviera el problema de la clasificación de señales de tráfico, por lo que se explicó en qué consistía dicho problema y cómo se iba a resolver, aportando definiciones de los procesos de entrenamiento, validación y testeo a seguir, las configuraciones de la red desarrollada, y todo lo necesario para definir el diseño completo de la red.

Finalmente, se realizaron los experimentos explicados, obteniendo una serie de resultados diferentes para cada uno, de manera que una vez terminados todos y cada uno, se obtuvo una red que cumplía con las especificaciones impuestas en la definición del problema, pudiendo comprobar que algunas decisiones que se tomaron inicialmente fueron acertadas. Una de esas decisiones fue la de usar las imágenes en escala de grises en lugar de usarlas en color.

A priori se podría pensar que el no detectar colores influiría de manera negativa en los resultados de la red, pero la realidad es que en escala de grises los resultados obtenidos han sido difícilmente mejorables y por encima de las especificaciones impuestas en un principio, ahorrando con este modelo u tiempo de computación muy valioso que permitió realizar más pruebas que si se hubiera elegido que los filtros de la red tuvieran 3 dimensiones en lugar de 1 para poder reconocer colores. Por tanto, se puede concluir también que esta decisión de diseño fue acertada, ya que la red puede reconocer de manera correcta las señales usando las diferentes formas y textos que las componen, pasando el color a ser una característica secundaria y de menor importancia, al menos en este dataset donde las señales son todas lo suficientemente diferentes entre sí.

En los entrenamientos se usó tanto una CPU como una GPU, observándose empíricamente la potencia de estas últimas respecto a las CPU's, puesto que en un principio al entrenar una red primeriza más pequeña que la del diseño final, se llegó a realizar un entrenamiento para 80 épocas y tardó del orden de 1 hora en terminar de ajustar los pesos. En cambio, al realizar los entrenamientos con la red final más profunda y la GPU, con 200 épocas tardaba del orden de 5 o 10 minutos, dependiendo de la GPU que Google Colab proporcionase en cada sesión. Google Colab es una plataforma online gratuita que proporciona acceso a GPU's para a realizar proyectos con Deep Learning, y se decidió usar esta plataforma debido a los largos tiempos de entrenamiento con CPU.

Por último, se comentarán una serie de mejoras y de trabajos futuros que se podrían realizar a partir del que se ha descrito en este documento, pero que quedan fuera de su alcance, pudiendo imaginar las posibles aplicaciones que tiene este sistema de clasificación de señales.

5.2 Futuros trabajos

En primer lugar, se podría ampliar el conjunto de señales identificables por la red, que actualmente es de 43 señales. Además, el dataset usado pertenece a señales de tráfico de Alemania, pero al ser de la Unión Europea, éstas serán prácticamente las mismas que en España. Sin embargo, se puede dar el caso de que en otros países tengan señales propias o que las señales con el mismo significado sean totalmente diferentes. Por eso, ampliar las señales identificables por la red permitiría que el sistema fuera más globalizado, pudiendo usarse en cualquier parte del mundo sin problemas.

Como se introdujo al final del capítulo anterior, una posible mejora futura sería usar un hardware específico para que la respuesta de la implementación en el sistema embebido sea más eficiente, equiparándose a la de un ordenador. Uno de estos dispositivos es el Neural Compute Stick 2 de Intel que, tal como se indica en [52], es un USB que contiene el hardware necesario para computar los algoritmos de Deep Learning, contando también con soporte para el sistema operativo Raspbian, que es el que usa la Raspberry Pi. Una de sus ventajas es la facilidad de uso frente a otros dispositivos con el mismo fin, ya que únicamente hay que conectarlo al puerto USB de la Raspberry Pi e instalar el software correspondiente a dicho dispositivo.

Una Raspberry Pi sin hacer uso de este dispositivo ronda un tiempo medio de respuesta de unos 400 ms, como ya se ha visto, y tiene además tiempos de respuesta individuales para cada caso algo alejados de dicho valor medio. Con el uso del Neural Compute Stick, se conseguiría reducir esa desviación estándar en una cantidad considerable, permitiendo que los valores de respuesta obtenidos individualmente se asemejen más al valor de respuesta medio. Además, dicho tiempo de respuesta bajaría un orden de magnitud, por lo que, si antes se obtuvo un tiempo de 400 ms, ahora será del orden de 40 ms, de modo que se puede comparar con la respuesta de 30 ms obtenida con la GPU de Google Colab. Estos resultados se pueden contrastar con los obtenidos en [53] por otro compañero de titulación.

Otro trabajo que se podría hacer en el futuro sería probarlo con un robot en funcionamiento sobre un escenario real. Para ello, se podría implementar un pequeño modelo de coche usando un miniordenador como por ejemplo una Raspberry Pi, que tiene soporte para Python, de manera que se le podría conectar una cámara con la que capture las imágenes de las señales y las clasifique, de modo que actúe en consecuencia.

El experimento sería realizar un circuito cerrado con un número de señales colocadas en él, de manera que el coche va avanzando por el circuito. De repente, se encuentra una señal, por ejemplo, una señal de STOP. La imagen capturada por la cámara se pasaría al programa controlador que está en la Raspberry Pi, donde se hace

el preprocesado de la imagen necesario para introducir los datos en la red, y una vez que se pasa por ésta se conseguirá clasificar la señal que aparece en la fotografía usando el modelo de red diseñado. Una vez que sepa de que señal se trata, debería haber otro programa que controle las acciones del coche, de manera que, al detectarse la señal de STOP, dicho programa debería mandar una orden a las ruedas para que se pare.

Finalmente, se podría pensar en el sistema diseñado como el 50 % de un sistema más completo de trabajo con señales de tráfico en tiempo real, y es que si se le añade a la red diseñada en este proyecto que permite saber qué señal está viendo en el entorno otra red que vaya por encima de ésta y que detecte si en la imagen hay o no alguna señal, se obtendría un sistema de detección más clasificación de señales de tráfico.

A este sistema conjunto se le pasaría una imagen de la cámara cada cierto intervalo de tiempo viendo si en esa imagen hay o no una señal de tráfico. Si la hubiera, se identificaría la bounding box que contiene la señal y se haría un preprocesado de dicha imagen de forma que se recorte la zona de dicha bounding box, se redimensione dicho recorte a 64x64 píxeles, y entonces se le pasaría a la primera red para ver de qué señal se trata. En caso de que no hubiera una señal, no se haría nada y el sistema ahorraría así recursos y capacidad de cómputo.

Además de poder identificar si hay o no una señal en la imagen capturada, se podrían usar también otras técnicas de Visión Artificial para, una vez identificada la zona en la que se encuentra, tomar algún tipo de medidas como la distancia a la que se encuentra la señal de la cámara.

De esta forma, puede usar esa información y complementarla con la que va a obtener de la red de clasificación de imágenes. Un ejemplo sería el de la señal de STOP en el sistema del coche autónomo descrito anteriormente, ya que puede que identifique bien una señal de STOP, pero si el programa que controla el vehículo no tiene bien implementado un protocolo para manejar la acción correspondiente, se podría parar justo al momento de detectarla, pudiendo no haber llegado aún a la señal de STOP y parando en medio de la calle. Si sabe qué distancia le queda para llegar, una vez detectada la señal podría simplemente seguir capturando con la red de detección la señal actualizando su distancia a la cámara, y parar en el momento en el que la distancia esté por debajo de cierto umbral.

Para realizarla, se podría usar YOLO, que usa la arquitectura Darknet inspirada en GoogleNet, como se puede ver en la Figura 5-1. Esta red es revolucionaria en la detección de objetos de cualquier tipo permitiendo crear alrededor de dicho objeto una bounding box. Lo único que sería necesario es un buen dataset de imágenes que permitan entrenar la red de manera correcta, indicando los límites de la bounding box de cada señal de cada imagen, así como la clase a la que pertenece, como se indica en [54].

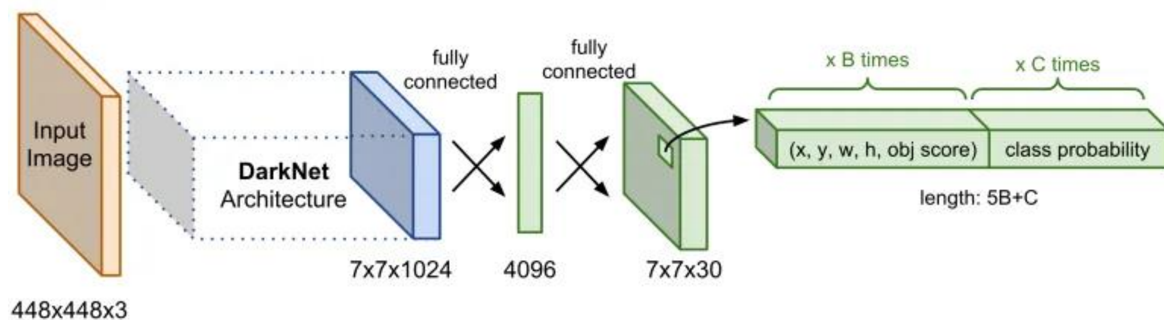


Figura 5-1: Arquitectura YOLO de 24 capas [54]

Según [55], para explorar la imagen, lo que hace la red de YOLO es dividirla en regiones, de manera que explora cada región prediciendo los límites de la bounding box y las probabilidades de cada región, que se ponderan con las divisiones de la imagen. La imagen es observada globalmente por una sola red, mientras que con los métodos basados en clasificadores se le pasan diferentes redes que la exploran a distintos niveles. Por ello, como lo hace

todo de una sola pasada, es increíblemente rápida, en concreto, del orden de magnitud de 1000 veces más rápidas que la R-CNN y 100 veces más rápida que las Fast R-CNN.

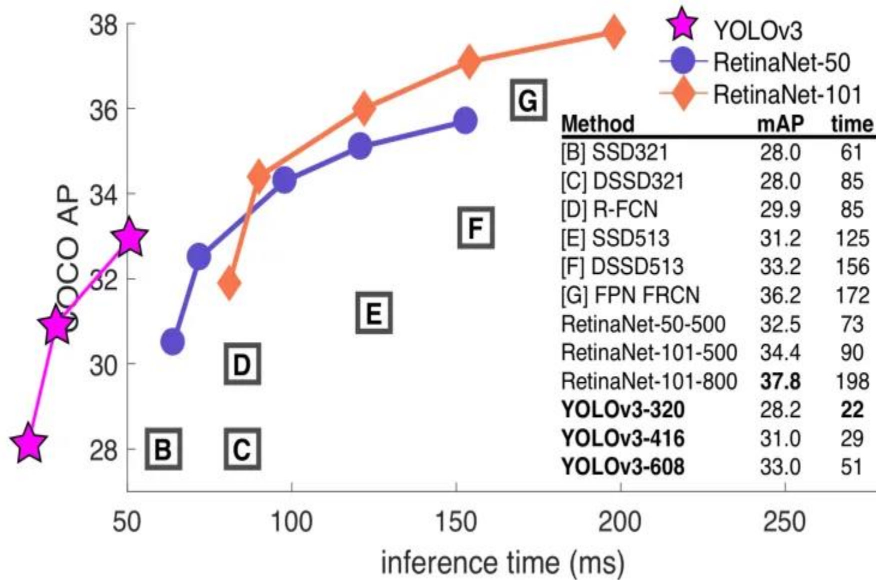


Figura 5-2: YOLOv3 sobre el dataset COCO [54]

La elección de YOLO sobre otras redes de detección se haría porque es la más rápida que existe. En la Figura 5-2 se puede ver una comparación de YOLOv3 respecto a otros métodos de detección en el dataset COCO. La métrica “mAP” que aparece en la gráfica permite controlar que no se detecten varias veces el mismo objeto y que sea capaz de distinguir varios objetos cercanos que se superpongan en la imagen, y controla que tanto la clase como la posición de la bounding box sea correcta.

Estas serían todas las conclusiones del trabajo y las posibles mejoras y experimentos que se podrían realizar en el futuro pero que quedan fuera del alcance de este TFG. Personalmente, la realización del proyecto a resultado muy enriquecedora a la hora de informarse sobre esta parte de la Inteligencia Artificial, habiendo adquirido unos conocimientos y una base sobre dicha temática que resultará muy útil en futuros proyectos.

6 BIBLIOGRAFÍA

[1] G. Julián, «Las redes neuronales: qué son y porqué están volviendo,» 21 Enero 2016. [En línea]. Available: https://www.xataka.com/robotica-e-ia/las-redes-neuronales-que-son-y-por-que-estan-volviendo. [Último acceso: 20 Julio 2020].

[2] Aprende Machine Learning, «Breve Historia de las Redes Neuronales Artificiales,» 12 Septiembre 2018. [En línea]. Available: https://www.aprendemachinelearning.com/breve-historia-de-las-redes-neuronales-artificiales/. [Último acceso: 20 Julio 2020].

[3] A.I, Data and Software Engineering, «Quick Benchmark Colab CPU GPU TPU (XLA-CPU),» 2019. [En línea]. Available: https://petamind.com/quick-benchmark-colab-cpu-gpu-tpu-xla-cpu/. [Último acceso: 20 Agosto 2020].

[4] Xataka, «CPU: qué es, cómo es y para qué sirve,» 2 Marzo 2020. [En línea]. Available: https://www.xataka.com/basics/cpu-que-como-sirve. [Último acceso: 20 Agosto 2020].

[5] Wikipedia, «Unidad de procesamiento gráfico,» 21 Junio 2020. [En línea]. Available: https://es.wikipedia.org/wiki/Unidad_de_procesamiento_gr%C3%A1fico. [Último acceso: 20 Julio 2020].

[6] Xataka, «Tarjeta gráfica: qué es, qué hay dentro y cómo funciona,» 27 Mayo 2020. [En línea]. Available: https://www.xataka.com/basics/tarjeta-grafica-que-que-hay-dentro-como-funciona. [Último acceso: 20 Julio 2020].

[7] R. Alonso, «¿Qué significa que tu tarjeta gráfica tenga arquitectura MCM?,» 25 Marzo 2020. [En línea]. Available: https://hardzone.es/reportajes/que-es/arquitectura-mcm-tarjeta-grafica/. [Último acceso: 20 Julio 2020].

[8] Google Cloud, «Cloud Tensor Processing Unit (TPU),» Google, 23 Junio 2020. [En línea]. Available: https://cloud.google.com/tpu/docs/tpus?hl=es-419. [Último acceso: 20 Julio 2020].

[9] Google Cloud, «Arquitectura del sistema,» Google, 1 Julio 2020. [En línea]. Available: https://cloud.google.com/tpu/docs/system-architecture?hl=es. [Último acceso: 20 Julio 2020].

[10] Wikipedia, «Visión artificial,» Wikipedia, 14 Agosto 2020. [En línea]. Available: https://es.wikipedia.org/wiki/Visi%C3%B3n_artificial. [Último acceso: 2 Septiembre 2020].

[11] Tomatech, «¿Qué sistemas de visión artificial existen?,» 25 Octubre 2019. [En línea]. Available: https://www.tatomatech.com/que-sistemas-de-vision-artificial-existen/. [Último acceso: 2 Septiembre 2020].

[12] Bcnvision, «Deep Learning, en el punto de mira de la visión artificial,» 6 Marzo 2019. [En línea]. Available: https://www.bcnvision.es/blog-vision-artificial/deep-learning-vision-artificial/. [Último acceso: 2 Septiembre 2020].

[13] M. Fortuño, «El impacto de la inteligencia artificial en la banca,» 1 Junio 2019. [En línea]. Available: https://www.elblogsalmon.com/productos-financieros/impacto-inteligencia-artificial-banca. [Último acceso: 4 Agosto 2020].

[14] Fintech, «Cinco aportaciones de la inteligencia artificial al sector financiero,» 2 Julio 2018. [En línea]. Available: https://www.bbva.com/es/cinco-aportaciones-inteligencia-artificial-sector-financiero/. [Último acceso: 4 Agosto 2020].

[15] J. Martínez, «¿Cómo afectará la inteligencia artificial al sector financiero?,» [En línea]. Available: https://www.finanzas.com/empresas-y-directivos/como-afectara-la-inteligencia-artificial-al-sector-financiero_13949882_102.html. [Último acceso: 4 Agosto 2020].

[16] IAT, «Inteligencia artificial en videojuegos: una mirada al pasado y futuro de la industria,» [En línea]. Available: https://iat.es/tecnologias/inteligencia-artificial/videojuegos/. [Último acceso: 5 Agosto 2020].

[17] 20minutos, «Así funciona el algoritmo de netflix que te incita a ver sus series y películas,» 10 Enero 2019. [En línea]. Available: https://www.20minutos.es/noticia/3533237/0/algoritmo-netflix-manipula-veas-series-peliculas/?autoref=true. [Último acceso: 5 Agosto 2020].

[18] R. Peco, «La inteligencia artificial ya rejuvenece a actores en cine y televisión,» 6 Noviembre 2019. [En línea]. Available: https://www.lavanguardia.com/tecnologia/20191106/471413348212/cine-series-hbo-inteligencia-artificial-productoras.html. [Último acceso: 5 Agosto 2020].

[19] IAT, «Inteligencia artificial en medicina: avances que salvan vidas,» [En línea]. Available: https://iat.es/tecnologias/inteligencia-artificial/medicina/. [Último acceso: 7 Agosto 2020].

[20] TICbeat, «Inteligencia Artificial en la salud: mejores avances de los últimos tiempos,» 8 Marzo 2020. [En línea]. Available: https://www.ticbeat.com/innovacion/inteligencia-artificial-en-la-salud-mejores-avances-de-los-ultimos-tiempos/. [Último acceso: 7 Agosto 2020].

[21] elDiario, «Inteligencia artificial en los coches: del ojo humano a los omnipresentes algoritmos,» 13 Noviembre 2019. [En línea]. Available: https://www.eldiario.es/motor/tecnologia/inteligencia-artificial-coches-omnipresentes-algoritmos_1_1255766.html. [Último acceso: 8 Agosto 2020].

[22] IAT, «Inteligencia artificial y robótica: el binomio del futuro,» [En línea]. Available: https://iat.es/tecnologias/inteligencia-artificial/robotica/. [Último acceso: 8 Agosto 2020].

[23] C. Rus, «GPT-3, el nuevo modelo de lenguaje de OpenAI, es capaz de programar, diseñar y hasta conversar sobre política o economía,» 20 Julio 2020. [En línea]. Available: https://www.xataka.com/robotica-e-ia/gpt-3-nuevo-modelo-lenguaje-openai-capaz-programar-disenar-conversar-politica-economia. [Último acceso: 8 Agosto 2020].

[24] A. Zhang, Z. C. Lipton, M. Li y A. J. Smola, «Modern Convolutional Neural Networks,» de *Dive into Deep Learning*, 2020, pp. 261-305.

[25] Y. LeCun, L. Bottou, Y. Bengio y P. Haffner, «Gradient-Based Learning Applied to Document Recognition,» Proc. of the IEEE, 1998.

[26] A. Krizhevsky, I. Sutskever y G. E. Hinton, «ImageNet Classification with Deep Convolutional Neural Networks,» 2012.

[27] K. Simonyan y A. Zisserman, «Very Deep Convolutioinal Networks for Large-Scale Image Recognition,»

2014.

[28] M. Lin, Q. Chen y S. Yan, «Network In Network,» 2013.

[29] C. Szegedy, «Going Deeper with Convolutions,» Computer Vision Foundation, 2015.

[30] K. He, X. Zhang, S. Ren y J. Sun, «Deep Residual Learning for Image Recognition,» Computer Vision Foundation, 2016.

[31] G. Huang, Z. Liu, L. van der Maaten y K. Q. Weinberger, «Densely Connected Convolutional Networks,» Computer Vision Foundation, 2017.

[32] HEREmobility, «Traffic AI: A Real-Life Use Case,» [En línea]. Available: https://mobility.here.com/learn/smart-transportation/traffic-ai-real-life-use-case. [Último acceso: 16 Agosto 2020].

[33] *Respiratory effects of air pollutant among nonsmoking traffic policemen of Patiala, India,* 2011.

[34] Alibaba Cloud, «Application of Machine Learning in Traffic Sign Detection and Fine-grained Classification,» 20 Noviembre 2019. [En línea]. Available: https://www.alibabacloud.com/blog/application-of-machine-learning-in-traffic-sign-detection-and-fine-grained-classification_595569. [Último acceso: 16 Agosto 2020].

[35] Bosch, «Better than a pair of eyes: Bosch camera with AI for driver assistance and automated driving,» 9 Septiembre 2019. [En línea]. Available: https://www.bosch-press.be/pressportal/be/en/press-release-18624.html. [Último acceso: 17 Agosto 2020].

[36] SecureWeek, «secureweek,» Enero 2020. [En línea]. Available: https://www.secureweek.com/reconocimiento-de-senales-de-trafico-con-ia/). [Último acceso: Agosto 2020].

[37] F. Lindner, U. Kressel y S. Kaelberer, «Robust Recognition of Traffic Signals,» 2004.

[38] Y. Lu, J. Lu, S. Zhang y P. Hall, «Traffic signal detection and classification in street views using an attention model,» 2018.

[39] J. Torres, DEEP LEARNING Introducción práctica con Keras., España: Independently published, 2018.

[40] J. Mariano Álvarez, «El perceptrón como neurona artificial,» 10 Junio 2018. [En línea]. Available: http://blog.josemarianoalvarez.com/2018/06/10/el-perceptron-como-neurona-artificial/. [Último acceso: 14 Agosto 2020].

[41] C. Ponal, «Redes Neuronales UT,» 11 Mayo 2017. [En línea]. Available: http://cipaponalredesneuronalesut.blogspot.com/2017/05/redes-monocapa.html. [Último acceso: 15 Agosto 2020].

[42] A. Ballesteros, «JRedesNeuronales,» Universidad de Málaga, [En línea]. Available: http://www.redes-neuronales.com.es/. [Último acceso: 14 Agosto 2020].

[43] D. Calvo, «Perceptrón Multicapa - Red Neuronal,» 8 Diciembre 2018. [En línea]. Available: https://www.diegocalvo.es/perceptron-multicapa/. [Último acceso: 16 Agosto 2020].

[44] G. Glenn y O. Del Rio, «Estudio y Aplicaciones de las Redes ART (Teoría de la Resonancia Adaptativa),» Bolivar, 2007.

[45] C. Gil, «Análisis de Componentes Principales (PCA),» Junio 2018. [En línea]. Available: https://rpubs.com/Cristina_Gil/PCA. [Último acceso: 16 Agosto 2020].

[46] M. Silva, «Aprendizaje por Refuerzo: Introducción al mundo del RL,» 28 Abril 2019. [En línea]. Available: https://medium.com/aprendizaje-por-refuerzo-introducci%C3%B3n-al-mundo-del/aprendizaje-por-refuerzo-introducci%C3%B3n-al-mundo-del-rl-1fcfbaa1c87. [Último acceso: 18 Agosto 2020].

[47] E. Freire y S. Silva, «Redes neuronales,» 14 Noviembre 2019. [En línea]. Available: https://medium.com/@bootcampai/redes-neuronales-13349dd1a5bb. [Último acceso: 19 Agosto 2020].

[48] InteractiveChaos, «Tutorial de Machine Learning,» [En línea]. Available: https://www.interactivechaos.com/manual/tutorial-de-machine-learning/. [Último acceso: 19 Agosto 2020].

[49] DeepAI, «deepai.org,» [En línea]. Available: https://deepai.org/machine-learning-glossary-and-terms/rmsprop. [Último acceso: 19 Agosto 2020].

[50] N. Araque, «Regularizando nuestra red: Dropout,» 20 Marzo 2019. [En línea]. Available: https://mc.ai/regularizando-nuestra-red-dropout/. [Último acceso: 2 Septiembre 2020].

[51] J. Martínez, «Más Allá del Accuracy: Precisión, Recall y F1,» Diciembre 2019. [En línea]. Available: https://datasmarts.net/es/mas-alla-del-accuracy-precision-recall-y-f1/. [Último acceso: 8 Septiembre 2020].

[52] A. Allan, «Getting Started with the Intel Neural Compute Stick 2 and the Raspberry Pi. Getting Started with the Intel's Movidius Hardware,» 8 Abril 2019. [En línea]. Available: https://medium.com/hacksters-blog/getting-started-with-the-intel-neural-compute-stick-2-and-the-raspberry-pi-6904ccfe963. [Último acceso: 23 Septiembre 2020].

[53] A. J. Toro Valderas, «Implementación de redes neuronales en Raspberry Pi 3 con Movidius Neural Compute Stick,» Sevilla, 2020.

[54] Aprende Machine Learning, «Modelos de Detección de Objetos,» 21 Agosto 2020. [En línea]. Available: https://www.aprendemachinelearning.com/modelos-de-deteccion-de-objetos/. [Último acceso: 2 Septiembre 2020].

[55] J. Redmon y A. Farhadi, «YOLO: Real-Time Object Detection,» 2018. [En línea]. Available: https://pjreddie.com/darknet/yolo/. [Último acceso: 2 Septiembre 2020].