



**Universidad
Rey Juan Carlos**

**ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA
INFORMÁTICA**

GRADO EN INGENIERÍA INFORMÁTICA

Curso Académico 2017/2018

Trabajo Fin de Grado

**RECONOCIMIENTO DE SEÑALES DE
TRÁFICO MEDIANTE TÉCNICAS DE
APRENDIZAJE PROFUNDO.**

Autor: David Robert Núñez

Director: Alfredo Cuesta Infante

AGRADECIMIENTOS

En primer lugar agradecer a mi tutor Alfredo Cuesta, por estar siempre disponible y dispuesto a resolver cualquier duda. También a mi familia, en especial a mis padres, que siempre me han apoyado.

RESUMEN

En este proyecto se pretende resolver un problema de reconocimiento de imágenes de señales de tráfico, para ello vamos a utilizar diferentes algoritmos de *machine learning* y *deep learning* entrenados sobre el mismo conjunto de imágenes de señales de tráfico y vamos a comparar sus resultados, para ver cual es la técnica más adecuada para resolver este problema.

ABSTRACT

This project aims to solve a problem of recognition of traffic signal images, for this we will use different algorithms of machine learning and deep learning trained on the same set of traffic signs images and we will compare their results, to see which is the most appropriate technique to solve this problem.

Índice

1. Introducción	1
1.1. Descripción del problema	1
1.2. Objetivos	2
1.3. Estado del arte	3
1.3.1. <i>Machine learning</i> y <i>Deep learning</i>	3
1.3.2. Reconocimiento de imágenes	5
1.3.3. Clasificación de señales de tráfico	6
2. Metodología	7
2.1. Herramientas utilizadas	7
2.2. El conjunto de datos	8
2.3. Preprocesado	11
2.4. Validación Cruzada	11

2.5. Algoritmos	13
3. Comparación de Algoritmos	15
3.1. Métodos clásicos	15
3.1.1. Random forest	15
3.1.2. Red neuronal	16
3.2. Transfer learning	19
3.2.1. VGG16/VGG19	20
3.2.2. Resnet50	21
3.2.3. Xception	23
3.3. CNNs Personalizadas	24
3.3.1. Simple CNN	25
3.3.2. Skip CoNet	26
3.3.3. Skip CoNet - Simple CNN	28
3.3.4. Mini VGG	29
4. Validación de resultados	33
4.1. Pruebas con el conjunto de test	33

4.2. Nuevos datos de test	35
5. Conclusiones	39
6. Bibliografía	43
Lista de Figuras	47
Anexos	50
A. Código	53

Capítulo 1

Introducción

1.1. Descripción del problema

El reconocimiento de objetos en imágenes dentro del área de la visión por ordenador lleva años presente. El problema del reconocimiento consiste en predecir a qué clase pertenece una imagen de entre un conjunto de clases predefinidas.

El reconocimiento de señales de tráfico tiene un gran número de aplicaciones relacionadas con ayudas a la conducción, los coches autónomos e incluso la creación de mapas, pero sigue siendo un problema complicado. Aunque las señales están diseñadas para ser fácilmente reconocibles a la vista, hay muchos factores que dificultan automatizar la tarea, como pueden ser distintas condiciones de luminosidad, condiciones climatológicas adversas, occlusiones parciales (árboles, postes...) o un mal mantenimiento (grafitis, pegatinas...). Todas estas dificultades están presentes en el conjunto de datos utilizado.

En los últimos tiempos gracias al aumento de la capacidad de procesamiento y a la cantidad de datos disponibles, ha resurgido un interés por el aprendizaje automático, principalmente por las redes neuronales. Cuando hablamos de aprendizaje automático o *machine learning* existen dos áreas principales: aprendizaje supervisado y aprendizaje no supervisado. La diferencia principal entre los dos tipos es que el aprendizaje supervisado se hace usando datos conocidos, o en otras palabras, tenemos un conocimiento previo de cuáles deberían ser los valores de salida para nuestras

muestras. Por lo tanto, el objetivo del aprendizaje supervisado es aprender una función que, dada una muestra de datos y resultados deseados, se aproxime mejor a la relación entre la entrada y la salida observable en los datos. El aprendizaje no supervisado, por otro lado, no tiene salidas etiquetadas, por lo que su objetivo es inferir la estructura natural presente dentro de un conjunto de datos.

En el aprendizaje supervisado, se proporcionan los datos de entrada y de salida deseados. Los datos de entrada y salida están etiquetados para su clasificación a fin de proporcionar una base de aprendizaje para el futuro procesamiento de datos.

Los datos de entrenamiento para aprendizaje supervisado incluyen un conjunto de ejemplos con datos de entrada emparejados a la salida deseada. En el aprendizaje supervisado para el procesamiento de imágenes, como es nuestro caso, se podrían proporcionar imágenes etiquetadas de señales en categorías tales como ‘stop’ y ‘ceda el paso’. Después de una cantidad suficiente de entrenamiento, el sistema debería ser capaz de distinguir y categorizar imágenes sin etiqueta, en cuyo momento se puede decir que el entrenamiento está completo. Para el desarrollo de este trabajo vamos a centrarnos en técnicas de aprendizaje supervisado.

1.2. Objetivos

Nuestro objetivo principal es construir un reconocedor de imágenes de señales de tráfico que sea capaz de funcionar con la mayor precisión que podamos alcanzar.

Para ello vamos a tener que aprender a desarrollar un proyecto completo de *machine learning*, que incluye el preprocessado de los datos, aprender a manejar herramientas específicas e implementar en código distintas técnicas y algoritmos de *machine learning* y *deep learning*, tanto clásicas como modernas, además de diseñar experimentos para compararlas entre ellas de forma efectiva, con el objetivo de encontrar el método que con más precisión clasifique las imágenes.

1.3. Estado del arte

1.3.1. *Machine learning* y *Deep learning*

El aprendizaje automático o *machine learning*, es una rama de la inteligencia artificial cuyo objetivo es desarrollar técnicas que sean capaces de generalizar comportamientos de manera automática. En vez de ser explícitamente programado, los algoritmos de *machine learning* aprenden a partir de una gran cantidad de ejemplos relativos a una tarea concreta y encuentran una estructura estadística que permite al sistema ‘descubrir’ reglas que le permiten automatizar la tarea en cuestión.

Para poder utilizar algoritmos de *machine learning* hacen falta tres cosas: datos de entrada, ejemplos de la salida esperada y una manera de medir si el algoritmo está funcionando como se espera.

Un algoritmo de *machine learning* transforma las entradas en salidas útiles, un proceso que como ya se ha explicado es aprendido por el método de exposición a múltiples pares de entrada/salida.

Los algoritmos de *deep learning* son un subconjunto de los de *machine learning*. Llamamos *deep learning* a una nueva forma de aprender de los datos de entrada, que pone énfasis en aprender por ‘capas’ de distintos niveles de abstracción. Llamamos ‘profundidad’ de un modelo al número de capas de representación que utiliza. Los modelos actuales pueden tener decenas y hasta centenares de capas de representación de los datos, y cada una de ellas aprende automáticamente por exposición a los datos de entrada. Estos modelos por capas se llaman redes neuronales, y en los últimos años no han dejado de crecer en profundidad y complejidad.

En el año 2010, aunque en la comunidad científica las redes neuronales habían caído prácticamente en desuso, un grupo de gente que aun seguía trabajando en ellas empezaron a realizar importantes avances, entre ellos estaban el grupo de Geoffrey Hinton, en la universidad de Toronto, Yoshua Bengio en la Universidad de Montreal, Yann LeCun en la Universidad de Nueva York e IDSIA en Suiza.

El momento de inflexión llegó en 2012, cuando el equipo de Hinton entró a participar en el *Large-scale image-classification challenge ImageNet*, una competición de visión artificial que consistía en clasificar imágenes en alta resolución en 1000 categorías diferentes, con un conjunto de entrenamiento de mas de un millón de imágenes. En la época era una tarea que entrañaba una elevada dificultad, en 2011 la mejor solución se basaba en técnicas clásicas de visión artificial y tenia una precisión de 74.3 %, medida que se calculaba dando por correcto la predicción del modelo siempre que la clase buscada se encontrara dentro de las 5 primeras clases predecidas por la red (*top-five accuracy*). En 2012 un grupo liderado por Alex Krizhevsky y supervisado por Geoffrey Hinton fue capaz de lograr aumentar esa precisión hasta el 83.6 %, un avance significativo. Desde entonces la competición ha sido dominada por diversas arquitecturas de redes neuronales convolucionales, en 2015 el ganador obtuvo una precisión de 96.4 %. [1].

Desde entonces las redes neuronales convolucionales o CNNs se han convertido en el método predominante para resolver una gran parte de los problemas de visión artificial.

El momento en el que una red neuronal (*machine learning*) pasa a ser una red neuronal profunda (*deep learning*) no está claro, en general se puede decir que a partir de tres o cuatro capas ocultas ya se trata de *deep learning*. En la Figura 1.1 podemos ver la estructura de una red neuronal simple y de otra que ya se considera profunda.

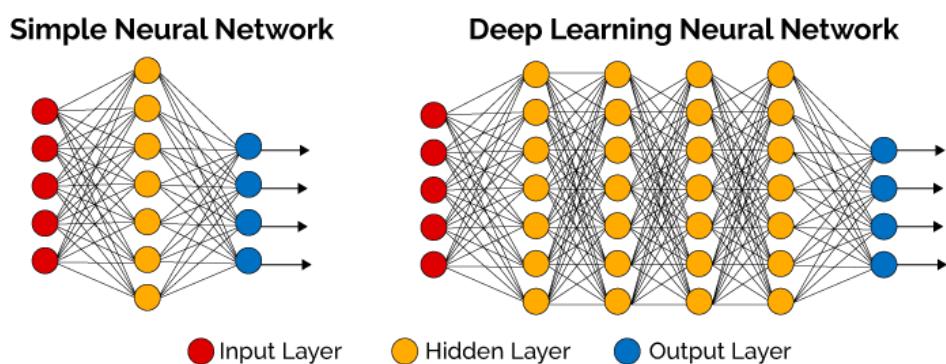


Figura 1.1: Comparación de redes neuronales simples y profundas

Fuente: Anónimo Internet

1.3.2. Reconocimiento de imágenes

Llamamos reconocimiento de imágenes a la tarea de categorizar imágenes de entrada en una de las clases predefinidas. Es un problema fundamental en el campo de la visión por computador y es la base para otras técnicas como la localización, la detección, o la segmentación. Aunque sea algo completamente natural para los humanos, es un problema muy complicado de automatizar.

Es importante resaltar el profundo cambio que han supuesto las técnicas de *deep learning* en el campo de la visión por computador. Tradicionalmente para resolver este tipo de problemas se empleaba una aproximación en dos fases. Primero se extraían de la imagen características (*features*) creadas a mano utilizando descriptores de características (*feature descriptors*) que eran dependientes del problema concreto que se quería resolver. Estas características se utilizaban de entrada para entrenar algún tipo de clasificador. El principal problema era que la precisión del sistema era altamente dependiente de las características escogidas, además al ser dependientes de la tarea en cuestión los algoritmos eran difíciles de reutilizar en tareas parecidas.

En los últimos años los modelos basados en técnicas de *deep learning* han dejado atrás estos desafíos. En particular las redes neuronales convolucionales o CNN han demostrado ser la mejor arquitectura para este tipo de problemas.

Las redes neuronales convolucionales son muy similares a las redes neuronales ordinarias, están formadas por neuronas que tienen pesos y sesgos que se pueden aprender. Cada neurona recibe algunas entradas y realiza un producto escalar al que le sigue una función de activación. Toda la red expresa una única función, desde los píxeles de la imagen sin formato en un extremo hasta el resultado de la función de coste (normalmente *Softmax*) en la última capa totalmente conexa.

Las entradas de una CNN son imágenes, este tipo de redes se aprovecha de la correlación espacial de los píxeles de las imágenes mediante una capa especial llamada, capa convolucional.

Los parámetros de la capa convolucional consisten en un conjunto de filtros que se pueden aprender. Cada filtro es pequeño espacialmente, pero se extiende a través de la

profundidad total del volumen de entrada. Por ejemplo, un filtro típico en una primera capa de una CNN podría tener un tamaño de 5x5x3 (filtro de 5x5 píxeles, y 3 si las imágenes tuvieran 3 canales de color). Durante la propagación hacia adelante o *forward propagation*, deslizamos cada filtro a través del ancho y alto del volumen de entrada y calculamos los productos escalares entre las entradas del filtro y la entrada en cualquier posición. A medida que deslizamos el filtro sobre el ancho y la altura del volumen de entrada produciremos un mapa de activación bidimensional que proporciona las respuestas de ese filtro en cada posición espacial. Intuitivamente, la red aprenderá los filtros que se activan cuando detecta algún tipo de característica visual, como un borde de alguna orientación o una mancha de algún color en la primera capa o, eventualmente, patrones enteros en las capas superiores de la red. Ahora, tendremos un conjunto completo de filtros en cada capa convolucional, y cada uno de ellos producirá un mapa de activación bidimensional por separado. Apilaremos estos mapas de activación a lo largo de la dimensión de profundidad y produciremos el volumen de salida.

Como ya se ha comentado anteriormente, los avances más significativos en el problema de clasificación de imágenes han surgido en el marco del *ImageNet Large Scale Visual Recognition Challenge*, una competición en la que la tarea es clasificar aproximadamente 1,2 millones de imágenes en 1000 clases distintas. Incluyendo cada año mejoras significativas como distintas funciones de activación, mecanismos de regularización, novedosas técnicas de optimización y nuevas formas de organizar la estructura de las CNNs.

Los avances en materia de redes neuronales convolucionales son increíblemente rápidos y cada pocos meses surgen nuevos modelos y técnicas que empujan un poco más allá el estado del arte.

1.3.3. Clasificación de señales de tráfico

El problema concreto que vamos a tratar, la clasificación de imágenes aplicada a las señales de tráfico, también ha sido estudiado con anterioridad. En el artículo de Yann leCun [2] tratan este mismo problema, alcanzando una precisión máxima del 99,17% en el mejor de los casos y con una arquitectura específica. Nuestro objetivo, además de comparar diversas aproximaciones con otras técnicas será acercarnos lo máximo posible a estas cifras.

Capítulo 2

Metodología

2.1. Herramientas utilizadas

Para la realización de este proyecto hemos utilizado distintas herramientas y librerías que detallaremos brevemente a continuación:

- Python 3: Ha sido el lenguaje de programación utilizado. Por su versatilidad, comunidad de desarrolladores, librerías de *machine learning* y recursos online es uno de los lenguajes más utilizados en el desarrollo de soluciones en el ámbito del aprendizaje automático y de la ciencia de datos.
- NumPy: Es una librería científica de Python, que agrega un gran número de funciones matemáticas de alto nivel, especialmente indicada para trabajar con vectores y matrices.
- TensorFlow: Es una librería *open source* para computación numérica de alto rendimiento que permite ser ejecutada tanto en CPU como en GPU. Originalmente creada por Google, tiene un fuerte soporte para *machine learning* y *deep learning*.
- Keras: Es una API de alto nivel que funciona sobre TensorFlow y que simplifica el proceso de trabajar con redes neuronales.
- Scikit-learn: Es una librería *open source* que proporciona herramientas útiles para trabajar en problemas de *machine learning* y análisis de datos.

- Pandas: Es una extensión de NumPy para la manipulación y análisis de datos en Python. Entre otras características ofrece estructuras de datos y operaciones para manipular tablas numéricas.
- TensorBoard: Una aplicación web que contiene una serie de herramientas de visualización que permite ver el grafo de tu modelo, mostrar gráficas sobre varias medidas e incluso mostrar visualizaciones de las capas intermedias de la red. Facilita enormemente la comprensión del funcionamiento del modelo y la búsqueda de errores.

El hardware utilizado para la realización del presente trabajo ha consistido principalmente en un ordenador portátil con un procesador i7 de quinta generación sin tarjeta gráfica dedicada, que ha resultado insuficiente para la ejecución de la mayor parte de los algoritmos ejecutados, llegando a tardar varias horas en entrenar determinados modelos, o incluso no siendo capaz de ejecutarlos.

Por ello para la ejecución de los algoritmos más costosos computacionalmente, se ha utilizado un equipo dedicado del grupo CAPO denominado ‘Corleone’ que cuenta con dos potentes unidades de procesamiento gráfico, una GeForce GTX TITAN X y una GeForce GTX TITAN Black. Para acceder de forma remota se ha utilizado el protocolo SSH y se ha empleado el sistema de control de versiones Git para actualizar el repositorio de trabajo que se desarrollaba en local. Sin acceso a estos recursos, hubiera sido prácticamente imposible la realización de este trabajo sin acudir a soluciones en la nube como AWS, con el coste económico asociado que eso hubiera tenido.

2.2. El conjunto de datos

El *dataset* o conjunto de datos utilizado para entrenar los modelos que se incluyen en este trabajo es el *German Traffic Sign Dataset* abreviado GTSRB. Es un conjunto de datos perfecto para nuestro objetivo de clasificación multiclas. Es importante conocer bien el *dataset* con el que vamos a trabajar, para comprenderlo correctamente y poder tomar las decisiones adecuadas.

Contiene 43 clases distintas de señales de tráfico, y un total de 51839 imágenes,

divididas en aproximadamente un 75 % para el conjunto de entrenamiento y un 25 % para el de test [3].



Figura 2.1: Imágenes de muestra del conjunto de datos

Fuente: [3]

El *dataset* se estructura en un directorio o carpeta para cada clase, cada directorio contiene un archivo CSV con anotaciones que incluyen el nombre del fichero y la clase a la que pertenece además de las imágenes de entrenamiento.

Las imágenes están agrupadas en conjuntos secuenciales, cada uno de ellos contiene treinta imágenes tomadas de la misma señal de tráfico real desde posiciones ligeramente distintas, además las imágenes también varían en tamaño. Cada una de las señales reales solo está presente en una de estas secuencias del *dataset*, esto quiere decir que la misma señal no aparece dos veces en el *dataset* en secuencias distintas, esto no impide que aparezcan varias señales pertenecientes a la misma clase. En la Figura 2.2 se puede observar con mayor claridad a qué nos referimos con ‘secuencia’. Pero aunque las señales están disponibles como una serie, el conjunto de test no tiene información temporal, por lo que no se tendrá en cuenta.

Cada imagen contiene una única señal de tráfico con un borde de un 10 % alrededor de la señal de al menos 5 píxeles, esto es así para permitir métodos basados en la detección de bordes. Las imágenes están en formato PPM y su tamaño varía entre 15x15 y 250x250 píxeles representadas en un rango de [0,255] en el espacio de color RGB. No se garantiza que las imágenes sean completamente cuadradas ni que la señal esté perfectamente centrada.

Comprobamos que el *dataset* no está balanceado correctamente, como podemos ver en la Figura 2.3, el número de imágenes varía bastante por cada clase, hay clases mucho mejor representadas que otras.



Figura 2.2: Secuencia de imágenes de la misma señal

Fuente: [4]

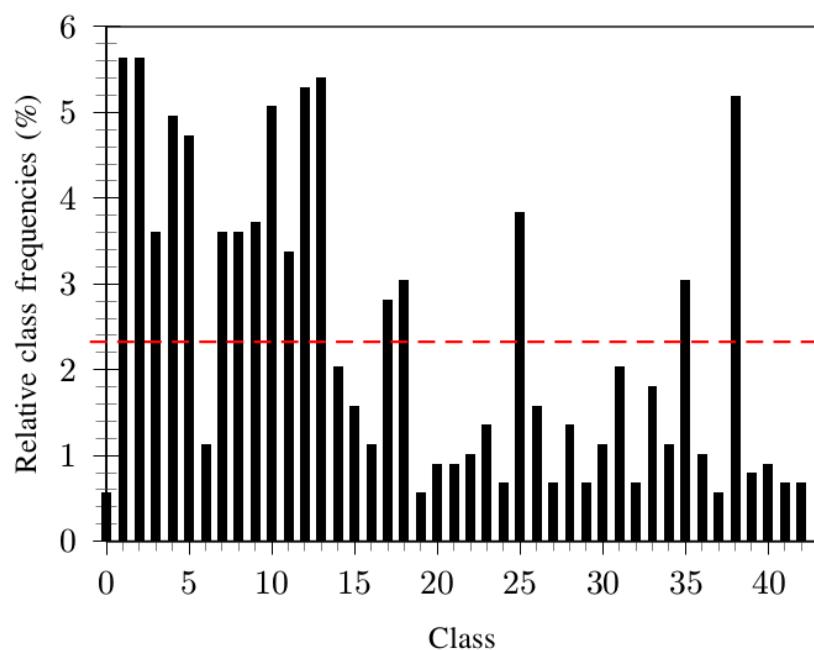


Figura 2.3: Número de imágenes de entrenamiento por clase

Fuente: [4]

2.3. Preprocesado

El preprocesado del conjunto de entrada tiene una gran importancia en el rendimiento posterior de los modelos de *machine learning* propuestos.

Las imágenes varían mucho en el tipo de iluminación y en el tamaño, por lo que en el preprocesado vamos a cambiar el espacio de color a HSV, que separa la intensidad de la imagen de la información de color y también vamos a utilizar una técnica llamada ecualización de histograma para ajustar el contraste de las imágenes, lo hacemos en canal V, que es el canal que contiene la información de brillo del espacio de color HSV. De esta forma aumentamos la solidez de nuestro sistema a los cambios de iluminación y reducimos el impacto de las sombras.

También redimensionamos las imágenes a un tamaño estándar de 48x48 o de 32x32 dependiendo del modelo utilizado, reescalamos los valores del espacio HSV y pasan a estar entre los valores 0 y 1 en vez de entre 0 y 255, además recortamos el cuadrado central para deshacernos del borde, ya que los métodos que se van a emplear no hacen uso de la detección de bordes y al acercar la señal facilitamos el reconocimiento de esta. Para ilustrar mejor los efectos, a continuación en la Figura 2.4 se muestran dos imágenes de señales de tráfico, a la izquierda, tal como aparece en el *dataset* y a la derecha después de aplicarle el preprocesado. Como vemos se aprecia una clara mejoría, la imagen preprocesada se distingue más claramente y está menos oscura que la original, además de tener menos borde, que no aporta nada a nuestro objetivo.

Aunque no es estrictamente parte del preprocesado, al cargar las imágenes las barajamos para que los lotes no estén correlacionados y el orden no influya en el entrenamiento.

2.4. Validación Cruzada

En los diferentes algoritmos que utilizaremos para clasificar señales de tráfico, vamos a utilizar una técnica conocida como ‘validación cruzada’ que divide el conjunto de entrenamiento en ‘ k ’ pliegues y entrena el modelo utilizando todos los pliegues menos

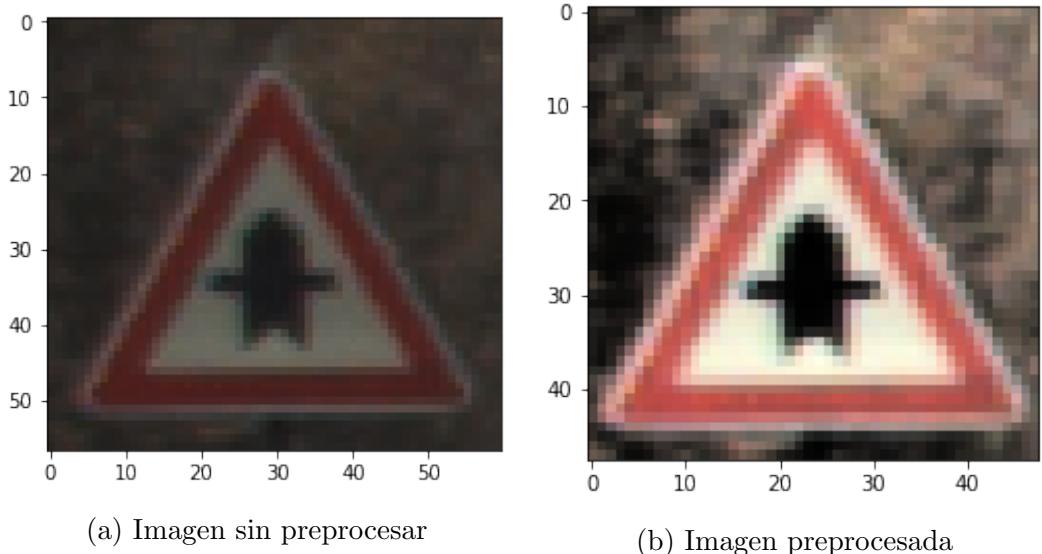


Figura 2.4: Imagen antes y después de aplicar el preprocessado

Fuente: Original

uno que se utilizará como conjunto de validación, a continuación se repite el proceso seleccionando otro de los pliegues como conjunto de validación y utilizando el resto para entrenar, se puede ver más claramente en la Figura 2.5. Esto se repite hasta que se haya entrenado k veces, cada una de ellas con un conjunto de entrenamiento ligeramente diferente y un conjunto de validación distinto. Esto nos dará ' k ' medidas distintas de precisión para cada uno de los modelos, escogeremos el modelo que tenga una precisión más cercana a la media, puesto que será el que mejor generalice en el conjunto de test [5].

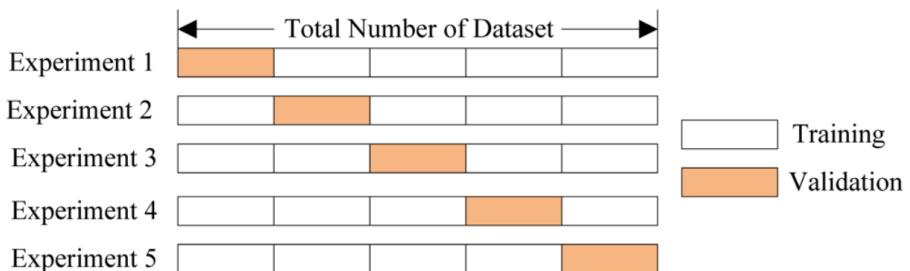


Figura 2.5: Validación cruzada

Fuente: Anónimo Internet

En concreto vamos a utilizar una versión estratificada que incluye la librería *scikit learn*, que reordena los datos para asegurarse de que cada pliegue es representativo del conjunto y está balanceado entre clases. Hacemos esto porque nuestro *dataset* tiene sobre-representación de algunas clases y si se escogieran de forma aleatoria podría dejar

clases sin representar.

2.5. Algoritmos

En la siguiente sección vamos a utilizar distintos métodos de clasificación para comparar los resultados de cada uno de ellos. Los clasificadores utilizados pueden dividirse en tres categorías, la primera corresponde a métodos clásicos de *machine learning*, en concreto *Random Forest* y una red neuronal clásica. El segundo tipo de algoritmos que utilizaremos son redes neuronales convolucionales que constituyen el estado del arte actualmente, estas arquitecturas son: *VGG16/VGG19*, *Resnet50* y *Exception*. Utilizaremos estas redes preentrenadas sobre *Imagenet* y aplicaremos *Transfer Learning* para poder utilizarlas en nuestro problema. Por último utilizaremos cuatro arquitecturas de redes neuronales convolucionales que entrenaremos desde cero.

Capítulo 3

Comparación de Algoritmos

Los algoritmos utilizados se dividen en tres tipos: Métodos clásicos, *Transfer Learning* y CNNs Personalizadas. Vamos a proceder a comentar los distintos clasificadores que utilizaremos y a ejecutarlos varias veces modificando los valores de los hiperparámetros para conseguir la mejor configuración posible de cada algoritmo.

3.1. Métodos clásicos

3.1.1. Random forest

El primer clasificador que utilizaremos es un *random forest*, también conocido en español como Bosques Aleatorios. Este algoritmo es un meta estimador de *bagging* que entrena un gran número de árboles de decisión no correlacionados, cada uno de ellos en un subconjunto del *dataset* inicial de forma aleatoria, y promedia los resultados para mejorar el poder predictivo y evitar el sobreajuste. La idea principal de los algoritmos de *bagging* es promediar muchos modelos con mucho ruido pero imparciales [6]. Los árboles se ajustan perfectamente a este tipo de algoritmos. En la Figura 3.1 podemos ver un ejemplo para el caso más sencillo de *random forest*, en el que solo promediamos dos árboles.

Empleamos la implementación de la librería de *machine learning scikit-learn* con

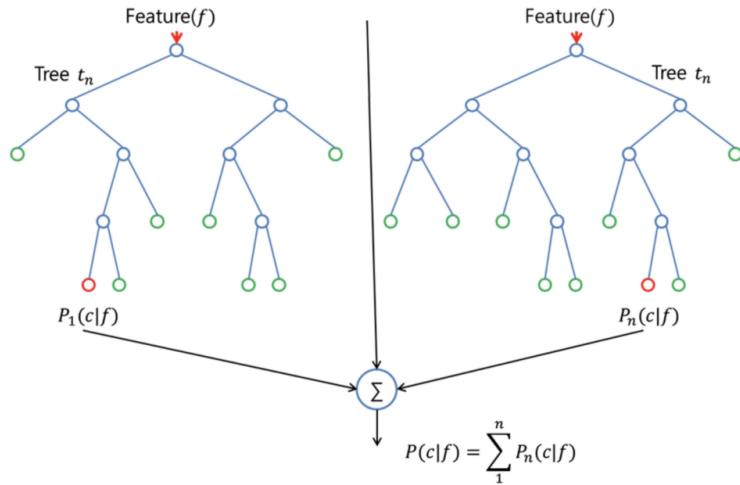


Figura 3.1: Representación de un *random forest* de 2 árboles

Fuente: [6]

varias configuraciones en la cantidad de árboles y la profundidad de estos, si utilizamos más de 500 árboles empezamos a tener problemas de rendimiento y el entrenamiento se interrumpe por fallos de memoria. A continuación, en la Figura 3.2 mostramos los resultados obtenidos.

Con este algoritmo, utilizando 500 árboles y una profundidad de 22, obtenemos una precisión en el conjunto de validación de 96,66 % y en el conjunto de test llegamos a una precisión del 85,03 %, un resultado bastante bueno para un algoritmo relativamente sencillo como este.

3.1.2. Red neuronal

Una red neuronal se compone de un conjunto de nodos conectados entre sí llamados neuronas artificiales o perceptrones, cada conexión entre los nodos puede transmitir una señal de una a otra, la neurona que recibe la señal la procesa y puede transmitir a su vez una señal a otras neuronas conectadas a ella. Las neuronas tienen unos pesos que se van ajustando conforme la red va aprendiendo. Los pesos van incrementando o disminuyendo la fuerza de la señal que se propaga de una neurona a otra, estas tienen una función de activación que determina si la señal se transmite a la siguiente

Resultados utilizando Random Forest		
Modelo	Precisión media en validación	Precisión en Test
Random Forest (100 arboles y profundidad 3)	37,28%	36,00%
Random Forest (300 arboles y profundidad 3)	37,99%	35,60%
Random Forest (100 arboles y profundidad 5)	53.88%	51.10%
Random Forest (300 arboles y profundidad 5)	54.40%	51.08%
Random Forest (100 arboles y profundidad 9)	79.09%	71.82%
Random Forest (300 arboles y profundidad 9)	81.10%	72.57%
Random Forest (100 arboles y profundidad 12)	89.54%	78.87%
Random Forest (300 arboles y profundidad 12)	90,10%	80.21%
Random Forest (300 arboles y profundidad 14)	93.41%	82.31%
Random Forest (100 arboles y profundidad 16)	94.39%	82.44%
Random Forest (100 arboles y profundidad 19)	95.36%	83.61%
Random Forest (300 arboles y profundidad 19)	96.07%	84.25%
Random Forest (100 arboles y profundidad 22)	95.68%	83.47%
Random Forest (500 arboles y profundidad 22)	96.66%	85.03%

Figura 3.2: Resultados obtenidos con el algoritmo *random forest*

Fuente: Original

neurona o no. Normalmente las neuronas están organizadas en capas, las señales viajan desde la capa de entrada a través de las capas ocultas hasta la capa de salida. Esto se conoce como *feedforward propagation*, pero el aprendizaje no se detiene ahí, las redes neuronales implementan lo que se conoce como *backpropagation*, este algoritmo calcula el gradiente de la función de coste con respecto a los pesos de la red y permite actualizar los pesos en cada pasada. De esta manera la red neuronal va aprendiendo [1].

Por ejemplo, en la Figura 3.3, la primera capa de perceptrones hace decisiones sencillas basadas en la entrada y el peso que tenga cada neurona, en la segunda capa las neuronas están tomando decisiones basadas en las decisiones tomadas previamente por la primera capa, esto es, está tomando decisiones a un nivel más complejo y más

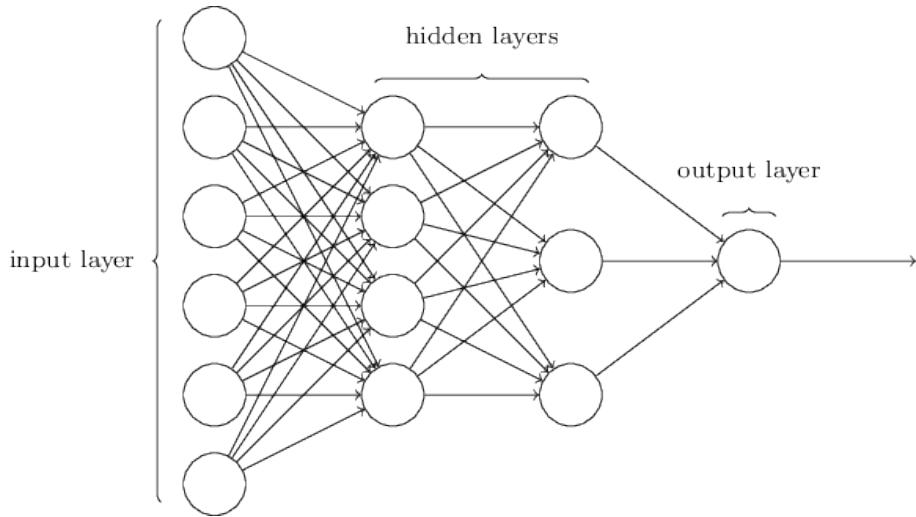


Figura 3.3: Esquema de una red neuronal

Fuente: [7]

abstracto que la primera capa de neuronas. De esta forma una red con muchas capas, va tomando decisiones cada vez más sofisticadas.

Se han implementado dos modelos en *Keras*, el primero con tres capas ocultas y el segundo con cuatro. Los dos tienen una capa de entrada de 1024 neuronas, entre cada capa se utilizan funciones de activación ReLU y en la capa de salida una función *softmax* que da una probabilidad sobre las cuarenta y tres clases de nuestro *dataset*. Como optimizador utilizamos descenso de gradiente estocástico (SGD) con momento y un *learning rate* que va decreciendo según avanza el entrenamiento. En la Figura 3.4 se muestran las capas intermedias utilizadas.

$$1024 \rightarrow 2048 \rightarrow 512 \rightarrow 128 \rightarrow 43$$

input h1 h2 h3 output

$$1024 \rightarrow 2048 \rightarrow 1024 \rightarrow 512 \rightarrow 128 \rightarrow 43$$

input h1 h2 h3 h4 output

Figura 3.4: Estructura de las redes neuronales utilizadas

Fuente: Original

Probamos cada una de las dos configuraciones detalladas anteriormente dos veces, una con imágenes de entrada de 32x32 y la otra de 48x48. Entrenamos la red durante 20 épocas. Los resultados se muestran en la Figura 3.5.

Resultados utilizando redes neuronales		
Modelo	Precisión media en validación	Precisión en Test
Neural Network 3H capas, 20 épocas, 32x32	97.03%	87.66%
Neural Network 3H capas, 20 épocas, 48x48	96.95%	87.96%
Neural Network 4H capas, 20 épocas, 32x32	97.60%	88.55%
Neural Network 4H capas, 20 épocas, 48x48	96.54%	88.02%

Figura 3.5: Resultados obtenidos con las redes neuronales de 3 y 4 capas ocultas

Fuente: Original

Con esta aproximación basada en redes neuronales clásicas alcanzamos una precisión en el conjunto de validación del 97,60 % y en el conjunto de test una precisión de 88,55 % con el modelo con cuatro capas ocultas e imágenes de 32x32.

3.2. Transfer learning

Una forma muy común y bastante efectiva de resolver un problema de *deep learning*, especialmente cuando nuestro *dataset* no es muy grande, es utilizar una red preentrenada. Una red preentrenada es una red guardada que ha sido previamente entrenada en un *dataset* de gran tamaño, que sea lo suficientemente grande y general que las características aprendidas por éste se puedan utilizar como un modelo genérico del mundo visual, y puedan utilizarse en una gran variedad de problemas de visión artificial. *Imagenet* es un *dataset* que con mas de un millón de imágenes y mil clases distintas que cumple estas características. Se pueden encontrar multitud de modelos en Internet preentrenados en *Imagenet*, nosotros vamos a utilizar *VGG16*, *VGG19*, *Resnet50* y *Xception* que están disponibles directamente en *Keras*. Para utilizar las características aprendidas por estos modelos en nuestro problema vamos a utilizar una técnica llamada *feature extraction*. Este método consiste en utilizar las representaciones aprendidas por una red para extraer características que puedan ser útiles en nuevos ejemplos. Estas características son entonces entrenadas en un nuevo clasificador.

En el caso de las redes neuronales convolucionales, la extracción de características consiste en coger la base convolucional de una red preentrenada, pasar nuestros nuevos datos de entrenamiento por ella y reentrenar un nuevo clasificador en la salida. Sólo cogemos las representaciones aprendidas en la base convolucional del modelo, porque las características son más genéricas y por lo tanto más fáciles de reutilizar [8]. Al utilizar estos modelos preentrenados no preprocesaremos las imágenes como hasta ahora, ya que cada red ha sido entrenada con imágenes preprocesadas de forma distinta, por lo que lo hacemos de la misma forma que en el entrenamiento original, por suerte *Keras* nos ofrece una función `preprocess_input()` para cada una de los modelos que utilizaremos.

3.2.1. VGG16/VGG19

El artículo que describe esta arquitectura [9] es de 2014, y obtuvo un error de 7,3% sobre *ImageNet* en el ILSVRC ese año, puede parecer reciente pero el campo ha avanzado mucho en los últimos años. La idea principal de esta arquitectura es que no hacían falta estructuras extrañas y complicadas para obtener una alta tasa de precisión, con solo una red neuronal convolucional ‘clásica’ con un número considerable de convoluciones de 3x3 y funciones de activación no lineales (ReLU) se podían alcanzar resultados sin precedentes hasta la fecha. Aunque esto no quiere decir que no incluyera novedades relevantes, como el uso de filtros con un tamaño de 3x3 en vez de 11x11 como AlexNet, otra popular arquitectura de redes convolucionales.

Una de las aportaciones de la arquitectura VGG es que demostraron que dos filtros convolucionales de 3x3 apilados uno a continuación de otro, tenían la misma capacidad de percibir un campo de visión que un filtro de 5x5, y que tres filtros sucesivos de 3x3 eran equivalentes a uno de 7x7. Esto permitía simular filtros más grandes con la ventaja computacional que los filtros más pequeños ofrecen (menos parámetros), además utilizar funciones ReLU entre cada filtro convolucional hace que la función de decisión sea más discriminativa. La idea de la profundidad también se vio reforzada, al añadir más capas cada vez con menos características, gracias a las capas de *pooling*, la profundidad del modelo crecía, reforzando la idea de reducir las dimensiones espaciales pero aumentar la profundidad. Por estos avances VGGNet es una de las arquitecturas más influyentes en el campo de la clasificación de imágenes y del aprendizaje profundo.

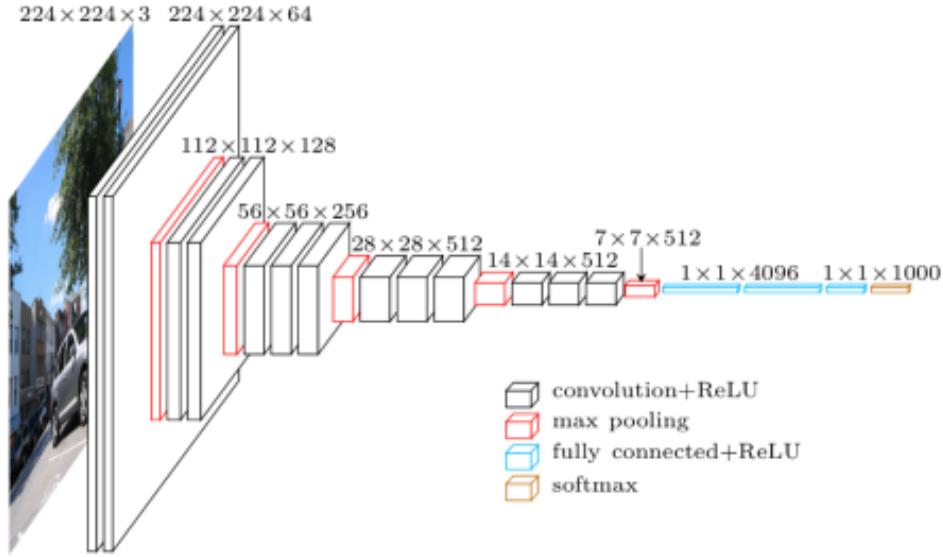


Figura 3.6: Arquitectura de la red VGG16

Fuente: [10]

Haciendo *feature extraction* en *VGG16* obtenemos una precisión del 96 % en el conjunto de validación y un 81 % en el conjunto de test y en *VGG19* una precisión del 96 % en el conjunto de validación y un 79 % en el conjunto de test como se puede ver en la Figura 3.7.

Resultados utilizando VGG16 y VGG19		
Modelo	Precisión media en validación	Precisión en Test
VGG16 (FE) + LogisticClassifier	96,00%	81,00%
VGG19 (FE) + LogisticClassifier	96,00%	79,00%

Figura 3.7: Resultados al hacer *transfer learning* con *VGG16* y *VGG19*

Fuente: Original

3.2.2. Resnet50

Desde la publicación del artículo [11] en 2015, las arquitecturas *resnet* han conseguido importantes mejoras en la precisión de diversas tareas relacionadas con la visión artificial. Esta arquitectura fue la primera en sobrepasar el nivel humano en el *dataset* de *ImageNet*, y las conexiones residuales son utilizadas en un número importante de arquitecturas actuales.

Por lo visto hasta ahora en la evolución de las redes convolucionales profundas, podría parecer que incrementar la profundidad de la red da lugar inevitablemente a mejores resultados, siempre que se mantenga controlado el sobreajuste, pero esto no es así. Al aumentar la profundidad de una red aparece el llamado ‘desvanecimiento de gradiente’, que consiste en que la señal que cambia los pesos de la red se vuelve muy pequeña por la profundidad, y cuando llega a las primeras capas casi ha desaparecido y la red no aprende correctamente. Este problema se contrarresta utilizando funciones de activación ReLU y añadiendo capas de normalización intermedias. Además de esto, cuando las redes más profundas empiezan a converger, aparece un problema de degradación: a medida que la profundidad de la red aumenta, la precisión se satura y luego se degrada rápidamente. Inesperadamente, tal degradación no es producto del sobreajuste, y agregar más capas a un modelo suficientemente profundo conduce a un mayor error durante el entrenamiento. Otro problema de incrementar la profundidad es que se incrementa el número de parámetros a optimizar, haciendo más difícil el entrenamiento.

Las redes residuales permiten el entrenamiento de redes muy profundas utilizando bloques residuales, que son micro arquitecturas modulares que se saltan conexiones. La idea es que las capas profundas tengan acceso directo a las características de capas anteriores, esto hace que la información se propague más fácilmente por la red y ayuda durante el entrenamiento a que los gradientes se propaguen hacia atrás de forma más eficiente.

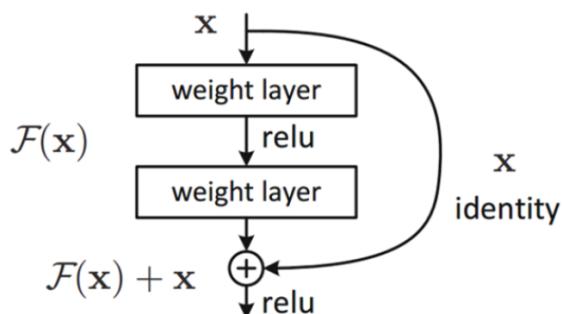


Figura 3.8: Bloque residual de *ResNet50*

Fuente: [12]

Haciendo *feature extraction* obtenemos una precisión del 68 % en el conjunto de validación y un 58 % en el conjunto de test, Figura 3.9. Un resultado especialmente deficiente.

Resultados utilizando ResNet50		
Modelo	Precisión media en validación	Precisión en Test
Resnet50 (FE) + LogisticClassifier	68,00%	58,00%

Figura 3.9: Resultados al hacer *transfer learning* con *ResNet50*

Fuente: Original

3.2.3. Xception

El nombre de *Xception* viene de *Xtreme inception* y como su nombre indica lleva las ideas y principios de la arquitectura *inceptionV3* de *Google* al extremo, haciendo uso también de conexiones residuales y de elementos clásicos propios de arquitecturas VGG.

La idea principal detrás de la arquitectura *Xception* es que la correlación de los canales y la correlación espacial no están lo suficientemente conectadas para que sea preferible mapearlas conjuntamente. Esto da lugar a la utilización de *depthwise separable convolutions* que consisten en convoluciones espaciales clásicas realizadas independientemente sobre cada canal de entrada seguidas de convoluciones 1x1 llamadas *pointwise convolutions*, proyectando los canales de salida en un nuevo canal espacial [13].

Primero buscamos correlaciones en el espacio bidimensional de las imágenes y a continuación en el espacio unidimensional. De forma intuitiva este mapeo 2D + 1D es mas fácil de aprender que un mapeo completo en tres dimensiones.

En resumen la arquitectura *Xception* apila 36 capas de *depthwise separable convolutions* con conexiones residuales.

Haciendo *feature extraction* obtenemos una precisión del 96 % en el conjunto de validación y un 78 % en el conjunto de test, podemos ver los resultados en la Figura 3.11.

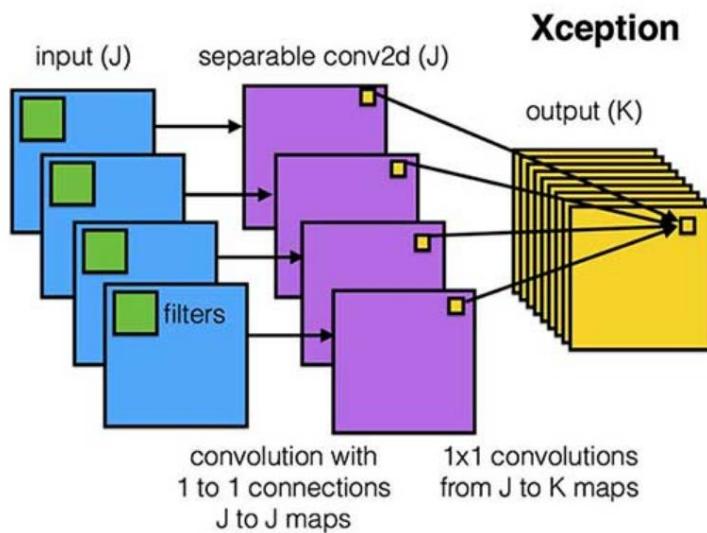


Figura 3.10: Esquema de una *pointwise separable convolution*

Fuente: [14]

Resultados utilizando Xception		
Modelo	Precisión media en validación	Precisión en Test
Xception (FE) + LogisticClassifier	96,00%	78,00%

Figura 3.11: Resultados al hacer *transfer learning* con *Xception*

Fuente: Original

3.3. CNNs Personalizadas

Viendo el éxito relativo obtenido utilizando modelos preentrenados, se opta por utilizar distintas arquitecturas de redes neuronales convolucionales especialmente diseñadas para nuestro problema, adaptadas a las necesidades concretas del reconocimiento de señales, teniendo en cuenta también las características del *dataset* con el que estamos tratando. Para el diseño de las diferentes redes utilizaremos la librería de *deep learning* *Keras*.

Una red neuronal convolucional es muy similar a una red neuronal clásica, pero incluye un tipo de operación especial llamada convolución que es especialmente efectiva para trabajar con imágenes. Para dar con el mejor modelo iremos probando diferentes configuraciones para compararlas entre ellas y quedarnos con la mejor

posible, intentando probar solo cambios significativos para evitar perdernos en la infinidad de modelos derivados que se pueden obtener modificando ligeramente los hiperparámetros y haciendo pequeños cambios en la configuración de la red.

3.3.1. Simple CNN

Para empezar utilizamos una arquitectura clásica, con 3 etapas convolucionales, consistentes en una capa convolucional con un *kernel* de 3x3 seguida de una capa de *pooling* y por último una de *dropout* después de cada capa de *pooling*. Empleamos funciones de activación de tipo ReLU, que se ha convertido en el estándar en los últimos tiempos. Después de la base convolucional del modelo, contamos con una capa densamente conexa, y otra que actúa como clasificador con 43 neuronas y una función *softmax*. Para la optimización utilizamos descenso de gradiente estocástico, SGD por sus siglas en inglés, con un *learning rate* que va decayendo según va aprendiendo el modelo, también probamos con RMSprop. Entrenamos el modelo durante 20 épocas, esto quiere decir que nuestra red recibe el conjunto de entrenamiento 20 veces.

Con esta configuración probamos el modelo al que vamos a llamar *Simple CNN*, con imágenes de entrada de un tamaño de 32x32x3 y 48x48x3, los resultados se muestran en la Figura (3.12).

Resultados utilizando Simple CNN		
Modelo	Precisión media en validación	Precisión en Test
Simple CNN (img 32x32), dropout (0.2, 0.3, 0.4, 0.5), 20 épocas	99.36%	96.04%
Simple CNN (img 48x48) dropout (0.2, 0.3, 0.4, 0.5), 20 épocas	99.58%	96.89%
Simple CNN (img 32x32) dropout (0.2, 0.3, 0.4, 0.5), 20 épocas, rmsprop	97.95%	93.86%
Simple CNN (img 48x48) dropout (0.2, 0.3, 0.4, 0.5), 20 épocas, rmsprop	99.23%	95.61%
Simple CNN (img 32x32), dropout (0.5), 20 épocas	99.19%	95.22%
Simple CNN (img 48x48) dropout (0.5), 20 épocas	99.36%	95.61%

Figura 3.12: Resultados de la arquitectura *Simple Cnn*

Fuente: Original

3.3.2. Skip CoNet

A continuación vamos a basarnos en el concepto de *skip connections* de arquitecturas modernas como *ResNet* así como algunas ideas sugeridas en el artículo [2]. En el artículo se propone la implementación de una red convolucional que a diferencia de modelos tradicionales en los que la salida de una capa alimenta únicamente a la siguiente, la salida de la primera etapa convolucional se divide en dos y alimenta tanto a la siguiente capa como directamente al clasificador que computa la salida, que a su vez también se alimenta de la salida de la segunda etapa convolucional. En la Figura 3.13 podemos ver más claramente el modelo de red propuesto.

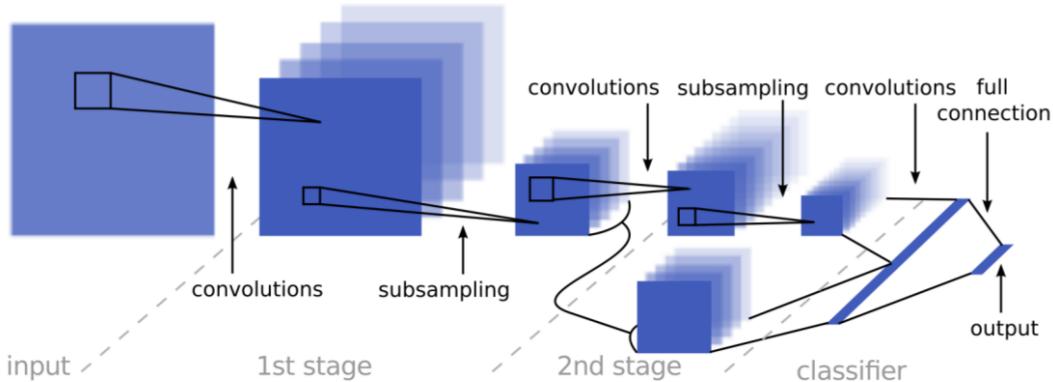


Figura 3.13: Arquitectura de la red Skip CoNet

Fuente: [2]

Estas conexiones no secuenciales que se saltan capas, tienen la ventaja de que ofrecen al clasificador la combinación de representaciones de distinto nivel de abstracción, la primera capa ofrece representaciones de detalles locales como esquinas y bordes, mientras que de la segunda se obtienen formas y estructuras más globales. La idea detrás de este tipo de arquitecturas es que combinar distintos tipos de escala de representación crea un clasificador más robusto.

La implementación concreta de la arquitectura que utilizamos consta de 2 etapas convolucionales, la primera de ellas cuenta con una capa convolucional con un *kernel* de 5x5 y una profundidad de 108, seguida de una capa de *max pooling* con un *kernel* de 2x2 con una capa de *batch normalization*, la salida de esta etapa se utiliza como entrada

de la segunda etapa, que consta de una capa convolucional con un *kernel* de 3x3 con una profundidad de 200 y una capa de *batch normalization*. En la tercera y última etapa se toman como entrada la salida de la primera y de la segunda y se concatenan para alimentar una capa densamente conexa de 300 neuronas ocultas a las que les sigue otra de 43 neuronas que actúa de clasificador con una función de activación *softmax*. Probamos varias versiones de la red con imágenes de entrada con un tamaño de 32x32 pixeles y 48x48 pixeles en tres canales. En la primera versión modificada añadimos una segunda capa de *pooling* después de la primera para reducir la dimensionalidad de las características de la primera etapa convolucional. En la segunda versión añadimos capas de *dropout* de 0,3 después de las etapas convolucionales, y una de 0,5 justo antes del clasificador. En la última versión que probamos incluimos los dos cambios anteriores y modificamos la etapa de clasificación añadiéndole dos capas con más neuronas justo antes de la capa de *softmax*, en concreto dos capas densamente conexas de 2048 y 512 neuronas respectivamente.

Resultados utilizando Skip CoNet		
Modelo	Precisión media en validación	Precisión en Test
Skip conect 32, 40 épocas, SIN dropout	99.40%	95.08%
Skip conect 48, 20 épocas, Sin dropout	99.32%	95.74%
Skip conect 32, 20 épocas, SIN dropout, otro MaxPool (2,2) en la primera etapa.	99.55%	94.95%
Skip conect 48, 20 épocas, SIN dropout, otro MaxPool (2,2) en la primera etapa.	99.47%	95.11%
Skip conect 32, 20 épocas, Dropout (0,3, 0,3, 0,5)	99.36%	95.91%
Skip conect 48, 20 épocas, Dropout (0,3, 0,3, 0,5)	99.24%	95.74%
Skip conect 32, 20 épocas, Dropout (0,3, 0,3, 0,5), otro MaxPool (2,2) en la primera etapa. Clasificador en 2 etapas(fc 2048 + fc 512)	99.41%	94.84%
Skip conect 48, 20 épocas, Dropout (0,3, 0,3, 0,5), otro MaxPool (2,2) en la primera etapa. Clasificador en 2 etapas(fc 2048 + fc 512)	99.55%	95.51%

Figura 3.14: Resultados al utilizar la arquitectura *Skip CoNet*

Fuente: Original

Como vemos en la Figura 3.14, obtenemos valores de precisión en el conjunto de test cercanos al 95 % en todas las variantes del modelo. Siendo la que mejor precisión alcanza el modelo al que añadimos *dropout*, entrenado con imágenes de 32x32, que

alcanza un 95,91 %, quedando algo por debajo del modelo anterior.

3.3.3. Skip CoNet - Simple CNN

No hemos conseguido superar en precisión a nuestra primera red neuronal *Simple CNN* por lo que vamos a probar con la arquitectura original añadiéndole conexiones no secuenciales como en la red *Skip CoNet*. Vamos a probar tres versiones de la arquitectura, en la primera solo conectaremos la salida de la primera etapa convolucional a la entrada del clasificador, en la segunda conectaremos solo la salida de la segunda etapa convolucinal y por último conectaremos la salida de la primera y la segunda al mismo tiempo.

Resultados utilizando Skip CoNet - Simple CNN		
Modelo	Precisión media en validación	Precisión en Test
Skip CoNet - Simple CNN, (img 48x48),20 épocas, con 1 residual shortcut en la primera stage, No dropout	98.25\%	92.12\%
Skip CoNet - Simple CNN, (img 48x48),20 épocas, con 1 residual shortcut en la primera stage, dropout (0.3, 0.4, 0.4, 0.5)	98.22\%	91.71\%
Skip CoNet - Simple CNN, (img 32x32),30 épocas, con 1 residual shortcut en la segunda stage, dropout (0.3, 0.4, 0.4, 0.5)	99.19\%	94.83\%
Skip CoNet - Simple CNN, (img 32x32),30 épocas, con 1 residual shortcut en la segunda stage, dropout (0.3, 0.4, 0.4, 0.5), GlogalAveragePooling2D	95.51\%	90.77\%
Skip CoNet - Simple CNN, (img 32x32),30 épocas, con 1 residual shortcut en la segunda stage, dropout (0.3, 0.4, 0.4, 0.5), SeparableConv2D	97.68\%	88.05\%
Skip CoNet - Simple CNN, (img 32x32),30 épocas, con 1 residual shortcut en la segunda stage, dropout (0.3, 0.4, 0.4, 0.5), 2 SeparableConv2D iguales por etapa	98.94\%	95.17\%
Skip CoNet - Simple CNN, (img 32x32),50 épocas, con 1 residual shortcut en la segunda stage, dropout (0.3, 0.4, 0.4, 0.5), 2 SeparableConv2D iguales por etapa	98.81\%	95.39\%
Skip CoNet - Simple CNN, (img 48x48), con 2 residual shortcuts, dropout (0.3, 0.4, 0.4, 0.5), 50 épocas	98.81\%	93.51\%
Skip CoNet - Simple CNN, (img 48x48) CON data augmentation, 100 épocas, con 2 residual shortcuts, AUN mas dropout (0.5 en todas)	95.95\%	86.56\%
Skip CoNet - Simple CNN, (img 48x48) CON data augmentation, 30 épocas, con 2 residual shortcuts, AUN mas dropout (0.5 en todas)	95.86\%	86.01\%
Skip CoNet - Simple CNN, (img 48x48) CON data augmentation, 30 épocas, con 2 residual shortcuts ,AUN mas dropout (0.5 en todas),FinalDense (1024)	98.25\%	90.89\%
Skip CoNet - Simple CNN, (img 32x32),30 épocas, con 1 residual shortcut, dropout (0.3, 0.4, 0.4, 0.5), FinalDense (1024)	98.36\%	91.73\%

Figura 3.15: Resultados al utilizar la arquitectura *Skip CoNet - Simple CNN*

Fuente: Original

De nuevo no superamos la precisión obtenida por a primera red, la máxima precisión obtenida por el modelo *Skip CoNet - Simple CNN* es de 98.81 % en validación y 95,39 % en test.

3.3.4. Mini VGG

La última red convolucional que vamos a probar está inspirada en la arquitectura VGG, pero adaptada al menor tamaño de nuestras imágenes, de ella sacamos la idea de apilar varias capas convolucionales antes de la capa de *pooling*. Ponemos una comparativa con la arquitectura *VGG16* para ver que a pesar del tamaño siguen una estructura similar. Un cambio introducido es la inclusión de capas de *dropout* para evitar el sobreajuste ya que nuestro *dataset* es relativamente pequeño.

Mini_VGG	VGG16
Conv2(32, 3x3) Conv2(32, 3x3) Dropout(0.2)	Conv2(64, 3x3) Conv2(64, 3x3)
Maxpool(2x2)	
Conv2(64, 3x3) Conv2(64, 3x3) Dropout(0.2)	Conv2(128, 3x3) Conv2(128, 3x3)
Maxpool(2x2)	
Conv2(128, 3x3) Conv2(128, 3x3) Dropout(0.2)	Conv2(256, 3x3) Conv2(256, 3x3) Conv2(256, 3x3)
Maxpool(2x2)	
	Conv2(512, 3x3) Conv2(512, 3x3) Conv2(512, 3x3)
Maxpool(2x2)	
	Conv2(512, 3x3) Conv2(512, 3x3) Conv2(512, 3x3)
Maxpool(2x2)	
Fc 512 Dropout(0.2)	Fc 4096 Fc 4096 Fc 1000
Softmax (43)	

Figura 3.16: Arquitectura de la red Mini VGG y VGG16

Fuente: Original

También vamos a incluir una técnica llamada *data augmentation* que consiste en modificar ligeramente las imágenes de entrada practicando rotaciones, *zoom*,

traslaciones o pequeños cambios de color entre otros ligeros cambios, el objetivo de este proceso es generar más datos de entrada a partir de los que ya tenemos. Es importante tener en cuenta que al ser nuestras imágenes señales de tráfico, no podemos practicar ciertas operaciones, como girarlas o deformarlas de manera agresiva o realizar simetrías. En la Figura 3.17 se muestran algunas imágenes de la misma señal modificadas con esta técnica.

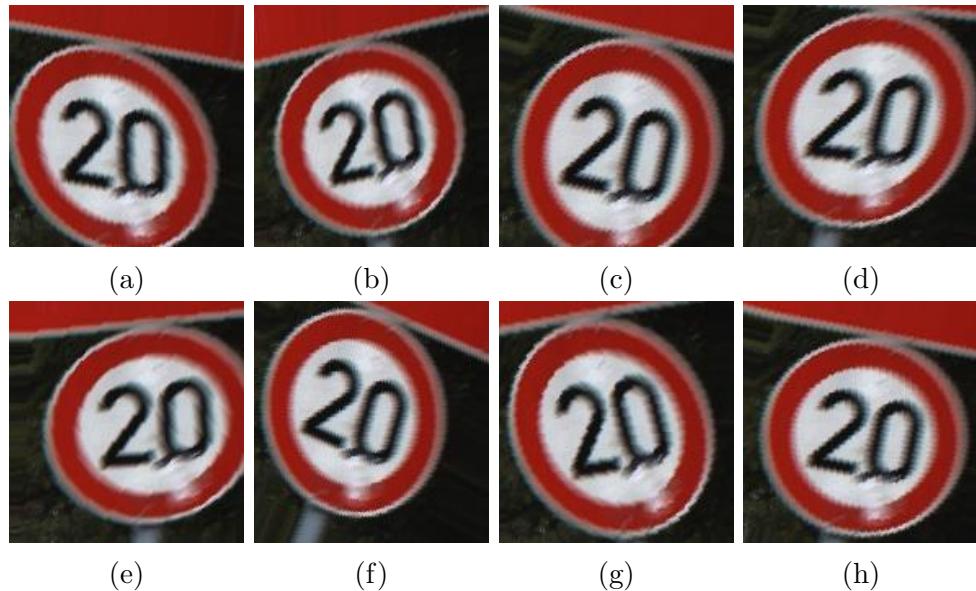


Figura 3.17: Ejemplos de imágenes generadas con *data augmentation* a partir de una única imagen

Fuente: Original

En la Figura 3.18 se indican los resultados obtenidos con las distintas versiones de la arquitectura *Mini VGG* que hemos utilizado.

Resultados utilizando Mini VGG		
Modelo	Precisión media en validación	Precisión en Test
Mini VGG (img 48x48), 20 épocas	99,63%	97,37%
Mini VGG (img 32x32), 20 épocas	99,35%	96,17%
Mini VGG (img 48x48) con data augmentation, 20 épocas	98,93%	96,99%
Mini VGG (img 48x48) con data augmentation, 50 épocas	99,11%	97,00%

Figura 3.18: Resultados al utilizar la arquitectura *Mini VGG*

Fuente: Original

Como vemos en la Figura 3.18, con la arquitectura *Mini VGG* y un tamaño de las imágenes de entrada de 48x48 conseguimos la precisión más alta de todos los modelos probados con un 99,63 % en el conjunto de evaluación y un 97,37 % en el conjunto de test.

Capítulo 4

Validación de resultados

Una vez entrenados los modelos anteriormente descritos, vamos a proceder a valorar los resultados obtenidos utilizando distintas técnicas, como puede ser, dividir el conjunto de test o probar qué resultados se obtienen con distintas imágenes de otras fuentes. De este modo simulamos el rendimiento de los algoritmos si se llevaran a producción.

4.1. Pruebas con el conjunto de test

Para un mayor rigor en la valoración de los resultados de los distintos modelos, vamos a dividir el conjunto de test que se proporciona con el GTSRB en cinco subconjuntos y probar los modelos con cada uno de ello, para de esta manera promediar el resultado obtenido y ver la desviación en la precisión de los modelos. La precisión es importante pero también lo es que nuestro modelo funcione de forma estable con diferentes imágenes, una elevada desviación típica indica que nuestro modelo es poco robusto, en cambio si es pequeña podemos estar más seguros de que funcionará con aproximadamente la misma precisión en diferentes situaciones. No incluimos los modelos preentrenados (*VGG16/VGG19*, *ResNet50* y *Xception*) por su mal rendimiento comparado con las demás aproximaciones. La poca precisión que alcanzan estos modelos preentrenados probablemente sea debida a que no generaliza bien a nuestro problema puesto que el conjunto de datos en el que fueron entrenadas estas redes en primer lugar (*Imagenet*) es demasiado general, incluyendo todo tipo

de especies de animales, herramientas diversas o vehículos de toda clase entre otras, incluyendo solo la clase ‘señal de tráfico’ de forma general, sin distinguir entre los distintos tipos, que es precisamente lo que pretendemos hacer nosotros, por lo que no resultan especialmente útiles para conseguir nuestro objetivo.

En la Figura 4.1 se muestra una tabla con los resultados para cada uno de los cinco subconjuntos de test de cada uno de los modelos probados, así como su desviación típica.

Comparación de resultados utilizando distintos modelos							
Modelo	Test 1	Test 2	Test 3	Test 4	Test 5	Avg Test	Test std
Random Forest 500 arboles y profundidad 22	85.55%	84.24%	85.23%	85.71%	84.40%	85.03%	0.60%
Neural Network 4 capas ocultas 32	89.15%	88.36%	88.24%	88.44%	88.56%	88.55%	0.32%
Simple Cnn 48	96.83%	97.15%	96.71%	97.27%	96.48%	96.89%	0.29%
Skip CoNet 32	95.88%	96.20%	95.72%	95.88%	95.84%	95.90%	0.60%
Skip CoNet-Simple Cnn	95.17%	94.89%	94.66%	94.73%	94.70%	94.83%	0.19%
Mini VGG	97.23%	97.39%	97.70%	97.94%	96.60%	97.37%	0.46%

Figura 4.1: Resultados del conjunto de test dividido en 5

Fuente: Original

De los métodos de aprendizaje automático considerados como clásicos, aquellos que no utilizan convoluciones para extraer información espacial, vemos como la red neuronal alcanza una mayor precisión que el *random forest*, presentando también una menor desviación típica.

Los cuatro modelos basados en redes convolucionales que incluimos funcionan notablemente mejor que las aproximaciones clásicas, y las diferencias en los resultados entre estos modelos son algo más pequeñas, entre el que menos precisión tiene y el que más, hay una diferencia de solo un 2,56 %. El modelo que mejor resultados obtiene es el *Mini VGG*, con un 97,37 % y aunque tiene una desviación típica algo mayor que el *Simple cnn*, que es el siguiente modelo con más precisión con un 96,89 %, la diferencia de 0,17 % entre ambos no es suficiente como para que sea significativa. Llama la atención que la red neuronal convolucional con menor precisión sea también la que menos desviación típica presenta. Podemos ver una gráfica de cajas con las precisiones y desviaciones típicas de cada modelo en la Figura 4.2 .

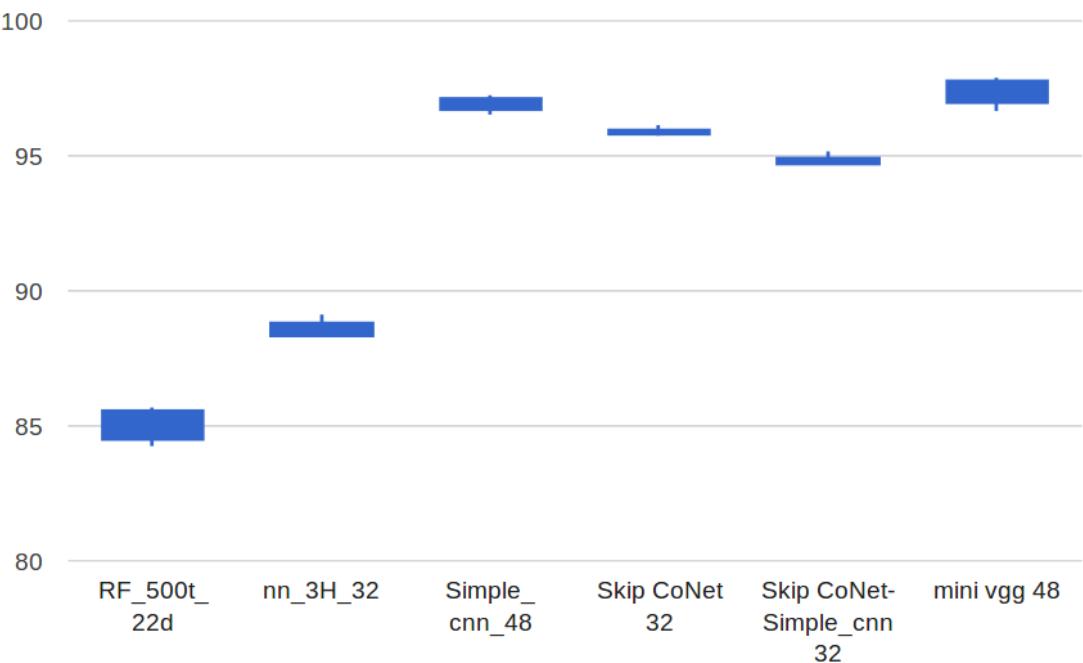


Figura 4.2: Gráfica de cajas de los diferentes modelos

Fuente: Original

4.2. Nuevos datos de test

Es interesante ver como nuestros modelos generalizan lo aprendido en imágenes que no solo nunca han visto sino que provienen de conjuntos de datos distintos, por lo que también vamos a probar nuestros modelos con imágenes de test que no se encuentran en el conjunto proporcionado por GTSRB, para ver que tal se comporta con ejemplos externos a los propuestos. En concreto vamos a probar nuestros modelos en el conjunto de test del *BelgiumTS Dataset*.

Este conjunto de datos es similar al que venimos utilizando pero las imágenes de señales de tráfico son distintas, en este hay 62 clases de señales distintas en vez de las 43 que tenía nuestro conjunto de datos, además las señales de limitación de velocidad corresponden todas a la misma clase, es decir, tanto la señal de límite de velocidad de 20km/h como la de 80km/h corresponderían a una misma clase, mientras que en el GTSRB cada una pertenecía a una clase diferenciada. También tenemos diferencias en el tipo de señales, hay señales que aparecen en un conjunto de datos y en el otro no y viceversa. Por último, incluso en señales que indican lo mismo hay pequeñas variaciones

en el dibujo debido a que provienen de señales de distintos países. En la Figura 4.3 se pueden ver ejemplos de señales iguales en los dos conjuntos de datos.

Para que los resultados sean lo más justos posible, vamos a utilizar solo los tipos de señales de tráfico del conjunto de test del *BelgiumTS Dataset* que también aparezcan en el *The German Traffic Sign Recognition Benchmark*, excluyendo las señales de límite de velocidad que, aunque están presentes en los dos conjuntos de datos como ya explicamos en el *BelgiumTS Dataset*, todas corresponden a una misma clase. Sin embargo sí que vamos a dejar las imágenes de las señales de la Figura 4.3, que aunque ligeramente distintas, representan la misma señal. Entendiendo que parte del poder generalizador de los algoritmos entrenados es el de poder superponerse a pequeñas diferencias como las que presentan estos imágenes en cada uno de los conjuntos de datos utilizados.

Después de eliminar todas las imágenes que no se corresponden con ninguna del conjunto de datos con el que hemos entrenado y por tanto no se podrían identificar correctamente, nos quedan un total de 1106 imágenes sobre las que probaremos la precisión de nuestros modelos. En la Figura 4.4 podemos observar como se comporta cada uno de los algoritmos propuestos.

El modelo que mejor funciona en el *BelgiumTS Dataset* es el *Mini VGG* con 92.13 % de precisión, que coincide con el que mejor precisión tenía en el conjunto de datos original, el segundo mejor es el primero que probamos, el *Simple Cnn*, con un 91.23 % a la cola se quedan prácticamente empatados los modelos con conexiones no secuenciales, el *Skip coNet* con un 87.61 % y el *Skip coNet - Simple Cnn* con 87.53 %, parece que estos dos modelos generalizan peor a imágenes nuevas. Cabe señalar también que sean menos robustos y que tengan una desviación típica mayor que en el conjunto de datos original.



Figura 4.3: Ejemplos de las diferencias en las mismas señales entre los dos conjuntos de datos utilizados (BelgiumTS y el GTSRB)

Fuente: Original

Resultados de los modelos convolucionales en el BelgiumTS Dataset							
Modelo	Test 1	Test 2	Test 3	Test 4	Test 5	Avg Test	Test std
Simple Cnn 48	92.34%	94.12%	89.14%	88.69%	91.86%	91.23%	2.04%
Skip CoNet 32	88.74%	90.05%	85.97%	87.78%	85.52%	87.61%	1.69%
Skip CoNet-Simple Cnn	87.43%	86.43%	86.43%	86.88%	90.50%	87.53%	1.55%
Mini VGG	90.54%	90.95%	92.76%	94.57%	91.86%	92.13%	1.44%

Figura 4.4: Resultados del *BelgiumTS Dataset* dividido en 5 subconjuntos

Fuente: Original

Capítulo 5

Conclusiones

A lo largo del desarrollo del presente trabajo he aprendido una gran cantidad de material, que no ha sido cubierto durante los estudios de grado, incluyendo: Preprocesado de imágenes, desarrollo de proyectos de aprendizaje automático de principio a fin, realización de experimentos para validar resultados, búsqueda de hiperparámetros, múltiples conceptos de diseño de arquitecturas de redes neuronales convolucionales, además del uso de herramientas y librerías específicas empleadas ampliamente en el entorno del aprendizaje automático.

Hemos probado diferentes algoritmos y aproximaciones al problema de la clasificación de imágenes, dos de ellos, el *random forest* y la red neuronal sin convoluciones, aproximaciones clásicas dentro del aprendizaje automático, mientras que los demás entran dentro del conjunto de técnicas modernas que denominamos aprendizaje profundo. Entre estos últimos diferenciamos entre las aproximaciones que hacen uso de redes preentrenadas y aquellas que se han entrenado desde cero. Mientras que las primeras, por el tipo de imágenes concretas que queremos clasificar, no han alcanzado una precisión elevada, quedando incluso por debajo de los métodos clásicos, los otros cuatro modelos de redes neuronales convolucionales han alcanzado elevados porcentajes de precisión de hasta un 97,37 %, aún así, no llega al estado del arte.

Para mejorar los resultados, se podrían realizar multitud de pruebas como: Realizar una búsqueda de hiperparámetros más exhaustiva, probar con diferentes optimizadores y *learning rates*, construir redes más o menos profundas, variar el número de neuronas en cada capa, aplicar distintas técnicas de regularización, probar diferentes tipos de

preprocesado en las imágenes de entrada, etc.

El presente trabajo podría también ampliarse en el futuro abarcando además el problema de la detección, que además de reconocer a qué clase pertenece una determinada señal, también sea capaz de localizarla en una escena con más objetos. Aunque se escapa del objetivo de este trabajo, se ha utilizado el módulo *dnn* de la librería *OpenCV* utilizando una versión preentrenada de la arquitectura *MobileNet* [15] combinada con *Single Shot Detector* [16], para realizar una prueba de concepto. Podemos ver en la Figura 5.1 un ejemplo en el que detecta a un caballo y a una persona.

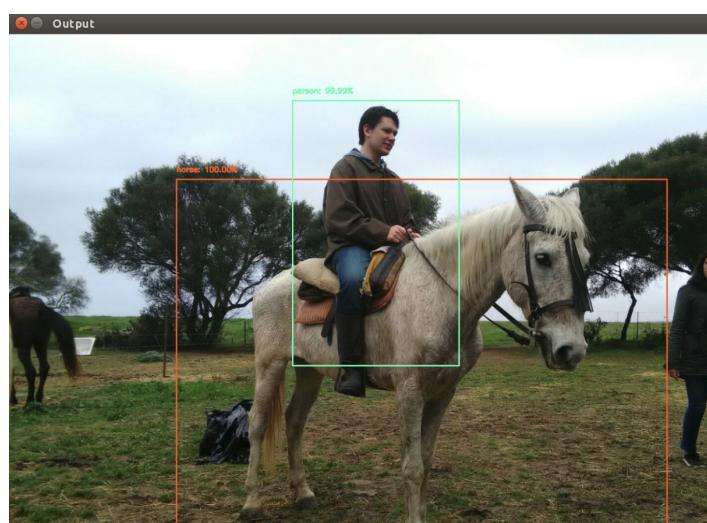


Figura 5.1: Prueba de detección de objetos

Fuente: Original

Diseñar redes neuronales convolucionales es una tarea compleja, en la que hay una gran cantidad de variables a tener en cuenta y siempre puedes probar nuevas versiones de una red para comprobar si el cambio introducido se traduce en una mejora en el resultado del modelo o por el contrario lo empeora. Durante la realización de esta investigación se han probado una gran cantidad de modelos distintos, y muy probablemente se podría haber mejorado la precisión realizando más pruebas, pero ejecutar estos modelos es una tarea que consume una cantidad de tiempo importante, aún cuando se utiliza una tarjeta gráfica de alta gama y eso nos limita la cantidad de versiones que es posible probar. En definitiva, a pesar de las limitaciones, hemos conseguido unos buenos resultados.

Los avances que en los últimos años se han desarrollado en el campo del aprendizaje profundo ha permitido resolver problemas con un nivel de precisión que antes

simplemente no era posible alcanzar, el campo de la visión por computador es uno de los más directamente beneficiados por estos avances, hace tan solo una década no podríamos haber conseguido estos resultados. Solo podemos imaginar lo que seremos capaces de hacer en los próximos años.

Capítulo 6

Bibliografía

- [1] Francois Chollet. *Deep learning with python*. Manning Publications Co., 2017.
- [2] Pierre Sermanet and Yann LeCun. Traffic sign recognition with multi-scale convolutional networks. In *Neural Networks (IJCNN), The 2011 International Joint Conference on*, pages 2809–2813. IEEE, 2011.
- [3] The German Traffic Sign Recognition Benchmark Web , Enero 2018. [Online].
- [4] Johannes Stallkamp, Marc Schlipsing, Jan Salmen, and Christian Igel. The German Traffic Sign Recognition Benchmark: A multi-class classification competition. In *IEEE International Joint Conference on Neural Networks*, pages 1453–1460, 2011.
- [5] Cross-Validation in Machine Learning , Marzo 2018. [Online].
- [6] The Random Forest Algorithm , Enero 2018. [Online].
- [7] Neural Networks and Deep Learning , Enero 2018. [Online].
- [8] Building powerful image classification models using very little data , 2018. [Abril Online].
- [9] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.
- [10] ImageNet: VGGNet, ResNet, Inception, and Xception with Keras , Abril 2018. [Online].

- [11] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.
- [12] Deep Learning for Image Recognition: why it’s challenging, where we’ve been, and what’s next ↗, Marzo 2018. [Online].
- [13] François Chollet. Xception: Deep learning with depthwise separable convolutions. *arXiv preprint arXiv:1610.02357*, 2016.
- [14] An Intuitive Guide to Deep Network Architectures ↗, Abril 2018. [Online].
- [15] Andrew G Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. Movenet: Efficient convolutional neural networks for mobile vision applications. *arXiv preprint arXiv:1704.04861*, 2017.
- [16] Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott Reed, Cheng-Yang Fu, and Alexander C Berg. Ssd: Single shot multibox detector. In *European conference on computer vision*, pages 21–37. Springer, 2016.
- [17] Waseem Rawat and Zenghui Wang. Deep convolutional neural networks for image classification: A comprehensive review. *Neural computation*, 29(9):2352–2449, 2017.
- [18] BelgiumTS Dataset ↗, Mayo 2018. [Online].
- [19] Pre-trained Deep Neural Net architectures for Flower Species Recognition ↗, Enero 2018. [Online].
- [20] Markus Mathias, Radu Timofte, Rodrigo Benenson, and Luc Van Gool. Traffic sign recognition—how far are we from the solution? In *Neural Networks (IJCNN), The 2013 International Joint Conference on*, pages 1–8. IEEE, 2013.
- [21] Keras Tutorial - Traffic Sign Recognition ↗, Enero 2018. [Online].
- [22] Traffic Signs Recognition cnn ↗, Mayo 2018. [Online].
- [23] Transfer Learning using Keras ↗, Marzo 2018. [Online].
- [24] Daniel Lerch. Fine-tuning en reconocimiento de imágenes mediante Deep Learning ↗, Abril 2018. [Online].
- [25] Transfer Learning - Machine Learning’s Next Frontier ↗, Abril 2018. [Online].

- [26] Transfer Learning: Leverage Insights from Big Data ↗, Marzo 2018. [Online].
- [27] A Gentle Introduction to Transfer Learning for Deep Learning ↗, Marzo 2018. [Online].
- [28] supervised learning ↗, Febrero 2018. [Online].
- [29] Supervised vs. Unsupervised Learning ↗, Marzo 2018. [Online].
- [30] Deep Learning made easy with Deep Cognition ↗, Marzo 2018. [Online].

Lista de Figuras

1.1.	Comparación de redes neuronales simples y profundas	4
2.1.	Imágenes de muestra del conjunto de datos	9
2.2.	Secuencia de imágenes de la misma señal	10
2.3.	Número de imágenes de entrenamiento por clase	10
2.4.	Imagen antes y después de aplicar el preprocesado	12
2.5.	Validación cruzada	12
3.1.	Representación de un <i>random forest</i> de 2 árboles	16
3.2.	Resultados obtenidos con el algoritmo <i>random forest</i>	17
3.3.	Esquema de una red neuronal	18
3.4.	Estructura de las redes neuronales utilizadas	18
3.5.	Resultados obtenidos con las redes neuronales de 3 y 4 capas ocultas . .	19
3.6.	Arquitectura de la red VGG16	21

3.7. Resultados al hacer <i>transfer learning</i> con <i>VGG16</i> y <i>VGG19</i>	21
3.8. Bloque residual de <i>ResNet50</i>	22
3.9. Resultados al hacer <i>transfer learning</i> con <i>ResNet50</i>	23
3.10. Esquema de una <i>pointwise separable convolution</i>	24
3.11. Resultados al hacer <i>transfer learning</i> con <i>Xception</i>	24
3.12. Resultados de la arquitectura <i>Simple Cnn</i>	25
3.13. Arquitectura de la red Skip CoNet	26
3.14. Resultados al utilizar la arquitectura <i>Skip CoNet</i>	27
3.15. Resultados al utilizar la arquitectura <i>Skip CoNet - Simple CNN</i>	28
3.16. Arquitectura de la red Mini VGG y VGG16	29
3.17. Ejemplos de imágenes generadas con <i>data augmentation</i> a partir de una única imagen	30
3.18. Resultados al utilizar la arquitectura <i>Mini VGG</i>	30
4.1. Resultados del conjunto de test dividido en 5	34
4.2. Gráfica de cajas de los diferentes modelos	35
4.3. Ejemplos de las diferencias en las mismas señales entre los dos conjuntos de datos utilizados (BelgiumTS y el GTSRB)	37
4.4. Resultados del <i>BelgiumTS Dataset</i> dividido en 5 subconjuntos	37

5.1. Prueba de detección de objetos	40
---	----

Anexos

Anexos A

Código

Junto con la presente memoria se incluye el código python utilizado. Las rutas que aparecen hacen referencia a ‘Corleone’, el ordenador en el que se han ejecutado la gran mayoría de las pruebas, para reproducir los resultados en otra máquina será necesario por tanto modificar estas rutas. La estructura de las carpetas en las que se encuentra el código se detalla a continuación:

```
traffic_sign_machine_learning
├── cnn6l
│   ├── cnn_multi_model_systematized.py
│   ├── cnn_data_augmentation.py
│   ├── test_systematized.py
│   └── test_belgium_dataset.py
├── nn
│   ├── nn_v2_systematized.py
│   └── test_systematized.py
├── rf
│   ├── random_forest_systematized.py
│   └── rf_test_systematized.py
├── resnet50
│   ├── resnet50_systematized.py
│   └── resnet50_test.py
├── vgg16
│   ├── vgg16_systematized.py
│   └── vgg16_test.py
└── vgg19
    ├── vgg19_systematized.py
    └── vgg19_test.py
└── xception
    ├── xception_systematized.py
    └── xception_test.py
```

Para cada uno de los distintos modelos tenemos un fichero Python de entrenamiento que contienen la palabra ‘systematized’, este fichero genera un modelo entrenado, que utilizamos en el fichero Python de test. En el caso de los modelos de redes neuronales convolucionales entrenados desde cero, también incluimos un fichero ‘cnn_data_augmentation.py’ que utilizamos para entrenar el modelo Mini VGG utilizando data augmentation, y el fichero ‘test_belgium_dataset.py’ que utiliza el conjunto de datos belga para las pruebas.