E.T.S. de Ingeniería Industrial,
Informática y de Telecomunicación

Estudio comparativo de modelos de clasificación automática de señales de tráfico

Grado en Ingeniería Informática

Trabajo Fin de Grado

Alumno: Arturo Silva González

Director: Daniel Paternain Dallo

Pamplona, 27 de febrero de 2020

upna
Universidad Pública de Navarra
Nafarroako Unibertsitate Publikoa

AGRADECIMIENTOS

Este trabajo fin de grado pone fin a mis estudios de Ingeniería Informática.

Me gustaría mostrar mis agradecimientos:

- A mis padres, no soy nada sin ellos.
- A Pascual, por todo lo que ha hecho por mí estos cuatro años en Pamplona.
- A mi familia, por su apoyo incondicional.
- A mis amigos, por estar siempre a mi lado.
- Y a mi director de TFG, Daniel, por sus consejos y por lo bien que se ha portado conmigo.

BREVE DESCRIPCIÓN

Este Trabajo Fin de Grado (TFG) pertenece al campo de Inteligencia Artificial, más concretamente al de Visión Artificial. Está divido en dos partes diferenciadas.

La primera parte es un estudio comparativo de un modelo de clasificación automático de imágenes de señales de tráfico, en la que haremos aproximación basada en Machine Learning tradicional y una basada en Redes Neuronales Convolucionales (Deep Learning). Para ello, analizaremos el rendimiento, la complejidad y el acierto que obtendremos al clasificar usando dichos modelos.

En la segunda parte, crearemos un detector de señales de tráfico en imágenes y vídeos basado en Aprendizaje Profundo. El algoritmo que se utilizará en este detector será YOLO "You Only Look Once".

PALABRAS CLAVE

- Visión Artificial,
- Aprendizaje Profundo,
- Detección de Objetos,
- Aprendizaje Supervisado,
- Redes Neuronales.

ÍNDICE

1. INTRODUCCIÓN Y OBJETIVOS

El término de Inteligencia Artificial (IA) fue acuñado en 1956 por unos de los padres de esta, John McCarthy. Sin embargo, desde hace unos pocos años, dicha rama de la informática ha experimentado un despegue en su investigación, desarrollo e implementación en el mundo real. Hoy en día, se está usando para localizar enfermedades, recomponer imágenes dañadas, reconocer objetos y personas, gestionar anuncios según nuestros gustos, predecir el precio de bienes, reconocimiento de palabras por voz, etc.

El objetivo de la Inteligencia Artificial es que las máquinas puedan ocuparse de tareas que los humanos consideramos inteligentes, permitiéndonos delegar ese trabajo. Gracias a que los ordenadores son capaces de hacer más cálculos por segundo que un cerebro humano, para ciertas tareas específicas ya nos han superado en efectividad, precisión y eficiencia.

Una rama de la IA es el Aprendizaje Automático (Machine Learning en inglés), cuyo objetivo es desarrollar técnicas que permitan a los computadores aprender. Una manera de hacer esto, es mediante el aprendizaje supervisado; que consiste en adiestrar a la máquina a partir de experiencias de las que conocemos su resultado. Cuando haya estrenado lo suficiente, queremos que sea capaz de funcionar bien en nuevas situaciones; a esta función se le denomina generalización. Como podemos deducir, los datos han pasado a ser una parte importantísima de esta rama, ya que, de no tener buenos datos, la dificultad de desarrollar sistemas precisos crece enormemente.

Dentro de la rama de Aprendizaje Automático apareció una nueva categoría que en nuestros días es muy popular: el Aprendizaje Profundo (Deep Learning en inglés). Esta se basa en teorías acerca de cómo funciona el cerebro humano; para ello se apoya en un sistema llamado Red Neuronal, compuesta por múltiples capas que permiten realizar abstracciones del conocimiento cuando el ordenador aprende, permitiéndole funcionar mejor en nuevas situaciones.

Una de las tareas cotidianas que nosotros consideramos inteligente es conducir, razón por la cual se ha estado trabajando arduamente en desarrollar un coche autónomo, es decir, que sea capaz de conducir por él mismo. El desafío está claro; crear un sistema de conducción que sea más seguro que un humano. Viendo los datos de la Dirección General de Tráfico (DGT), sabemos que un 33% de los accidentes se debe a distracciones y un 29% a exceso de velocidad; es decir, se podría reducir más de un 50% los casos de accidentes. Esto es posible gracias a que las máquinas no presentan distracciones y respetarían el código de circulación siempre que fuera posible.

Para ello desarrollar tan complejo sistema, es necesario que nuestro coche autónomo sea capaz de reconocer el trazado, los peatones, los vehículos y las señales de tráfico. Una vez reconocidas, actuaría de un modo determinado a esa situación.

Este Trabajo Fin de Grado va a estudiar distintitos métodos basados en Aprendizaje Automático tradicional y Aprendizaje Profundo en la clasificación de señales de tráficos en imágenes, con el objetivo de determinar cuál es mejor y por qué. Además, se creará un sistema de detección de señales de tráfico a partir de imágenes y vídeos, basado en Redes Neuronales Profundas. Es decir, se va a centrar en lo que sería la primera etapa del coche autónomo: el reconocimiento.

1.1. Objetivos

El objetivo de este trabajo es desarrollar un algoritmo de detección y clasificación de señales de tráfico automático. El desarrollo de este objetivo lo vamos a realizar en dos fases: fase de clasificación, que consistirá en crear unos modelos que a partir de una señal sea capaz de decirnos de cuál se trata; y una fase de detección, que será la encargada de a partir de una imagen con señales de tráfico detectarlas y decir a qué clase pertenecen.

Para la primera fase, realizaremos una comparativa entre **Machine Learning tradicional** y **Deep Learning**. Buscando el modelo de reconocimiento que mejor rendimiento proporcione.

Para la segunda fase, crearemos un detector de imágenes de tráfico basado en **YOLO** "You Only Look Once", un modelo de Deep Learning formado por Redes Neuronales Convolucionales, el cual permitirá identificar señales de tráfico en una imagen.

Este TFG se dividirá en 4 capítulos.

- En el capítulo 1 expondremos los objetivos de este proyecto.
- En el capítulo 2 explicaremos en qué consiste el Machine Learning, qué entendemos por Machine Learning tradicional y en qué se basa el Deep Learning y la Red Neuronal Convolucional.
- En el capítulo 3 abordaremos el problema de clasificación de señales de tráfico. Primero con Machine Learning tradicional y extracción de características; después, lo haremos con modelos de Deep Learning. Finalmente, analizaremos los resultados que hemos obtenido.
- En el capítulo 4, explicaremos cómo funciona un detector de objetos basado en aprendizaje profundo, implementaremos uno para la detección de señales de tráfico y analizaremos los resultados.

2. PRELIMINARES

2.1. Machine Learning

El Aprendizaje Automático (Machine Learning, **ML**) consiste en algoritmos que aprenden patrones a partir de datos. Así que, lo que se busca es la generalización, es decir, que se comporte correctamente ante datos nuevos.

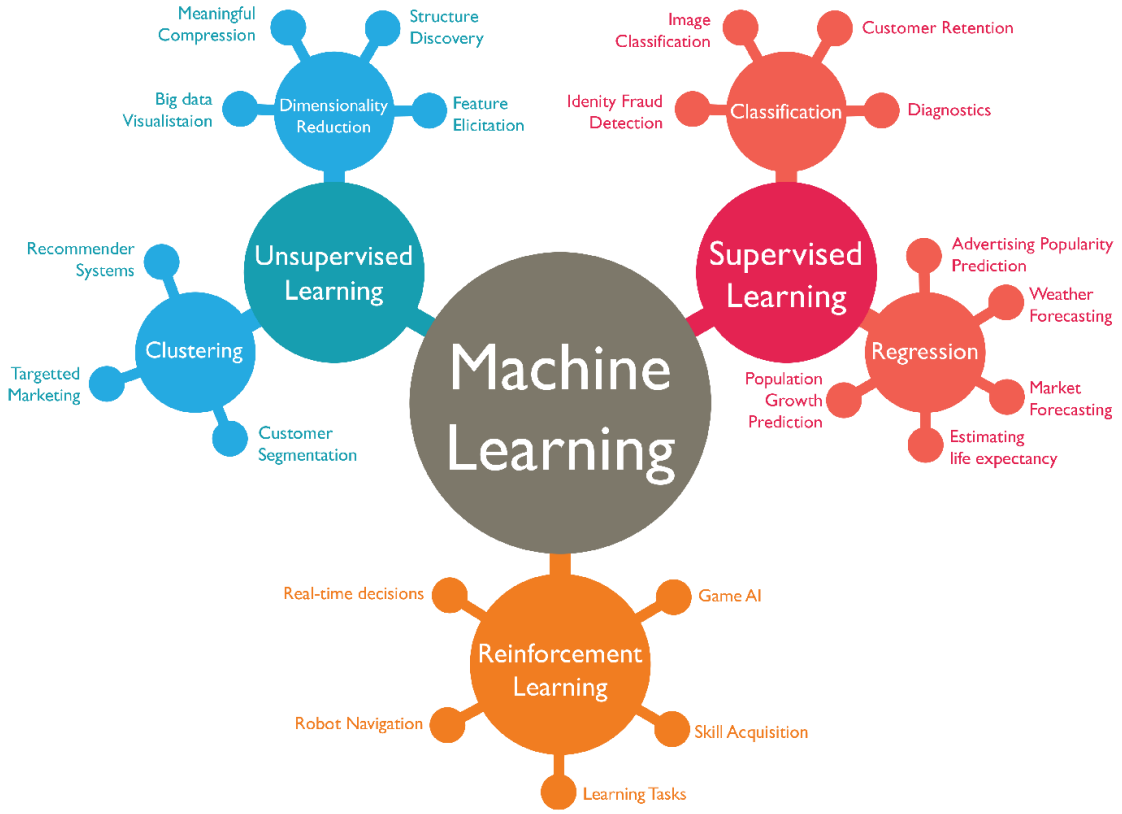


Ilustración 1 – Ramas Machine Learning

El aprendizaje automático es un nuevo paradigma respecto a la programación clásica. Esta última se centraba en a partir de unas reglas y de unos datos dar un resultado. En cambio, el ML a partir de unos datos y unas respuestas, crea unas reglas para esas circunstancias; por lo que, estaría usando unas reglas que no han sido explícitamente codificadas por programadores.

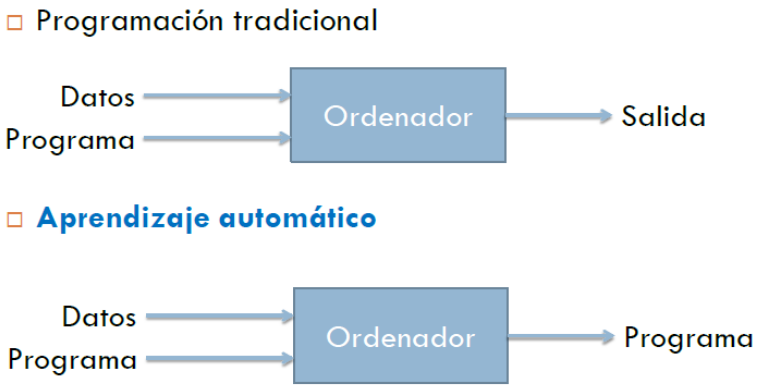


Ilustración 2 - Programación clásica vs Aprendizaje automático

Algunos de los modelos más conocidos son:

- **Aprendizaje Supervisado (Supervised Learning)**: necesita datos previamente etiquetados para aprender a realizar una tarea. Con estos, lo que conseguimos es adiestrar al sistema para que pueda resolver problemas futuros. Dentro de este tipo nos podemos encontrar dos tipos distintos de problemas.
    - **Regresión**: saber el valor numérico de una variable continua. Por ejemplo, el precio de una vivienda.
    - **Clasificación**: saber a qué clase pertenece un ejemplo. Por ejemplo, si una imagen es un coche o un barco.
- **Aprendizaje No Supervisado (Unsupervised Learning)**: son los algoritmos de aprendizaje automático que no necesitan de datos etiquetados. Sin embargo, sí que es necesario algún tipo de indicación previa para analizar y comprender la información que se está usando para aprender.
- **Reforzamiento por aprendizaje (Reinforcement Learning)**: el computador aprende de los refuerzos de la práctica, basándose en éxitos o fracaso que tenga. Es decir, aprender por su cuenta.

En nuestro TFG vamos a basarnos en el **Aprendizaje Supervisado**, de tipo **Clasificación**. Es decir, contaremos con un conjunto de datos con imágenes de las señales de tráfico y la clase a la que pertenecen cada una de ellas.

2.1.1. Qué consideramos Machine Learning Tradicional

Para este Trabajo Fin de Grado, con el fin de poder designar de forma rápida qué técnica usamos, se va a considerar como Machine Learning Tradicional a toda técnica que no tenga que ver con el Deep Learning.

Sin embargo, como podemos ver en la siguiente ilustración, el Deep Learning es una subrama del Machine Learning.

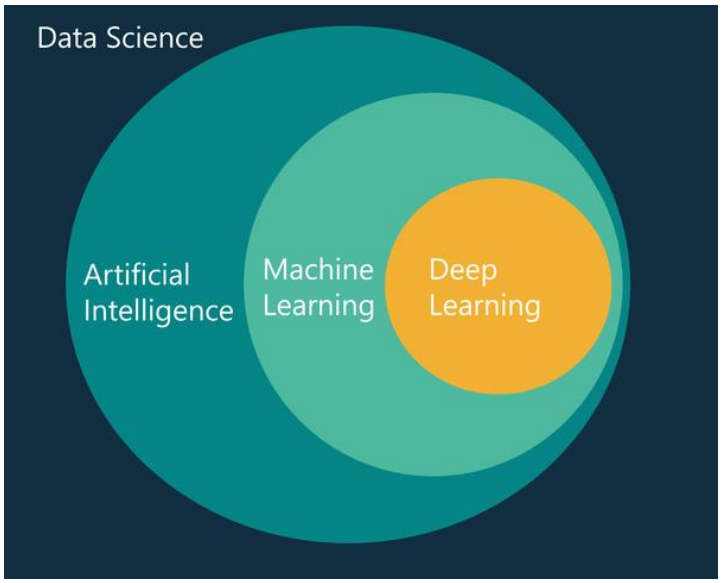


Ilustración 3 - Aprendizaje Automático y Aprendizaje Profundo

Unos modelos típicos de Machine Learning Tradicional serían:

- **Regresión**: trata de encontrar para unas características de entrada una salida en forma de variable continua. Se podría aplicar un modelo de regresión a la predicción del precio de una vivienda, por ejemplo.
- **Regresión logística**: para unos atributos trata de encontrar un súper plano que separe a dos o más clases. Está orientado a la clasificación y da buenos resultados para ejemplos sencillos.
- **Naive-Bayes**: este modelo de aprendizaje se basa en el aprendizaje estadístico para clasificar ejemplos. Ha dado muy buenos resultados para clasificar correo electrónico en SPAM y no SPAM.
- **SVM**: hablaremos de él en profundidad más adelante. Se trata de encontrar el súper plano que separe a dos clases, con la mayor distancia entre ellas.

Siendo $\hat{y}$ el resultado del modelo, x_i un parámetro del dato de entrada, w_i el valor del peso asociado al parámetro y g() la función de activación.

2.2. Deep Learning

El Aprendizaje Profundo es un conjunto de algoritmos de Aprendizaje Automático que se centran en emular el enfoque de aprendizaje de los seres humanos, pudiendo modelar abstracciones de alto nivel de los datos utilizando arquitecturas computacionales que admiten operaciones no lineales.

2.2.1. En qué se basa

La arquitectura computacional más usada para el Deep Learning es la **Red Neuronal**. Estas redes neuronales están formadas por múltiples capas. Las neuronas de una capa están conectadas con las adyacentes por medio de conexiones, las cuales tienen asociado un peso.

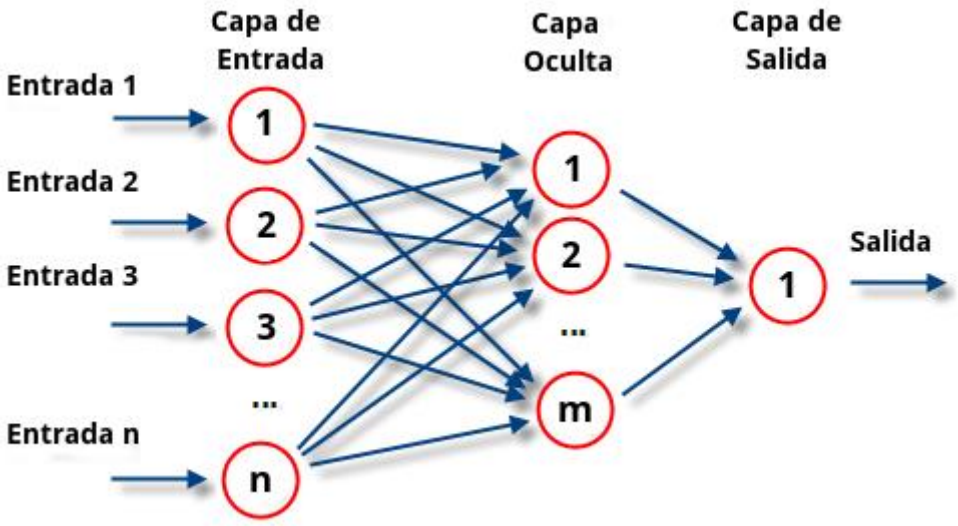


Ilustración 4 - Red Neuronal

El valor de cada neurona de la **capa de entrada** es un parámetro de un ejemplo concreto.

Para obtener el resultado de una neurona de una capa oculta, haríamos la suma ponderada de los pesos de las conexiones por su parámetro correspondiente. Una vez obtenido el resultado de esa suma, aplicaríamos una **función de activación** para dicha neurona. De esta manera, podemos encadenar resultados de distintas neuronas de una capa anterior pudiendo crear un modelo de separación de clases no lineal.

Las funciones de activación más conocidas son las siguientes:

- **Sigmoide**: transforma los valores introducidos a una escala (0, 1). Los valores altos tienden de forma asintótica a 1 y los valores muy bajos tienden de manera asintótica a 0. Habitualmente se utiliza en la última capa en problemas de clasificación binarios.

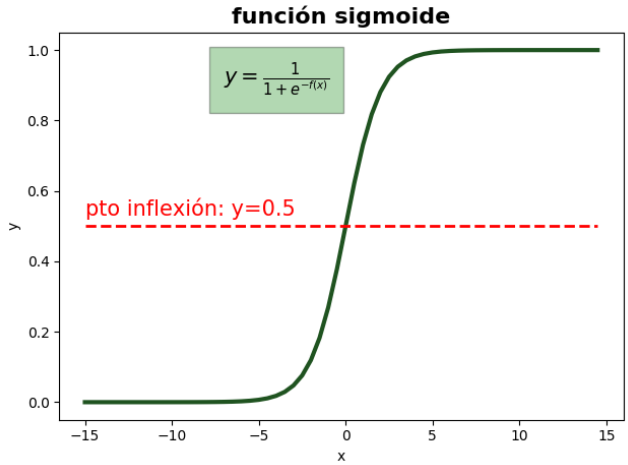


Ilustración 5 - Función sigmoide

- **Tangente hiperbólica**: transforma los valores a una escala (-1, 1), los valores altos tienen una asíntota a 1, y los bajos a -1. Es muy similar a la sigmoide, satura y acaba con el gradiente, tiene una lenta convergencia, centrada en 0, tiene buen desempeño en redes recuerrentes y es usada para decidir entre una opción y su contraria. Su fórmula directa es: $f(x) = \frac{2}{1+e^{-2x}} - 1$.

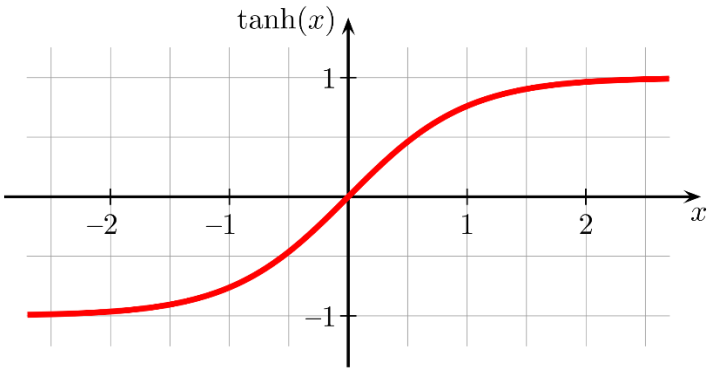


Ilustración 6 - Función tangente hiperbólica

- **ReLU** o rectificador (Rectified Lineal Unit): esta función transforma los valores introducir anulando los valores negativos y dejando los positivos tal cual entran. Sus características son: solo se activa si son positivos, no está acotada, se comporta bien en imágenes con un bajo coste computacional.

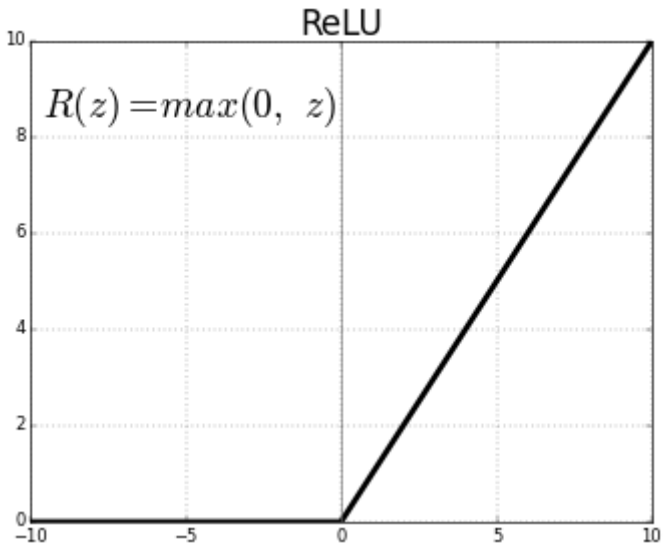


Ilustración 7 - Función ReLU

Cuando se crea la Red Neuronal, los pesos de las conexiones son aleatorios. Durante el aprendizaje, se irán ajustando hasta llegar a los valores que minimicen la función de error de dicha red.

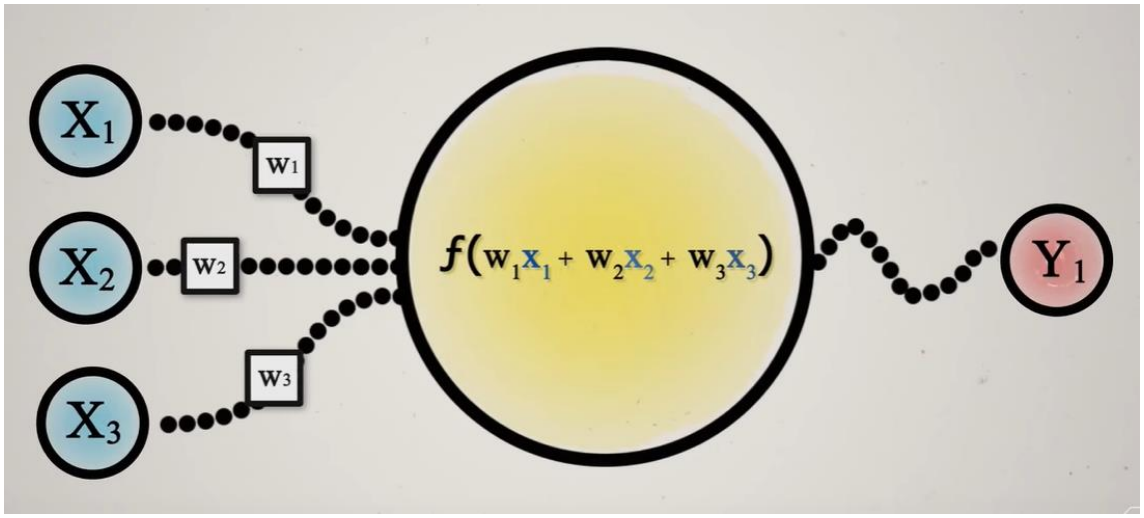


Ilustración 8 - Funcionamiento neurona

El funcionamiento de una red neuronal consta de dos partes:

- **Propagación hacia adelante (Forward propagation)**: para obtener el resultado de salida de una red neuronal, tenemos que calcular por capas el valor de cada una de la neurona. Hacemos esto hasta llegar a las neuronas de salida. Como hemos podido ver, los valores de una neurona de la capa i depende de los valores de las neuronas de las capas i-1. Sin embargo, los valores de las neuronas de una misma capa son independientes entre sí; esto nos permite un **paralelizar** las operaciones.
- **Propagación hacia atrás o retropropagación (Backpropagation)**: esta es la parte más interesante de las redes neuronales. Recordemos que tenemos una función de coste para la red neuronal, que nos mide la calidad de las soluciones que esta está aportando. Después de hacer la propagación hacia adelante, calculamos el error de la red neuronal con los resultados obtenidos. Después, obtenemos las derivadas parciales del error con respecto a los pesos que unen la última capa oculta con la de salida. Repetimos este

proceso con las capas anteriores. Para finalizar ajustaríamos los pesos de cada neurona con el objetivo de reducir el error. Repetiríamos este proceso por cada ejemplo.

2.2.2. La Red Neuronal Convolucional

La Red Neuronal Convolucional (Convolutional Neuronal Network, **CNN** o **ConvNet**) es una clase de red de aprendizaje profundo, aplicada en el análisis visual de imágenes. Las principales aplicaciones que tiene son reconocimiento de imágenes y vídeos, clasificación de imágenes y procesamiento del lenguaje natural.

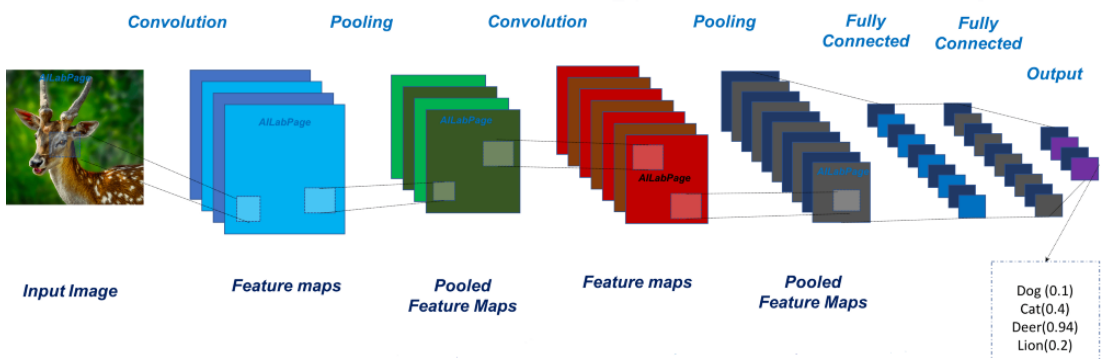


Ilustración 9 - Red Neuronal Convolucional

Este tipo de Red Neuronal, por medio de entrenamiento y aprendizaje supervisado, intenta imitar cómo funciona las neuronas en la corteza visual de nuestro cerebro. Por lo tanto, estas tienen muchas capas ocultas especializadas, las cuales siguen una jerarquía. Las primeras capas irán detectando líneas y curvas, las siguientes irán cada vez abstrayéndose más hasta poder identificar formas complejas.

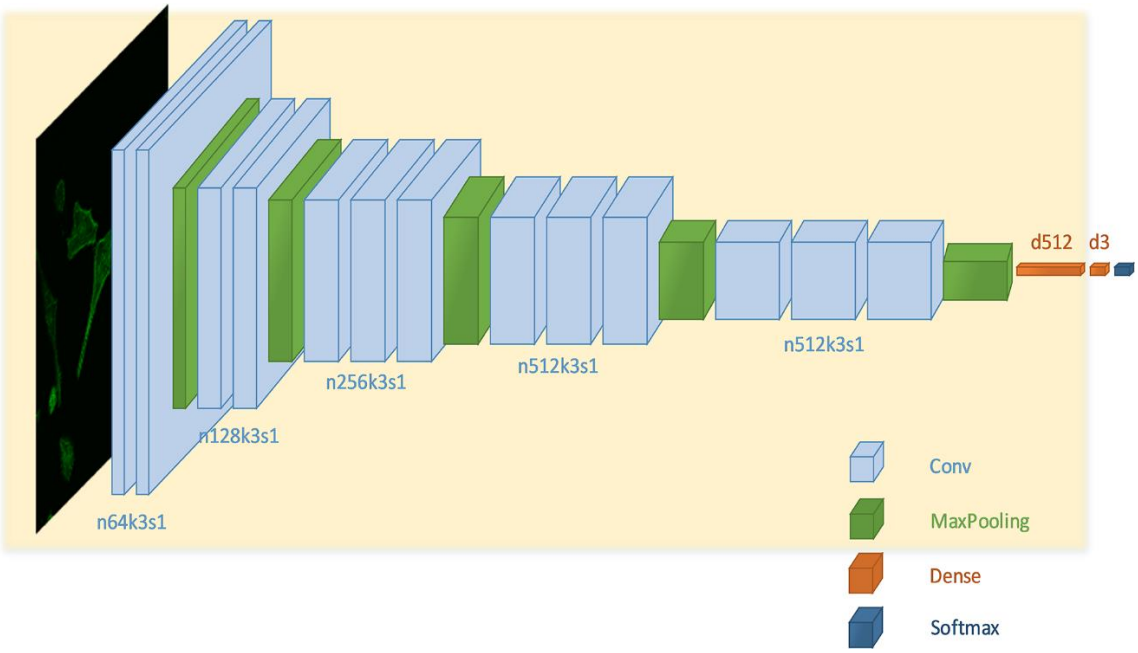


Ilustración 10 - Capas de una Red Neuronal Convolucional

A continuación, vamos a mostrar las formas que sería capaz de reconocer cada capa de una CNN. Realmente, son las formas que hace que una neurona de esa capa maximice su activación. [1]

Ilustración 11 - Capa 1 de una CNN

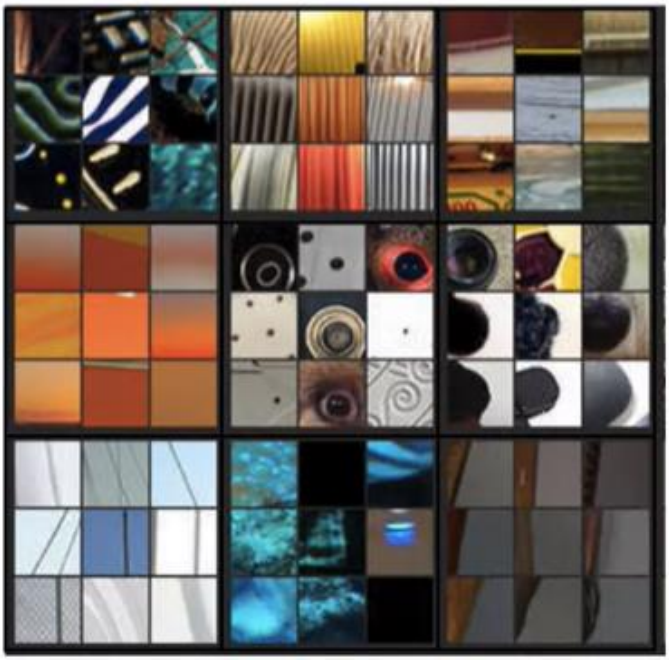


Ilustración 12 - Capa 2 de una CNN

Ilustración 13 - Capa 3 de una CNN

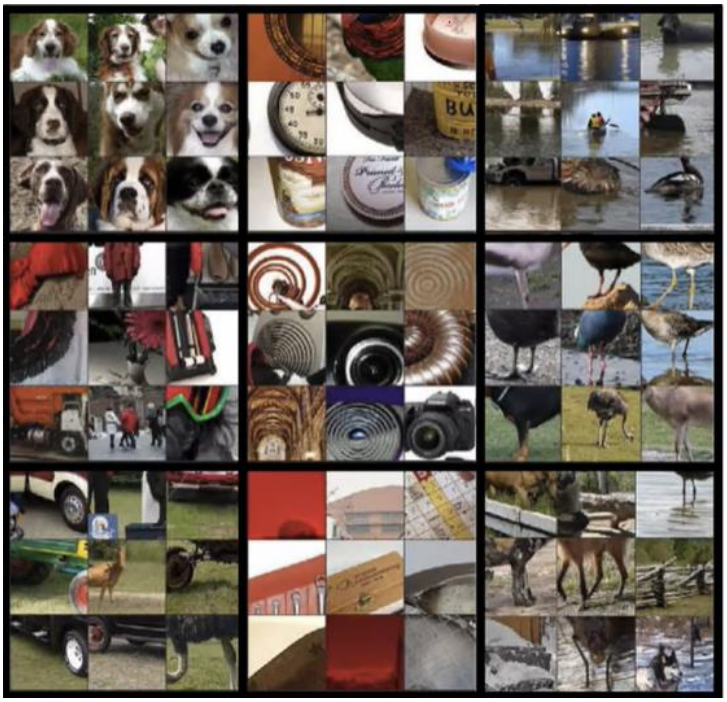


Ilustración 14 - Capa 4 de una CNN

2.2.1.1. Capas Convolucionales

Como hemos dicho, la capa de entrada de una Red Neuronal Convolucional tiene como neuronas de entrada los píxeles de la imagen. Es decir, una imagen de 32x32 tendría 1024 neuronas de entrada. No obstante, eso solo sería si la imagen es en blanco y negro, si es en color ya pasamos a 32x32x3, que son 3072 neuronas de entrada.

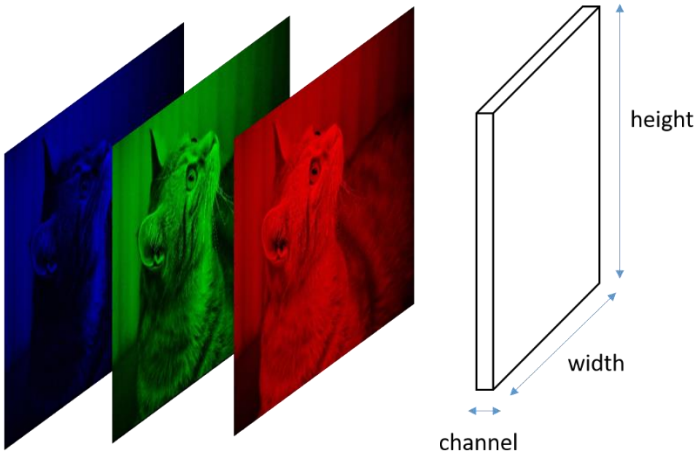


Ilustración 15 - Imagen RGB

Como se ha explicado antes, los valores de los píxeles de una imagen son muy parecidos con los adyacentes, no sería eficiente usarlos todos los valores de píxeles, ya que de esta manera aumentaríamos desproporcionalmente la cantidad de cómputo. En este momento es cuando entra en escena la operación que le da nombre a este tipo de Red Neuronal, la **Convolución**.

La convolución consiste en tomar grupos de píxeles cercanos de la imagen de entrada e ir haciendo el **producto escalar** contra una pequeña matriz que se llama **filtro** o **kernel**. Este filtro recorre todas las neuronas de entrada (de izquierda a derecha y de arriba a abajo), generando una nueva matriz de salida, la cual será nuestra nueva capa de neuronas ocultas.

Una convolución depende del **tamaño** del **kernel** y del tamaño saltos (**strides**) que haga con la convolución, dando como resultado una matriz de un tamaño que depende de esos 2 valores. Para que una operación de convolución con saltos de un píxel de una imagen de la misma dimensión que la original haría falta poner un **relleno** a la imagen.

Podemos calcular la dimensión (resolución) de la matriz resultante tras aplicar una convolución con la siguiente función:

$$resolución = \left(\frac{n + 2p - f}{s} + 1\,, \frac{n + 2p - f}{s} + 1\right)$$

Siendo:

- **n**: número píxeles en una fila o columna de la imagen original.
- **f**: número de píxeles de un filtro, en una fila o columna.
- **p**: número de píxeles de relleno que ponemos a la imagen.
- **s**: número de saltos que damos al deslizar el filtro.

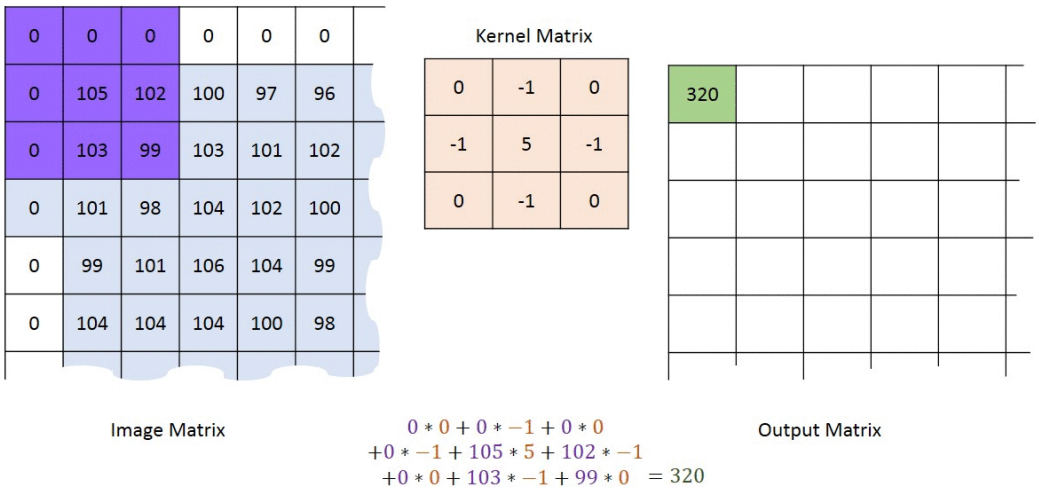


Ilustración 16 - Ejemplo de convolución en imagen

Los valores del filtro serán al principio valores aleatorios, como en toda red neuronal. Mediante la retropropagación ajustaremos los valores.

En una Red Neuronal Convolucional no aplicamos solamente un solo kernel, sino que usamos muchos, es decir, tendremos un **conjunto de filtros** en cada capa.

En caso de contar con una imagen en blanco y negro de tamaño 32x32, podríamos tener para la primera capa, por ejemplo, 16 filtros; por lo que obtendríamos 16 matrices de salida. El filtro podría ser de tamaño 3x3x1, en caso de ser una imagen a color tendría que ser 3x3x3. Por lo tanto, aplicando cada filtro obtenemos una matriz de salida de 32x32x1 por cada filtro; y teniendo en cuenta que tenemos 16 filtros, en total tenemos unas 16 384 neuronas en la **primera capa oculta** para la imagen en blanco y negro.

Cuando convolucionamos los kernels con la imagen, obtenemos una nueva imagen con ciertas características de la imagen original que ha sido filtrada. A esto se le llama mapeo de características (**Feature Mapping**)

Para obtener el valor de esa neurona, recordamos que hasta ahora solo habríamos hecho una suma ponderada de píxeles, le pasamos una función de activación. La más usada es la función **ReLU**, ya que es la que mejores resultados experimentales ha dado.

$$f(x) = \max(0, x)$$

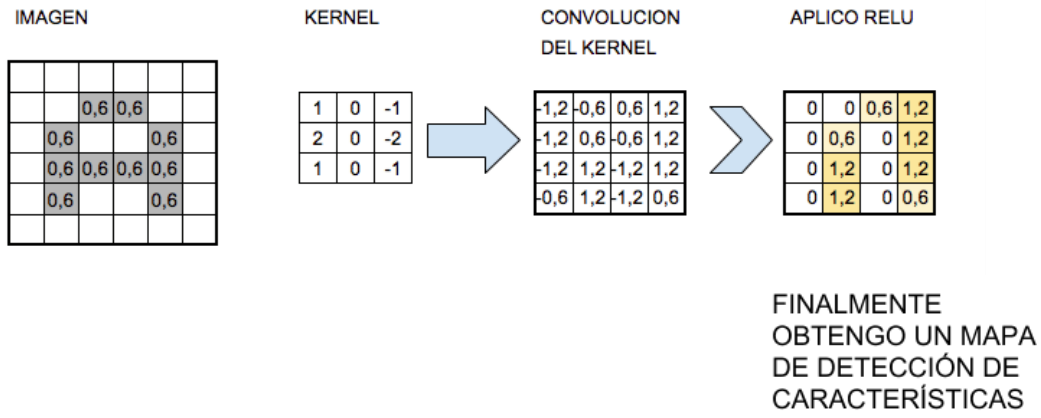


Ilustración 17 – Cálculo del valor de las neuronas en la 1ª capa oculta

2.2.1.2. Capas de Pooling

Hasta ahora, no hemos reducido aún la dimensión de las características, hemos obtenido unas nuevas matrices que muestran características destacadas de la imagen original con la misma dimensión de esta.

Sin embargo, es conveniente reducir la dimensión de dichas características para no aumentar significativamente el número de parámetros de la red. Por eso, nos vamos a quedar con las características más importantes que detectó cada filtro. La operación usada para realizar esto se le llamada submuestreo (**Subsampling**). Siendo en imágenes la más común el **Max-Pooling**, una agrupación los valores máximos. El tamaño de la matriz que obtenemos depende de la dimensión de la reducción.

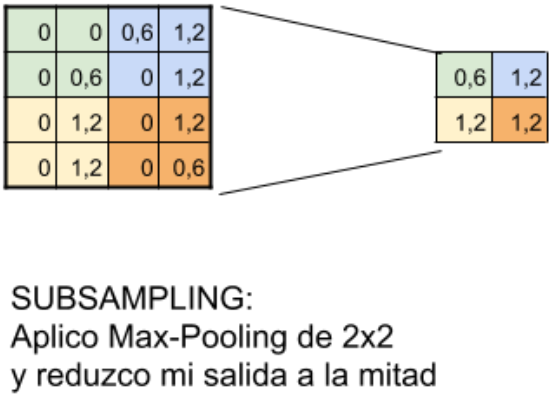


Ilustración 18 - Max-Pooling de 2x2

Si seguimos con el ejemplo anterior, al hacer un Max-Pooling de 2x2 en la capa oculta, en la que tenemos 16 384 neuronas, pasaríamos a tener 4 096 neuronas. Como podemos ver, la reducción es bastante grande.

Con todo lo anterior, ya tendríamos nuestra primera capa de convolución que detecta características primitivas como líneas o bordes.

Vamos a representar en una tabla las dimensiones de esta primera parte. Consideramos que las operaciones de convolución son con relleno, por lo que la salida tiene la misma dimensión que la imagen de entrada. Es decir, considerando p el número de píxeles de relleno cumpliría que $p = \frac{f-1}{2}$.

Imagen de entrada	Filtros	Feature Mapping	Salida al aplicar Max-Pooling 2x2
32x32x1	16 filtros de 3x3	32x32x16	16x16x16

Tabla 1 - Capa 1 de CNN a partir de imagen en blanco y negro

En caso contar con imágenes a color sería:

Imagen de entrada	Filtros	Feature Mapping	Salida al aplicar Max-Pooling 2x2
32x32x1	16 filtros de 3x3x3	32x32x16	16x16x16

Tabla 2 - Capa 1 de CNN a partir de imagen a color

Es decir, con imágenes a color y con las capas de neuronas de las siguientes capas, estamos haciendo la Convolución en Volúmenes. Por lo que la profundidad del filtro siempre tiene que ser igual a la profundidad del volumen que queremos convolucionar.

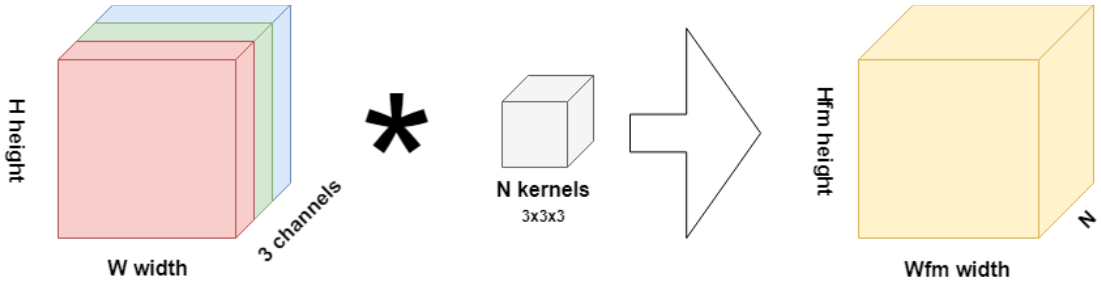


Ilustración 19 - Ejemplo de convolución con N filtros

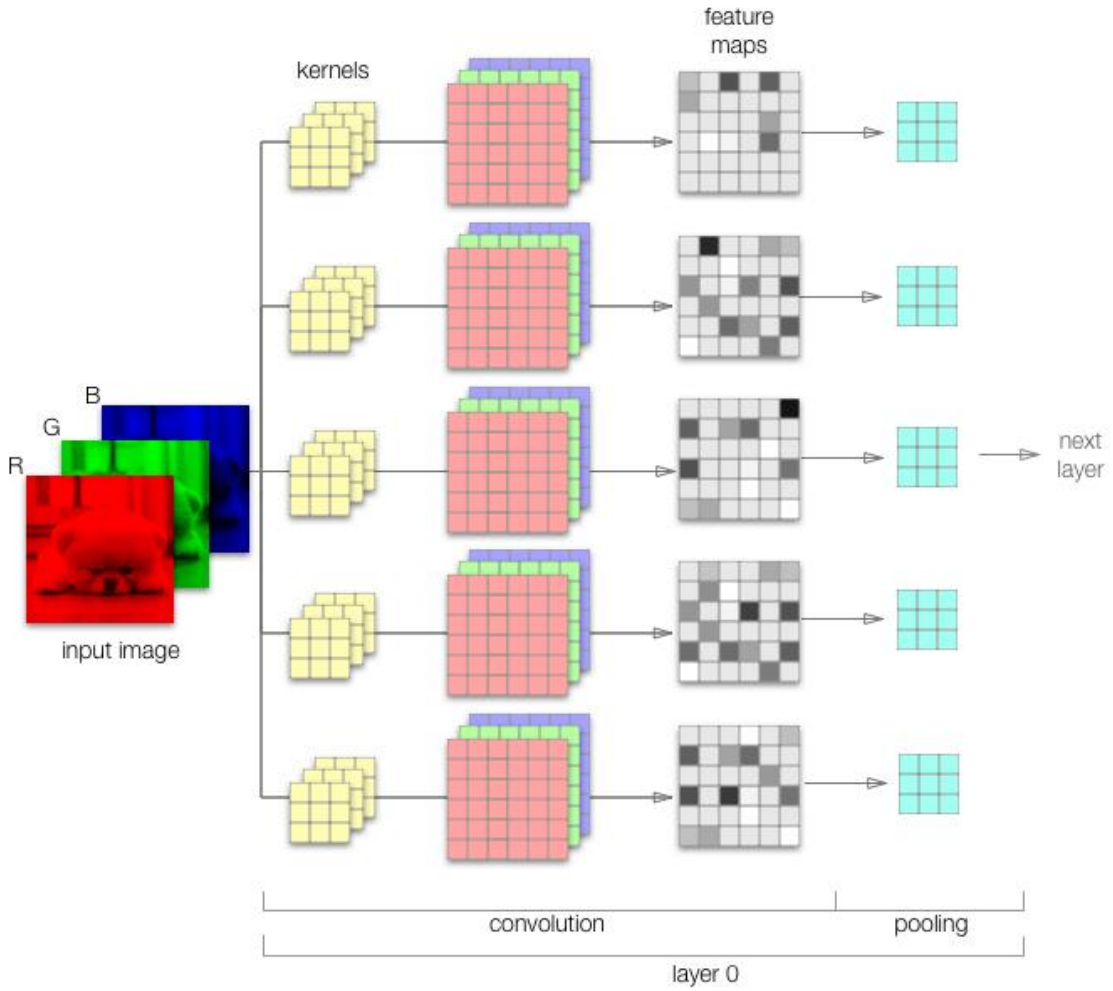


Ilustración 20 - Primera capa de CNN

Lo siguiente, sería juntarlo con otra capa convolucional para que esta se abstrajera y fuera capaz de reconocer figuras más complejas. Obtendríamos la siguiente tabla.

Imagen de entrada	Filtros	Feature Mapping	Salida al aplicar Max-Pooling 2x2
16x16x16	64 filtros de 3x3x16	16x16x64 (Aplicamos ReLU)	8x8x64

Tabla 3 - Capa 2 de CNN a partir de imagen a color

Como se puede intuir, cuando se realiza la convolucional por capas hay muchas operaciones que son independientes unas de otras, lo cual, permite una alta **paralelización**. Pudiendo así, con el hardware y la implementación correcta, reducir el tiempo de cálculo.

2.2.1.3. Capas Fully-Connected

Tras encadenar varias capas convolucionales, tomaríamos la última capa a la que le hemos hecho Max-Pooling. En nuestro caso, 8x8x64, siendo (altura x anchura x canal); aplanaríamos esta, obteniendo una **capa tradicional** de redes neuronales. Así que, pasamos de un volumen de 8x8x64 a 4096 neuronas.

Las partes de la Red Neuronal Convolucional que son Redes Neuronales Multicapa tradicionales se les da el nombre de **Fully Connected**. Ya que estas neuronas están completamente

conectadas con las de las sucesivas capas, es decir, una neurona de una capa está conectada con todas las neuronas de la siguiente.

La función de esta última parte de las CNN es coger los resultados del proceso de convolución/agrupación y usarlo para clasificar imágenes. Podemos agrupar varias capas como estas, tal como sucede en la Red Neuronal Tradicional.

2.2.1.4. Capas de salida

En la capa de salida, si quisiéramos predecir 2 clases, podríamos usar una función de activación sigmoide. Tal que si es y si el resultado es mayor que 0.5 predecirlo como la 1ª clase. Sin embargo, si tenemos más 2 clases lo que tendríamos que la función **Softmax**.

Softmax consiste en que la capa de salida tenga tantas neuronas como clases a predecir. En nuestro caso, sería el número de señales de tráfico que queremos clasificar. El valor de cada neurona, la cual tiene asociado una neurona, será la probabilidad de ser esa clase. Estaríamos haciendo un uno contra todos (**One vs. All**).

La función softmax, o función exponencial normalizada, es $f(z)_j = \frac{e^{z_j}}{\sum_{k=1}^{K} e^{z_k}}$ para j = 1, …, K. Siendo j asociada a una neurona de la capa de salida. Al final, la suma de los valores de las neuronas de salida es 1.

Por lo tanto, la clase a predecir será la asociada a la neurona de la capa de salida con el valor más alto. [2]

Como podemos ver, al usar Deep Learning para clasificar señales, no hacemos una extracción de características previa; sino que ya nuestro modelo es el que lo hace. De esta forma, se especializa el tipo de problema.

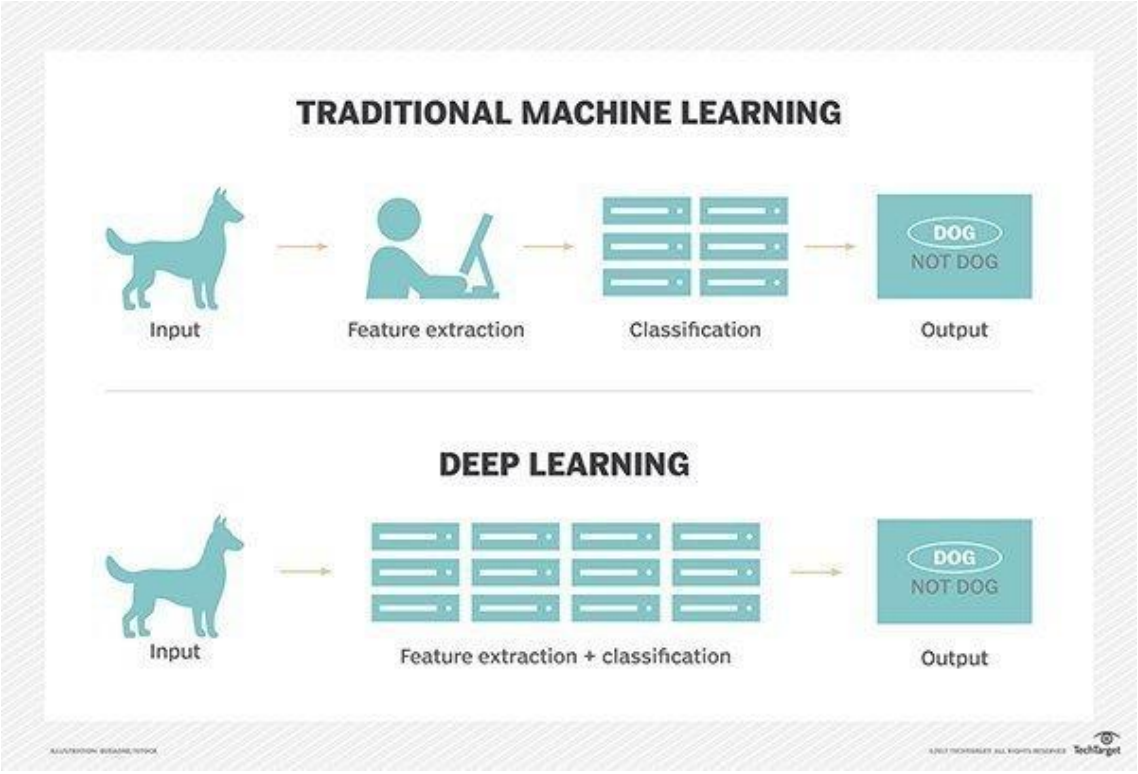


Ilustración 21 - ML tradicional vs Deep Learning

2.2.1.5. Aprendizaje

La Red Neuronal Convolucional, como las tradicionales, aprende en el proceso de Backpropagation, en el que ajusta los pesos de sus interconexiones con el objetivo de minimizar la función de error.

En el caso de las CNN, los pesos que justa son los **pesos** de los distintos **filtros**. Volviendo a nuestro caso de ejemplo, si tenemos una imagen en blanco y negro de 32x32x1 y en la primera capa convolucional usamos 16 filtros de 3x3x1, los pesos a ajustar son 144. En caso de estar en el ejemplo con una imagen a color serían 432 pesos. Muchos menos de lo que serían los pesos en caso usar una Red Neuronal tradicional que usara como parámetros todos los píxeles de la imagen.

2.2.1.5.1. Descenso por gradiente

El algoritmo principal de optimización para el aprendizaje automático es el descenso por gradiente. Se usa tanto para Redes Neuronales como para otros modelos de Machine Learning.

Es un método general de **minimización** para nuestra **función de coste** (o error, la cual depende de nuestro modelo). El método consiste en obtener la primera derivada de la función de coste para cada parámetro, con lo que podemos calcular la pendiente en ese punto específico; es decir, cómo **crece la función en ese punto**. Como lo que se busca es el **punto mínimo de la función** (derivada de la función igualada a cero por cada variable), seguiremos la dirección contraria a la pendiente (la 1ª derivada), e iremos bajando en cada **iteración** esa pendiente hasta el punto mínimo, con cada parámetro.

En resumen, calculamos $\frac{\partial error}{\partial \theta_i} = \nabla f$ para todo i = 1, …, n. Siendo n el número de parámetros y ∇f el gradiente de la función y $\frac{\partial error}{\partial \theta_i}$ la derivada del error respecto al parámetro i. A la hora de actualizar los parámetros haremos la operación $\theta_i = \theta_i - \alpha \frac{\partial error}{\partial \theta_i}$. Para todo i = 1, …, n.

Como vemos, entra también en la ecuación el parámetro α, que es cuánto avanzamos en cada iteración. Es decir, regula la velocidad de convergencia al mínimo. La elección del correcto valor de alfa es muy importante, ya que de elegir uno muy pequeño la velocidad sería muy lenta, y si eligiésemos uno muy grande podría no converger nunca.

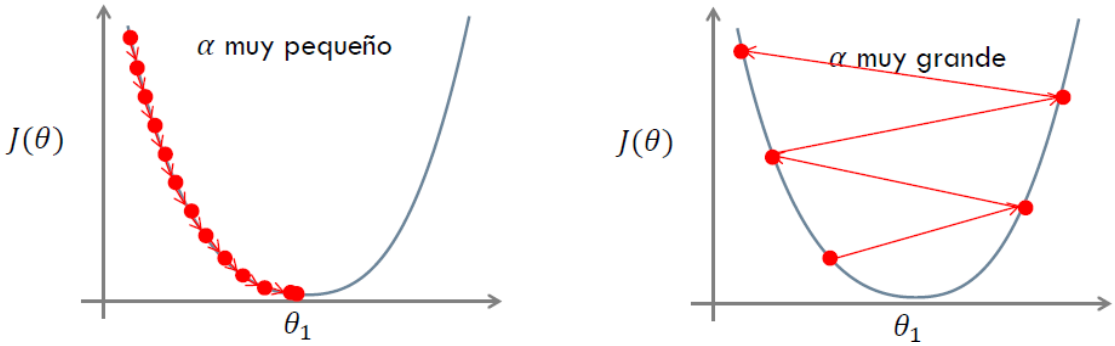


Ilustración 22 - Elección de alfa para el descenso por gradiente

A la hora de realizar el aprendizaje de nuestro modelo, contamos con varias opciones para actualizar los parámetros según el número de muestras (ejemplos) que utilizamos.

- **Descenso del gradiente por lotes** (o **batch**): utilizamos todos los datos. El gradiente se calcula usando todos los ejemplos. Eso es un problema, ya que puede haber un momento en que las variaciones son mínimas y haya un estancamiento.

- **Descenso del gradiente estocástico**: se introduce un único ejemplo aleatorio en cada iteración. Con esto, se dificulta el estancamiento. Sin embargo, es bastante lento y solo aprovecha la información que puede dar una muestra.
- **Descenso del gradiente estocástico en mini-lotes** (**mini-batch**): introducimos un número determinado de muestras en cada iteración. Con esto, podemos aprovechar las ventajas de los dos anteriores métodos, irá rápida la convergencia y tenemos arbitrariedad de ejemplos. La elección del número de muestras dependerá de nuestra capacidad del hardware, se suelen coger número potencias de 2, como 64; esto es debido a la distribución de la memoria RAM o VRAM.

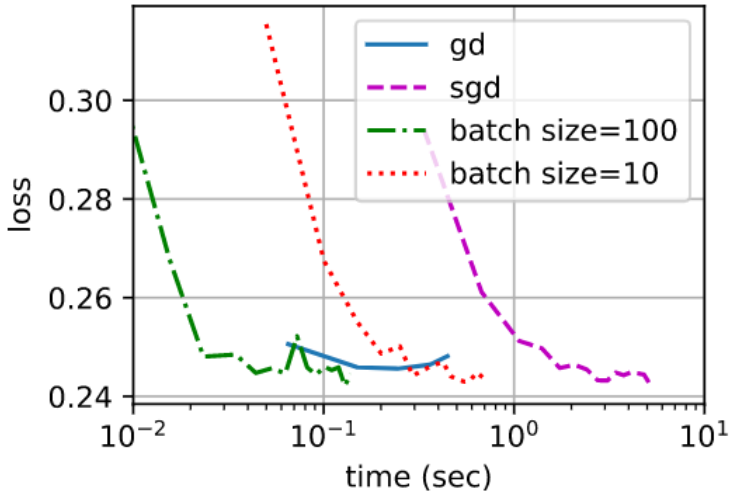


Ilustración 23 - Velocidad de convergencia según tamaño de lote

Por lo tanto, elegiríamos el tamaño de mini-batch y ejecutaríamos el algoritmo del gradiente hasta llegar al punto mínimo de la función de coste. Si la función solo tiene un mínimo, encontraremos siempre el mínimo de la función sin ningún problema. Recordemos que, en el siguiente ejemplo, el valor del gradiente es un vector de 2 componentes (plano X e Y), donde nos marca hacia dónde tenemos que movernos para aumentar el coste más rápido; si hacemos le cambiamos el signo, nos dice en qué sentido tenemos que ir para reducirlo más rápido. Cuando mayor es el módulo, mayor es la pendiente.

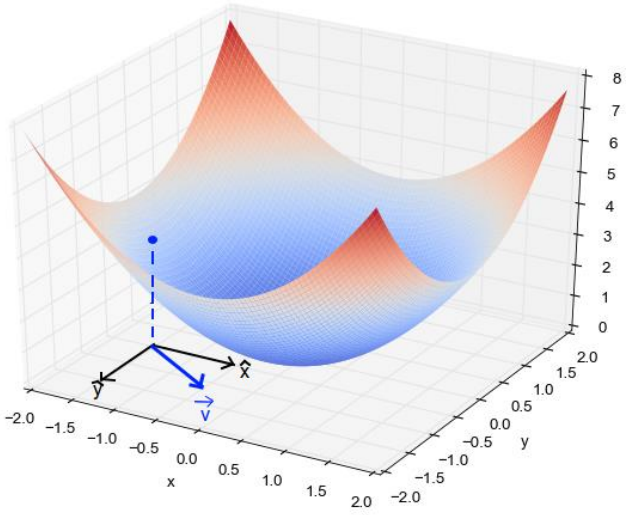


Ilustración 24 - Ejemplo con un solo mínimo local

No obstante, esto no suele pasar. La mayoría de las funciones de coste de los modelos aprendizaje automático son más complejas y presentan mínimos locales, los cuales dificultan la convergencia al mínimo global.

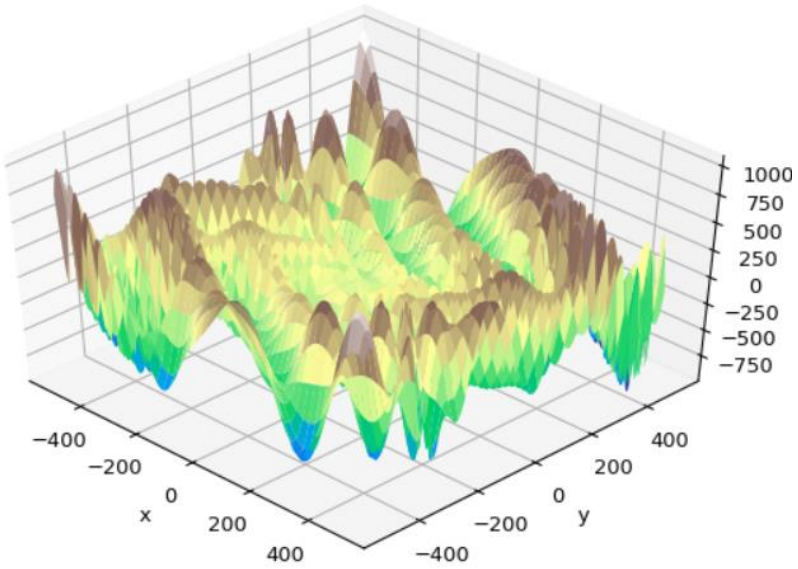


Ilustración 25 - Ejemplo con muchos mínimos locales

A la hora de entrenar, por lo tanto, tendremos que evitar quedarnos atascados en ningún mínimo global, ajustar el número de **épocas** (**epochs**, una iteración; se han usado todas las muestras) y elegir el valor del **factor de aprendizaje** (alfa, si cambiará conforme avanzamos o tendrá siempre el mismo valor durante todo el entrenamiento).

Se ha ido mejorando la implementación del algoritmo del descenso por gradiente para evitar los problemas mencionados anteriormente, además de, mejor la velocidad de convergencia.

El método **Adam** (**Adaptative Moment Estimation**) es el método de optimización más utilizado en el Machine Learning. Su éxito es la unión de varios métodos que mejoraban determinados aspectos del descenso por gradiente.

Es una combinación de los métodos:

- **RMSProp** (**Root Mean Square Propagation**): adapta la **tasa de aprendizaje** para cada parámetro. Con eso conseguimos acelerar el aprendizaje, ya que el valor de alfa por cada parámetro aumentará o disminuirá con el fin de acercarnos más rápido al mínimo o no alejarnos de esa trayectoria.
- **Momentum**: se actualizan los **parámetros de la red** añadiendo un término adicional que tiene en cuenta el valor de actualización aplicada en la iteración anterior. De tal forma, que con esto conseguimos que avance más rápido cuando nos vemos en la dirección correcta. Además de que se reducen posibles desviaciones no deseadas.

Así que, con Adam conseguimos potenciar el cambio de parámetros de la red y el cambio de la tase de aprendizaje. Consiguiendo evitar mínimos locales y encontrar el global. Esto es lo que le hace, el método de optimización más usado; y el que nosotros vamos a emplear también. [3]

Podemos ver en la siguiente ilustración, que el algoritmo es el que más rápido reduce el coste por cada época.

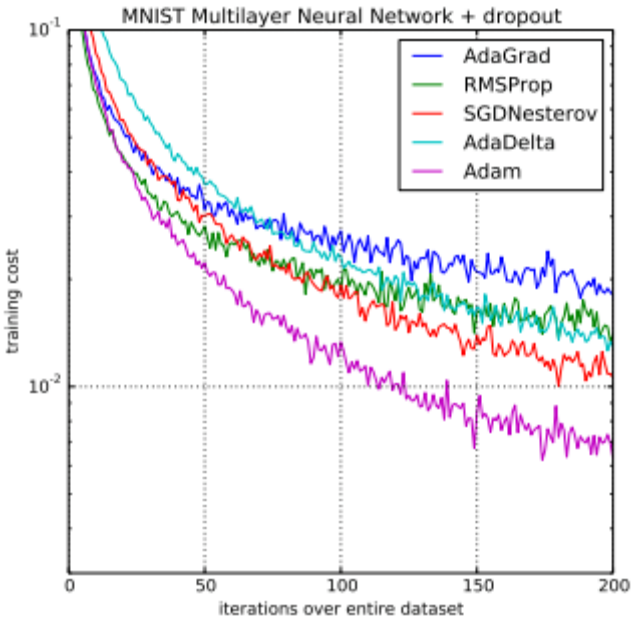


Ilustración 26 - Rendimiento de distintos algoritmos de optimización

2.2.1.5.2. Mejorar velocidad de aprendizaje

Como se ha explicado anteriormente, a la hora de introducir los datos de entrada en una Red Neuronal Convolucional se normalizan con el objetivo de mejorar la velocidad aprendizaje y evitar que parámetros más altos, no por ello más importantes, influyan en el entrenamiento.

Sin embargo, lo anterior solo lo hacemos en la 1ª capa, después ya no lo hacemos. En el momento de realizar los cálculos de una capa, podemos tener valores con distintas magnitudes. Esto dificulta el entrenamiento, ya que la activación de la función podría hacer que el gradiente a penas cambiara, o que pesos que serán importantes tengan un valor muy bajo.

Asimismo, hay una razón muy importante que hace que funcione; se le conoce como Cambio Covariable (**Covariate Shift**). Ya que, si has aprendido anteriormente que una entrada determinada produce una salida determinada, pero cambian los rangos de los valores, tendremos que reaprender otra vez esa relación. Esto es bastante frecuente en las imágenes, ya que, dependiendo de la calidad de la imagen, esta puede tener valores más altos o no.

De estos problemas, nace la solución de la **Normalización por Lotes** (**Batch Normalization**), encargada de normalizar los datos obtenidos tras calcular una capa (normalmente antes de la función de activación). Con esto, acabamos teniendo unos valores normalizados que están en [0, 1].

Con esto, hemos conseguido reducir la distribución de valores, por lo que los valores son más estables. Así que, las capas posteriores pueden más fácil ajustar los pesos.

Input: Values of x over a mini-batch: $\mathcal{B} = \{x_{1...m}\}$;
Parameters to be learned: γ, β
Output: $\{y_i = \mathrm{BN}_{\gamma,\beta}(x_i)\}$

$$\mu_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m} x_i \qquad \text{// mini-batch mean}$$

$$\sigma^2_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m}(x_i - \mu_{\mathcal{B}})^2 \qquad \text{// mini-batch variance}$$

$$\widehat{x}_i \leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma^2_{\mathcal{B}} + \epsilon}} \qquad \text{// normalize}$$

$$y_i \leftarrow \gamma\widehat{x}_i + \beta \equiv \mathrm{BN}_{\gamma,\beta}(x_i) \qquad \text{// scale and shift}$$

Ilustración 27 - Fórmulas Normalización por Lotes

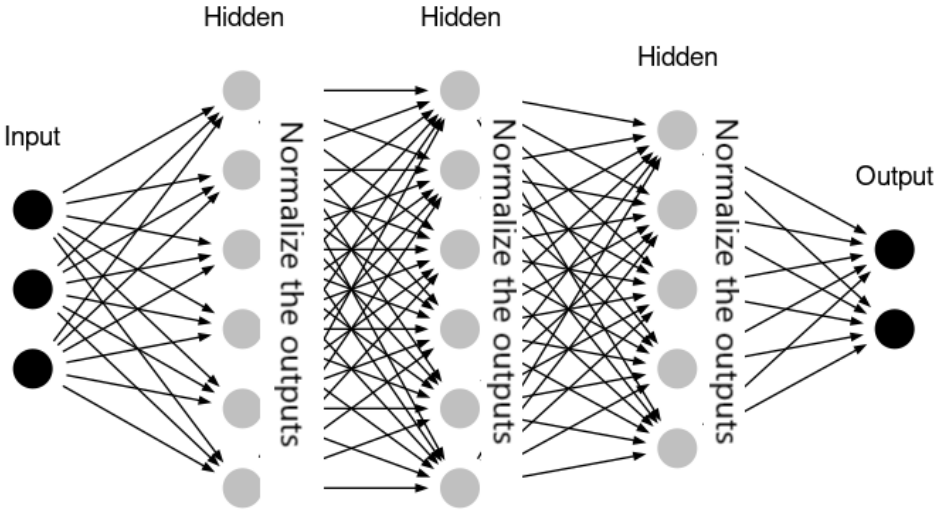


Ilustración 28 - Esquema Normalización por Lotes en CNN

- x_i son los valores de las neuronas que obtenemos tras hacer la convolución (se suele usar los valores antes de la función de la activación).
- γ, β son los valores que podemos hacer aprender al modelo, o los podemos fijar, para que sigan una determinada distribución. En caso de seguir la distribución estándar serían 1 y 0 respectivamente.

2.2.1.5.2. Evitar el sobreaprendizaje

Ya sabemos que es muy importante tener un modelo que generalice bien, esto se consigue no complicando el modelo para que se ajuste de una manera muy rígida a los datos de entrenamiento.

Para evitar estos casos en las Redes Neuronales Convoluciones se recurre a una técnica de regularización muy eficiente patentada por Google, denominada como **Dropout**.

Esta técnica consiste en darle una probabilidad determinada a todas las conexiones neuronales de una capa. Durante el aprendizaje, si obtenemos esa probabilidad en una conexión, esta será "desaparecerá" y haremos durante el aprendizaje como que no existe, no ajustando su peso.

Aplicando esto, obtenemos un modelo regularizado que nos permite dar mejores resultados con el conjunto de prueba, y además, hemos conseguido aumentar la velocidad de aprendizaje, ya que no tenemos que ajustar los valores de los pesos que han "desaparecido".

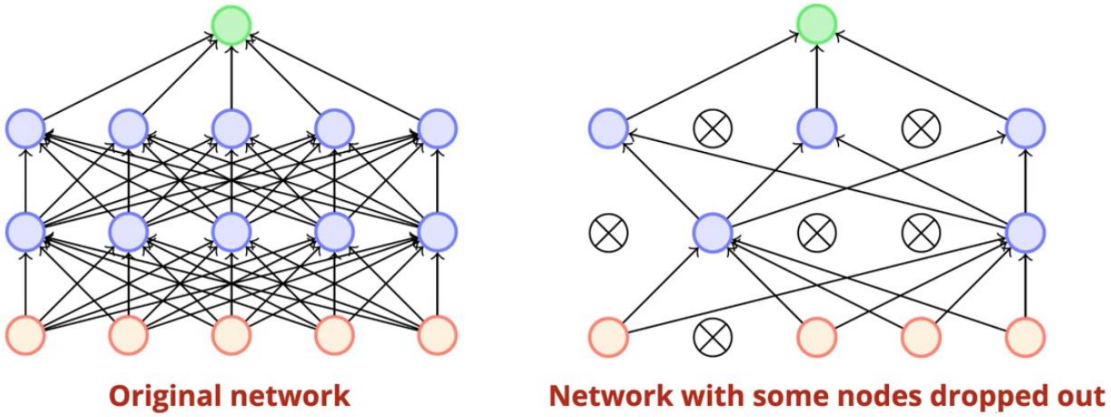


Ilustración 29 - Ejemplo de Dropout en red neuronal

2.3. Tecnologías y entorno de trabajo

Este trabajo lo hemos desarrollado en la plataforma **Google Colaboratory**. Es un proyecto de Google creado para ayudar a difundir la educación e investigación en machine learning. Es un Jupyter Notebook que no requiere ninguna instalación, ya que se ejecuta enteramente en la nube. Pudiendo entrenar nuestros modelos de forma gratuita en una GPU K80.

Para usar los modelos Machine Learning tradicionales usaremos la biblioteca **Scikit-learn**, biblioteca para el aprendizaje automático de software libre para **Python** escrita en C y Python, por lo que obtendremos un gran rendimiento. De esta librería usaremos **LinearSVC**, que explicaremos más tarde con detalle.

Ilustración 30 - Logo Scikit-learn

Para la extracción de características usaremos la biblioteca **Scikit-image**, biblioteca de software libre para el procesamiento de imágenes en Python, escrita también en C y Python. Está diseñada para funcionar con la biblioteca NumPy. De esta librería usaremos Histogram of Oriented Gradient (**HOG**), el cual será explicado a continuación.

Ilustración 31 - Logo Scikit-image

El uso de extensión de Python **NumPy** será esencial en nuestro proyecto. Agrega soporte para vectores y matrices, proporcionando una biblioteca de funciones matemáticas de alto nivel, implementada en C y Python.

Ilustración 32 - Logo NumPy

Para nuestros modelos de Deep Learning usaremos **Tensorflow 2.0**, una biblioteca de código abierto para aprendizaje automático desarrollado por Google. En esta versión 2.0 se incluye la biblioteca Keras, la cual es una API de lata nivel escrita en Python que corre sobre TensorFlow. Con ella, podremos construir redes neuronales profundas de una manera rápida y sencilla. Esta biblioteca permite entrenar y ejecutar los modelos en GPUs, por lo que reduciremos drásticamente el tiempo de aprendizaje para estos modelos.

Ilustración 33 - Logo TensorFlow

2.4. Análisis de resultados

A la hora de enfrentarnos a problemas de clasificación sucede que la precisión (aciertos del clasificador) no nos da toda la información de la calidad de un clasificador. Por ejemplo, si tuviésemos un modelo que nos dice si es cáncer o no, podríamos tener un 1% de error; es decir, acertamos si es o no cáncer un 99%. No obstante, si siempre decimos no cáncer y un 0.5% de los casos son de cáncer, el clasificador no es nada bueno.

Por eso, se suele usar una **matriz de confusión**. Para clasificación binaria, tiene la siguiente forma.

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

Ilustración 34 - Matriz de confusión para clasificación binaria

- VP: cantidad de positivos clasificados correctamente.
- VN: cantidad de negativos clasificados correctamente.
- FP: cantidad de negativos que se han clasificado como positivos.
- FN: cantidad de positivos clasificados como negativos.

De esto, diferenciaremos entre:

- **Precisión**: de todos los predichos como positivos, cuántos son correctos. $Precisión = \frac{VP}{VP + FP}$.
- **Recall**: de todos los que son positivos, cuántos hemos predicho como tal. $Recall = \frac{VP}{VP + FN}$.
- **Exactitud** (**Accuracy**): son las clases que ha predicho correctamente el clasificador. $Exactitud = \frac{VP + VN}{VP + VN + FP + FN}$

En nuestro caso, para las señales de tráfico tenemos 43 clases, la matriz de confusión no va a ser una tabla de 4x4. Lo que haremos será, por cada clase, calcular su precisión y recall. Por ejemplo, para la señal de 120 km/h, la **precisión** se calculará como las señales predichas y que son de 120 entre todas las señales que hemos predicho como 120. En cambio, el **recall** se calculará como las predichas y que son de 120 entre las todas las señales que son de 120 en el dataset.

Existe una medida que nos permite comparar la precisión respecto al recall. Se llama **Valor-F** (**F1-score**). Es una media armónica que combina los valores de precisión y recall.

$$F_1 Score = 2 \cdot \frac{Precisión \cdot Recall}{Precisión + Recall}$$

3. PROBLEMA DE CLASIFICACIÓN DE SEÑALES DE TRÁFICO

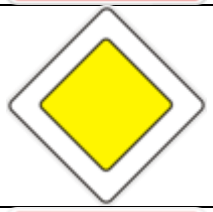
En este capítulo vamos a desarrollar y comparar algoritmos de clasificación de señales de tráfico. En primer lugar, mostraremos el conjunto de datos al que nos enfrentamos. A continuación, desarrollaremos los algoritmos utilizados y qué resultados han obtenido en el problema.

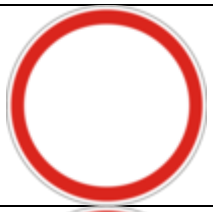
3.1. Conjunto de datos

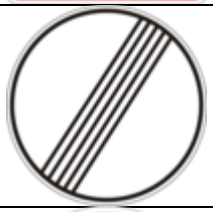
El conjunto de datos (**dataset**) que se va a usar para entrenar a los clasificadores es **GTSRB** (The German Traffic Sign Recognition Benchmark) [4].

Está formado por 39 209 imágenes en el conjunto de entrenamiento y por 12 631 en el conjunto de pruebas (test). Las señales están clasificadas en 43 clases, con la distribución que puede verse a continuación:

Código clase	Nombre señal	Número de imágenes	Señal
0	Límite velocidad a 20km/h	210	
1	Límite velocidad a 30km/h	2220	
2	Límite velocidad a 50km/h	2250	
3	Límite velocidad a 60km/h	1410	
4	Límite velocidad a 70km/h	1980	
5	Límite velocidad a 80km/h	1860	

6	Fin límite velocidad a 80 km/h	420	
7	Límite velocidad a 100km/h	1440	
8	Límite velocidad a 120km/h	1410	
9	Prohibido adelantar	1470	
10	Prohibido adelantar turismos por camiones	2010	
11	Peligro intersección	1320	
12	Calzada con prioridad	2100	
13	Ceda el paso	2160	
14	Stop	780	

15	Prohibido circulación	630	
16	Prohibido camiones	420	
17	Dirección prohibida	1110	
18	Precaución	1200	
19	Peligro curva a la izquierda	210	
20	Peligro curva a la derecha	360	
21	Peligro curvas peligrosas	330	
22	Peligro resalto por obras	390	
23	Peligro calzada deslizable	510	
24	Peligro estrechamiento de carril	270	

25	Peligro obras	1500	
26	Peligro semáforo	600	
27	Peligro paso de cebra	240	
28	Peligro cruce peatones	540	
29	Peligro bicicletas	270	
30	Peligro nieve	450	
31	Peligro animales salvajes	780	
32	Fin prohibiciones	240	
33	Obligatorio giro a la derecha	689	

34	Obligatorio giro a la izquierda	420	
35	Obligatorio recto	1200	
36	Obligatorio derecha o recto	390	
37	Obligatorio izquierda o recto	210	
38	Obligatorio paso a la derecha	2070	
39	Obligatorio paso a la izquierda	300	
40	Obligatorio intersección sentido giro	360	
41	Fin prohibido adelantar	240	
42	Fin prohibido adelantar camiones	240	

Tabla 4 - Clases de señales GTSRB

3.2. Machine Learning tradicional

Para una primera aproximación a nuestro problema, vamos a utilizar aprendizaje automático tradicional. La extracción de características se hará a través de HOG y para el modelo de reconocimiento se empleará SVMs.

3.2.1. Histograma de gradientes orientados (HOG)

A la hora construir el modelo de aprendizaje automático tradicional, tenemos que pensar qué datos vamos a usar para construir el modelo. Para este problema, disponemos imágenes en las hay una señal, las cuales queremos reconocer diciendo qué tipo de señal es.

Una imagen se puede representar como una matriz donde cada elemento es la intensidad del píxel en esa posición. Si la imagen es a color, está formada por 3 matrices, cada una representa un color del **RGB** (Red Green Blue).

Para desarrollar un modelo de aprendizaje automático podríamos hacer que cada píxel fuera un parámetro de entrada. Sin embargo, esto no es una buena idea, ya que una imagen de mucha resolución – por ejemplo, 1920x1080 tendría 2 073 600 parámetros de entrada – tendríamos que poner un peso por cada píxel, lo que haría muy ineficiente el modelo. Además, las imágenes tienen unas características especiales, y es que los píxeles adyacentes normalmente tienen una intensidad muy parecida.

La **extracción de características** nos permite reducir la dimensión de los datos de entrada, permitiendo así hacer un modelo más eficiente a la hora de predecir la clase a la que pertenece la imagen. Como es obvio, la tasa de éxito del clasificador depende totalmente de lo buena que es dicha extracción. Para cambiar la dimensionalidad de los píxeles de una imagen se utilizan descriptores de imágenes.

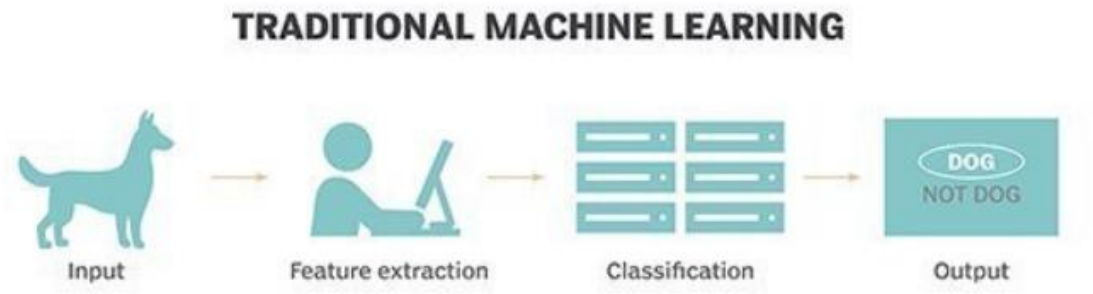


Ilustración 35 - Aprendizaje Automático en clasificación de imágenes

HOG (Histogram of Oriented Gradients) es un **descriptor visual** usado en procesamiento de imágenes y visión por ordenador, destinado a la detección de objetos. Esta técnica cuenta las ocurrencias de una orientación del gradiente en determinadas partes de la imagen.

Una imagen está compuesta por bloques, los cuales están formados por celdas. Una celda puede pertenecer a más de un bloque. Las celdas están formadas por píxeles adyacentes.

Usamos las celdas para calcular los gradientes de esos píxeles, elegimos los gradientes más frecuentes en cada una de las celdas. Con el fin de normalizar los resultados y así poder obtener menos varianza provocada por la iluminación y las sombras se normalizan los distintos bloques.

Los parámetros que determinan el número de dimensión final del descriptor HOG.

- Tamaño de celdas.
- Gradiente con signo o sin signo.

- Número de intervalos del histograma. Es decir, en cuántas partes discretizaremos los gradientes.
- Celdas que colocamos a cada uno de los bloques.

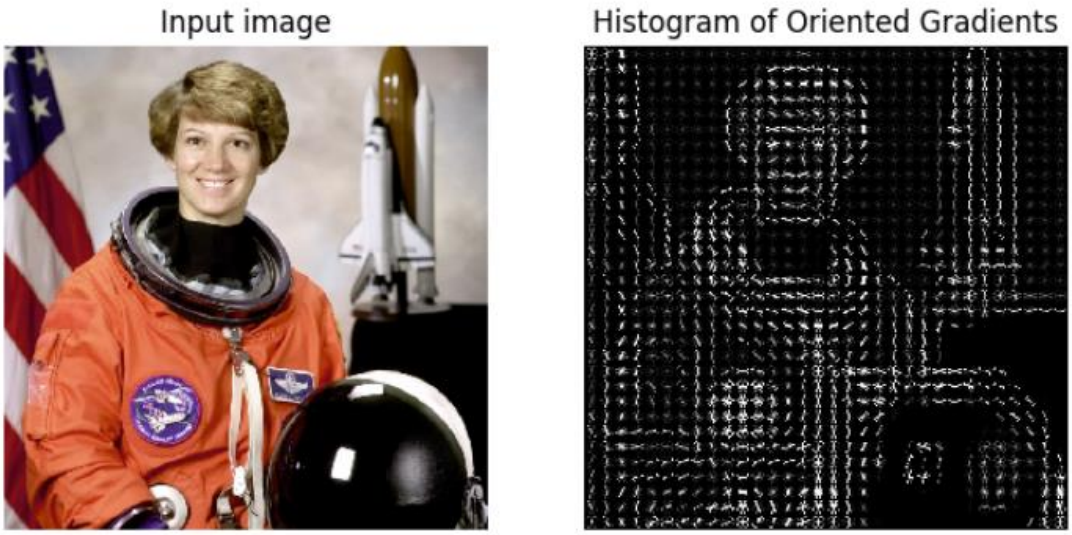


Ilustración 36 - Ejemplo de HOG

Hemos elegido HOG como extractor de características ya que es el que mejor resultados experimentales ha dado en tareas de clasificación de imágenes.

3.2.2. Máquinas de Soporte Vectorial (SVM)

La máquina de vectores soporte (Support Vector Machine, **SVM**) son un conjunto de algoritmos de aprendizaje supervisado. Están orientados a problemas de aprendizaje supervisado, como clasificación y regresión.

La idea del SVM es conseguir un clasificador de margen óptimo, es decir, que sea capaz de separar clases con el mínimo riesgo. Por lo tanto, conseguimos tener una buena generalización. Esto se consigue diferenciando 2 clases en un **hiperplano** con un espacio de separación lo más amplio posible.

Los ejemplos que definen el margen de separación son llamados **vectores soporte**. Son los únicos utilizados a la hora de construir el hiperplano óptimo.

El parámetro que controla la regularización es C.

- **C alto**: no se regulariza. Es decir, aumenta el riesgo de sobreaprendizaje. Intentaremos obtener un hiperplano que nos separe todos los ejemplos.
- **C bajo**: podría no ajustarse bien ni a los datos de entrenamiento.

SVM no devuelve una probabilidad, es un clasificador discriminativo.

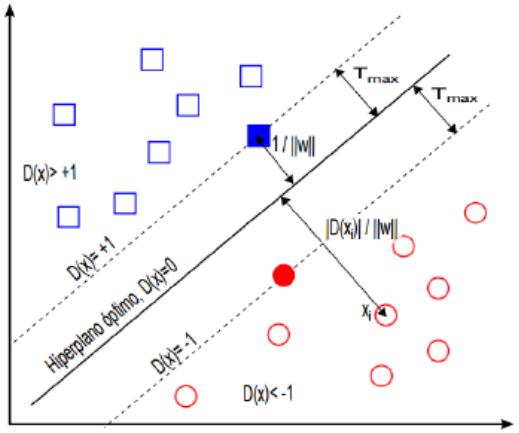


Ilustración 37 – SVM, obtención de hiperplano óptimo de separación de clases

No obstante, muchas veces no es posible separar las clases de una manera tan clara. Ya que puede ser que un SVM deba trata con curvas no lineales de separación. Debido esta limitación usamos funciones **Kernel**, la cual nos permite mapear el espacio de características inicial a un nuevo espacio de características de mayor dimensión, pero que permite hacer una separación más simple.

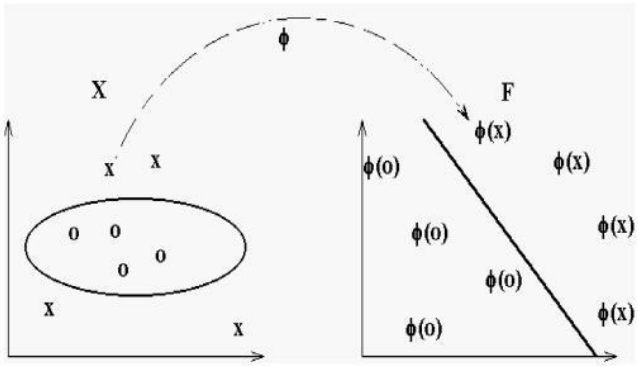


Ilustración 38 - Ejemplo de función Kernel

SVM solo sirve para clasificación binomial. Sin embargo, nuestro problema de señales de tráfico consta de 43 clases. Para ello, usaremos el método de uno contra todos (**One vs. All**, **OVA**), es decir, creamos tantos modelos SVM como clases. Teniendo, por ejemplo, un modelo que sea señal 120km/h vs resto. Si dice que se ha predicho la señal 120km/h, diremos que es señal 120. Habría otra opción, la cual sería implementar el uno contra uno. En cambio, esto no es inviable computacionalmente, ya que habría que hacer $\frac{n(n-1)}{2}$ modelos, es decir, 903 para nuestro problema.

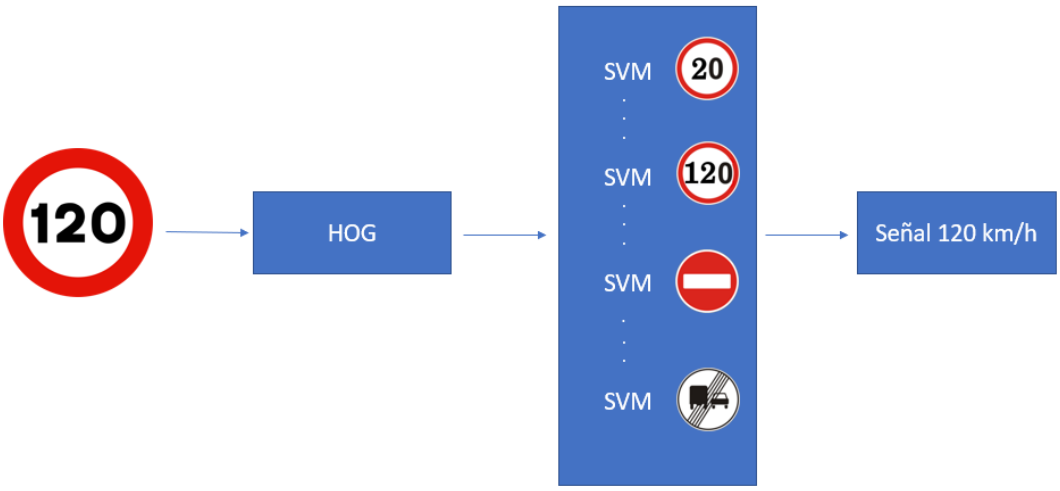


Ilustración 39 - Esquema ML tradicional para GTSRB

3.2.3. Clasificando señales de tráfico

Para resolver nuestro problema vamos a usar las clases **HOG** de Scikit-image y **LinearSVC** de Scikit-learn. A continuación, vamos a presentar los parámetros que vamos a probar para cada uno de ellos. Para terminar, presentaremos los resultados según la distinta combinación de configuraciones.

3.2.3.1. Parámetros HOG

Los parámetros de HOG que vamos a usar para obtener la mejor configuración posible son:

- **Orientación**: representa las orientaciones en las que vamos a discretizar los distintos gradientes.
- **Píxeles por celda**: serán los píxeles adyacentes por los que estarán compuestos las celdas.
- **Celdas por bloque**: a la hora de normalizar los valores con el fin de que la luz no perturbe los resultados, hacemos una normalización a nivel de bloque. Con este valor decimos cuántas celdas vamos a usar para normalizar sus valores por cada bloque. Recordamos que una celda puede estar en distintos bloques.

3.2.3.2. Parámetros SVM

Los parámetros que vamos a usar de la función **LinearSVC** (de Scikit-learn), modelo de SVM en el que el kernel que usa es **Lineal**, son:

- **C**: el parámetro de regularización. Cuanto más grande, más probabilidad de sobreaprendizaje.
- **Peso de clase**: este parámetro siempre será el mismo valor. Lo usamos ya que nuestro dataset no tiene para cada clase el mismo número de ejemplo, por lo que al tener una proporción podemos hacer un error ponderado.

3.2.3.3. Resultados

Vamos a fijar la configuración de HOG y analizaremos los resultados de SVMs con distintos parámetros.

Los resultados de **precisión**, **recall** y **f1-score** son la **media absoluta** para cada una de las señales.

3.2.3.3.1 Primera configuración HOG

Obtenemos por cada imagen un vector de características con la siguiente configuración:

- **Orientaciones**: 9.
- **Píxeles por celdas**: (8, 8).
- **Celdas por bloque**: (3, 3).

Tenemos, por cada imagen, un vector de **324 características** a partir de un dataset de 32x32x3 (3 072) de resolución.

3.2.3.3.1.1. Resultados en el conjunto de entrenamiento

Iteraciones	C	Precisión	Recall	F1-Score	Exactitud (Accuracy)
10 000	0.1	0.81	0.87	0.84	0.84
10 000	1	0.83	0.88	0.85	0.85
10 000	10	0.86	0.90	0.87	0.86
1 000	0.1	0.81	0.87	0.84	0.84
1 000	1	0.83	0.88	0.85	0.85
1 000	10	0.86	0.90	0.87	0.86

Tabla 5 - Resultados en train con configuración 1 de HOG para SVM

3.2.3.3.1.2. Resultados en el conjunto de test

Iteraciones	C	Precisión	Recall	F1-Score	Exactitud (Accuracy)
10 000	0.1	0.70	0.73	0.71	0.76
10 000	1	0.71	0.72	0.71	0.76
10 000	10	0.70	0.71	0.70	0.75
1 000	0.1	0.70	0.73	0.71	0.76
1 000	1	0.71	0.72	0.71	0.76
1 000	10	0.70	0.71	0.70	0.75

Tabla 6 - Resultados en test con configuración 1 de HOG para SVM

Como hemos podido comprobar, si fijamos C = 10, la predicción en train da mejores resultados. Sin embargo, en test funciona peor. Por lo tanto, donde el modelo da mejores resultados es en C = 1 o C = 0.1. Asimismo, con 1 000 iteraciones es suficiente, la calidad del clasificador no mejora; por eso, a partir de ahora vamos a usar solo 1 000 iteraciones para aprender el clasificador.

3.2.3.3.2. Segunda configuración HOG

Obtenemos por cada imagen un vector de características con la siguiente configuración:

- **Orientaciones**: 9.
- **Píxeles por celdas**: (4, 4).
- **Celdas por bloque**: (3, 3).

Tenemos, por cada imagen, un vector de **2 916 características** a partir de un dataset de 32x32x3 (3 072) de resolución. Podemos observar que al final hemos obtenido más características de las originales. Vamos a evaluar cuáles son los resultados que obtenemos.

3.2.3.3.2.1. Resultados en el conjunto de entrenamiento

C	Precisión	Recall	F1-Score	Exactitud (Accuracy)
0.1	0.98	0.99	0.98	0.97
1	0.99	0.99	0.99	0.99
10	1.00	1.00	1.00	1.00

Tabla 7 - Resultados en train con configuración 2 de HOG para SVM

3.2.3.3.2.2. Resultados en el conjunto de test

C	Precisión	Recall	F1-Score	Exactitud (Accuracy)
0.1	0.86	0.86	0.86	0.88
1	0.85	0.85	0.84	0.87
10	0.83	0.83	0.83	0.86

Tabla 8 - Resultados en test con configuración 2 de HOG para SVM

3.2.3.3.3. Tercera configuración HOG

Obtenemos por cada imagen un vector de características con la siguiente configuración:

- **Orientaciones**: 9.
- **Píxeles por celdas**: (8, 8).
- **Celdas por bloque**: (4, 4).

Por imagen, hemos obtenido un vector de **144 características** a partir de 32x32x3 (3 072). Podemos comprobar que, al poner más celdas por bloques, se reducen el número de parámetros.

3.2.3.3.3.1. Resultados en el conjunto de entrenamiento

C	Precisión	Recall	F1-Score	Exactitud (Accuracy)
0.1	0.79	0.85	0.81	0.82
1	0.82	0.87	0.84	0.84
10	0.83	0.87	0.85	0.84

Tabla 9 - Resultado en train con configuración 3 de HOG para SVM

3.2.3.3.3.2 Resultados en el conjunto de test

C	Precisión	Recall	F1-Score	Exactitud (Accuracy)
0.1	0.68	0.72	0.69	0.75
1	0.70	0.72	0.70	0.75
10	0.70	0.71	0.70	0.75

Tabla 10 - Resultado en train con configuración 3 de HOG para SVM

3.2.3.3.4. Cuarta configuración HOG

Obtenemos por cada imagen un vector de características con la siguiente configuración:

- **Orientaciones**: 9.
- **Píxeles por celdas**: (4, 4).
- **Celdas por bloque**: (4, 4).

Por imagen, disponemos de **3 600 características**, obtenidas a partir de unas de 32x32x3 (3 072).

3.2.3.3.4.1. Resultados en el conjunto de entrenamiento

C	Precisión	Recall	F1-Score	Exactitud (Accuracy)
0.1	0.98	0.98	0.98	0.97
1	0.99	0.99	0.99	0.98
10	1.00	1.00	1.00	0.99

Tabla 11 - Resultado en train con configuración 4 de HOG para SVM

3.2.3.3.4.2. Resultados en el conjunto de test

C	Precisión	Recall	F1-Score	Exactitud (Accuracy)
0.1	0.87	0.87	0.87	0.89
1	0.86	0.86	0.86	0.88
10	0.84	0.85	0.84	0.87

Tabla 12 - Resultado en test con configuración 4 de HOG para SVM

3.2.3.3.5. Quinta configuración HOG

Obtenemos por cada imagen un vector de características con la siguiente configuración:

- **Orientaciones**: 5.
- **Píxeles por celdas**: (8, 8).
- **Celdas por bloque**: (3, 3).

Por imagen, hemos extraído de **180 características**.

3.2.3.3.5.1. Resultados en el conjunto de entrenamiento

C	Precisión	Recall	F1-Score	Exactitud (Accuracy)
0.1	0.74	0.80	0.77	0.78
1	0.76	0.82	0.78	0.79
10	0.78	0.83	0.80	0.81

Tabla 13 - Resultado en train con configuración 5 de HOG para SVM

3.2.3.3.5.2. Resultados en el conjunto de entrenamiento

C	Precisión	Recall	F1-Score	Exactitud (Accuracy)
0.1	0.65	0.68	0.66	0.72
1	0.65	0.67	0.65	0.71
10	0.65	0.67	0.65	0.71

Tabla 14 - Resultado en test con configuración 5 de HOG para SVM

3.2.3.3.6. Sexta configuración HOG

Obtenemos por cada imagen un vector de características con la siguiente configuración:

- **Orientaciones**: 5.
- **Píxeles por celdas**: (4, 4).
- **Celdas por bloque**: (3, 3).

Por imagen, hemos obtenido **1 620 características**.

3.2.3.3.6.1. Resultados en el conjunto de entrenamiento

C	Precisión	Recall	F1-Score	Exactitud (Accuracy)
0.1	0.97	0.97	0.97	0.95
1	0.98	0.99	0.98	0.97
10	0.99	0.99	0.99	0.99

Tabla 15 - Resultado en train con configuración 6 de HOG para SVM

3.2.3.3.6.2. Resultados en el conjunto de test

C	Precisión	Recall	F1-Score	Exactitud (Accuracy)
0.1	0.85	0.85	0.85	0.87
1	0.85	0.84	0.84	0.86
10	0.83	0.83	0.83	0.86

Tabla 16 - Resultado en test con configuración 6 de HOG para SVM

3.2.3.3.7. Séptima configuración HOG

Obtenemos por cada imagen un vector de características con la siguiente configuración:

- **Orientaciones**: 11.
- **Píxeles por celdas**: (4, 4).
- **Celdas por bloque**: (3, 3).

Por imagen, hemos obtenido **396 características**.

3.2.3.3.7.1. Resultados en el conjunto de entrenamiento

C	Precisión	Recall	F1-Score	Exactitud (Accuracy)
0.1	0.83	0.87	0.85	0.84
1	0.85	0.89	0.87	0.86
10	0.87	0.91	0.89	0.87

Tabla 17 - Resultado en train con configuración 7 de HOG para SVM

3.2.3.3.7.2. Resultados en el conjunto de test

C	Precisión	Recall	F1-Score	Exactitud (Accuracy)
0.1	0.71	0.74	0.71	0.77
1	0.71	0.72	0.71	0.76
10	0.71	0.71	0.70	0.76

Tabla 18 - Resultado en test con configuración 7 de HOG para SVM

3.2.3.3.8. Octava configuración HOG

Obtenemos por cada imagen un vector de características con la siguiente configuración:

- **Orientaciones**: 11.
- **Píxeles por celdas**: (4, 4).
- **Celdas por bloque**: (3, 3).

Por imagen, hemos obtenido **3 564 características**.

3.2.3.3.8.1. Resultados en el conjunto de entrenamiento

C	Precisión	Recall	F1-Score	Exactitud (Accuracy)
0.1	0.98	0.99	0.99	0.98
1	0.99	0.99	0.99	0.99
10	1.00	1.00	1.00	1.00

Tabla 19 - Resultado en train con configuración 8 de HOG para SVM

3.2.3.3.8.2. Resultados en el conjunto de test

C	Precisión	Recall	F1-Score	Exactitud (Accuracy)
0.1	0.86	0.86	0.86	0.88
1	0.84	0.84	0.84	0.86
10	0.83	0.83	0.83	0.85

Tabla 20 - Resultado en test con configuración 8 de HOG para SVM

3.2.3.3.9. Sin configuración HOG

Vamos a usar los **3 072 píxeles** como características.

3.2.3.3.9.1. Resultados en el conjunto de entrenamiento

C	Precisión	Recall	F1-Score	Exactitud (Accuracy)
0.1	0.67	0.72	0.66	0.68
1	0.83	0.87	0.84	0.85
10	0.92	0.94	0.93	0.92
15	0.93	0.95	0.94	0.93
20	0.94	0.96	0.94	0.94

Tabla 21 - Resultado en train sin HOG para SVM

3.2.3.3.9.2. Resultados en el conjunto de test

C	Precisión	Recall	F1-Score	Exactitud (Accuracy)
0.1	0.58	0.60	0.55	0.61
1	0.73	0.77	0.73	0.78
10	0.81	0.82	0.81	0.85
15	0.82	0.82	0.82	0.86
20	0.83	0.83	0.82	0.86

Tabla 22 - Resultado en test sin HOG para SVM

3.2.3.4. Análisis de resultados

Podemos observar que tiene mucho mayor impacto en el rendimiento del modelo la extracción de características que cualquier configuración que podamos cambiar en el SVM, pudiendo pasar de un acierto del **72%** (5ª configuración) al **89%** (4ª configuración) en la elección de la mejor configuración para HOG.

Asimismo, la configuración de SMV que mejor resultado da es con **C = 0.1**, ya que de esta forma regulariza el modelo generalizando mejor para los resultados desconocidos. Además, hemos podido comprobar que a partir de 1000 iteraciones no se mejoran los resultados para el modelo.

La **configuración** de **HOG y SVM** con la que **mejores resultados** se obtienen, 89% de acierto, es con 9 orientaciones, 4 píxeles por celdas y 4 celdas por bloque a la hora de obtener las orientaciones en HOG (píxeles y celdas tanto en altura como en anchura). Y con C en 0.1. Hemos obtenido más características de las originales, de 3 072 a **3 600 características**. Esto es posible ya que hemos creado más bloques con celdas de píxeles que no son adyacentes, por lo que se aumentaba dicho número.

La **configuración** de **HOG y SVM** que **peores resultados** ha dado, **71%**, es con 5 orientaciones, 8 píxeles por celdas y 8 celdas por bloque. Con C = 1. Suceden aquí dos cosas, obtenemos tan solo **180 características** por señal a clasificar, teniendo 3 072 originales. Además, hacemos con ese valor de C que el modelo sobreaprenda, perdiendo capacidad de generalización.

También, es importante comentar los resultados obtenidos al no extraer características y usar los píxeles como valores para el SVM. Vemos que con **C = 0.1**, obtenemos en test unos resultados de **61%**, los cuales son los peores obtenidos. Sin embargo, si elegimos un valor **C = 15**, llevamos a obtener una exactitud del **86%**. Esto se debe a que los valores de píxeles son incómodos para aprender; ya que, en varias imágenes pueden distar mucho los valores de los píxeles, siendo de la misma clase. Por esta razón, al poner un valor de C alto, forzamos a que el modelo se adapte a esos cambios de intensidades.

En los resultados anteriores, al haber muchas configuraciones usadas no he creído conveniente poner los resultados de clasificación para cada clase señal. Sin embargo, para la mejor configuración de HOG y SVM sí que vamos a poner esos resultados, para poder tener una idea más pormenorizada de las diferencias con los modelos de Deep Learning que veremos a continuación.

	precision	recall	f1-score	support
Límite velocidad (20km/h)	0.63	0.77	0.69	60
Límite velocidad (30km/h)	0.87	0.82	0.84	720
Límite velocidad (50km/h)	0.81	0.87	0.84	750
Límite velocidad (60km/h)	0.79	0.82	0.80	450
Límite velocidad (70km/h)	0.95	0.92	0.93	660
Límite velocidad (80km/h)	0.73	0.73	0.73	630
Fin límite velocidad (80km/h)	0.89	0.76	0.82	150
Límite velocidad (100km/h)	0.83	0.87	0.85	450
Límite velocidad (120km/h)	0.78	0.84	0.81	450
No adelantar	0.90	0.91	0.91	480
No adelantar vehículos < 3.5 T	0.94	0.92	0.93	660
Intersección	0.91	0.78	0.84	420
Calzada con prioridad	0.98	0.99	0.98	690
Ceda el paso	0.99	0.99	0.99	720
Stop	0.98	0.93	0.95	270
Prohibido circulación	0.90	0.96	0.93	210
No pasar vehículos > 3.5 T	0.97	0.95	0.96	150
Dirección prohibida	0.99	0.96	0.98	360
Precaución	0.94	0.85	0.89	390
Curva peligrosa izquierda	0.84	0.90	0.87	60
Curva peligrosa derecha	0.81	0.93	0.87	90
Doble curvas peligrosas	0.92	0.84	0.88	90
Carretera con baches	0.91	0.87	0.89	120
Peligro carretera deslizante	0.84	0.83	0.83	150
Estrechamiento derecha	0.88	0.88	0.88	90
Peligro obras	0.83	0.91	0.87	480
Peligro semáforo	0.80	0.72	0.76	180
Paso de cebra	0.86	0.92	0.89	60
Colegio	0.81	0.90	0.85	150
Bicicletas	0.77	0.87	0.82	90
Precaución nieve/hielo	0.58	0.55	0.57	150
Animales salvajes	0.95	0.96	0.95	270
Fin prohibiciones	0.74	0.80	0.77	60
Girar a la derecha	0.95	0.94	0.94	210
Girar a la izquierda	0.93	0.97	0.95	120
Recto	0.99	0.92	0.95	390
Recto o derecha	0.93	0.97	0.95	120
Recto o izquierda	0.92	0.90	0.91	60
Mantenerse a la derecha	0.97	0.98	0.97	690
Mantenerse a la izquierda	0.89	1.00	0.94	90
Rotonda	0.84	0.94	0.89	90
Fin no adelantar	0.82	0.70	0.76	60
Fin no adelantar vehículos < 3.5 T	0.87	0.77	0.82	90
accuracy			0.89	12630
macro avg	0.87	0.87	0.87	12630
weighted avg	0.89	0.89	0.89	12630

Ilustración 40 - Resultados del mejor clasificador SVM con HOG

3.3. Deep Learning

En este capítulo vamos a dar un paso adelante y vamos a desarrollar un nuevo clasificador automático de imágenes basado en deep learning en lugar de utilizar un extractor de características y un modelo de clasificación.

Como hemos explicado anteriormente, las Redes Neuronales Convolucionales son, en teoría, la mejor alternativa para crear modelos de aprendizaje automático en la clasificación y detección de imágenes.

Empezaremos presentando una red neuronal convolucional hecha por mí – la llamares CNN simple – para comparar de forma más directa la diferencia entre el modelo tradicional y uno convolucional. Después, utilizaremos modelos famosos que suelen dar muy buenos resultados experimentales, estos serán LexNet-5, VGG-16, VGG-19, y ResNet-50.

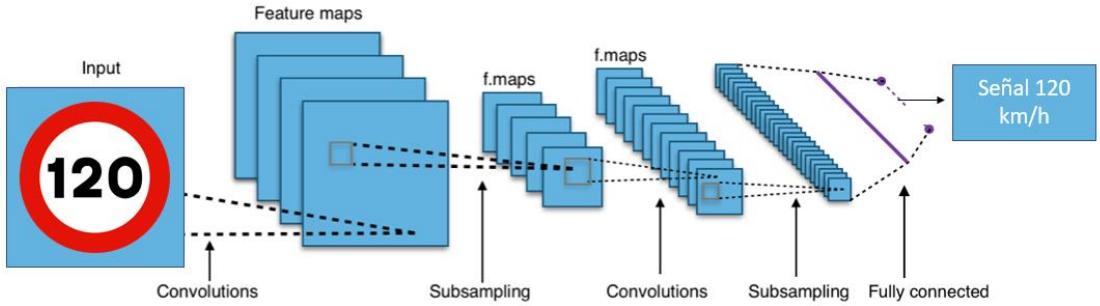


Ilustración 41 - Esquema CNN para GTSRB

Los **hiperparámetros** que vamos a emplear para esta parte son:

- **Número de épocas**: 30.
- **Factor de aprendizaje** (**alfa**): 0.001.
- **Tamaño de lote**: 64 imágenes.

Con el número de ejemplo que tenemos, vamos a acabar haciendo 612 pasadas hasta hacer una época, es decir, recorrer complemente el dataset de aprendizaje.

También, para aumentar el número de ejemplos, disponemos de una función de TensorFlow con la que podemos aumentar un poco más el número de ejemplo, variando ligeramente las imágenes que usamos para entrenar rotándolas levemente.

3.3.1. CNN simple

He creado para resolver la clasificación de señales de tráfico un modelo CNN simple. Este no tiene ninguna configuración en espacial, simplemente sigue los fundamentos de una Red Neuronal Convolucional.

Puede resultar interesante probar este modelo, ya que más tarde probaremos otros que han resultado muy buenos para la clasificación de imágenes. Así que, podremos comprar el rendimiento de una ConvNet estándar con estos más avanzados, y también ver la diferencia de rendimiento que puede haber entre una CNN simple y un modelo de aprendizaje automático tradicional.

La red, estará compuesta por **3 Capas Convolucionales**.

- En la primera usaremos **8 filtros** de tamaño **3x3** con relleno (padding, same convolution). Es decir, la imagen no se reducirá de tamaño. Aplicaremos la función de activación **RELU**, haremos una **Normalización por Lotes** y para terminar haremos **MaxPooling 2x2**.
- La segunda estará compuesta por **16 filtros** de tamaño **3x3**. En el resto, la misma configuración.
- En la tercera capa vamos a poner **32** filtros **3x3**. El resto de la configuración igual.

Las últimas **2 capas Full-Connected** tendrán la siguiente configuración: **128 neuronas** con activación **RELU**. Haremos **Normalización por Lotes** y **Dropout** de 0.5 de probabilidad.

La capa de **salida** será evidentemente, una con **43 neuronas de salida**, con la función de activación **Softmax**.

A continuación, se muestra el resumen del modelo:

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 32, 32, 8)         224
_________________________________________________________________
activation (Activation)      (None, 32, 32, 8)         0
_________________________________________________________________
batch_normalization (BatchNo (None, 32, 32, 8)         32
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 16, 16, 8)         0
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 16, 16, 16)        1168
_________________________________________________________________
activation_1 (Activation)    (None, 16, 16, 16)        0
_________________________________________________________________
batch_normalization_1 (Batch (None, 16, 16, 16)        64
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 8, 8, 16)          0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 8, 8, 32)          4640
_________________________________________________________________
activation_2 (Activation)    (None, 8, 8, 32)          0
_________________________________________________________________
batch_normalization_2 (Batch (None, 8, 8, 32)          128
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 4, 4, 32)          0
_________________________________________________________________
flatten (Flatten)            (None, 512)               0
_________________________________________________________________
dense (Dense)                (None, 128)               65664
_________________________________________________________________
activation_3 (Activation)    (None, 128)               0
_________________________________________________________________
batch_normalization_3 (Batch (None, 128)               512
_________________________________________________________________
dropout (Dropout)            (None, 128)               0
_________________________________________________________________
flatten_1 (Flatten)          (None, 128)               0
_________________________________________________________________
dense_1 (Dense)              (None, 128)               16512
_________________________________________________________________
activation_4 (Activation)    (None, 128)               0
_________________________________________________________________
batch_normalization_4 (Batch (None, 128)               512
```

```
_________________________________________________________________
dropout_1 (Dropout)          (None, 128)               0
_________________________________________________________________
dense_2 (Dense)              (None, 43)                5547
_________________________________________________________________
activation_5 (Activation)    (None, 43)                0
=================================================================
Total params: 95,003
Trainable params: 94,379
Non-trainable params: 624
```

3.3.1.1 Resultados

A continuación, se muestran los resultados obtenidos con la Red Neuronal Convolucional más simple.

	Precisión	**Recall**	**F1-Score**	**Exactitud (Accuracy)**
Train	0.95	0.94	0.94	0.95
Test	0.89	0.88	0.88	0.91

Tabla 23 - Resultados de CNN Simple

3.3.2. LeNet-5

El modelo de LeNet-5 original empezaría con una imagen en blanco y negro, sin embargo, en nuestro caso haremos que funcione con una imagen a color para que no esté en clara desventaja con los demás modelos.

Los saltos en la convolución son de 1.

El sufijo 5 es porque usamos siempre filtros de tamaño 5x5.

Compuesta por **2 Capas Convolucionales**:

- La 1ª capa usa **6 filtros** de tamaño **5x5**. Usamos **convolución válida (valid convolution)**, es decir, se reduce el tamaño de la imagen 32x32 a 28x28. La función de activación que usaremos es una **RELU**. Hacemos **MaxPooling de 2x2**.
- En la 2ª capa usamos 16 filtros de tamaño 5x5. En lo demás es la misma configuración que la capa anterior.

A continuación, creamos 2 capas Full-Connected.

- 1ª capa tendrá **120 neuronas**, con activación **RELU**, **Normalización por Lotes** y **Dropout** de 0.5.
- La 2ª capa igual, pero con **84 neuronas**.

La capa de salida se trate, evidentemente, de una capa con 43 neuronas y **Softmax**.

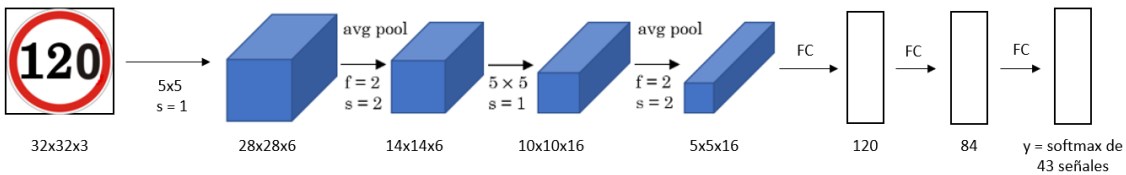


Ilustración 42 - Esquema de LeNet 5 para el dataset GTSRB (cambiar ilustración)

El modelo:

```
Model: "lenet5"
_________________________________________________________________
```

```
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_16 (Conv2D)           (None, 28, 28, 6)         456
_________________________________________________________________
activation_22 (Activation)   (None, 28, 28, 6)         0
_________________________________________________________________
batch_normalization_20 (Batc (None, 28, 28, 6)         24
_________________________________________________________________
max_pooling2d_6 (MaxPooling2 (None, 14, 14, 6)         0
_________________________________________________________________
conv2d_17 (Conv2D)           (None, 10, 10, 16)        2416
_________________________________________________________________
activation_23 (Activation)   (None, 10, 10, 16)        0
_________________________________________________________________
batch_normalization_21 (Batc (None, 10, 10, 16)        64
_________________________________________________________________
max_pooling2d_7 (MaxPooling2 (None, 5, 5, 16)          0
_________________________________________________________________
flatten_4 (Flatten)          (None, 400)               0
_________________________________________________________________
dense_6 (Dense)              (None, 120)               48120
_________________________________________________________________
activation_24 (Activation)   (None, 120)               0
_________________________________________________________________
batch_normalization_22 (Batc (None, 120)               480
_________________________________________________________________
dropout_4 (Dropout)          (None, 120)               0
_________________________________________________________________
flatten_5 (Flatten)          (None, 120)               0
_________________________________________________________________
dense_7 (Dense)              (None, 84)                10164
_________________________________________________________________
activation_25 (Activation)   (None, 84)                0
_________________________________________________________________
batch_normalization_23 (Batc (None, 84)                336
_________________________________________________________________
dropout_5 (Dropout)          (None, 84)                0
_________________________________________________________________
dense_8 (Dense)              (None, 43)                3655
_________________________________________________________________
activation_26 (Activation)   (None, 43)                0
=================================================================
Total params: 65,715
Trainable params: 65,263

Non-trainable params: 452
```

3.3.2.1 Resultados

	Precisión	**Recall**	**F1-Score**	**Exactitud (Accuracy)**
Train	0.98	0.97	0.97	0.97
Test	0.92	0.91	0.91	0.93

Tabla 24 - Resultados de LeNet 5

3.3.3. VGG-16

Este modelo de Red Neuronal Convolucional se centra en reducir el número de hiperparámetros, pero proporciona una red lo suficiente profunda que la hace capaz de abstraerse de los píxeles conforme pasan las capas.

Las configuraciones de las **Capas Convolucionales** son:

- La 1ª parte consta de 2 capas con **64 filtros 3x3**, relleno (**same convolution**), activación **RELU** y **Normalización por Lotes**. Otra capa exactamente igual. Para terminar, haríamos un **MaxPooling de 2x2**.
- La 2ª parte tiene la misma configuración de la parte anterior, pero usando **128 filtros**.
- La 3ª parte igual, pero con **256 filtros**.
- La 4ª parte con **512 filtros**, pero esta vez repetimos esto 3 veces en vez de 2.
- La 5ª parte igual que la parte 4, aplicamos **512 filtros** 3 veces.

Tras esto, como siempre, vendrían las **FC** (Full-Connected):

- 2 capas con **4 096 neuronas**, **Normalización por Lotes** y **Dropout** de 0.5.

La capa de salida, como siempre, 43 neuronas de salida con **Softmax**.

El siguiente dibujo muestra una modelo VGG-16 para imágenes de 224x224x3. En nuestro caso serían de 32x32x3.

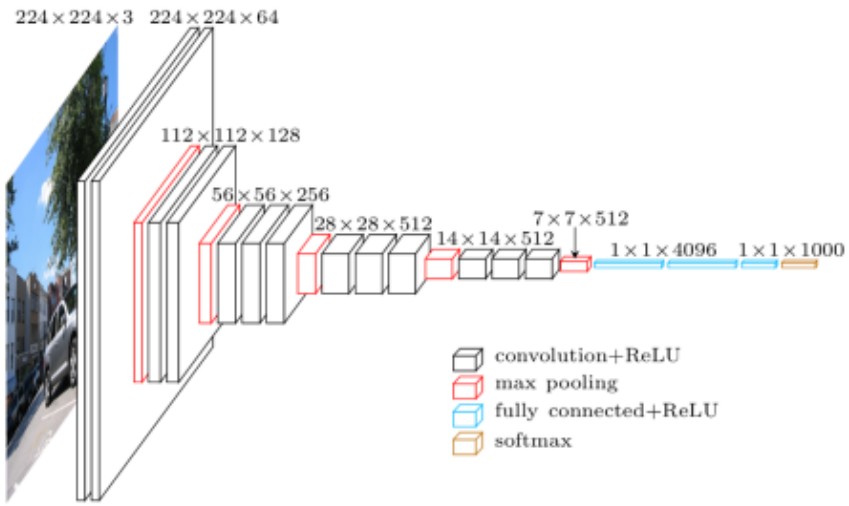


Ilustración 43 - VGG 16 para imágenes de 224x224 y salida de 1000 clases

Los parámetros detallados de la red se muestran a continuación:

```
Model: "vgg16"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_3 (Conv2D)            (None, 32, 32, 64)        1792
_________________________________________________________________
activation_6 (Activation)    (None, 32, 32, 64)        0
_________________________________________________________________
batch_normalization_5 (Batch (None, 32, 32, 64)        256
_________________________________________________________________
conv2d_4 (Conv2D)            (None, 32, 32, 64)        36928
_________________________________________________________________
activation_7 (Activation)    (None, 32, 32, 64)        0
_________________________________________________________________
batch_normalization_6 (Batch (None, 32, 32, 64)        256
_________________________________________________________________
max_pooling2d_3 (MaxPooling2 (None, 16, 16, 64)        0
_________________________________________________________________
conv2d_5 (Conv2D)            (None, 16, 16, 128)       73856
_________________________________________________________________
activation_8 (Activation)    (None, 16, 16, 128)       0
_________________________________________________________________
```

```
batch_normalization_7 (Batch (None, 16, 16, 128)       512
_________________________________________________________________
conv2d_6 (Conv2D)            (None, 16, 16, 128)       147584
_________________________________________________________________
activation_9 (Activation)    (None, 16, 16, 128)       0
_________________________________________________________________
batch_normalization_8 (Batch (None, 16, 16, 128)       512
_________________________________________________________________
max_pooling2d_4 (MaxPooling2 (None, 8, 8, 128)         0
_________________________________________________________________
conv2d_7 (Conv2D)            (None, 8, 8, 256)         295168
_________________________________________________________________
activation_10 (Activation)   (None, 8, 8, 256)         0
_________________________________________________________________
batch_normalization_9 (Batch (None, 8, 8, 256)         1024
_________________________________________________________________
conv2d_8 (Conv2D)            (None, 8, 8, 256)         590080
_________________________________________________________________
activation_11 (Activation)   (None, 8, 8, 256)         0
_________________________________________________________________
batch_normalization_10 (Batc (None, 8, 8, 256)         1024
_________________________________________________________________
conv2d_9 (Conv2D)            (None, 8, 8, 256)         590080
_________________________________________________________________
activation_12 (Activation)   (None, 8, 8, 256)         0
_________________________________________________________________
batch_normalization_11 (Batc (None, 8, 8, 256)         1024
_________________________________________________________________
conv2d_10 (Conv2D)           (None, 8, 8, 512)         1180160
_________________________________________________________________
activation_13 (Activation)   (None, 8, 8, 512)         0
_________________________________________________________________
batch_normalization_12 (Batc (None, 8, 8, 512)         2048
_________________________________________________________________
conv2d_11 (Conv2D)           (None, 8, 8, 512)         2359808
_________________________________________________________________
activation_14 (Activation)   (None, 8, 8, 512)         0
_________________________________________________________________
batch_normalization_13 (Batc (None, 8, 8, 512)         2048
_________________________________________________________________
conv2d_12 (Conv2D)           (None, 8, 8, 512)         2359808
_________________________________________________________________
activation_15 (Activation)   (None, 8, 8, 512)         0
_________________________________________________________________
batch_normalization_14 (Batc (None, 8, 8, 512)         2048
_________________________________________________________________
max_pooling2d_5 (MaxPooling2 (None, 4, 4, 512)         0
_________________________________________________________________
conv2d_13 (Conv2D)           (None, 4, 4, 512)         2359808
_________________________________________________________________
activation_16 (Activation)   (None, 4, 4, 512)         0
_________________________________________________________________
batch_normalization_15 (Batc (None, 4, 4, 512)         2048
_________________________________________________________________
conv2d_14 (Conv2D)           (None, 4, 4, 512)         2359808
_________________________________________________________________
activation_17 (Activation)   (None, 4, 4, 512)         0
_________________________________________________________________
batch_normalization_16 (Batc (None, 4, 4, 512)         2048
_________________________________________________________________
conv2d_15 (Conv2D)           (None, 4, 4, 512)         2359808
_________________________________________________________________
activation_18 (Activation)   (None, 4, 4, 512)         0
_________________________________________________________________
batch_normalization_17 (Batc (None, 4, 4, 512)         2048
_________________________________________________________________
```

```
flatten_2 (Flatten)          (None, 8192)              0
_________________________________________________________________
dense_3 (Dense)              (None, 4096)              33558528
_________________________________________________________________
activation_19 (Activation)   (None, 4096)              0
_________________________________________________________________
batch_normalization_18 (Batc (None, 4096)              16384
_________________________________________________________________
dropout_2 (Dropout)          (None, 4096)              0
_________________________________________________________________
flatten_3 (Flatten)          (None, 4096)              0
_________________________________________________________________
dense_4 (Dense)              (None, 4096)              16781312
_________________________________________________________________
activation_20 (Activation)   (None, 4096)              0
_________________________________________________________________
batch_normalization_19 (Batc (None, 4096)              16384
_________________________________________________________________
dropout_3 (Dropout)          (None, 4096)              0
_________________________________________________________________
dense_5 (Dense)              (None, 43)                176171
_________________________________________________________________
activation_21 (Activation)   (None, 43)                0
=================================================================
Total params: 65,280,363
Trainable params: 65,255,531
Non-trainable params: 24,832
_________________________________________________________________
None
```

3.3.3.1. Resultados

Los resultados son los siguientes:

	Precisión	Recall	F1-Score	Exactitud (Accuracy)
Train	1.00	1.00	1.00	1.00
Test	0.96	0.94	0.95	0.97

Tabla 25 - Resultados de VGG-16

3.3.4. VGG-19

VGG-19 es una Red Neuronal Convolucional con unan estructura muy similar a la VGG-16. Se diferencia únicamente en tener 19 capas, en vez de 19.

Las configuraciones de las **Capas Convolucionales** son:

- En una 1ª parte 2 capas convolucionales, **64 filtros** 3x3 con relleno, **Normalización por Lotes**, **RELU** y tras esas **2 capas**, una de MaxPooling de 2x2.
- La 2ª parte igual que la anterior, pero con **128 filtros**.
- La 3ª parte hacemos ponemos **4 capas** convolucionales y **256 filtros**, hacemos **MaxPooling** de 2x2.
- La 4ª parte y 5ª parte consiste en hacer lo mismo, pero con **512 filtros**.

La parte FC de la Red Neuronal Convolucional son, como en la VGG-16, 2 capas de 4096 neuronas.

Para terminar, una capa de 43 neuronas con **SoftMax**.

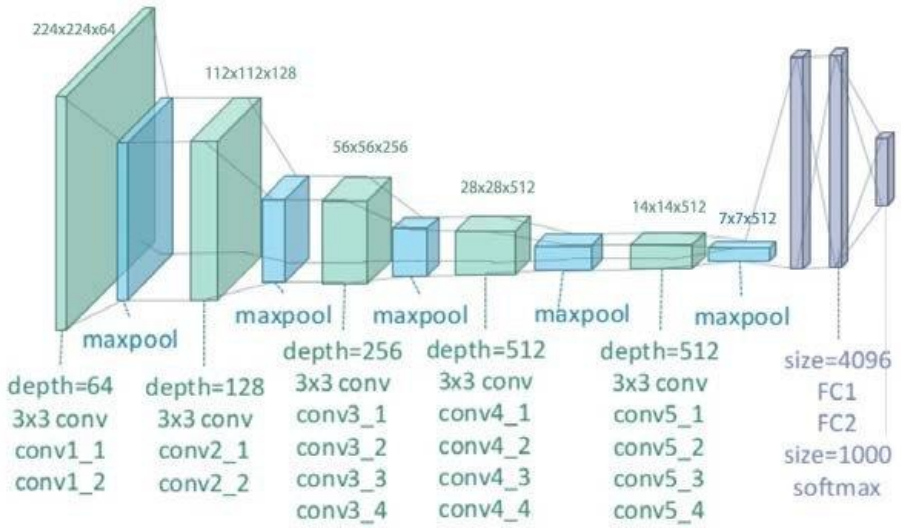


Ilustración 44 - Modelo VGG-19

Los parámetros que forman la red:

```
Model: "vgg19"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_18 (Conv2D)           (None, 32, 32, 64)        1792
_________________________________________________________________
activation_27 (Activation)   (None, 32, 32, 64)        0
_________________________________________________________________
batch_normalization_24 (Batc (None, 32, 32, 64)        256
_________________________________________________________________
conv2d_19 (Conv2D)           (None, 32, 32, 64)        36928
_________________________________________________________________
activation_28 (Activation)   (None, 32, 32, 64)        0
_________________________________________________________________
batch_normalization_25 (Batc (None, 32, 32, 64)        256
_________________________________________________________________
max_pooling2d_8 (MaxPooling2 (None, 16, 16, 64)        0
_________________________________________________________________
conv2d_20 (Conv2D)           (None, 16, 16, 128)       73856
_________________________________________________________________
activation_29 (Activation)   (None, 16, 16, 128)       0
_________________________________________________________________
batch_normalization_26 (Batc (None, 16, 16, 128)       512
_________________________________________________________________
conv2d_21 (Conv2D)           (None, 16, 16, 128)       147584
_________________________________________________________________
activation_30 (Activation)   (None, 16, 16, 128)       0
_________________________________________________________________
batch_normalization_27 (Batc (None, 16, 16, 128)       512
_________________________________________________________________
conv2d_22 (Conv2D)           (None, 16, 16, 256)       295168
_________________________________________________________________
activation_31 (Activation)   (None, 16, 16, 256)       0
_________________________________________________________________
batch_normalization_28 (Batc (None, 16, 16, 256)       1024
_________________________________________________________________
conv2d_23 (Conv2D)           (None, 16, 16, 256)       590080
_________________________________________________________________
activation_32 (Activation)   (None, 16, 16, 256)       0
_________________________________________________________________
batch_normalization_29 (Batc (None, 16, 16, 256)       1024
_________________________________________________________________
```

```
conv2d_24 (Conv2D)           (None, 16, 16, 256)       590080
_________________________________________________________________
activation_33 (Activation)   (None, 16, 16, 256)       0
_________________________________________________________________
batch_normalization_30 (Batc (None, 16, 16, 256)       1024
_________________________________________________________________
conv2d_25 (Conv2D)           (None, 16, 16, 256)       590080
_________________________________________________________________
activation_34 (Activation)   (None, 16, 16, 256)       0
_________________________________________________________________
batch_normalization_31 (Batc (None, 16, 16, 256)       1024
_________________________________________________________________
max_pooling2d_9 (MaxPooling2 (None, 8, 8, 256)         0
_________________________________________________________________
conv2d_26 (Conv2D)           (None, 8, 8, 512)         1180160
_________________________________________________________________
activation_35 (Activation)   (None, 8, 8, 512)         0
_________________________________________________________________
batch_normalization_32 (Batc (None, 8, 8, 512)         2048
_________________________________________________________________
conv2d_27 (Conv2D)           (None, 8, 8, 512)         2359808
_________________________________________________________________
activation_36 (Activation)   (None, 8, 8, 512)         0
_________________________________________________________________
batch_normalization_33 (Batc (None, 8, 8, 512)         2048
_________________________________________________________________
conv2d_28 (Conv2D)           (None, 8, 8, 512)         2359808
_________________________________________________________________
activation_37 (Activation)   (None, 8, 8, 512)         0
_________________________________________________________________
batch_normalization_34 (Batc (None, 8, 8, 512)         2048
_________________________________________________________________
conv2d_29 (Conv2D)           (None, 8, 8, 512)         2359808
_________________________________________________________________
activation_38 (Activation)   (None, 8, 8, 512)         0
_________________________________________________________________
batch_normalization_35 (Batc (None, 8, 8, 512)         2048
_________________________________________________________________
conv2d_30 (Conv2D)           (None, 8, 8, 512)         2359808
_________________________________________________________________
activation_39 (Activation)   (None, 8, 8, 512)         0
_________________________________________________________________
batch_normalization_36 (Batc (None, 8, 8, 512)         2048
_________________________________________________________________
conv2d_31 (Conv2D)           (None, 8, 8, 512)         2359808
_________________________________________________________________
activation_40 (Activation)   (None, 8, 8, 512)         0
_________________________________________________________________
batch_normalization_37 (Batc (None, 8, 8, 512)         2048
_________________________________________________________________
conv2d_32 (Conv2D)           (None, 8, 8, 512)         2359808
_________________________________________________________________
activation_41 (Activation)   (None, 8, 8, 512)         0
_________________________________________________________________
batch_normalization_38 (Batc (None, 8, 8, 512)         2048
_________________________________________________________________
conv2d_33 (Conv2D)           (None, 8, 8, 512)         2359808
_________________________________________________________________
activation_42 (Activation)   (None, 8, 8, 512)         0
_________________________________________________________________
batch_normalization_39 (Batc (None, 8, 8, 512)         2048
_________________________________________________________________
max_pooling2d_10 (MaxPooling (None, 4, 4, 512)         0
_________________________________________________________________
flatten_6 (Flatten)          (None, 8192)              0
_________________________________________________________________
```

```
dense_9 (Dense)              (None, 4096)              33558528
_________________________________________________________________
activation_43 (Activation)   (None, 4096)              0
_________________________________________________________________
batch_normalization_40 (Batc (None, 4096)              16384
_________________________________________________________________
dropout_6 (Dropout)          (None, 4096)              0
_________________________________________________________________
flatten_7 (Flatten)          (None, 4096)              0
_________________________________________________________________
dense_10 (Dense)             (None, 4096)              16781312
_________________________________________________________________
activation_44 (Activation)   (None, 4096)              0
_________________________________________________________________
batch_normalization_41 (Batc (None, 4096)              16384
_________________________________________________________________
dropout_7 (Dropout)          (None, 4096)              0
_________________________________________________________________
dense_11 (Dense)             (None, 43)                176171
_________________________________________________________________
activation_45 (Activation)   (None, 43)                0
=================================================================
Total params: 70,595,179
Trainable params: 70,567,787
Non-trainable params: 27,392
_________________________________________________________________
None
```

3.3.4.1. Resultados

	Precisión	**Recall**	**F1-Score**	**Exactitud (Accuracy)**
Train	1.00	1.00	1.00	1.00
Test	0.97	0.97	0.97	0.98

Tabla 26 - Resultados de VGG-19

3.3.5. ResNet-50

Este modelo de Red Neuronal Convolucional fue introducido por Microsoft en 2015.

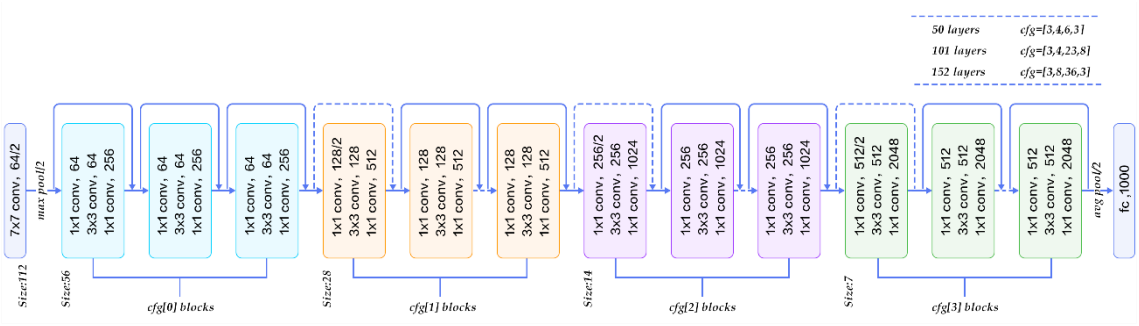


Ilustración 45 - Diagrama Resnet 50

Está formada por **bloques residuales**, se trata en tomar $a^{[l]}$ y acercarlo hasta $a^{[l+1]}$, justo antes de aplicar la función de activación.

Según la fórmula: $a^{[l+2]} = g(Z^{[l+2]} + a^{l})$.

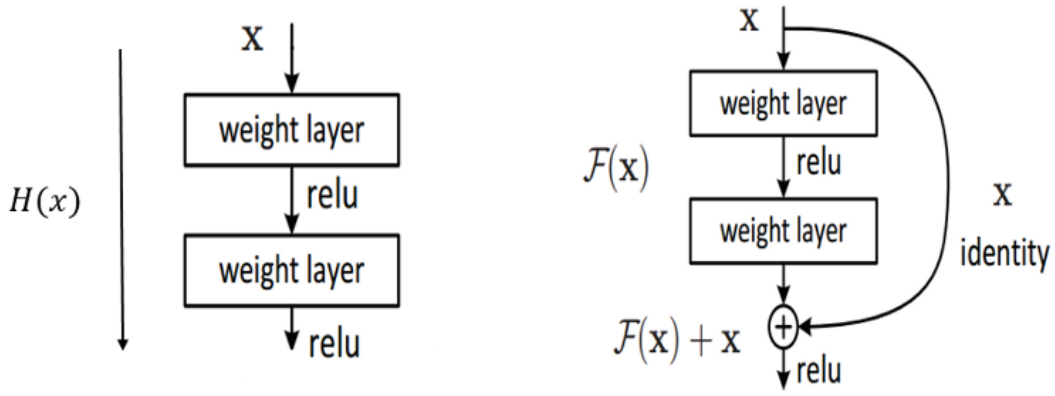


Ilustración 46 - Conexión estándar vs Bloque residual

Según la teoría, tener una Red Neuronal Profunda solo debería ayudar a que diera mejores resultados. Sin embargo, en la práctica no pasa esto porque muchas veces no tiene el tiempo para que aprende lo suficiente. Por lo que, al agregar más capas, los resultados eran peores.

Lo que pasa con ResNet, es que a medida que el número de capas crece, puede seguir bajando el error que comete la capa. No hay una perdida aceptable de rendimiento. Así que, podemos crear redes más profundas sin realmente hacerlas menos eficaces en el conjunto de entrenamiento.

Lo comentado anterior es posible porque con ResNet calculamos el residuo que hay de diferencia entre esas capas. Supongamos que en una capa normal está el mapeo H(x). El residio sería F(x) = H(x) – x. Con ResNet hacemos que nuestra red intente aprender F(x) + x.

Cada **bloque residual ResNet** consta de una serie de capas y **un mapeo de identidad** que agrega una entrada de bloque a la salida. La cual, si tiene distinto tamaño, la ajustará con relleno.

Esta idea es muy efectiva. Antes, había un problema de cancelación del gradiente, dado por la minimización de la función de error en la retroprogramación de las capas anteriores. Es decir, al final los errores eran tan pequeños que no se permitían que la red aprendiera. No obstante, esta red solucionó este problema, permitiendo usar Redes Neuronales mucho más profundas. [5]

3.3.5.1. Resultados

Al tener ResNet 50 capas, las 30 épocas que usábamos para entrenar los anteriores modelos no son suficientes. A continuación, vamos a mostrar los resultados que obtenemos cada 30 épocas hasta llegar a 90, que es cuando ya no obtenemos mejor rendimiento.

	Precisión	**Recall**	**F1-Score**	**Exactitud (Accuracy)**
Train	0.13	0.12	0.07	0.16
Test	0.12	0.12	0.07	0.16

Tabla 27 - Resultados de ResNet-50 para 30 épocas

	Precisión	**Recall**	**F1-Score**	**Exactitud (Accuracy)**
Train	0.57	0.27	0.24	0.33
Test	0.51	0.26	0.24	0.32

Tabla 28 - Resultados de ResNet-50 para 60 épocas

	Precisión	**Recall**	**F1-Score**	**Exactitud (Accuracy)**
Train	0.92	0.87	0.87	0.89
Test	0.86	0.79	0.78	0.83

Tabla 29 - Resultados de ResNet-50 para 90 épocas

En esta red, tenemos en total 23 675 819 parámetros. Menos que para la VGG16 o VGG19, sin embargo, el número de capas es mucho mayor.

3.3.6. Análisis de los resultados

Hemos podido comprobar que el modelo de **Red Neuronal Convolucional simple** hemos conseguido tener un acierto del **91%**. Sin embargo, este ha sido el peor modelo que hemos encontrado.

Al probar **LeNet-5**, el cual era un poco más profundo que el anterior, conseguimos llegar hasta el **93%** de acierto y mejorando todas las demás medidas de resultados en 3%.

Con **VGG-16** tuvimos el mayor salto de rendimiento, consiguiendo llegar a un **97%** de acierto. Esto es un 4% más respecto a LeNet-5. Pudimos mejorar tanto los resultados ya que este modelo tiene una profundidad mucho mayor que los anteriores, consiguiendo tener una capacidad de abstracción mayor. Cabe destacar, que como el dataset empleaba unas imágenes de entrada de 32x32x3, tuvimos que poner menor capas de agrupación de 2x2, ya que, si no, se hubieran reducido muchísimo el volumen de las capas de convolucionales, dando mucho peores resultados.

VGG-19 consigue un increíble **98%** de acierto. Como explicamos anteriormente, este modelo lo único que ha hecho es aumentar el número de capas convolucionales y de filtros que usamos. Con esto, pudimos mejorar aún más la capacidad de abstracción de las capas, mejorando el reconocimiento un codiciado 1% más. Cabe destacar, que también conseguimos obtener un 97% en precisión y recall, obteniendo un modelo muy fiable.

ResNet-50, en cambio, nos ha dado malos resultados. Al entrenar 30 épocas vimos que el resultado era pésimo, cosa normal, ya que aún se tenía que entrenar mucho más. Al llevar 90 épocas es cuando conseguimos el mayor rendimiento, un **83%** de acierto en test. Sin embargo, este es el peor resultado respecto a los demás modelos de CNN entrenados. La explicación a esto es que el dataset utilizado tenía solo una dimensión de 32x32x3, una vez hemos pasado unas pocas capas de max-pooling, perdemos demasiadas características; lo cual hace que el

modelo sea muy malo. En caso de haber contado, por ejemplo, con una entrada de 224x224x3, hubiésemos obtenido mucho mejores resultados.

Por lo tanto, la mejor Red Neuronal Convolucional para la clasificación de señales de tráfico es la VGG-19. Que da, en test para cada clase, como resultados:

	precision	recall	f1-score	support
Límite velocidad (20km/h)	0.94	1.00	0.97	60
Límite velocidad (30km/h)	0.99	1.00	0.99	720
Límite velocidad (50km/h)	1.00	0.99	1.00	750
Límite velocidad (60km/h)	0.97	0.99	0.98	450
Límite velocidad (70km/h)	1.00	0.98	0.99	660
Límite velocidad (80km/h)	0.97	0.99	0.98	630
Fin límite velocidad (80km/h)	1.00	0.83	0.91	150
Límite velocidad (100km/h)	1.00	1.00	1.00	450
Límite velocidad (120km/h)	0.99	0.94	0.96	450
No adelantar	0.98	1.00	0.99	480
No adelantar vehículos < 3.5 T	1.00	0.99	0.99	660
Intersección	0.90	1.00	0.94	420
Calzada con prioridad	1.00	0.98	0.99	690
Ceda el paso	0.99	1.00	0.99	720
Stop	0.99	1.00	1.00	270
Prohibido circulación	0.93	0.99	0.96	210
No pasar vehículos > 3.5 T	0.99	0.96	0.98	150
Dirección prohibida	1.00	0.97	0.99	360
Precaución	0.99	0.96	0.98	390
Curva peligrosa izquierda	1.00	0.97	0.98	60
Curva peligrosa derecha	0.93	1.00	0.96	90
Doble curvas peligrosas	1.00	0.77	0.87	90
Carretera con baches	1.00	0.98	0.99	120
Peligro carretera deslizante	1.00	0.93	0.96	150
Estrechamiento derecha	0.98	0.98	0.98	90
Peligro obras	0.99	0.99	0.99	480
Peligro semáforo	0.98	0.98	0.98	180
Paso de cebra	0.95	1.00	0.98	60
Colegio	0.96	1.00	0.98	150
Bicicletas	1.00	1.00	1.00	90
Precaución nieve/hielo	0.93	0.86	0.89	150
Animales salvajes	1.00	0.98	0.99	270
Fin prohibiciones	0.75	1.00	0.86	60
Girar a la derecha	0.99	1.00	1.00	210
Girar a la izquierda	0.99	1.00	1.00	120
Recto	1.00	0.98	0.99	390
Recto o derecha	0.99	1.00	1.00	120
Recto o izquierda	1.00	0.98	0.99	60
Mantenerse a la derecha	0.99	1.00	1.00	690
Mantenerse a la izquierda	0.99	1.00	0.99	90
Rotonda	0.89	0.97	0.93	90
Fin no adelantar	0.85	0.93	0.89	60
Fin no adelantar vehículos < 3.5 T	0.78	0.84	0.81	90
accuracy			0.98	12630
macro avg	0.97	0.97	0.97	12630
weighted avg	0.98	0.98	0.98	12630

Ilustración 47 - Resultados de VGG-19 en test

3.4. Análisis de resultados de Deep Learning frente a Machine Learning tradicional

El mejor modelo de Machine Learning tradicional, compuesto por un **HOG** con una configuración de 9 orientaciones, 4x4 píxeles por celda y 4x4 celdas por bloque, ha convertido nuestras señales 32x32x3 en un vector de 3 600 características. Como modelo usamos una **SVM** con C = 0.1, es decir, penalizando el sobreaprendizaje, conseguimos una tasa de acierto de 89%, con un 0.87 como media F1-Score.

La mejor CNN obtenida es la **VGG-19**, la cual nos ha proporcionado un 98% de acierto, con un 97 de F1-Score.

A la vista de estos resultados, queda demostrado que emplear el aprendizaje profundo para este problema otorga muchos mejores resultados que usar un modelo de aprendizaje automático tradicional; consiguiendo de esta manera casi un 10% más de acierto.

En cuanto al tiempo de ejecución de los mejores modelos, SVM es claramente más rápido que VGG-19, consiguiendo un tiempo de ejecución de 2.43 segundos frente a los 85.5 de la VGG-19 en la predicción del conjunto de prueba (12 631 imágenes), 7.38 respecto a 192.90 en el conjunto de entrenamiento. Es decir, la SVM es **35 veces más rápida** que la VGG-19. Sin embargo, tampoco sería tan importante la diferencia de velocidad, ya para predecir un solo ejemplo tarda 1.15 segundos; suficiente para reconocer una señal de tráfico a tiempo.

4. DETECCIÓN DE SEÑALES DE TRÁFICO CON DEEP LEARNING

Hasta ahora, hemos estado enfrentándonos a un problema de clasificación, donde teníamos muchas señales y nosotros solo teníamos que decir a qué clase pertenecía. Sin embargo, esto no es lo que nos encontramos en la realidad, ahí tenemos una imagen grande y tenemos que localizar la señal y después determinar a qué clase pertenece.

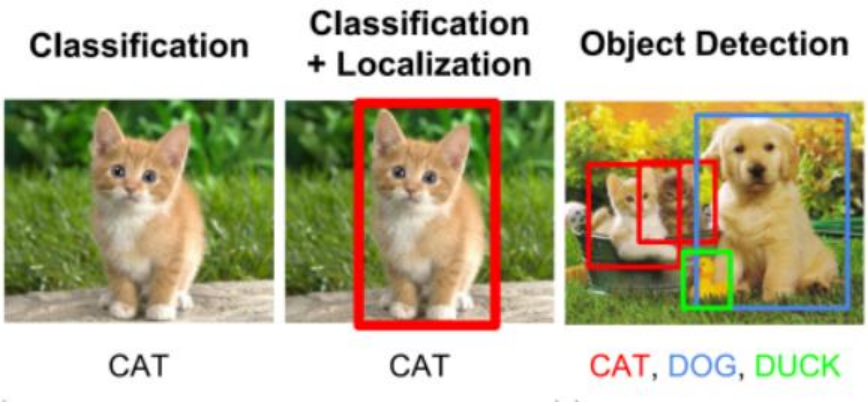


Ilustración 48 - Ejemplo de clasificación y detección

Por lo tanto, nuestro objetivo sería pasarle una imagen de una carretera y que detectara las señales de tráfico.

Ilustración 49 - Ejemplo de nuestro detector de señales de tráfico

Podríamos usar la técnica de la **ventana deslizante**, la cual consiste en pasar por una imagen un recuadro de un tamaño fijado y realizar en cada una de las posiciones un problema de clasificación. Este método tiene tres principales inconvenientes: da muchos falsos positivos, es muy ineficiente computacionalmente y puede ser que el tamaño de la ventana deslizante no encaje bien con el objeto en la imagen. Por lo que lo descartamos.

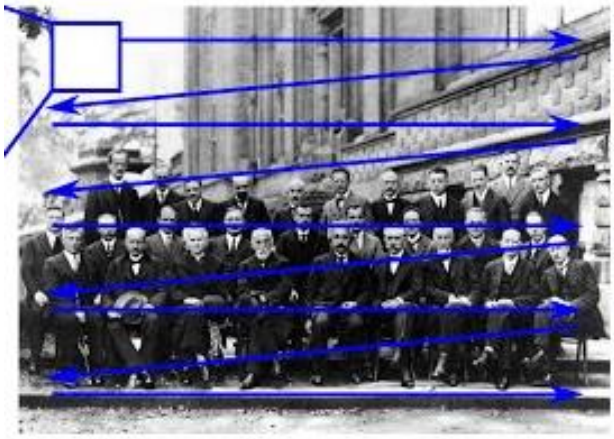


Ilustración 50 - Ejemplo de detección de caras con ventana deslizante

El Aprendizaje Profundo proporciona 2 tipos de algoritmos para la detección de objetos:

- **Algoritmos basados en clasificación**: trabajan en 2 etapas. Seleccionamos de la imagen regiones donde pueden estar esos objetos. Después, usando Redes Neuronales Convolucionales clasificamos las regiones. Esta solución es lenta, ya que hay que hacer una predicción por cada región. De este grupo, se crea una Red basada en Regiones usando Redes Neuronales Convolucionales (**RCNN** – Region-based Convolutional Neural Network), de donde salen Fast-RCNN y Faster-RCNN. A continuación, explicaremos brevemente cuál es la dinámica de estos.
- **Algoritmos basados en regresión**: en vez de seleccionar partes de una imagen, vamos a predecir clases y cuadro delimitadores de clase (**bounding boxes**) para toda la imagen, ejecutando una sola vez el algoritmo. Aquí es donde entra nuestro algoritmo **YOLO**, de ahí su nombre, You Only Look Once.

4.1. Detección basada en regiones

Como hemos dicho, se les llama R-CNN y significa Region-base Convolutional Neural Network. El funcionamiento de dicha red consta de 3 fases:

- Escaneo la imagen de entrada para detectar objetos posibles. Se usa un algoritmo llamado **Selective Search**, que extrae aproximadamente 2 000 regiones de una imagen.
- Se implementa una **CNN** en la parte superior de **cada región**.
- Se extrapola la salida de cada Red Neuronal Convolucional para introducirla en:
    - Una **SVM** para clasificar la región.
    - Realiza una **regresión lineal** para restringir el cuadro delimitador del objeto.

En resumen, generamos las regiones posibles. Extraemos las características, clasificamos esas regiones en función de las características extraídas. De esta manera, hemos convertido un problema de detección de objetos, en uno de clasificación. Este esquema es muy intuitivo, pero es muy lento.

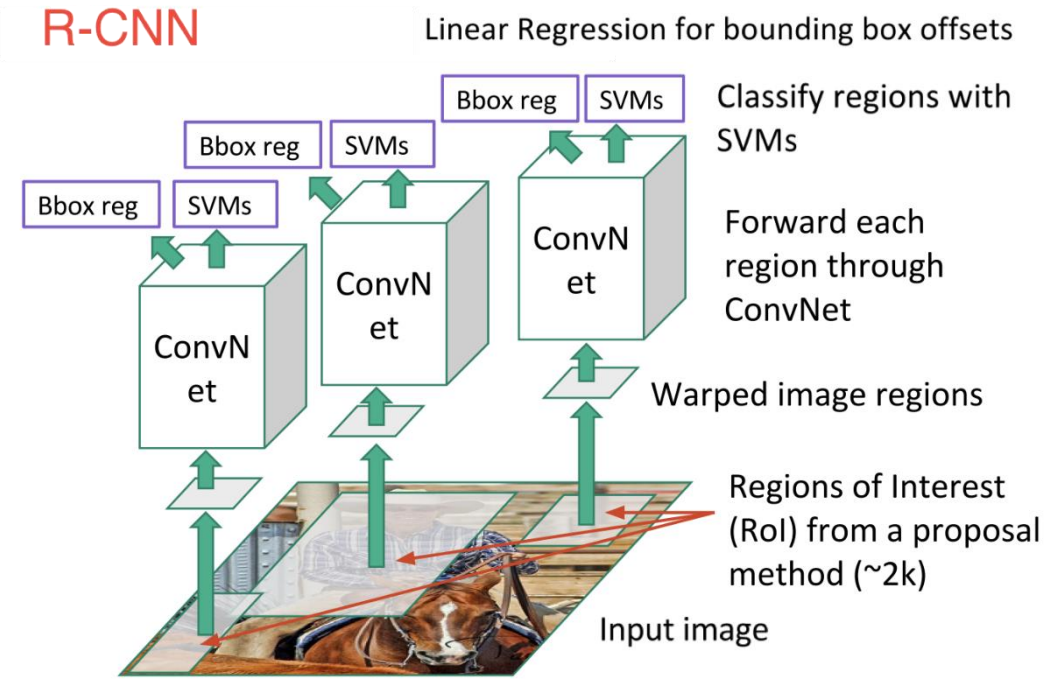


Ilustración 51 - Esquema del funcionamiento de una RCNN

4.1.1. Fast R-CNN

Hay una versión mejorada de R-CNN, que busca que se aumente la velocidad de predicción esta es conocida como **Fast R-CNN**. Lo consigue por estos dos aspectos:

- En vez de trabajar en 2 000 regiones sobrepuestas, usando 2 000 CNNs, empleamos una sola Red Neuronal Convolucional para dividir la imagen.
- En lugar de usar una SVM, la reemplazamos por una función SoftMax. Usando, también, dicha red neuronal para realizar la selección de regiones propuestas.

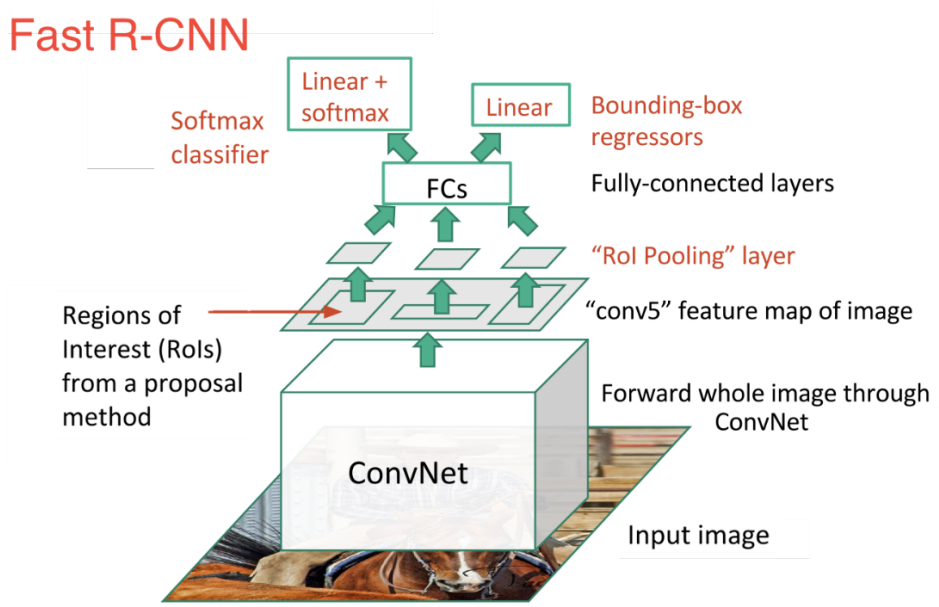


Ilustración 52 - Esquema de Fast R-CNN

Como podemos ver, nos hemos cargado 2 000 CNN y SVMs para clasificar las regiones. Con esto, hemos conseguido mejorar mucho en velocidad. Por lo tanto, al tener una SoftMax para generar

la probabilidad de las clases en la imagen y solo usar una CNN para toda la imagen. No obstante, aún queda un defecto importante, el algoritmo **Selective Search** para proponer las regiones donde están los objetos.

4.1.2. Faster R-CNN

Para este modelo, se introduce una Red de Proposición de Regiones (**RPN – Region Proposal Network**). Su funcionamiento es el siguiente:

- En la última capa de la CNN inicial, una ventana de 3x3 se mueve a través del mapa de características para asignarle un tamaño más pequeño. Por ejemplo, 256.
- Para cada posición por la que pasa la ventana deslizante, el RPN genera múltiples regiones basadas en uniones espaciales de dimensiones fijas, llamadas cajas de anclaje (**anchor boxes**). Cada propuesta consiste en:
    - Una puntuación por la presencia de un objeto en particular (clase de objeto que detecta).
    - 4 coordenadas que representan al cuadro delimitador de la región (**Bounding Box**).

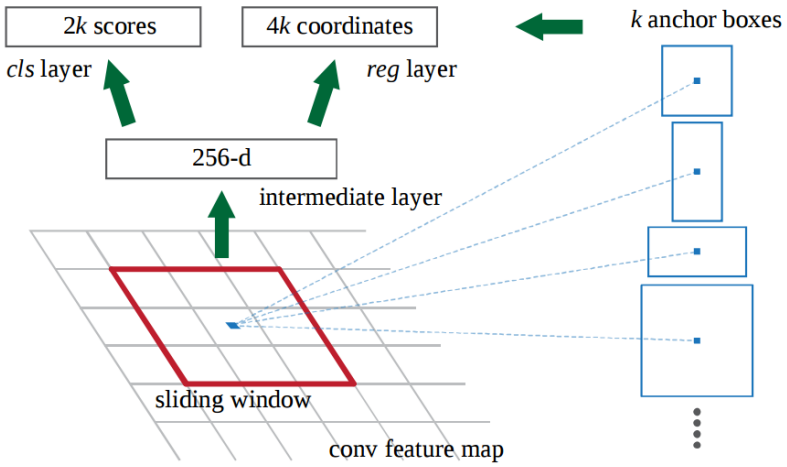


Ilustración 53 - Esquema del funciomiento de RPN

Es decir, en la última de capa de características (la que mejor ha abstraído), teniendo en cuenta los k tipos de anchor boxes que lo rodean; para cada cuadro, se muestra si esta tiene un objeto y cuáles son las coordenadas del cuadro. Solo sirve para elegir regiones, la detección precisa del objeto viene después. Si se supera un umbral, esa casilla (anchor box) se clasificará como posible región.

Una vez tenemos las regiones, las ejecutamos en el Fast R-CNN. Finalmente añadiríamos unas capas de Pooling y otras FC. La última capa, como siempre, será una SoftMax para cada objeto a reconocer. Tendremos un regreso de cuadros delimitadores para cada objeto.

Con esto, podemos ver que Faster R-CNN viene a ser un RPN (Region Proposal Network) y una Fast R-CNN.

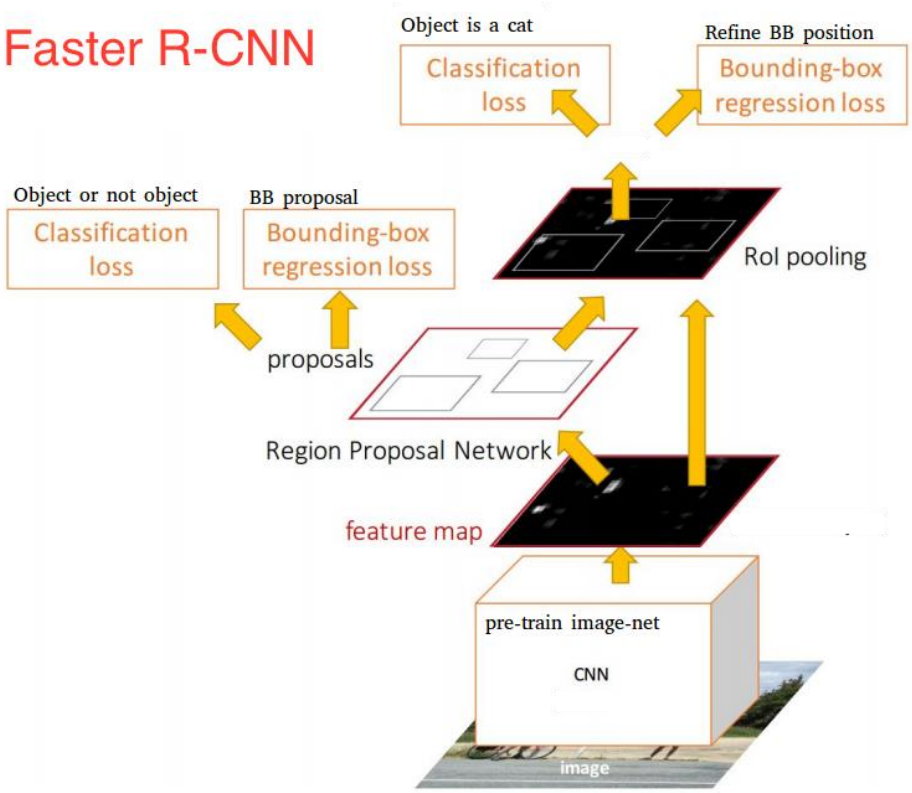


Ilustración 54 - Esquema de Faster R-CNN

Con todo esto, podemos ver que el Faster R-CNN logra una mejor velocidad y precisión. Puede parecer un poco complejo, pero en esencia estamos haciendo lo mismo que en el R-CNN clásico. Identificar posibles regiones de objetos y clasificarlos. [6]

4.2. Detección basada en regresión

Para hacer regresión en una imagen y obtener así los resultados de una imagen, podríamos intentar crear una Red Neuronal Convolucional con 5 neuronas de salida. La primera nos indicaría clase que es, la segunda y tercera nos podría decir la parte superior izquierda de la señal y las últimas 2 la esquina inferior derecha.

Este método tendría muchos problemas, el principal es que solo podríamos identificar un objeto en una imagen. Además, como resulta evidente, habría una falta de generalización para nuevos ejemplos, dependiendo fuertemente de la ubicación del objeto en la posición de la imagen.

YOLO, publicado en 2015, es el algoritmo más popular en la actualidad por su rapidez y precisión a la hora de predecir.

Ilustración 55 - Precisión respecto a velocidad para los distintos algoritmos de detección de objetos

El funcionamiento de YOLO es complejo y teniendo en cuenta la extensión de este TFG, se va a explicar de una manera sencilla y no muy profunda, obviando los detalles más técnicos en cuestión.

4.2.1. Funcionamiento de YOLO

YOLO divide una imagen en SxS celdas de tamaño fijo. Cada celda se encarga de predecir solo un objeto. En la práctica, normalmente las celdas son 19x19, aunque cuando fue creado el algoritmo el número de celdas que se usaban eran 7x7.

Ilustración 56 - División de una imagen en celdas

A la hora de identificar el objeto, YOLO usa un **cuadro delimitador** (**bounding box**). Este viene definido por 5 descriptores:

1. **bx:** centro del objeto según x (filas).
2. $\mathbf{b_y}$: centro del objeto según y (columnas).
3. $\mathbf{b_w}$: anchura de la caja.
4. $\mathbf{b_h}$: altura de la caja.
5. $\mathbf{p_c}$: valor que indica la probabilidad de que haya un objeto en esa celda.

Estos descriptores no toman como referencia los píxeles, sino que, al dividir una imagen en celdas, usaremos un rango de [0, 1]. Tomándose como el punto (0, 0) la esquina superior izquierda de la celda, y el punto (1, 1) para la esquina inferior derecha. El centro del objeto tendrá un valor entre [0, 1].

Sin embargo, la altura y la anchura tomarán como referencia proporcional al tamaño de la caja 1, por lo que, si el objeto es más grande que una celda, podrá superior el 1 en ese caso. P_c, evidentemente, será de valor [0, 1], ya que indica la probabilidad de que haya un objeto.

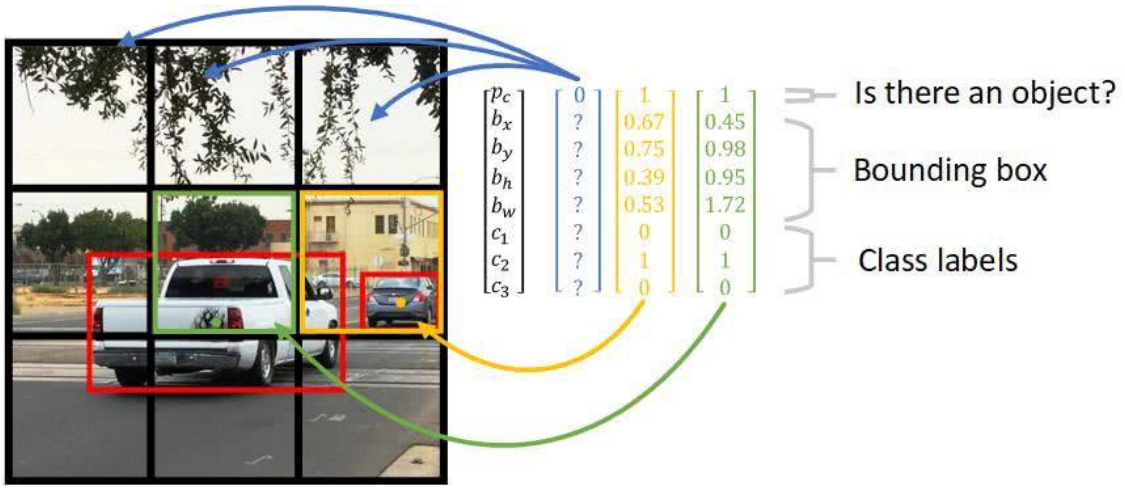


Ilustración 57 - Ejemplo de cuadros delimitadoras en YOLO

Así que, de esta forma, si nuestra CNN tiene un 7x7 como número de celdas en una imagen y predice 20 clases; tendremos una capa de salida de 7x7x(5 + 20).

Presenta una limitación nuestro modelo, no podríamos reconocer objetos distintos, con distintas formas, de estar su centro en las mismas celdas. Aquí es donde entre el uso de distintas distintos bounding boxes, en la práctica se suelen emplear 5.

Además, se suelen usar **cuadros de anclaje** (**anchor boxes**) con un tamaño prefijado para conseguir limitar el aprendizaje a unas formas predeterminadas y no acabar sobreaprendiendo en exceso. Normalmente, YOLO suele usar 5 tipos de anchos boxes distintos.

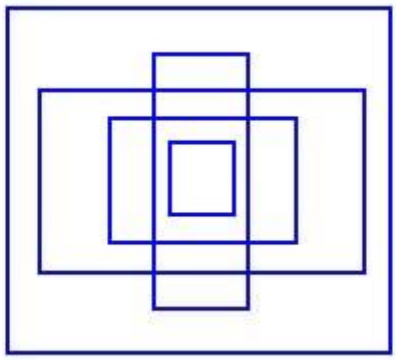


Ilustración 58 – Los 5 cuadros de anclaje predeterminados por YOLO

Con toda la configuración explicada anterior, la capa de salida de YOLO, en caso de tener un tamaños de celdas de 7x7, predecir 20 clases y usar 2 cajas delimitadores distintas (fueron las características con las que se desarrolló la primera implementación de YOLO), tendríamos una **capa de salida** de 7x7x(2*5 + 20). Generalizando, la capa de salida sería de **SxSx(#Bounding Boxes*5 + #Clases)**.

La medida que usamos para evaluar el funcionamiento del detector de objetos es Intersección sobre Unión (**IoU** – Intersection over Union). Viene determinado por la siguiente fórmula:

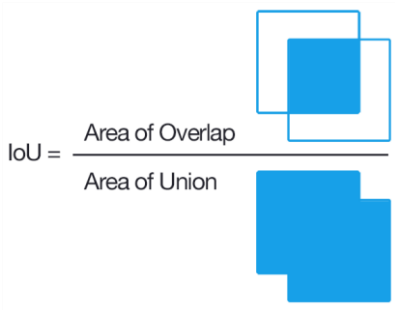


Ilustración 59 - Fórmula de IoU

A la hora de entrenar y probar, podemos calcular el IoU de la solución frente a lo predicho. Con esto, tenemos una buena medida. Consideraremos que la predicción es correcta si IoU ≥ 0.5; sin embargo, podríamos hacer que tuviera un mayor valor con tal de ser más estricto. Si fuera perfecta, IoU sería 1.

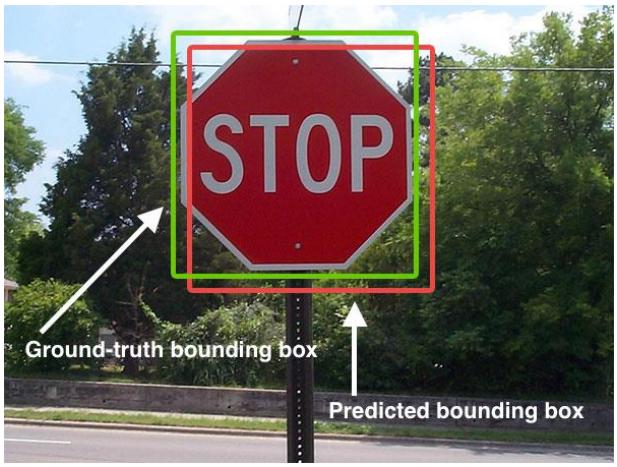


Ilustración 60 - Ejemplo de caja delimitadora en predicción frente a real

Sin embargo, a la hora de clasificar señales nos encontramos con el siguiente problema: puede ser que muchas celdas adyacentes predigan el mismo objeto. Para solucionar esto, vamos a emplear la supresión de no máximos (**non-max suppression**).

Cuando estamos prediciendo, hay muchas celdas de las SxS que puede ser que informen de que hay un objeto en su celda (p_c). Esto puede hacer que obtengamos un mayor número de objetos detectados de los que realmente hay en la imagen. Con supresión de no máximos eliminaremos las múltiples detecciones del mismo objeto y buscaremos obtener una sola.

Primero examinamos las probabilidades asociadas con cada una de las detecciones (p_c). Seleccionamos la detección más segura (la más alta), la que tiene más alta probabilidad. A continuación, vamos a eliminar los bounding boxes que tengan una menor probabilidad, pero un alto IoU respecto a él.

En caso de que hubiera más de una clase, haríamos el mismo procedimiento para cada clase detectada independientemente.

Ilustración 61 - Ejemplo de supresión de no máximos

4.2.2. Estructura de YOLO

La red YOLO para el dataset Pascal VOC (el dataset sobre el que se basó la creación de la red), el cual se compone de 20 clases a predecir, 7x7 celdas en la imagen y con 2 objetos en cada celda.

De ahí, obtenemos una salida de 7x7x(2*5 + 20). Recordamos que ese 5 marcan las coordenadas del objeto en cuestión.

YOLO tiene más de 63 000 000 millones de parámetros.

Name	Filters	Output Dimension
Conv 1	7 x 7 x 64, stride=2	224 x 224 x 64
Max Pool 1	2 x 2, stride=2	112 x 112 x 64
Conv 2	3 x 3 x 192	112 x 112 x 192
Max Pool 2	2 x 2, stride=2	56 x 56 x 192
Conv 3	1 x 1 x 128	56 x 56 x 128
Conv 4	3 x 3 x 256	56 x 56 x 256
Conv 5	1 x 1 x 256	56 x 56 x 256
Conv 6	1 x 1 x 512	56 x 56 x 512
Max Pool 3	2 x 2, stride=2	28 x 28 x 512
Conv 7	1 x 1 x 256	28 x 28 x 256
Conv 8	3 x 3 x 512	28 x 28 x 512
Conv 9	1 x 1 x 256	28 x 28 x 256
Conv 10	3 x 3 x 512	28 x 28 x 512
Conv 11	1 x 1 x 256	28 x 28 x 256
Conv 12	3 x 3 x 512	28 x 28 x 512
Conv 13	1 x 1 x 256	28 x 28 x 256
Conv 14	3 x 3 x 512	28 x 28 x 512
Conv 15	1 x 1 x 512	28 x 28 x 512
Conv 16	3 x 3 x 1024	28 x 28 x 1024
Max Pool 4	2 x 2, stride=2	14 x 14 x 1024
Conv 17	1 x 1 x 512	14 x 14 x 512
Conv 18	3 x 3 x 1024	14 x 14 x 1024
Conv 19	1 x 1 x 512	14 x 14 x 512
Conv 20	3 x 3 x 1024	14 x 14 x 1024
Conv 21	3 x 3 x 1024	14 x 14 x 1024
Conv 22	3 x 3 x 1024, stride=2	7 x 7 x 1024
Conv 23	3 x 3 x 1024	7 x 7 x 1024
Conv 24	3 x 3 x 1024	7 x 7 x 1024
FC 1	-	4096
FC 2	-	7 x 7 x 30 (1470)

Ilustración 62 - Lista de componentes YOLO original

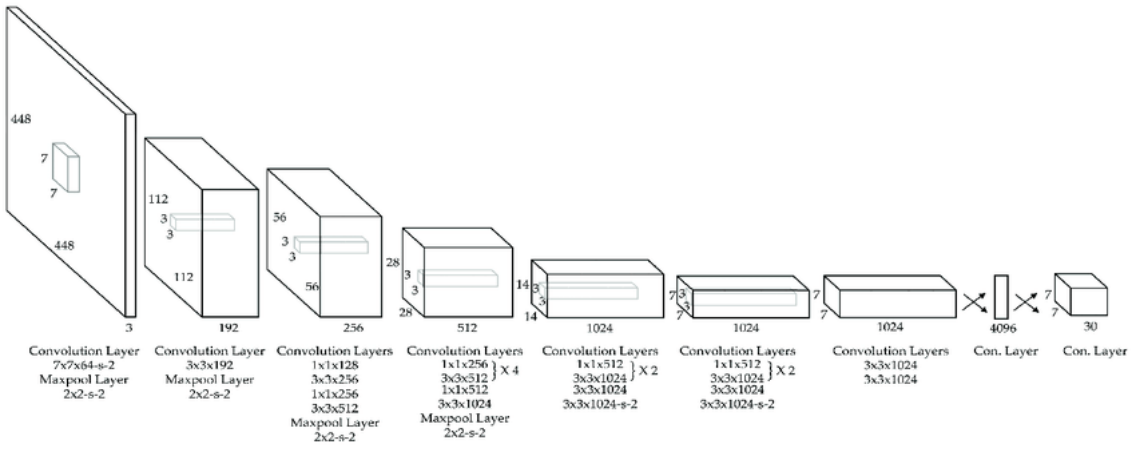


Ilustración 63 - Esquema de YOLO

4.2.3. Cómo entrena YOLO

Características del entrenamiento de YOLO:

- Primero, entrena las 20 primeras capas convolucionales, con un tamaño de entrada de 224x224.
- Segundo, aumenta la resolución a 448x448.
- Entrena toda la red con la nueva resolución.
- Usa un aumento de los datos (**data augmentation**) con escalado, traslación, ajuste de la exposición y saturación de manera aleatoria. [7]

4.2.4. Medida de error

Para medir el error de YOLO usaremos las funciones ya explicados de Precisión y Recall. Sin embargo, también entrará en juego la Intersección sobre Unión (IoU). Consideraremos como una clase bien predicha, aquella que supere el umbral de **IoU ≥ 0.5**.

YOLO tiene una **función de error** propia, la cual es la más importante. Está dividida en 2 partes: el error de localización y el error de clasificación.

La función Precisión Media (Average Precision – **AP**) es el área de la curva precisión-recall.

$$AP = \int_0^1 p(r)dr$$

Sin embargo, la medida con la que nos guiaremos para saber si está funcionando bien YOLO es el **Promedio de Precisión Media** (Mean Average Precision - **mAP**). Tiene en cuenta, por lo tanto, la precisión, el recall y el IoU (que determina si se ha detectado un objeto o no). Esta nos da el promedio de la precisión media de todas las clases en total.

4.2.5. YOLOv3

Para este proyecto vamos a usar la versión 3 YOLO. Se publicó YOLOv1, en el año 2015; en el 2017 salió una nueva versión de YOLOv2 que mejoraba la red haciéndola más precisa y rápida a la hora de predecir.

La última versión de YOLO es la YOLOv3, la cual, si bien no ha mejorado la precisión de YOLOv2, sí que la ha hecho más rápida.

4.2.6. Entorno de trabajo

Para poder acceder a la red YOLOv3 hemos usado la implementación **Darknet** [8]. Esta Red Neuronal está escrita en **C** y es de código libre, usa las librerías de **CUDA** para poder usar el cómputo de las GPUs. Sin embargo, vamos a usar un fork de la misma, hecho por **AlexeyAB** [9], el cual además de muchas mejores, es compatible con Windows.

Las características del equipo que va a ejecutar YOLOv3 son:

- Ubuntu 18.04 LTS.
- Intel Core i7 8750H.
- 16GB de RAM.
- Nvidia GeForce 1050Ti.

Como acabamos de comentar, necesitamos tener instalador los drivers **NVIDIA CUDA Toolkit** para poder entrenar la red YOLO a una alta velocidad. CUDA Toolkit es una librería que nos permite codificar algoritmos para ser ejecutadas en las gráficas Nvidia y así poder aprovecharnos del gran paralelismo que nos proporciona una GPU.

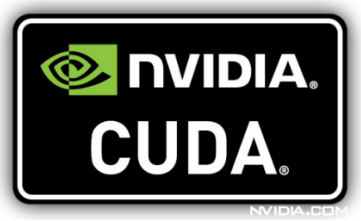


Ilustración 64 - Logo NVIDIA CUDA

Dentro de NVIDIA CUDA Tooklit, vamos a utilizar un módulo en especial: **NVIDIA CUDA Deep Neural Network library** (**cuDNN**). Es una librería orientada a GPU para redes neuronales profundas; implementa funciones como propagación hacia adelante, retropropagación, agrupación, normalización y capas de activación.

Ilustración 65 - Logo cuDNN

4.2.7. Conjunto de datos GTSDB

Una vez ya tenía todo el entorno creado para poder empezar a aprender un modelo de detección de señales de tráfico, encontré el dataset The German Traffic Sign Detection Benchmark (**GTSDB**) [10].

Tiene las siguientes características:

- 900 imágenes, divididas en 600 para entrenamiento y 300 para evaluación.
- 43 clases de señales (las mismas que para el GTSRB).
- Fichero con los píxeles en los que están las señales para cada imagen.

Ilustración 66 - Muestra de GTSDB

La implementación de YOLO que estamos usando necesita que se le pase, por cada imagen, la posición de cada señal que aparece según el formato de YOLO; es decir, con el punto medio de la imagen y la anchura y altura del cuadro delimitador en rango [0, 1]. Para ello, se desarrolló un script que convertía las etiquetas en píxeles a etiquetas YOLO, creando un fichero txt por cada imagen.

4.2.7.1. Resultados

Como explicamos anteriormente, la medida que usaremos para referirnos a la calidad del detector será el Promedio de Precisión Media **mAP**, considerando como detección válida aquella que tenga una Intersección sobre Unión mayor que 0.5.

Los resultados que se han obtenido fueron tras 14 000 épocas, cuando ya tras la época 12 000 no se conseguía reducir el error.

Con esta explicación, los resultados obtenidos tras para el dataset GTSDB son:

Precisión	Recall	F1-Score	mAP
0.63	0.65	0.64	0.445

Tabla 30 - Resultados YOLOv3 en GTSDB

Es decir, los resultados son bastantes malos para el conjunto de test, podemos ver que tenemos tan solo obtenemos un 44.5% de precisión en detección y clasificación (mAP). Para el conjunto de entrenamiento los resultados son bastante buenos, es decir, está sobreaprendiendo mucho.

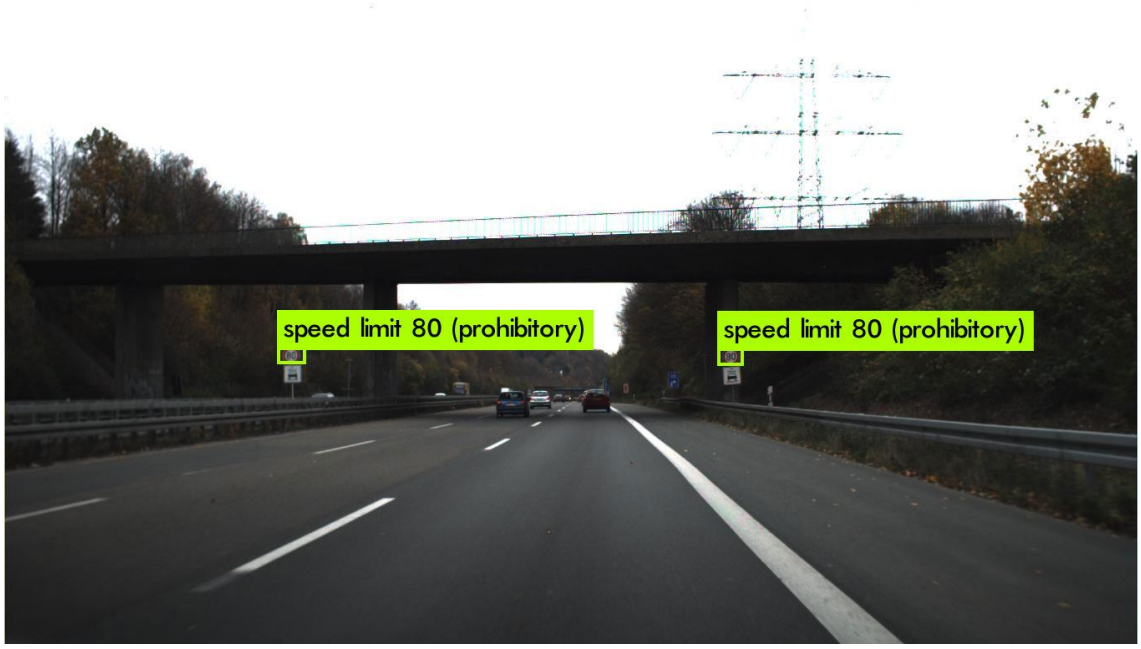


Ilustración 67 – Ejemplo 1 de predicción de train en GTSDB

Ilustración 68 - Ejemplo 2 de predicción de train en GTSDB

Ilustración 69 - Ejemplo 1 de predicción de test en GTSDB

Ilustración 70 - Ejemplo 2 de predicción de test en GTSDB

Por lo tanto, podemos intuir que el problema de que no funcione tan bien este detector es que tenemos muy pocas imágenes para entrenar 43 clases. Además, las de velocidad y de giro son muy parecidas entre ellas. Asimismo, algunos ejemplos no tenían muy buena calidad de iluminación.

Por lo que, resulta imposible mejorar la calidad del detector con tan pocos ejemplos.

4.2.8. Conjunto de datos Mapillary Traffic Sign

Mapillary es una compañía sueca que se dedica al desarrollo de mapas y a la comercialización de conjunto de datos para el desarrollo de asistentes de conducción para visión por computador.

Ilustración 71 - Logo de Mapillary

La compañía ofrece, entre muchos de sus servicios, el Mapillary Traffic Sign Dataset [11]. Este, tiene varias opciones; una opción comercial, la cual hay que pagar por usarla y otra opción de investigación. Solicité a esta compañía el uso del conjunto de datos de investigación, el cual tiene las siguientes características:

- 100 000 imágenes en alta resolución (52 000 completamente anotadas y 48 000 parcialmente).
- 300 clases de señales de tráfico identificadas con sus cajas delimitadoras.
- Señales pertenecientes a los 5 continentes.
- Variedad de clima, estación y hora del día.

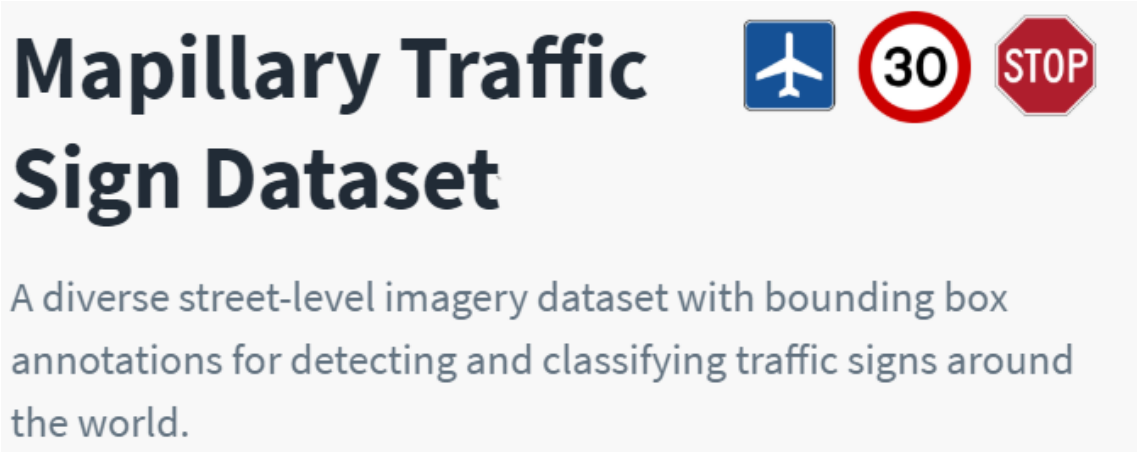


Ilustración 72 - Mapillary Traffic Sign Dataset

Como bien sabemos, una de las cosas más importantes para crear un modelo de aprendizaje es contar con un dataset de calidad, y este sin duda es el mejor que se puede encontrar.

Solicité acceso a este conjunto de datos y se me concedió. Para mi sorpresa me encontré con un dataset en bruto: en un directorio tenía todas las imágenes y en otra las etiquetes pertenecientes a cada imagen, por ningún lado tenía un fichero en el que se me explicara de qué región era la foto ni nada. Tuve que desarrollar un script para clasificar las señales, quitar las que no quería y convertir las etiquetas al formato YOLO.

De 300 señales, me quedé con 102, las cuales son solo las que son oficiales en Europa. Además, seleccioné imágenes del conjunto completamente etiquetado, ya que a la hora de crear el detector quería tener un dataset donde todas las señales conocidas de la imagen están etiquetadas; de la otra forma, podríamos estar maleducando a la red.

La razón por la que elegí 102 clases de señales, y no 43 como en el dataset GTSDB, fue porque el objetivo era construir el mejor detector posible. Es decir, que reconociera el mayor número de señales posible y, además, tuviera la mejor precisión. Con este conjunto de datos, parecía posible el desafío.

En resumen, nuestro nuevo conjunto de datos tiene:

- 102 clases de señales.
- 12 546 imágenes de entrenamiento.
- 3 137 imágenes de test.
- Un tamaño total de 14GB.

Las 102 clases son:

regulatory--bicycles-only--g1
regulatory--dual-path-bicycles-and-pedestrians--g1
regulatory--end-of-no-parking--g1
regulatory--end-of-priority-road--g1
regulatory--end-of-prohibition--g1
regulatory--end-of-speed-limit-zone--g1
regulatory--give-way-to-oncoming-traffic--g1
regulatory--go-straight--g1
regulatory--go-straight-or-turn-left--g1
regulatory--go-straight-or-turn-right--g1
regulatory--height-limit--g1
regulatory--keep-left--g1
regulatory--keep-right--g1
regulatory--maximum-speed-limit-100--g1
regulatory--maximum-speed-limit-120--g1
regulatory--maximum-speed-limit-20--g1
regulatory--maximum-speed-limit-30--g1
regulatory--maximum-speed-limit-50--g1
regulatory--maximum-speed-limit-60--g1
regulatory--maximum-speed-limit-70--g1
regulatory--maximum-speed-limit-80--g1
regulatory--maximum-speed-limit-90--g1
regulatory--maximum-speed-limit-led-100--g1
regulatory--maximum-speed-limit-led-60--g1
regulatory--maximum-speed-limit-led-80--g1
regulatory--mopeds-and-bicycles-only--g1
regulatory--no-bicycles--g1
regulatory--no-entry--g1
regulatory--no-heavy-goods-vehicles--g1
regulatory--no-left-turn--g1
regulatory--no-motor-vehicles-except-motorcycles--g1
regulatory--no-motorcycles--g2
regulatory--no-overtaking--g1
regulatory--no-overtaking-by-heavy-goods-vehicles--g1
regulatory--no-parking--g1
regulatory--no-pedestrians--g1
regulatory--no-right-turn--g2
regulatory--no-stopping--g1
regulatory--no-u-turn--g1
regulatory--no-u-turn--g2
regulatory--no-u-turn--g3
regulatory--no-vehicles-carrying-dangerous-goods--g1
regulatory--one-way-left--g1
regulatory--one-way-right--g1
regulatory--one-way-right--g2
regulatory--one-way-straight--g1
regulatory--pass-on-either-side--g1
regulatory--passing-lane-ahead--g1
regulatory--pedestrians-only--g1
regulatory--priority-over-oncoming-vehicles--g1
regulatory--priority-road--g1
regulatory--radar-enforced--g1
regulatory--reversible-lanes--g2
regulatory--road-closed--g2
regulatory--road-closed-to-vehicles--g1
regulatory--road-closed-to-vehicles--g3
regulatory--roundabout--g1
regulatory--shared-path-bicycles-and-pedestrians--g1
regulatory--shared-path-pedestrians-and-bicycles--g1
regulatory--stop--g1
regulatory--stop-signals--g1
regulatory--turn-left--g1
regulatory--turn-left-ahead--g1
regulatory--turn-right--g1
regulatory--turn-right-ahead--g1
regulatory--u-turn--g1
regulatory--weight-limit--g1
regulatory--wrong-way--g1
regulatory--yield--g1
warning--bicycles-crossing--g1
warning--children--g1
warning--crossroads--g1
warning--crossroads-with-priority-to-the-right--g1
warning--curve-left--g1
warning--curve-right--g1
warning--double-curve-first-left--g1
warning--double-curve-first-right--g1
warning--double-turn-first-right--g1
warning--junction-with-a-side-road-perpendicular-left--g1
warning--junction-with-a-side-road-perpendicular-right--g1
warning--narrow-bridge--g3
warning--other-danger--g1
warning--pedestrians-crossing--g1
warning--pedestrians-crossing--g5
warning--railroad-crossing--g3
warning--railroad-crossing-with-barriers--g1
warning--railroad-crossing-without-barriers--g1
warning--road-bump--g1
warning--road-narrows--g1
warning--road-narrows-left--g1
warning--road-narrows-right--g1
warning--roadworks--g1
warning--roadworks--g3
warning--roadworks--g6
warning--roundabout--g1
warning--slippery-road-surface--g1
warning--traffic-merges-right--g2
warning--traffic-signals--g1
warning--two-way-traffic--g1
warning--uneven-road--g1
warning--wild-animals--g1
warning--winding-road-first-right--g3

4.2.8.1. Resultados

Los resultados obtenidos en el conjunto test en Mapillary Traffic Sign Dataset, en la época 40 000 son:

Precisión	Recall	F1-Score	mAP
0.43	0.36	0.39	0.162

Tabla 31 - Resultados YOLOv3 en Mapillary Traffic Sign Dataset

Estos resultados los hemos sacado cuando ya el detector llevaba más de 3 000 épocas sin bajar el error. Como vemos, obtenemos un 16.2% de mAP, mucho peor que antes. Esto se debe principalmente a que tenemos muchas más del doble de clases a identificar, las cuales, además son muy parecidas entre ellas.

Aunque, si nos fijamos, anteriormente en el dataset GTSDB teníamos para 43 señales 600 imágenes, en este caso, para 102 señales tenemos más de 12 000 imágenes. La proporción es de 13.95 contra 117.64.

A continuación, vamos a mostrar algunos resultados de la detección en train y test.

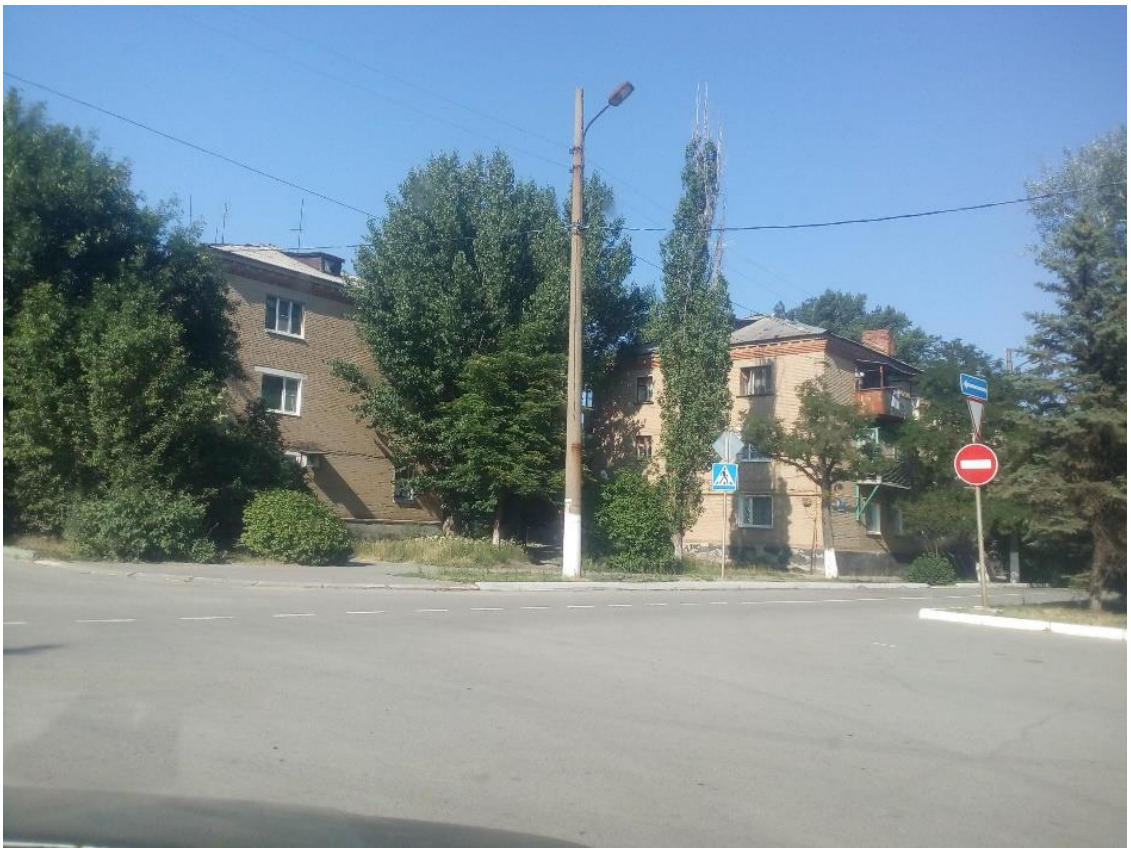


Ilustración 73 - Ejemplo a predecir en train de Mapillary

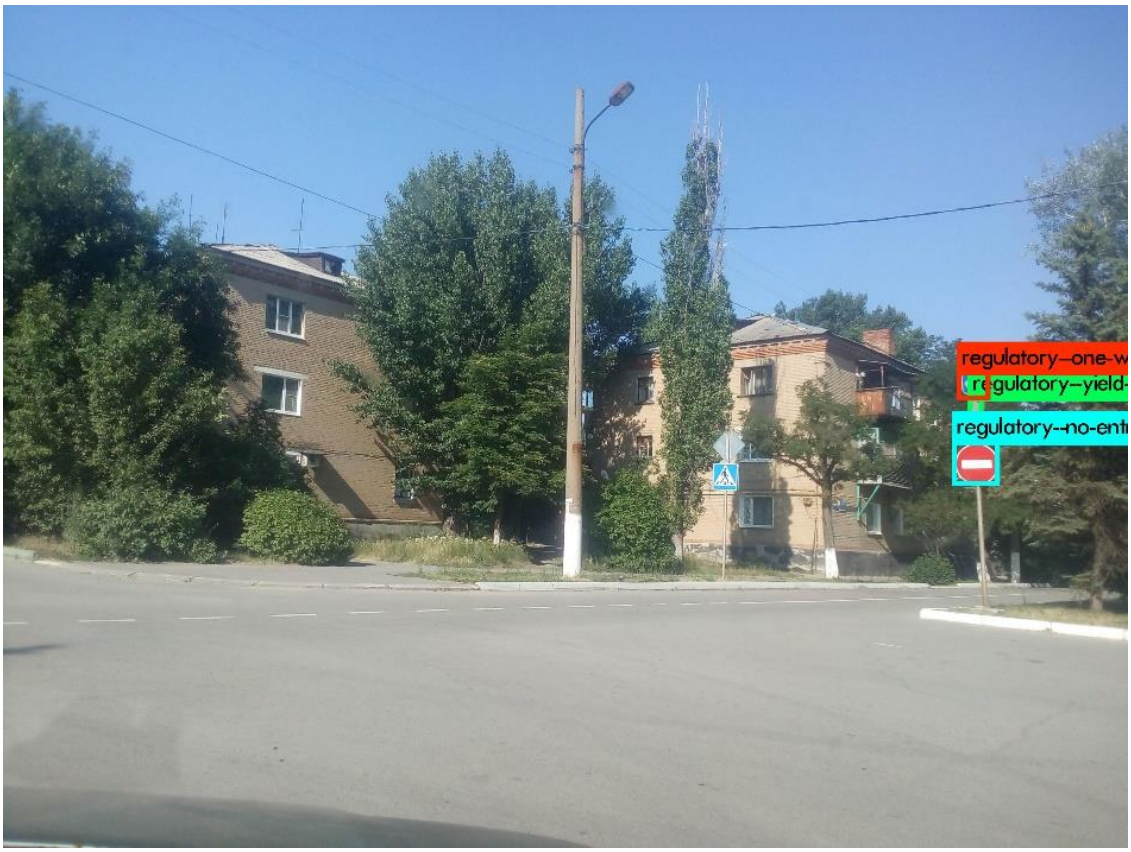


Ilustración 74 - Resultado de predicción en train de Mapillary

Ilustración 75 – Ejemplo a predecir en test de Mapillary

Ilustración 76 - Resultado de predicción en test de Mapillary

4.2.9. Conjunto de datos GTSDB con superclases

Con las dos pruebas realizadas anteriormente, podemos intuir que hay un problema a la hora de entrenar un modelo YOLO que detecte todas las clases de señales de tráfico directamente. [12]

Más tarde, si quisiésemos detectar la clase exacta de la señal, podríamos recortar la señal que hemos encontrado y usar un modelo de clasificación, como los entrenados anteriormente, para reconocer de manera concreta su clase. De esta manera, podríamos tener un detector más rápido y sencillo, y unos modelos creados específicamente para detectar las diferencias sutiles entre las señales de tráfico.

Siguiendo esta premisa, el conjunto original de GTSDB ha pasado a tener 9 clases. Estas, están agrupadas de la siguiente manera:

Número superclase	Nombre superclase	Números clases agrupadas	Clases agrupadas
0	Prohibido	0, 1, 2, 3, 4, 5, 7, 8, 9, 10, 16	Límites de velocidad, prohibido adelantar turismos y camiones
1	Peligro	11, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31	Todas las señales de peligro
2	Fin prohibido	6, 32, 41, 42	Final prohibido velocidad, fin restricción, fin no adelantar turismos y camiones
3	Obligatorio	33, 34, 35, 36, 37, 38, 38, 40	Obligatorio direcciones de circulación
4	Calzada con prioridad	12	Calzada con prioridad
5	Ceda el paso	13	Ceda el paso
6	Stop	14	Stop
7	No circular	15	Prohibida circulación
8	Prohibido el paso	17	Dirección prohibida

Tabla 32 - Agrupación de las clases GTSDB

4.2.9.1. Resultados

Los resultados que se han para el conjunto de test del dataset GTSB, obtenidos tras XXX épocas son:

Precisión	Recall	F1-Score	mAP
0.89	0.94	0.92	0.740

Tabla 3330 - Resultados YOLOv3 en GTSDB

Como podemos ver, los resultados son muchísimos mejores respecto a los obtenidos anteriormente. Pasando de un 44.5% de Promedio de Precisión Media en el GTSDB de 43 clases, a un 74% mAP con el de 9 superclases.

Con esto, se confirma que YOLOv3 funciona mucho mejor con clases que representan a objetos de distintas formas.

A continuación, vamos a mostrar cómo serían las detecciones para los mismos ejemplos que hemos usado con el GTSDB de 43 clases.

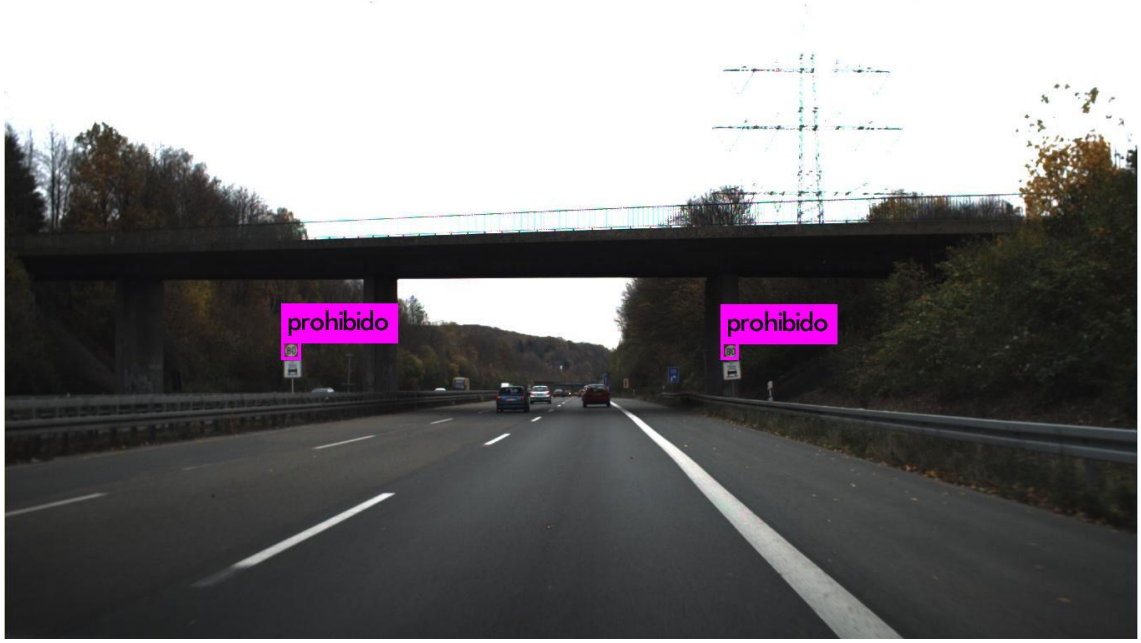


Ilustración 77 - Ejemplo 1 de predicción de train en GTSDB con 9 clases

Ilustración 78 - Ejemplo 2 de predicción de train en GTSDB con 9 clases

Ilustración 79 - Ejemplo 1 de predicción de test en GTSDB de 9 clases

Ilustración 80 - Ejemplo 2 de predicción de test en GTSDB de 9 clases

En adición a esto, he decido probar cómo funcionaría el detector en una circunstancia real. Para ello, he grabado en vídeo un pequeño trayecto por Pamplona. Puede verse en el siguiente enlace: https://youtu.be/B3iVZ7hYC7w.

4.2.10. Conclusiones del funcionamiento de YOLO

YOLO es un gran modelo para la detección de objetos en tiempo real. Sin lugar a duda, proporciona muy buenos resultados en la detección de objetos con **distinta forma** y diferencias claras entre ellas.

Asimismo, existen diferentes modelos de YOLO que dan buenos resultados en circunstancias específicas. En este trabajo fin de grado, se ha querido evaluar el rendimiento del modelo estándar **YOLOv3**.

Hemos podido comprobar que le afecta mucho la calidad y cantidad de imágenes. Sin embargo, lo que de verdad es determinante para el reconocimiento es la **forma que tienen los objetos** que queremos **detectar**. Ya que, si tienen una forma muy parecida, aumenta la probabilidad de error en clasificación.

Siguiendo la conclusión anterior, hemos adaptado el dataset GTSDB a 9 superclases, las cuales eran de formas diferentes; pudimos ver que la detección y la clasificación mejoraba mucho.

La mejor solución para la detección de señales de tráfico sería usar el último detector obtenido. Una vez identificadas, podríamos ejecutar; por ejemplo, una VGG-19 para obtener el tipo del que se trata.

En caso de haber querido detectar las 102 señales del último conjunto de datos, tendríamos que haber creado nuestro propio dataset de señales recortándolas de las imágenes originales que Mapillary nos proporcionó. Para luego, mandar aprender a nuestro modelo para la clasificación las señales. Además, por otro lado, tendríamos que entrenar YOLO con un número de clases menor que 10.

Cabe destacar, que cuanta más **definición** tenga la imagen de entrada para la detección, mejor precisión se obtiene, tanto para la clasificación como para la detección. En nuestro estudio, aunque teníamos un dataset con calidad mayor que 1920x1080, solo pudimos entrenar a YOLO con una resolución de **416x416** por falta de capacidad Hardware. Por lo que, está claro que hemos perdido capacidad de detección por esa circunstancia.

5. CONCLUSIONES Y LÍNEAS FUTURAS

Después de todo este Trabajo Fin de Grado hemos podido sacar unas conclusiones interesantes.

En primer lugar, el Deep Learning supone la mejor alternativa para la clasificación de señales de tráfico, arrojando hasta un 98% de precisión en la detección. Sin embargo, el tiempo tanto para el entrenamiento como para la predicción aumentan en más 35 veces respecto al modelo de Machine Learning tradicional. Aun así, dicho aumento de tiempo no perjudica a la solución del problema real, ya que no se necesita una velocidad de reconocimiento instantánea. Por lo que, para la clasificación de señales de tráfico, la Red Neuronal Convolucional es la opción idónea. Además, aumentar la cantidad de capas y parámetros ayudaba a mejorar aún más la precisión de detección, ya que esto favorecía la capacidad de abstracción de la red neuronal. Aunque, la ResNet-50 no llegó a funcionar bien porque la dimensión de entrada de las imágenes era demasiado pequeña para tantas capas. De esta primera prueba, la red que mejores resultados a aportado ha sido la VGG-19.

En segundo lugar, para la detección de señales de tráfico, hemos probado el detector de aprendizaje profundo basado en regresión YOLOv3 para tres diferentes datasets. Hemos podido apreciar que tiene gran influencia en el rendimiento: la variedad y calidad de imágenes con la que lo entrenamos, como la cantidad y formas de las clases a predecir. Aunque tuviésemos un conjunto de datos muy grande y variado, al querer identificar en la imagen muchas señales parecidas entre ellas, no conseguiríamos que YOLO funcionara de una manera consistente; fallando en la clasificación por señales parecidas entre ellas. De esta manera, al adaptar un dataset a 9 superclases según su forma, conseguimos mejorar en gran medida la detección. En suma, al aumentar la definición de entrada de la red, tanto para el entrenamiento como para la predicción, los resultados en la clasificación mejoraban de manera significativa.

Tras toda esta experiencia, la línea futura más prometedora para el desarrollo de un detector de señales de tráfico sería, con el dataset de Mapillary, dividir las 102 clases en no más de 20 superclases basadas en las formas de las señales. Más tarde, obtener todas ellas recortadas y entrenar a 20 clasificadores VGG-19 distintos, cada uno especializado en detectar las subclases pertinentes. Lo siguiente, evidentemente, sería educar a nuestro YOLOv3 hasta conseguir un buen resultado de detección. Para finalizar, conectaríamos el YOLOv3 con los clasificadores específicos, de tal manera que, cuando detecte una señal, se la pase recortada al clasificador asociado a esa superclase y la reconozca. Con esto, estaríamos aprovechándonos de los puntos fuertes de cada una de las técnicas y conseguir reconocer de manera específica la señal.

También, resultaría muy interesante poder probar distintos modelos de YOLOv3 para comparar el rendimiento respecto a cada uno de ellos. Además, sería muy positivo poder entrenar a nuestro detector con una mejor resolución de entrada – para que pudiera tener obtener mejores detalles de las señales – algo que nos ha sido imposible por la falta de recursos Hardware.

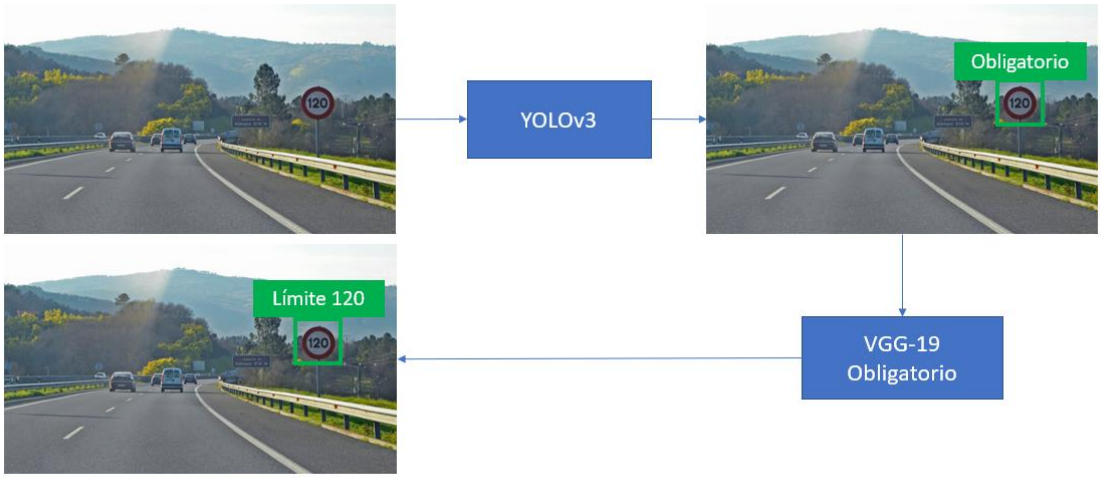


Ilustración 81 - Esquema de la mejor aproximación para el detector de señales

BIBLIOGRAFÍA

[1] A. Ng, K. Katanforoosh y Y. Bensouda Mourri, «Programa especializado Aprendizaje profundo,» deeplearning.ai, [En línea]. Available: https://es.coursera.org/specializations/deep-learning.

[2] J. J. Huerta Fernández, «www.aprendemachinelearning.com,» Diciembre 2018. [En línea]. Available: https://www.aprendemachinelearning.com/como-funcionan-las-convolutional-neural-networks-vision-por-ordenador/.

[3] J. Durán, «https://medium.com/,» Septiembre 2019. [En línea]. Available: https://medium.com/metadatos/todo-lo-que-necesitas-saber-sobre-el-descenso-del-gradiente-aplicado-a-redes-neuronales-19bdbb706a78.

[4] J. Stallkamp, M. Schlipsing, J. Salmen y C. Igel, «Man vs. Computer: Benchmarking Machine Learning Algorithms for Traffic Sign Recognition,» 20 Febrero 2012. [En línea]. Available: http://dx.doi.org/10.1016/j.neunet.2012.02.016.

[5] J. Xu, «https://www.deeplearningitalia.com/,» [En línea]. Available: https://www.deeplearningitalia.com/guia-para-arquitecturas-de-redes-profundas/.

[6] J. Xu, «https://www.deeplearningitalia.com/,» [En línea]. Available: https://www.deeplearningitalia.com/uso-del-aprendizaje-profundo-para-el-reconocimiento-de-objetos/.

[7] M. Menegaz, «https://hackernoon.com/,» [En línea]. Available: https://hackernoon.com/understanding-yolo-f5a74bbc7967.

[8] J. Redmon, «Darknet: Open Source Neural Networks in C,» 2013 - 2016. [En línea]. Available: http://pjreddie.com/darknet/.

[9] Alexey, «Windows and Linux version of Darknet Yolo v3 & v2 Neural Networks for object detection (Tensor Cores are used),» [En línea]. Available: https://github.com/AlexeyAB/darknet#datasets.

[10] S. Houben, J. Stallkamp , J. Salmen , M. Schlipsing y C. Igel, «Detection of Traffic Signs in Real-World Images: The {G}erman {T}raffic {S}ign {D}etection {B}enchmark,» 2013. [En línea]. Available: http://benchmark.ini.rub.de/?section=gtsdb&subsection=news.

[11] Mapillary, «Mapillary Traffic Sign Dataset,» [En línea]. Available: https://www.mapillary.com/dataset/trafficsign.

[12] Á. L. Igareta Herraiz, «SaferAuto: sistema de asistencia a la conducción,» 2019. [En línea]. Available: https://riull.ull.es/xmlui/handle/915/14512.

[13] M. Maj, «https://appsilon.com/,» [En línea]. Available: https://appsilon.com/object-detection-yolo-algorithm/.

ILUSTRACIONES

Ilustración 31 – Fuente: https://scikit-image.org/_static/img/logo.png
Ilustración 32 – Fuente: https://upload.wikimedia.org/wikipedia/commons/thumb/1/1a/NumPy_logo.svg/1163px-NumPy_logo.svg.png
Ilustración 33 – Fuente: https://upload.wikimedia.org/wikipedia/commons/thumb/1/11/TensorFlowLogo.svg/522px-TensorFlowLogo.svg.png
Ilustración 34 – Fuente: https://rpubs.com/chzelada/275494
Ilustración 35 – Fuente: https://cdn.ttgtmedia.com/rms/onlineImages/what_is-traditional_vs_deep_learning_desktop.jpg
Ilustración 36 – Fuente: https://scikit-image.org/docs/dev/_images/sphx_glr_plot_hog_001.png
Ilustración 37 – Fuente: apuntes de la asignatura Aprendizaje Formal.
Ilustración 38 – Fuente: https://upload.wikimedia.org/wikipedia/commons/b/b1/Svm_8_polinomial.JPG
Ilustración 39 – Fuente: propia.
Ilustración 40 – Fuente: resultados propios.
Ilustración 41 – Fuente: dibujo propio.
Ilustración 42 – Fuente: dibujo propio.
Ilustración 43 – Fuente: https://miro.medium.com/max/588/0*qrMVR8XCPceU7dnP.png
Ilustración 44 – Fuente: https://www.researchgate.net/profile/Clifford_Yang/publication/325137356/figure/fig2/AS:670371271413777@1536840374533/Illustration-of-the-network-architecture-of-VGG-19-model-conv-means-convolution-FC-means.jpg
Ilustración 45 – Fuente: http://www.jesusutrera.com/articles/img/resnet.png
Ilustración 46 – Fuente: http://www.jesusutrera.com/articles/img/a7im001.png
Ilustración 47 – Fuente: propia captura.
Ilustración 48 – Fuente: https://appsilon.com/wp-content/uploads/2018/08/types.png
Ilustración 49 – Fuente: ejemplo detector de señales tráfico.
Ilustración 50 – Fuente: https://sites.google.com/site/5kk73gpu2012/_/rsrc/1354215989995/assignment/viola-jones-face-detection/sliding_window.jpg?height=199&width=400
Ilustración 51 – Fuente: https://www.deeplearningitalia.com/wp-content/uploads/2018/06/2.png
Ilustración 52 – Fuente: https://www.deeplearningitalia.com/wp-content/uploads/2018/06/3-1.png
Ilustración 53 – Fuente: https://www.deeplearningitalia.com/wp-content/uploads/2018/06/4-1.png
Ilustración 54 – Fuente: https://www.deeplearningitalia.com/wp-content/uploads/2018/06/5-1.png
Ilustración 55 – Fuente: https://cv-tricks.com/wp-content/uploads/2017/12/Various-detectors-2.jpg
Ilustración 56 – Fuente: https://i.stack.imgur.com/zlhvo.png
Ilustración 57 – Fuente: https://i.stack.imgur.com/aUcNf.jpg
Ilustración 58 – Fuente: https://miro.medium.com/max/630/1*8Q8r9ixjTiKLi1mrF36xCw.jpeg
Ilustración 59 – Fuente: https://www.pyimagesearch.com/wp-content/uploads/2016/09/iou_equation.png
Ilustración 60 – Fuente: https://www.pyimagesearch.com/wpcontent/uploads/2016/09/iou_stop_sign.jpg
Ilustración 61 – Fuente: https://appsilon.com/wp-content/uploads/2018/08/nonmax-1.png
Ilustración 62 – Fuente: https://hackernoon.com/understanding-yolo-f5a74bbc7967