CNN project

Jose Luis Martínez Martínez

October 2025

1 Breve introducción a las CNN

Las Redes Neuronales Convolucionales (CNN) son un tipo de estructuras con aplicación al Aprendizaje Automatizado (ML) que le añaden al 'clásico' fully-conected perceptrón multicapa bloques de tratamiento/filtrado (operación conocida como convolución).

En la explicación nos limitaremos a dilucidar aspectos relacionados con este componente añadido de convolución, pues es la parte que caracteriza a este tipo de Redes Neuronales. Además puesto que su aplicación está mayoritariamente enfocada a la visión computacional (clasificación de imagines, detección de objetos...) y puesto que ésta será la aplicación que le daré para el proyecto; la explicación, tanto en su contenido como en su forma, estará ajustada a este ámbito.

Dicho esto, la forma más sencilla de entenderlas es a través de sus partes. Los bloques convolucionales están participados de los siguientes actores:

1. **Feature (Input):** La imagen de entrada que queremos analizar.
2. **Kernel (Filtro):** El objeto que recorrerá toda la imagen y que abstraerá una característica concreta de la misma (es el que realiza la convolución)
3. **Feature Map (Output):** La abstracción que optenemos una vez la imagen ha sido 'escaneada' por el Kernel.

Para una comprensión más profunda quizás ayude entender que cada uno de estos objetos es realmente una matriz nm donde las dimensiones representan la cantidad de pixeles de que están construidos.

La siguiente imagen muestra una simplificación de como operarían estos agentes en un bloque convolucional, donde el Kernel iría recorriendo la imagen y aplicando la convolución (que, como veremos, es simplemente la suma de una multiplicación elemento por elemento entre el Kernel y el cacho equivalente en el Feture) hasta formar la imagen abstraída/filtrada del Feature Map.

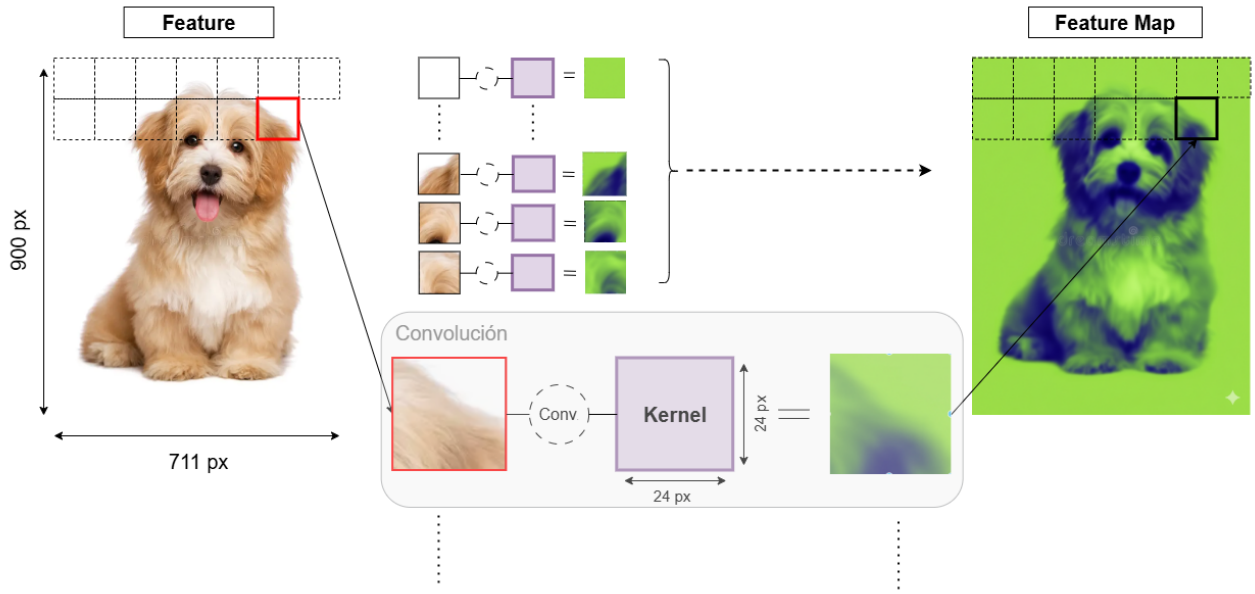

Figure 1: Representación vaga de un Bloque Convolucional

Si bien esta imagen puede ayudarnos a tener una idea visual de lo que esta ocurriendo, antes o después las simplificaciones acaban por no ser suficiente y debemos pasar a un plano más objetivo si queremos entender verdaderamente lo que esta ocurriendo. Por este motivo vamos a dejar a un lado las imágenes y a expresar todo en función de matrices. Pero antes seria conveniente explicar como una imagen puede ser evaluada como una matriz:

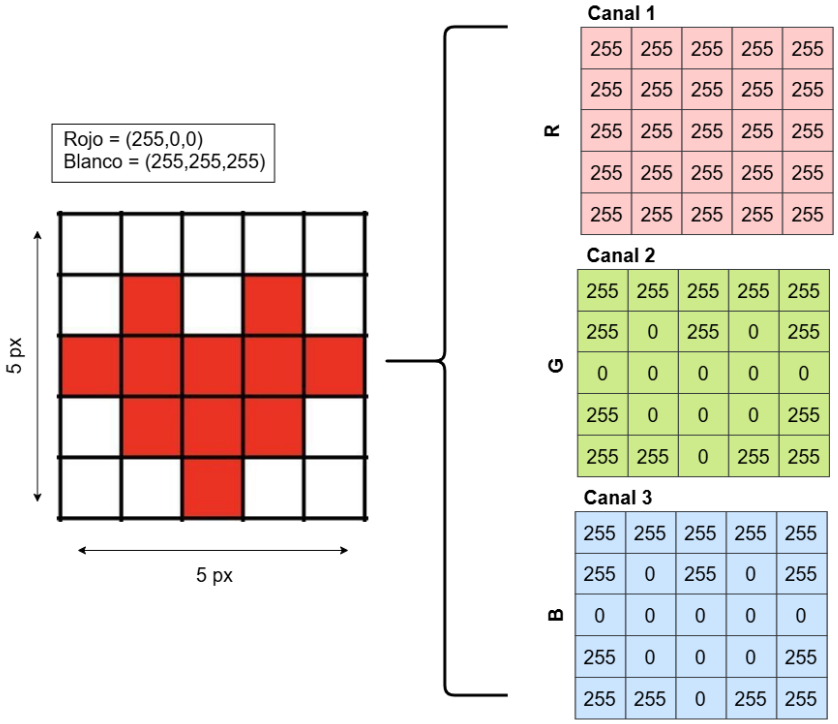

Figure 2: De Imagen a Datos tratables (Matriz)

La imagen que se ve muestra como el Feature es dividido en tres matrices diferentes, representando cada una de ellas el valor que ocuparía ese pixel en el sistema de coloreado RGB en cada uno de los tres estadios de la tupla $(R, G, B) \rightarrow R \equiv red, G \equiv Green, B \equiv blue$. Cuando tratamos con imágenes esta es una practica

habitual y nos introduce a lo que se conoce como **Canales de Entrada**, que seria cada una de las matrices que metemos como input en el Bloque Convolucional. De la misma forma tendríamos **Canales de Salida** cuya cuantia vendrá dada por el numero de Kernels que estipulemos; porque sí: podemos poner tantos Kernels como queramos y la idea es que cada uno de ellos *aprenda* a diferenciar un aspecto concreto de las imagenes, lo que le dará criterio para la ulterior clasificación. El numero de estos Canales marcara la **Profundidad** de nuestro Bloque.

Una vez normalizados los valores del RGB entre 0 y 1, la imagen del corazón atravesando un Bloque convolucional se vería algo así:

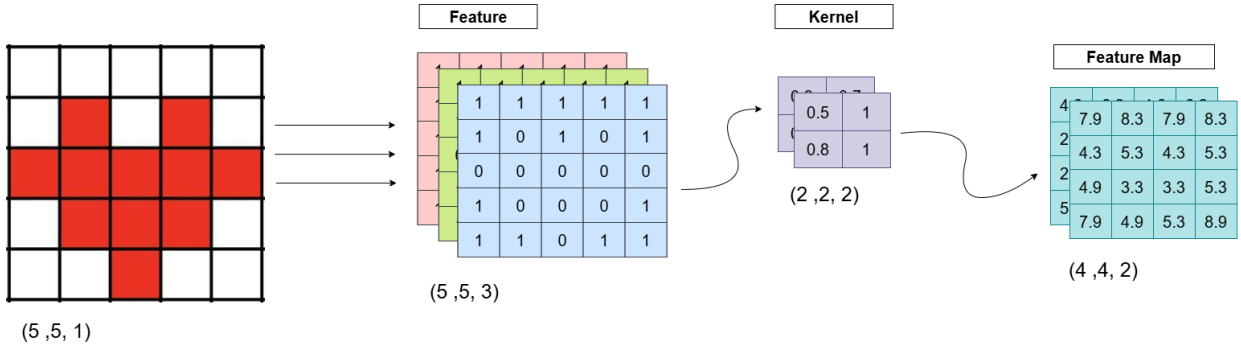

Figure 3: Imagen atravesando Bloque Convolucional, con 3 Canales de Entrada y 2 Canales de Salida

Aquí podemos ver ya como la imagen se descompone en los tres Canales de Entrada habituales, una vez pasan por el filtrado del Kernel ontenemos un Feature Map con 2 Canales de Salida (uno por cada Kernel que tenemos).

Quizás os estéis preguntando, cómo es que hemos obtenidos los valores de este Feature Map final; pues bien, todo lo que hemos hecho no es mas que la sumatoria de una multiplicación matricial elemento por elemento (se multiplican los valores en posiciones coincidentes), a esta operación es a la que se le pone el nombre de Convolución y da nombre a este tipo de estructuras. De momento vamos a coger simplemente uno de los canales del Feature, digamos el canal Azul:

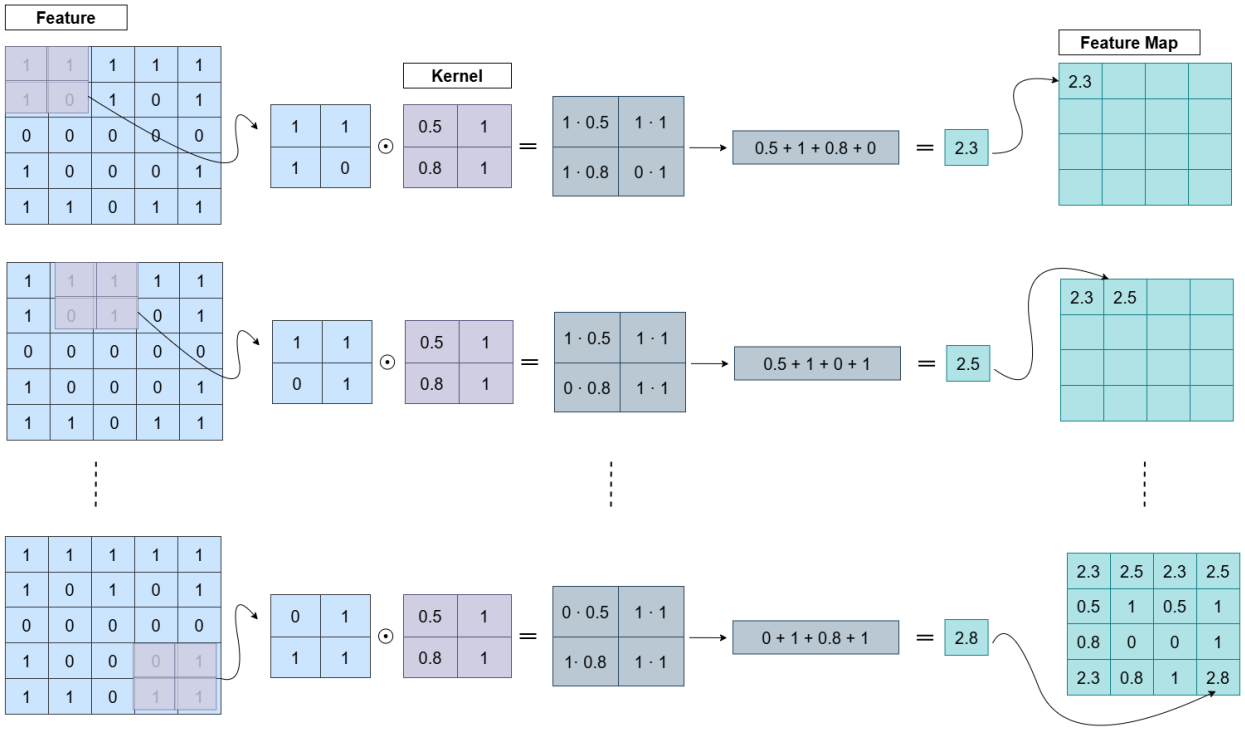

Figure 4: Proceso de Convolución para un solo Canal de entrada

Como se puede ver, el Kernel va escaneando todo el Feature y va rellenando los valores del Feature Map con el agregado de la multiplicación elemento a elemento que realiza sobre el cacho de Feature correspondiente. Ahora bien, en nuestro caso y en la mayoría de casos reales (a menos que las imágenes estén en monocromático) siempre tendremos estos 3 Canales de Entrada, no solamente 1. En estos casos lo que se hace es computar esta Matriz final para cada uno de los canales, y después sumar las tres matrices, de tal forma que cada renglón este ocupado por la suma de los tres renglones respectivos de cada uno de estas matrices finales. Entonces si repetimos este proceso para los 2 Canales de entrada restantes tendríamos:

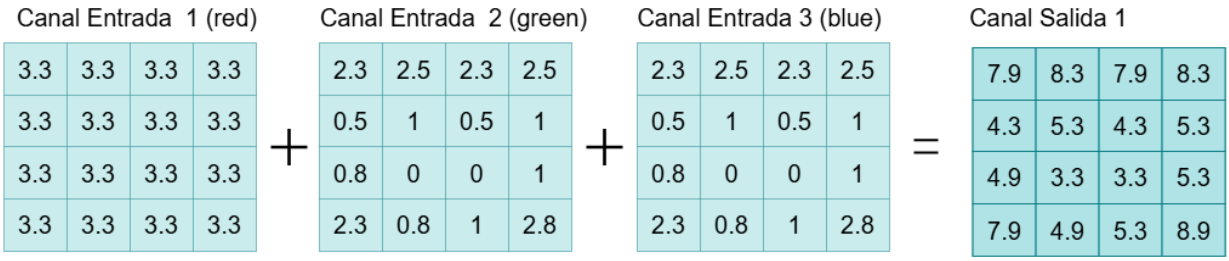

Figure 5: Agreagdo final entre diferentes Canales de Entrada

Cabe decir que en nuestro ejemplo contamos no sólo con un Kernel sino con dos, por tanto este proceso se repetirá de forma idéntica para los valores de este segundo Kernel, obteniendo un Feature Map de Profundidad 2. Controlar las dimensiones de antemano facilitará el proceso de programación de estas redes (en la practica el numero de Kernels utilizados es bastante más elevado), pero lo importante es entender que está ocurriendo, después, aunque las dimensiones se tornen absurdamente grandes, sabrás que simplemente es una repetición agran escala de este proceso.

Todo esto está bien, pero el objetivo detrás de toda esta estructura es *aprender* a *clasificar* imágenes, y

por el momento ni hemos enseñado a nadie ni hemos clasificado nada. Como hemos mencionado antes, las CNN son una combinación de Bloques Convolucionales y Perceptrones Multicapa (Redes Neuronales al uso), todo este tratamiento con el Kernel y las Capas de Salida y todo eso sólo sirve para simplificar la imagen y crear un contexto bajo el cual la estructura pueda extraer patrones; patrones suficientemente simples como para construir un criterio generalizable pero suficientemente complejos como para que su predicción no sea una basura. De esta parte es de la que se encarga el Bloque Convolucional, donde cada uno de los Kernels (Canales de Salida) representará una característica destacable o diferenciable que le ayude a distinguir entre las diferentes etiquetas de que cuenta su base de datos de entrenamiento. En efecto, podríamos alimentar a un Perceptrón directamente con la imagen en crudo (utilizando una ristra de nodos de entrena formada por todos los pixeles de la imagen), y entrenarlo para que clasificara entre ciertas etiquetas, y esto para según que casos nos daría resultados decentes, el problema comienza cuando intentamos tratar con imágenes más complejas, formadas por mayor cantidad de pixeles; no es viable alimentar un Perceptron con pongamos $1000x1000$ nodos de entrada donde cada nodo este conectado con todo el resto, el entrenamiento sería muy engorroso y a penas se podría extraer información útil, por no hablar de que con este sistema el más mínimo cambio en la imagen (digamos voltear la imagen horizontalmente) haría que la Red la interpretara como algo diametralmente distinto; una de las ventajas más notorias de las CNN es que consiguen que el análisis de patrones (con este sistema de Kernels) no dependa tanto del orden de los datos sino de su coherencia local (el Kernel absorbe información por clusters, es como si entendiera la imagen por trozos y luego la asimilara en su conjunto en lugar de percibirla como algo puramente estático).
Una vez tenemos la imagen tratada y tamizada en estos Canales de Salida generados con los distintos Kernels, el Perceptrón ya puede asimilar mejor la información y por tanto tiene más sentido unificar todos los valores que hemos obtenido en los distintos Canales de Salida (hacer un *array* con ellos) y utilizarlos de nodos de entrada, para ahora sí entrenarlo para clasificar entre las distintas etiquetas. Continuando con nuestro ejemplo tendríamos algo como esto:

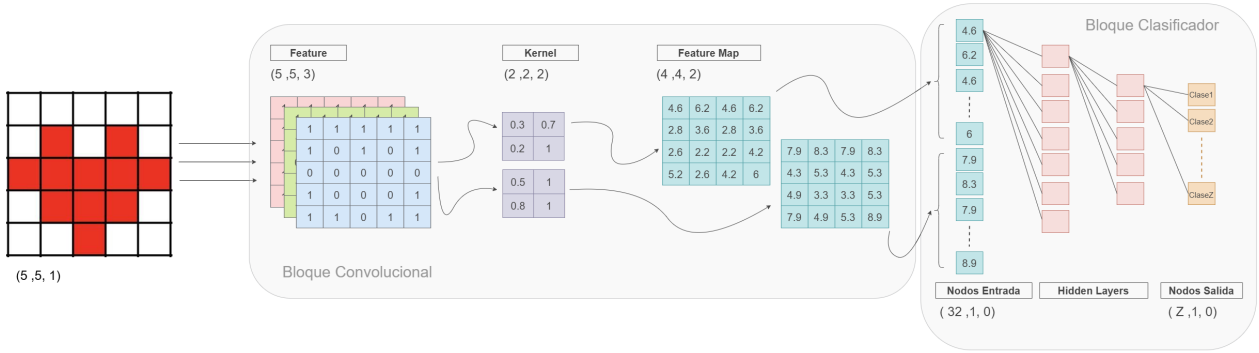

Figure 6: Representación vaga de CNN completa

Entonces, como hemos visto, una CNN es una especie de tubería que va destilando una imagen para luego alimentar a una Red Neuronal encargada de clasificarla. Con esto ya hemos cubierto el tema de como tal estructura es capaz de recibir una imagen y devolver una etiqueta; pero como cualquier otra Red Neuronal, este proceso requiere de un entrenamiento previo, de lo contrario la elección de los distintos parámetros encargados de la *toma de decisión* serían puramente aleatorios (como el caso de los ejemplos que hemos mostrado) y la predicción sería un fiasco. En este apartado no voy a meterme en profundidad en este tema simplemente que sepais que el método de entrenameinto puede variar pero casi siempre está dirigido por métodos de *Gradiente Descendiente*, bajo el cual se calcula el efecto marginal (derivada/gradiente) que tiene cada parámetro sobre la Función de Perdida y se actualiza su valor en función de este. Los parámetros a entrenar del Bloque de Clasificación son los habituales; pero el Bloque Convolucional también posee parámetros propios: estos parámetros vienen representados por los valores del Kernel (que también se les llama *weights*) y un *bias* único por Kernel (común a todo el Kernel).

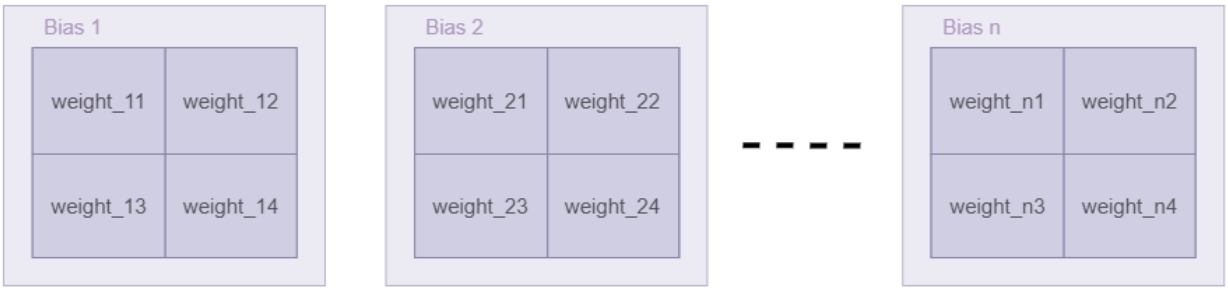

Figure 7: Conjunto de parámetros de un Bloque Convolucional

Y ya para terminar este apartado simplemente comentar que a la hora de diseñar el BLoque COnvolucional, hay tres variables que se tienen en cuenta a cerca del Kernel y que afectan a las dimensiones de las imágenes filtradas y varían según el objetivo del que lo diseña (*Hyperparámetros*):

1. **Size:** que ya hemos visto y que representa que dimensiones queremos que tenga el Kernel.
2. **Padding:** debido a la naturaleza del filtrado muchas veces se pierde información de los bordes, es por esto que en la matriz del Feature se suele añadir un Margen (formado por un borde de ceros al rededor de la matriz).
3. **Stride**: que representa cuantas columnas se desplaza el Kernel entre una convolución y otra durante el proceso de filtrado. Es un valor que controla un poco la densidad del Feature Map.

2 Implementación: ResNet

Todo lo que hemos visto anteriormente cierra un poco el marco teórico de las CNN y nos da una visión general de lo que esta ocurriendo bajo estas estructuras; a la hora de traerlas a la practica otros factores como la eficiencia computacional o la convergencia del entrenamiento (optimización) cobran mucha relevancia y se tienen en cuenta para diseñar este tipo de Redes Neuronales. Esto, entre otras cosas, hace que a lo largo de la vida de las CNN hayan sido muchas las propuestas que se han entregado. La implementación que voy a explicar a continuación (y la que usaremos para el proyecto) es un tipo especial de CNN que se diseñó para aliviar la carga de las CNN profundas y evitar problemáticas que los investigadores se encontraban a la hora de querer entrenar este tipo de CNN con muchas capas (principalmente el problema del *vanishing gradient*). Como pone en el título estas CNN son las Residual Nets o ResNet que reciben el nombre a colación de su mayor aportación: algo llamado Bloques Residuales, capaces de mantener presentes durante todo el entreno a las capas anteriores, evitando perdida de información (algo así como las Recurrent Neural Networks (RNN) para las series temporales).

Pero vamos por partes:

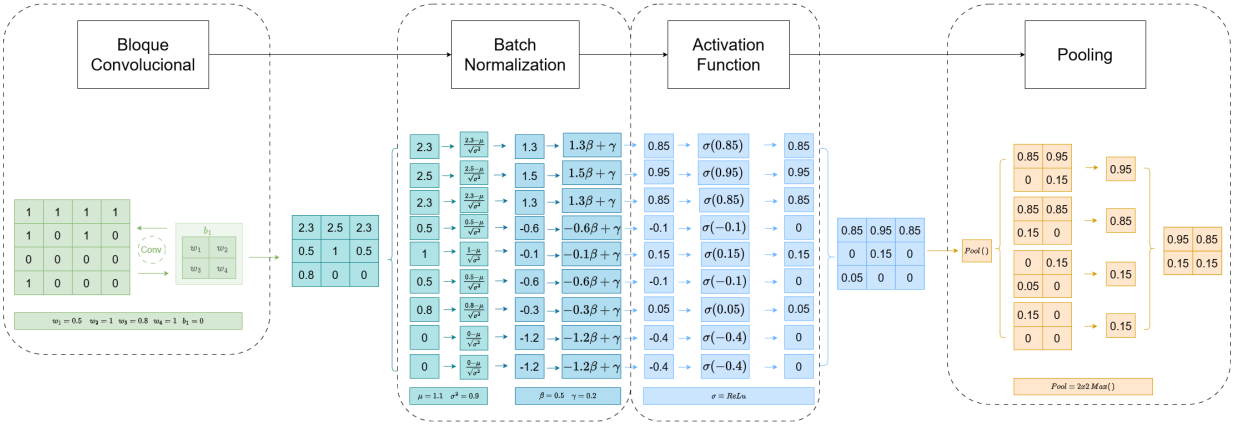

Figure 8: Ejemplo más real de una CNN con un único Canal de entrada y un único Canal de Salida.

Esto que habéis visto es un ejemplo de a lo que me refiero cuando digo que en la practica otros factores se tienen en cuenta (dirigidos a facilitar la implementación de estas estructuras teóricas) y es un ejemplo más certero de lo que nos ponemos encontrar en la vida real. Como veis, no es tan diferente a lo que habíamos visto, simplemente introduce 3 bloques ajenos a la Convolución, destinados a mejorar el rendimiento de la Red. No me extenderé mucho en ellos, pero es importante presentarlo pues a la hora de escribir el código deberemos tenerlos en cuenta:

1. **Batch Normalization:** Su objetivo es acelerar el entrenamiento por medio de la reducción de algo llamado *Internal Covariate Shift*. Esto se hace Normalizando el Bach de entrado y reescalando los valores normalizados con 2 parámetros (β, γ). Parámetros que serán actualizados por medio del entrenamiento (forman parte de la optimización).

2. **Activation Function:** La hemos mencionado ya antes pero no la hemos dibujado en el flujo. Tienen la capacidad de introducir *non-linearities* en la función lo que permite que la Red Neuronal asimilé y colija muestras de datos con estructura más dinámicas (e.j no lineales: como suele ser el caso de casi toda la realidad).

3. **Pooling:** Enfocado en la reducción de complejidad de la Red (acción que suele mejorar la generalización de las predicciones y agiliza la convergencia). Se trata de reducir el numero de valores dentro de un Canal de Salida. Esto se hace principalmente por 2 métodos, cogiendo el valor Máximo de cierta bolsa o cogiendo el valor Promedio de cierta bolsa, dentro de ese Canal de Salida.

Ahora estamos más preparados para entender la arquitectura de las ResNet:

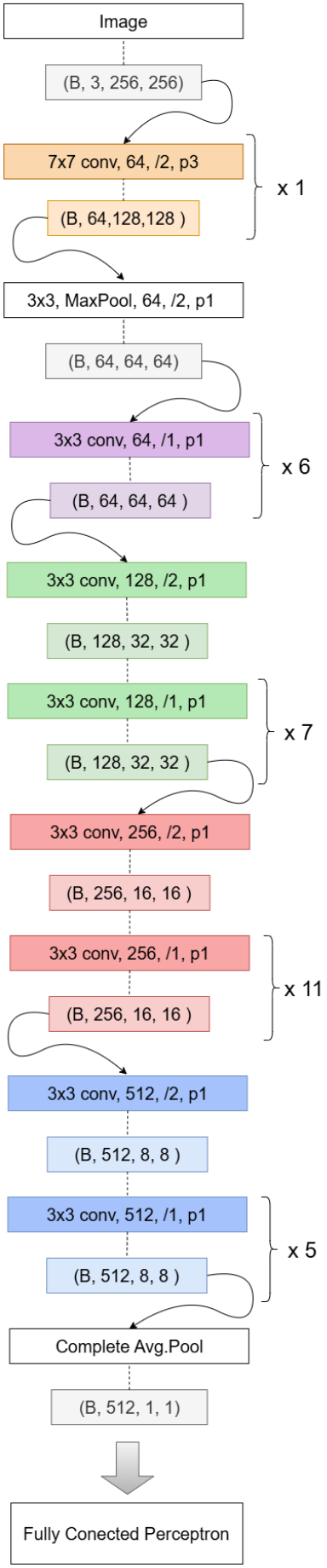

Figure 9: Abstracción de Cadena ResNet donde: (B,64,128,128) $\equiv$ (batch size, channels, height, width); (7x7 conv, 64, /2, p3)$\equiv$ (size, number of kernels, stride, padding)

La imagen anterior es un croquis de la arquitectura de las ResNet y está basado en el paper original que la desarrolla; es una buena primera impresión de lo que son, pero obvia la parte más importante que habíamos mencionado antes: los Bloques Residuales. La imagen que os muestro a continuación si que está extraída directamente del paper original. En ella se ve como las parejas de bloques están agrupadas por una flecha que parece estar proyectando el resultado pasado en relaciones futuras. Pues bien a cada uno de estos pares se les conoce como Bloques residuales y están involucrados en algo llamado *Shortcuts*, que de alguna forma aligera la computación de la Red y le permite no olvidar estancias pasadas dentro de las CNN tan profundas como es la que nos atiene. Y es lo que trataré de explicar a continuación.

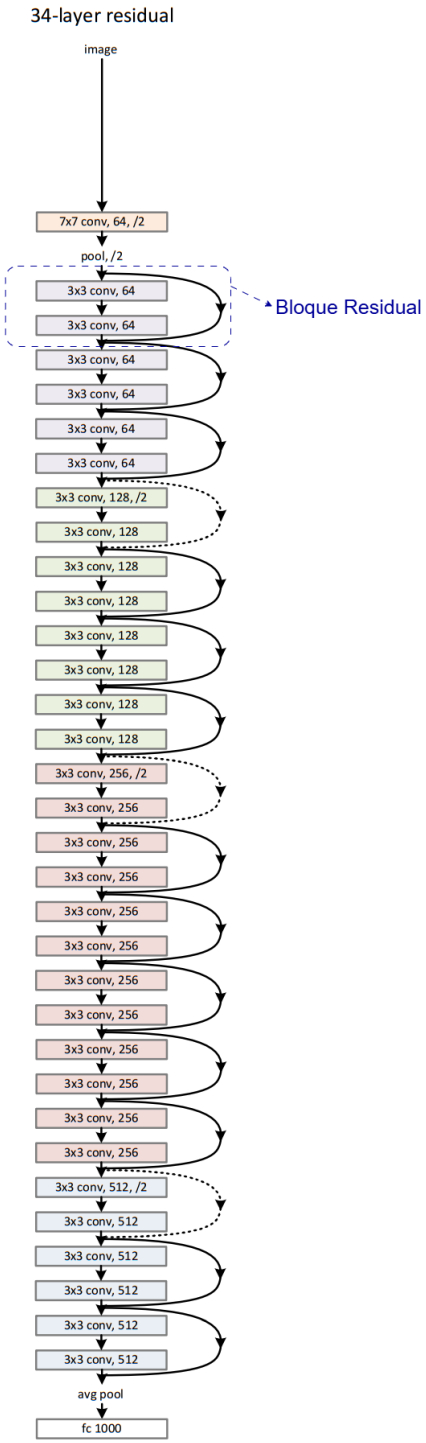

Figure 10: CNN con Bloques Residuales

Esta idea de introducir Residuos esta respaldada por una hipótesis que los autores desarrollan en su paper pero que no desarrollare aquí. La idea fundamental que subyace es la de que resulta más fácil (o menos complejo) entrenar (aprender) en relación a un Residuo, entendiéndose por éste la diferencia entre algo que ya se tiene y el siguiente resultado que se pretende obtener; que sobre el resultado en crudo. Es decir obtener diferencias basadas en resultados anteriores es más sencillo que tratar de aprender algo completamente de nuevas. cómo se materializa esto?:

1. $\mathbf{H}(\mathbf{x})$: Nuestro objetivo último (*Underlying Mapping*)
2. $\mathbf{F}(\mathbf{x})$: Lo que hacemos que aprenda (*Residual Mapping*)
3. $\mathbf{x}$: El resultado que ya conociamos (*Input*)

Con esto podemos definir el Residuo como:

$$F(x) = H(x) - x$$

Y recuperar el Underlying Mapping una vez aprendido el Residual Mapping con la siguiente expresión:

$$H(x) = F(x) + x$$

Ahora sí, entenderemos por Shortcut a estas conexiones de a dos que se observan en la figura 10 y que en nuestro caso representan un mapeo de el anterior output a modo de input en los subsiguientes bloques, a través de la formula que acabamos de ver.

A continuación muestro un esquema más programático de como se reflejará esta estructura en un código, junto con una intuición visual de lo que está ocurriendo durante el proceso de refinamiento que forman los bloques Convolucionales, que complementa a la figura 9 de antes:

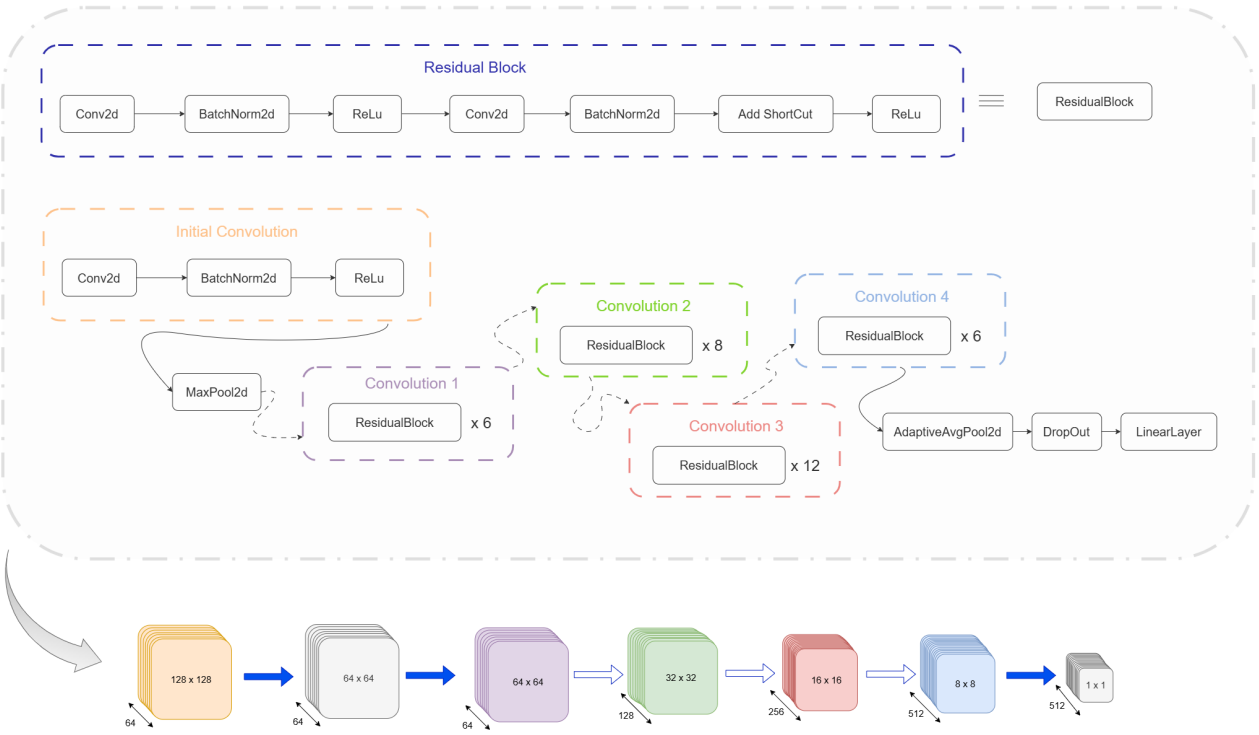

Figure 11: Esquema Flujo de Código de la ResNEt, más representación visual de tratamiento de la imagen

Para finalizar os mostraré un ejemplo real de lo que ocurre con la imagen (tan solo de algunos canales de salida) una vez entra en este torbellino convolucional. Esto se representa normalizando los valores y representándolos con un mapa de calor dónde los números más altos representan zonas más cálidas:

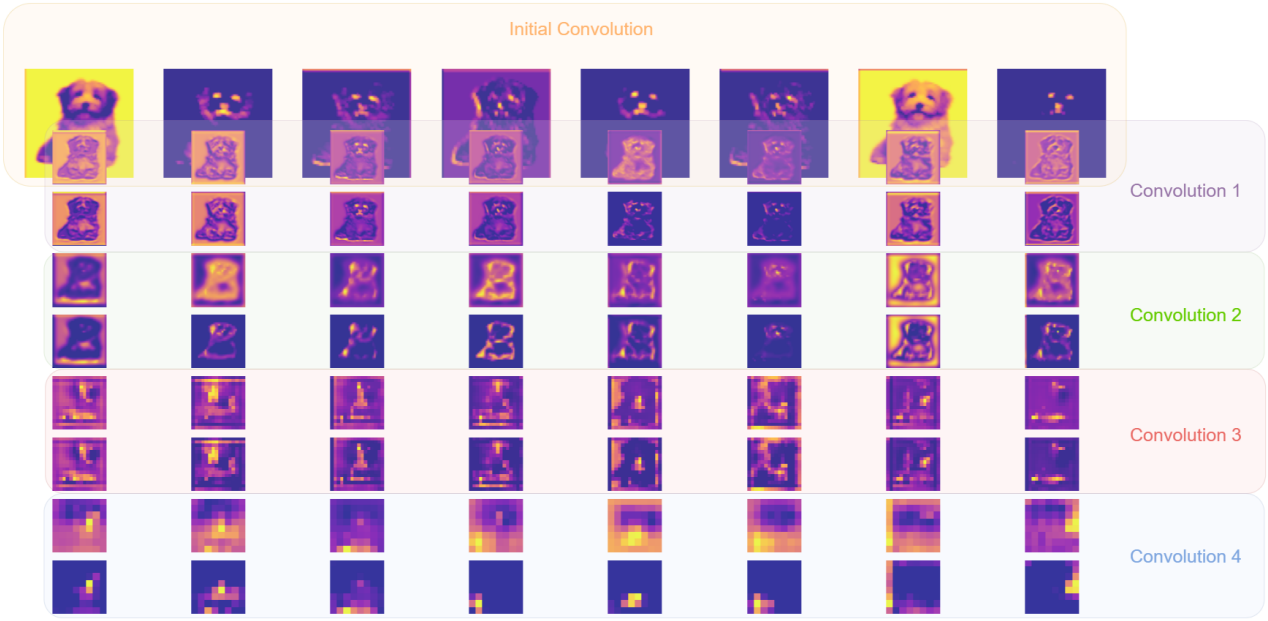

Figure 12: Primeros 8 canales de salida para las diferentes Capas Convolucionales de la ResNet