

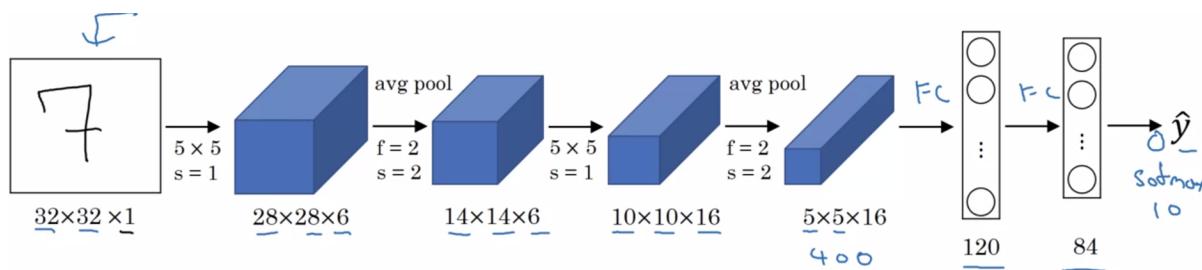
Deep Convolutional Models - Case Studies

• LeNet - 5

En sus inicios, LeNet-5 se utilizó para reconocer dígitos en una imagen en blanco y negro. Una de sus particularidades es que opera con capas de average pooling (en vez de max pooling), y tampoco se utiliza padding en las capas, por lo que siempre se reduce la dimensionalidad capa a capa. Para compensar el decrecimiento de estas dimensiones, a medida que se avanza más profundo en la red, la cantidad de canales (*nch*) aumenta. Además, es una red con pocos parámetros por aprender, en este caso tiene alrededor de 60k de parámetros, cuando otro tipo de redes tiene millones de parámetros por aprender.

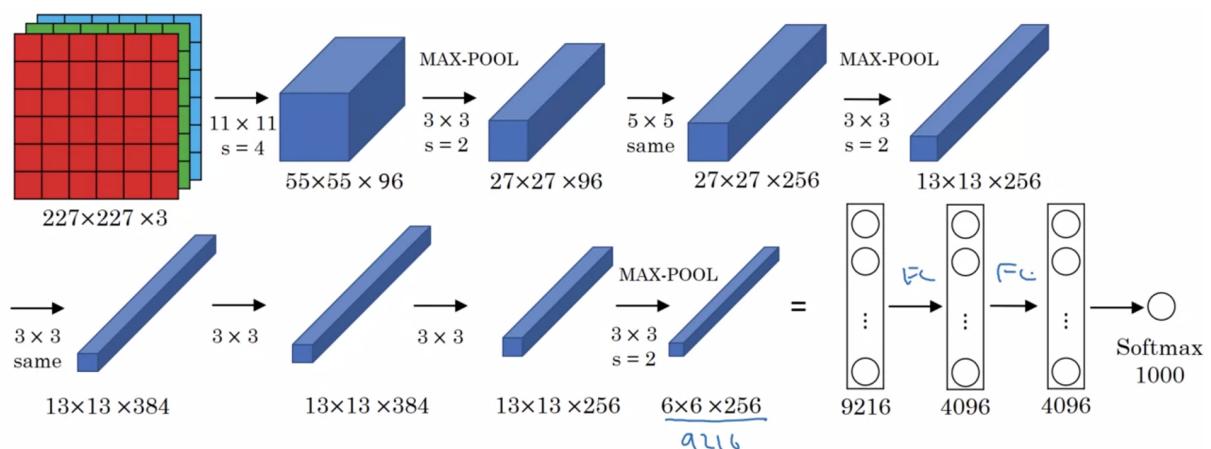
Tambien sigue el siguiente patrón entre capas:

CONV → *POOL* → *CONV* → *POOL* → *FC* → *FC* → *OUTPUT*



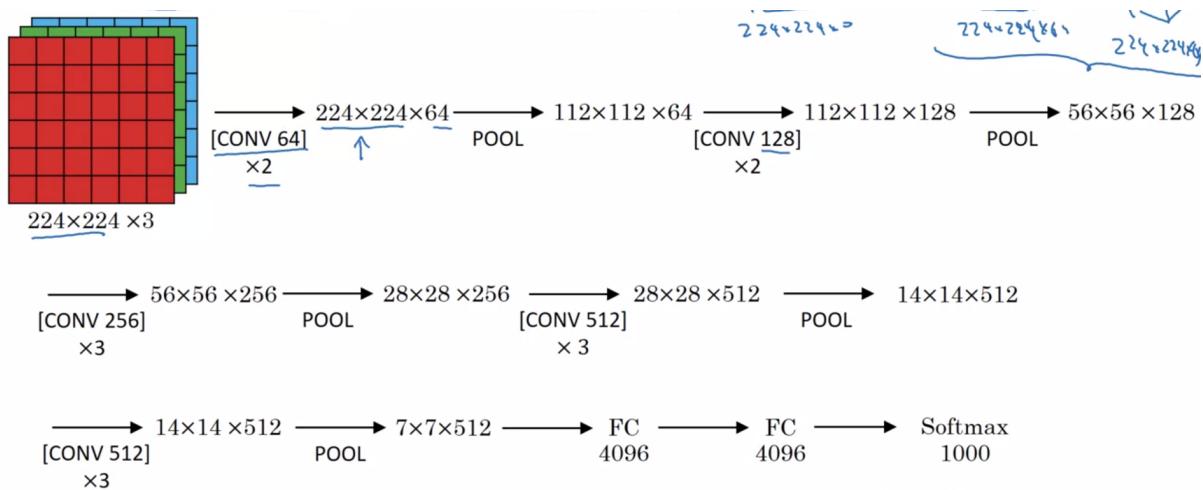
• AlexNet

Tiene varias similitudes con LeNet, pero muchisimo mas grande. Este tipo de red tiene alrededor de 60 millones de parámetros por aprender. Otro aspecto por el cual es mejor este tipo de red que la LeNet es que utiliza ReLU como función de activación en las capaz FC.



• VGG-16

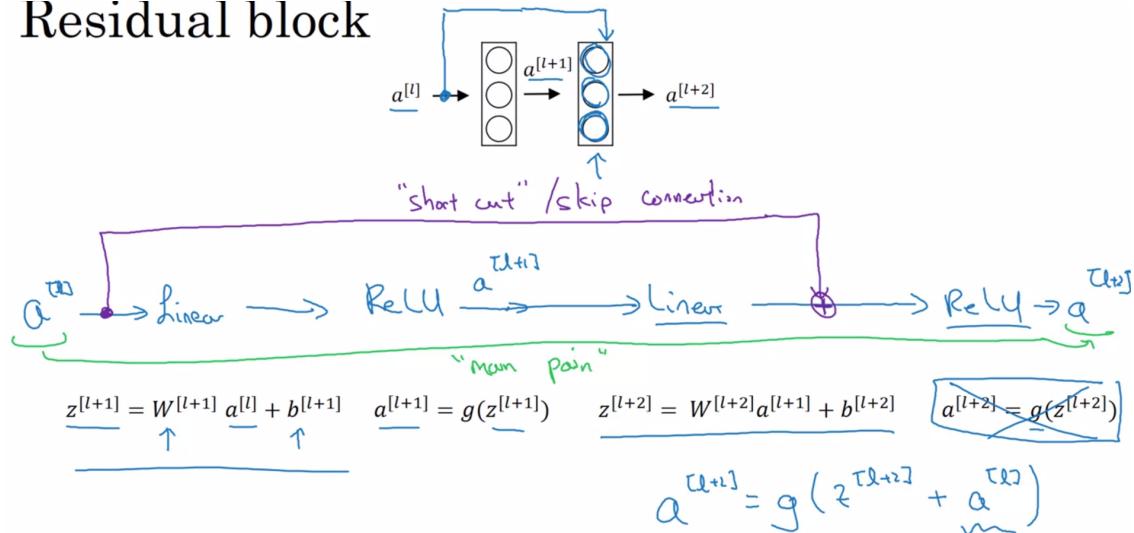
La particularidad de este tipo de red es que utiliza filtros de 3×3 con stride = 1, y el tipo de convolución que se realiza es del tipo same. Además, las capas de max pooling son de 2×2 con un stride de 2. Es una red con muchas capas y con muchos parámetros por aprender (aproximadamente 138 millones de parámetros). Si bien con las capas de pooling disminuye la dimensionalidad de las primeras dos dimensiones, cuando se pasa por las capas de convolución, se duplica la cantidad de canales.



• ResNets

Las redes muy grandes son complicadas de entrenar por la gran complejidad que tienen los gradientes para actualizar los parámetros. La particularidad de las ResNets es que se pueden saltar conexiones (**skip connections**), es decir, tomar la salida de una red del principio para alimentar una capa que está mucho más profunda en la red. Las ResNets generan algo que se llama **residual blocks**,

Residual block

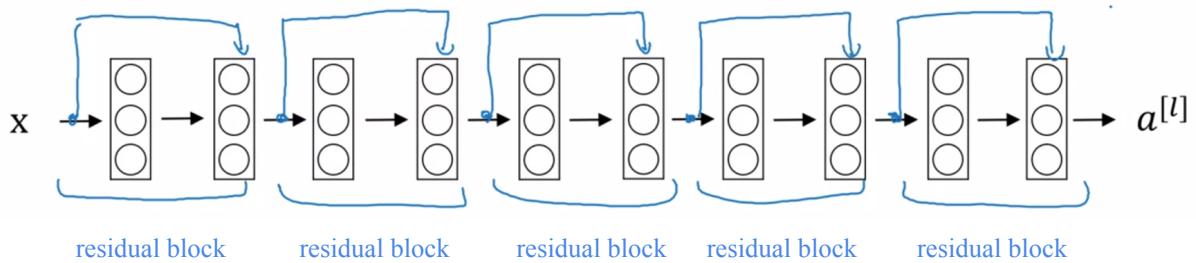


Este shortcut puede skippear las capas que se quieran, la muestra de $a[l]$ se inserta antes de la ReLU, por lo tanto, la ecuación que determina la salida del residual block será:

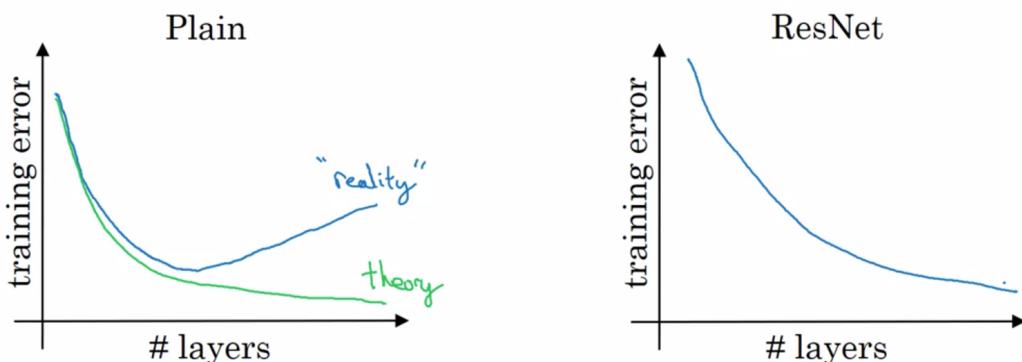
$$a^{[l+2]} = g(z^{[l+2]} + a^{[l]}) = g(W^{[l+2]}a^{[l+1]} + b^{[l+2]} + a^{[l]})$$

donde $a[l]$ es el shortcut que ingresa a la ReLU de la capa $l+2$.

Una aplicación de estos bloques se puede ver en esta red plana:



En una red convencional, a medida que aumentamos la cantidad de capas, el error en el entrenamiento teóricamente debería disminuir, pero en la práctica, el error disminuye hasta un determinado valor, y luego comienza a aumentar. Con las ResNets, a pesar de tener muchas capas, el error con el aumento de las capas sigue disminuyendo.

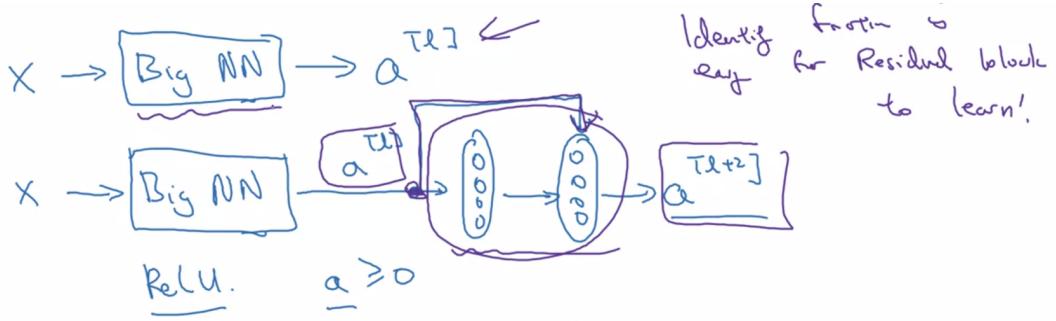


¿Por qué las ResNets funcionan tan bien?

Suponiendo que tengo una gran red, y le quiero acoplar un residual block en la salida, se cumple que:

$$a^{[l+2]} = g(z^{[l+2]} + a^{[l]}) = g(W^{[l+2]}a^{[l+1]} + b^{[l+2]} + a^{[l]})$$

En el caso de que $W=0$ y $b=0$, la salida será igual a $a[l]$, ya que en la salida de la BigNN se utiliza ReLU. Por ello añadir un residual block no afecta la performance.

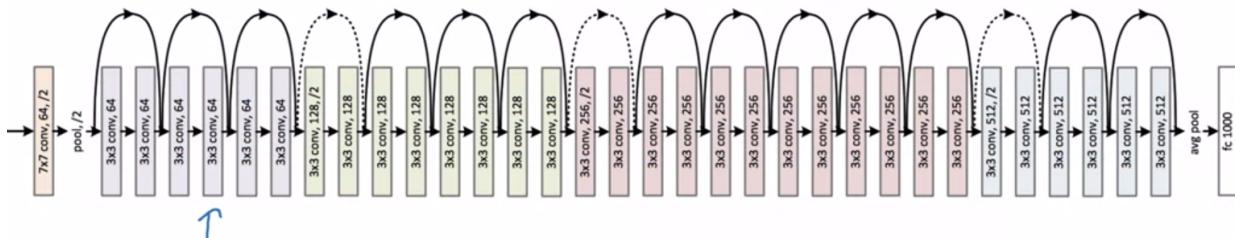


$$\begin{aligned}
 & \text{ReLU.} \quad \underline{a \geq 0} \\
 a^{[l+2]} &= g\left(\underline{\frac{z^{[l+2]}}{w^{[l+2]}}} + \underline{a^{[l]}}\right) \\
 &= g\left(\cancel{w^{[l+2]}} \cancel{a^{[l+2]}} + \cancel{b^{[l+2]}} + a^{[l]}\right) = g(a^{[l]}) \\
 & \quad \text{If } \cancel{w^{[l+2]}} = 0, \cancel{b^{[l+2]}} = 0 \quad \underline{= a^{[l]}}
 \end{aligned}$$

Plain



ResNet



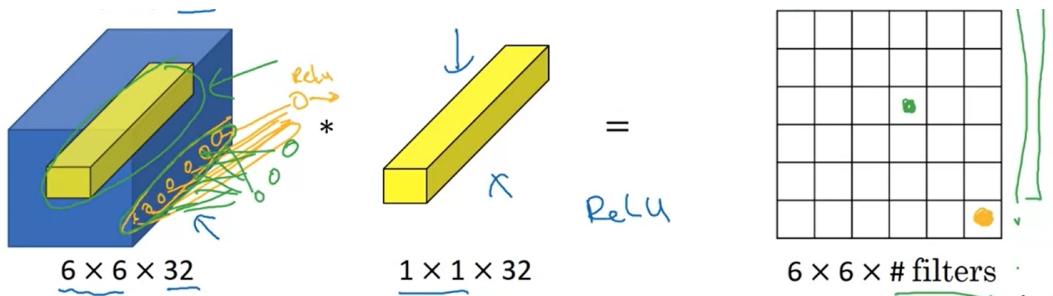
Una particularidad es que las capas convolucionales utilizan la convolución del tipo ‘same’ y suelen ser de 3x3.

• Networks in Networks and 1x1 Convolutions

Si hacemos la operación con una matriz 2D, es lo mismo que multiplicar la matriz por el número del filtro.

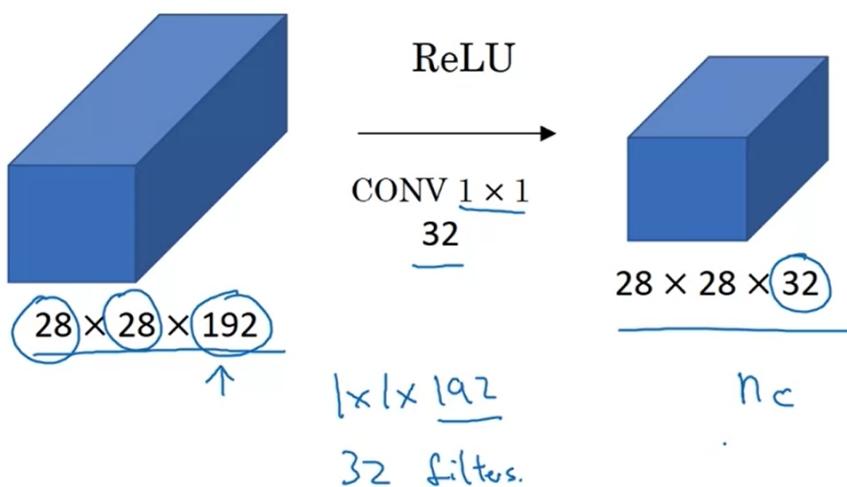
$$\begin{array}{|c|c|c|c|c|c|} \hline
 1 & 2 & 3 & 6 & 5 & 8 \\ \hline
 3 & 5 & 5 & 1 & 3 & 4 \\ \hline
 2 & 1 & 3 & 4 & 9 & 3 \\ \hline
 4 & 7 & 8 & 5 & 7 & 9 \\ \hline
 1 & 5 & 3 & 7 & 4 & 8 \\ \hline
 5 & 4 & 9 & 8 & 3 & 5 \\ \hline
 \end{array}
 \quad *
 \quad
 \begin{array}{|c|} \hline
 2 \\ \hline
\end{array}
 \quad =
 \quad
 \begin{array}{|c|c|c|c|c|c|} \hline
 2 & 4 & 6 & \dots \\ \hline
 \end{array}$$

En el caso de trabajar con volúmenes, la operación es más influyente.



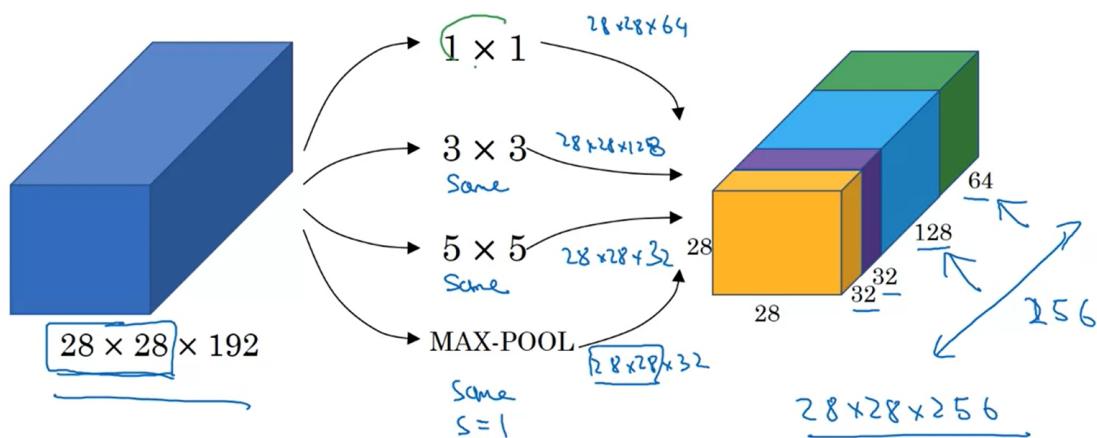
Básicamente, es como aplicar una fully connected network a cada uno de los volúmenes de tamaño $1 \times 1 \times 32$.

Una aplicación de las 1×1 convolutions es cuando se quiere mantener n_h y n_w pero disminuir n_c .

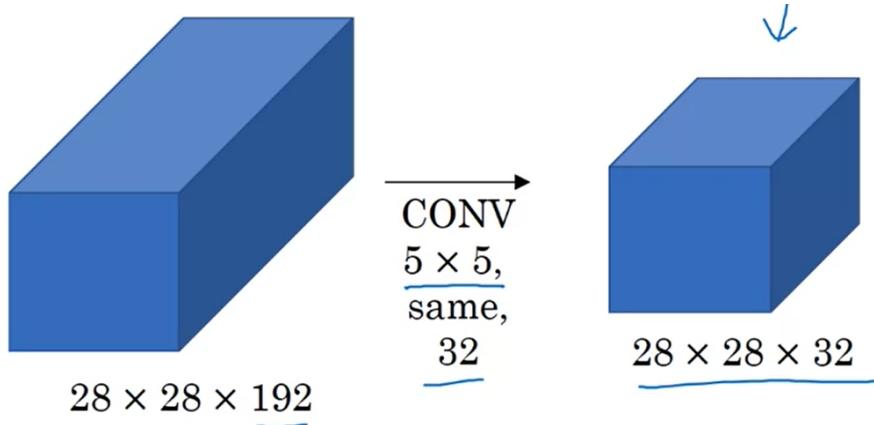


• Inception Network

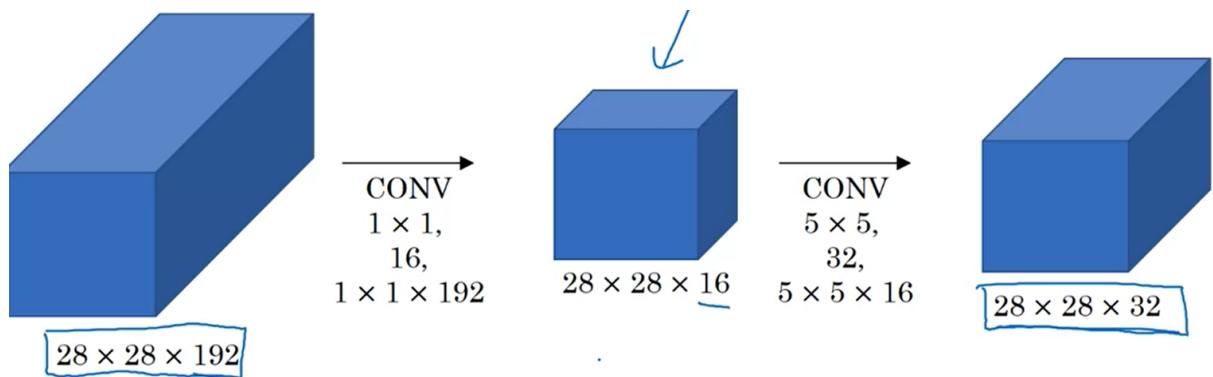
Cuando se diseña una capa convolucional, hay que decidir las dimensiones de los filtros y la cantidad de filtros. En las Inception Network, podemos tener distintos tamaños de filtros y cantidad de filtros por capa, y luego concatenar los resultados en un hipervolumen.



La ventaja es que no tenemos que preocuparnos por el tamaño de los filtros, la red por si sola aprenderá y refinará los pesos más representativos para cada aplicación. La gran desventaja es el costo computacional. Por ejemplo, en este caso, la cantidad de parámetros que la red deberá aprender es de alrededor de 120 Millones.



Para reducir de manera muy drástica la cantidad de parámetros se utilizan las **1x1 convolutions**. En este caso, el primer volumen se convoluciona con 16 filtros de $1 \times 1 \times 192$, generando un volumen de $28 \times 28 \times 16$, esta capa se la conoce como **bottleneck**. Luego, se convoluciona la bottleneck con 32 filtros de $5 \times 5 \times 16$, resultando en un volumen de $28 \times 28 \times 32$.



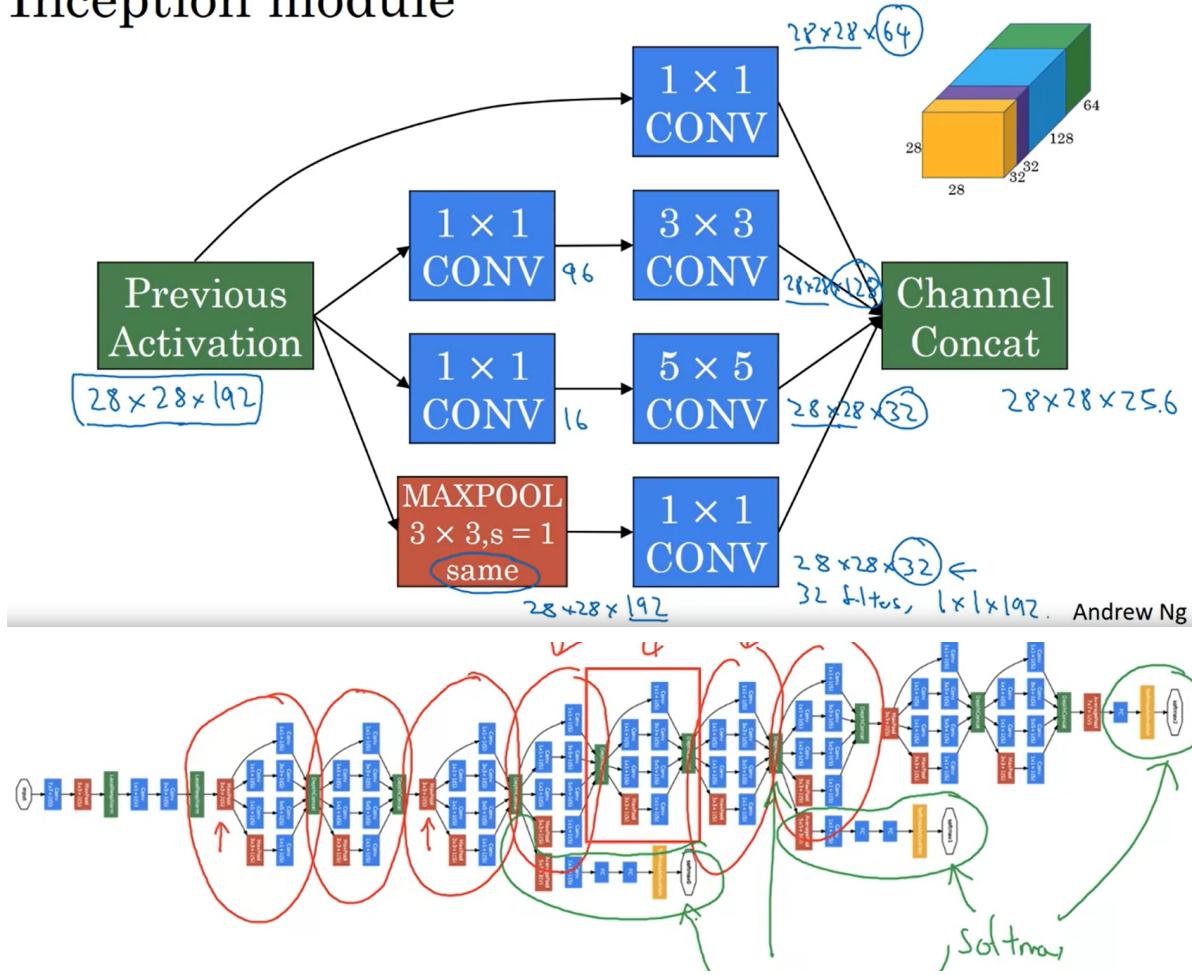
Calculando la cantidad de parámetros:

- De la entrada a la bottleneck hay: $(28 \times 28 \times 16) \times (192 \times 1 \times 1) = 2.4 \text{ M}$
- De la bottleneck a la salida hay: $(28 \times 28 \times 32) \times (5 \times 5 \times 16) = 10 \text{ M}$

Por lo tanto, la cantidad de parámetros total es de 12.4 M, aproximadamente **10 veces menos comparado a hacer la convolución directamente**.

Una Inception Network toma la salida de una capa anterior, y aplica distintos filtrados, donde los resultados los apila todos en un hipervolumen en la salida. Para evitar mucho gasto computacional, se aplican 1x1 convolutions.

Inception module



Hay algunas ramas alternativas en el medio de la red, las cuales permiten saber si en las capas intermedias ya puede predecir bien o todavía le falta. Ayudan a evitar el overfitting.