

# **REDES NEURONALES CONVOLUCIONALES Y SIMULACIONES DE MONTECARLO**



**Universidad Pedagógica y Tecnológica de Colombia**

**LUIS ARIEL MESA M.  
Estudiante**

# AGENDA

1. **Inteligencia Artificial**
2. **Redes Neuronales Artificiales**
3. **Redes Neuronales Convolucionales**
4. **Simulación Monte Carlo**



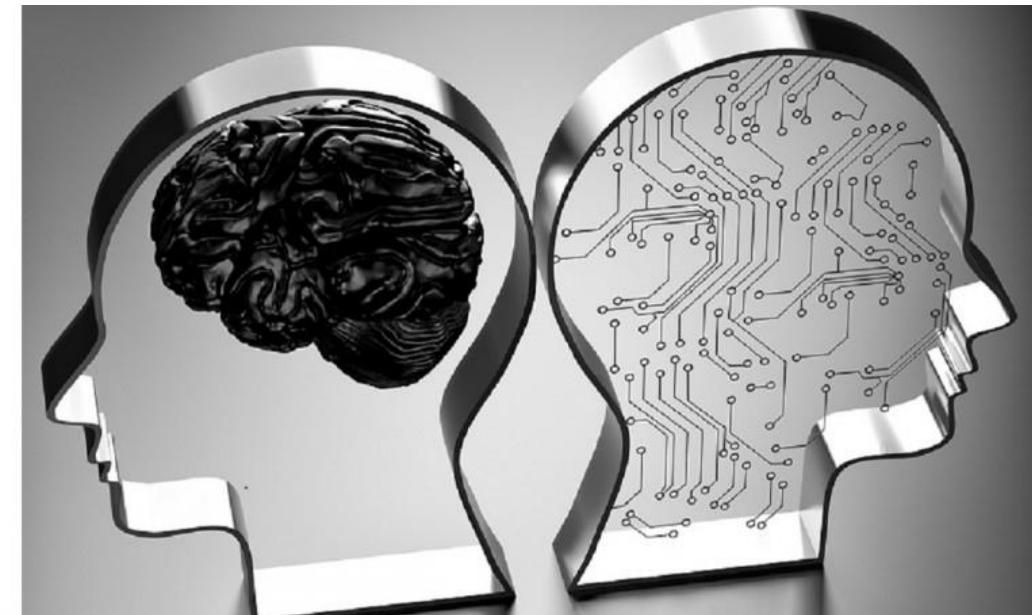
**Uptc<sup>®</sup>**  
Universidad Pedagógica y  
Tecnológica de Colombia



# INTELIGENCIA ARTIFICIAL

La inteligencia artificial (IA), desde su creación en la década de los 60's, ha pretendido emular el funcionamiento y capacidad cognitiva de la mente humana, dotando a las máquinas de cualidades complejas como percibir, razonar, aprender y resolver problemas, necesarias para crear nuevas aplicaciones y ambientes de desarrollo.

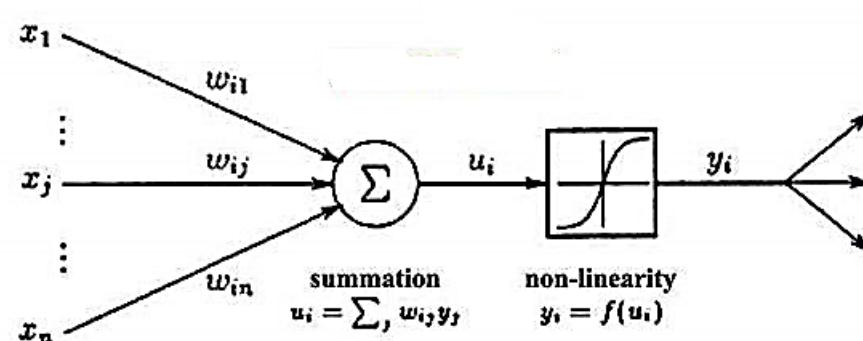
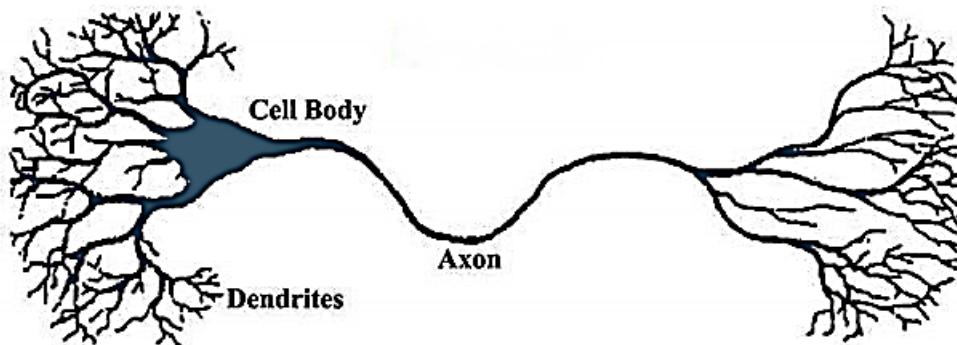
De esta forma, la IA ha venido apalancando tendencias innovadoras e impulsando nuevas transformaciones tecnológicas en diferentes campos al servicio de la humanidad. La IA integra diversas técnicas y conceptos como Machine Learning, Fuzzy logic, Sistemas expertos, Data Mining, Redes Bayesianas, Redes Neuronales Artificiales (ANN), entre otras, que les permiten abordar múltiples campos de aplicación.



# RED NEURONALES ARTIFICIALES - ANNs

Las ANNs, inspiradas funcionalmente en las neuronas biológicas de algunos seres vivos, son una de las técnicas más importantes de este conjunto, que han permitido incorporar el concepto de aprendizaje automatizado al funcionamiento de las máquinas.

Allí, un sistema de enlaces de neuronas, interactúan entre sí para producir un estímulo de salida, donde cada conexión contiene un peso numérico que se ajusta continuamente con la información de entrada y les permiten obtener un aprendizaje artificial.



Son un modelo computacional que consiste en un conjunto de unidades, llamadas neuronas artificiales, conectadas entre sí para transmitir señales.

Las neuronas del cerebro están compuestas de **dendritas, el soma y el axón**:

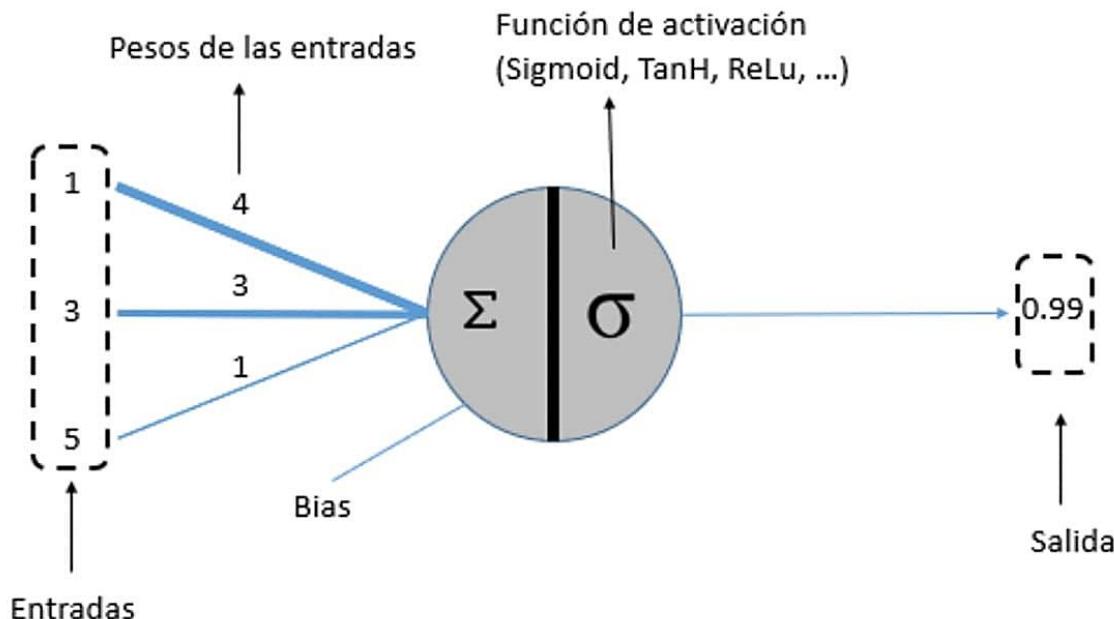
Las dendritas se encargan de captar los impulsos nerviosos que emiten otras neuronas. Estos impulsos, se procesan en el soma y se transmiten a través del axón que emite un impulso nervioso hacia las neuronas contiguas. Una zona de conexión entre una neurona y otra, conocida como **sinapsis**



# FUNCIONAMIENTO DE UNA ANN

En el caso de las neuronas artificiales, la suma de las entradas multiplicadas por sus pesos asociados determina el “impulso nervioso” que recibe la neurona. Este valor, se procesa en el interior de la célula mediante una función de activación que devuelve un valor que se envía como salida de la neurona.

## Detalle de una neurona artificial (perceptrón)

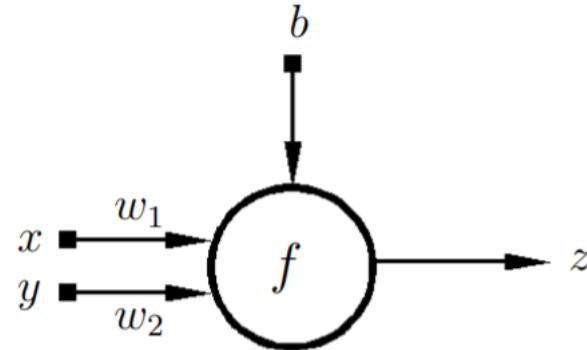


La información de entrada atraviesa la red neuronal (donde se somete a diversas operaciones) produciendo unos valores de salida.



# FUNCIONAMIENTO DE UNA ANN

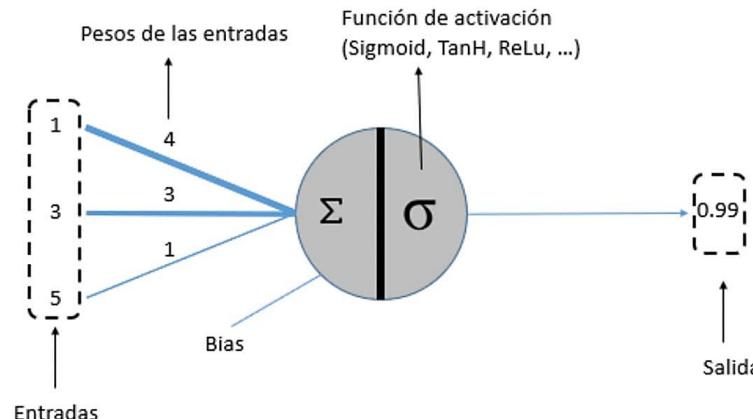
## Modelo matemático



Las entradas  $x$  e  $y$  son el estímulo que la neurona artificial recibe del entorno que la rodea, y la salida  $z$  es la respuesta a tal estímulo. La neurona se adapta al medio circundante y aprende de él modificando el valor de sus pesos sinápticos  $w_1$  y  $w_2$  y su término aditivo  $b$  (conocido también como Bias)

Modelo de McCulloch-Pitts para una neurona artificial

$$z = f(w_1 x + w_2 y + b)$$



F recibe el nombre de **función de activación**

# FUNCIONAMIENTO DE UNA ANN

## Funciones de activación mas comunes

	Función	Rango	Gráfica
Identidad	$y = x$	$[-\infty, +\infty]$	
Escalón	$y = sign(x)$ $y = H(x)$	$\{-1, +1\}$ $\{0, +1\}$	
Sigmoidea	$y = \frac{1}{1+e^{-x}}$ $y = tgh(x)$	$[0, +1]$ $[-1, +1]$	
Gaussianas	$y = Ae^{-Bx^2}$	$[0, +1]$	
Sinusoidal	$y = A \sen(\omega x + \varphi)$	$[-1, +1]$	

La función de activación devuelve una salida que será generada por la neurona dada una entrada o conjunto de entradas.

Cada una de las capas que conforman la red neuronal tienen una función de activación que permitirá reconstruir o predecir.



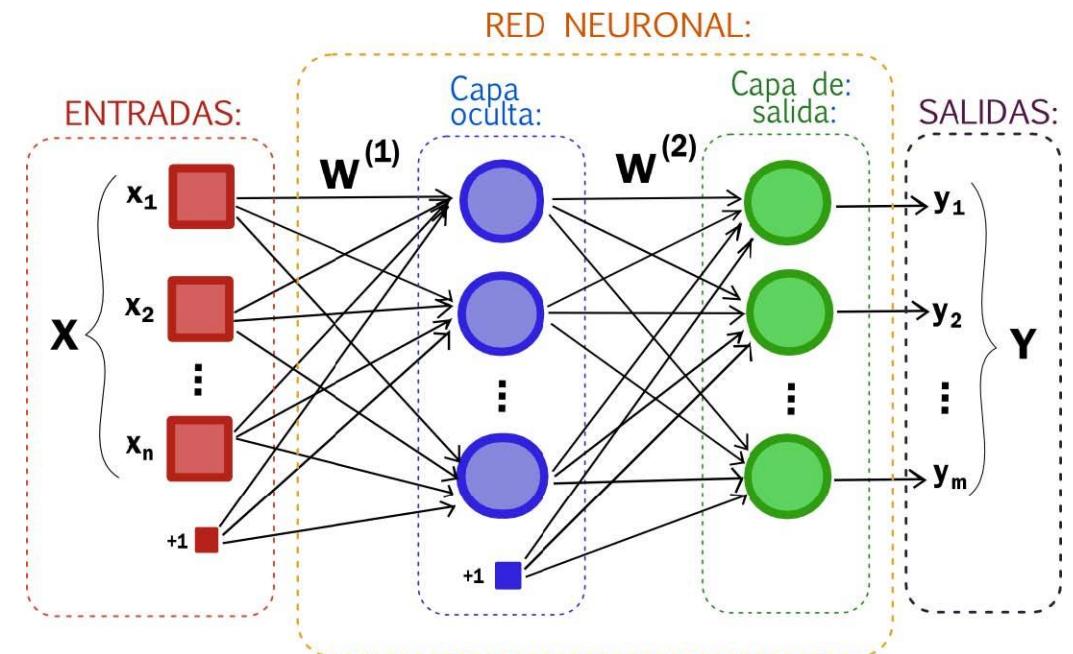
# FUNCIONAMIENTO DE UNA ANN

Las neuronas de la red agrupadas en capas que forman la red neuronal artificial. La información de entrada atraviesa la red neuronal (donde se somete a diversas operaciones) produciendo unos valores de salida.

**Capa Neuronal:** es un conjunto de neuronas cuyas entradas provienen de una capa anterior (o de los datos de entrada en el caso de la primera capa) y cuyas salidas son la entrada de una capa posterior.»

## TIPOS DE CAPAS

- **Capa de entrada:** neuronas que reciben como entrada los datos reales que alimentan a la red neuronal.
- **Capa de salida :** es la última capa o el resultado visible de la red.
- **Capas ocultas:** capas que se sitúan entre la capa de entrada y la capa de salida. Se denominan así porque se desconocen tanto los valores de entrada como los de salida.





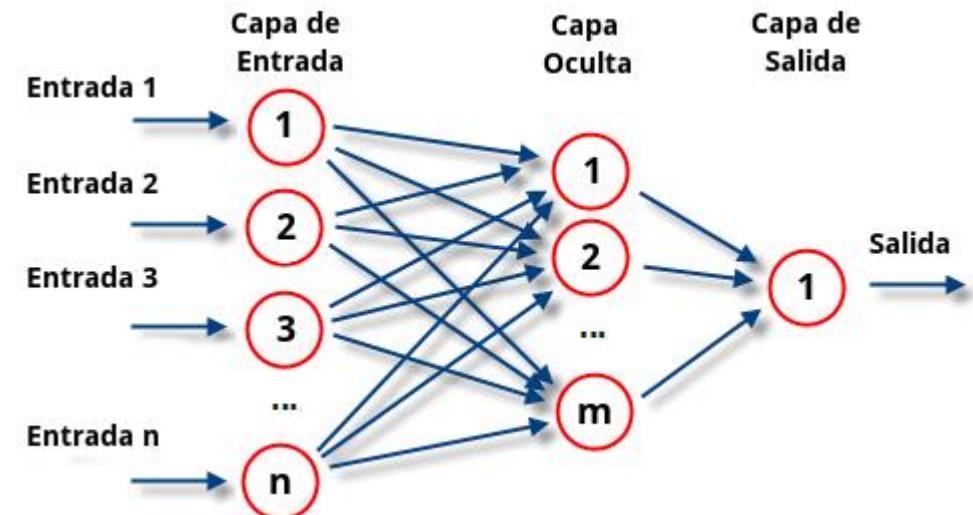
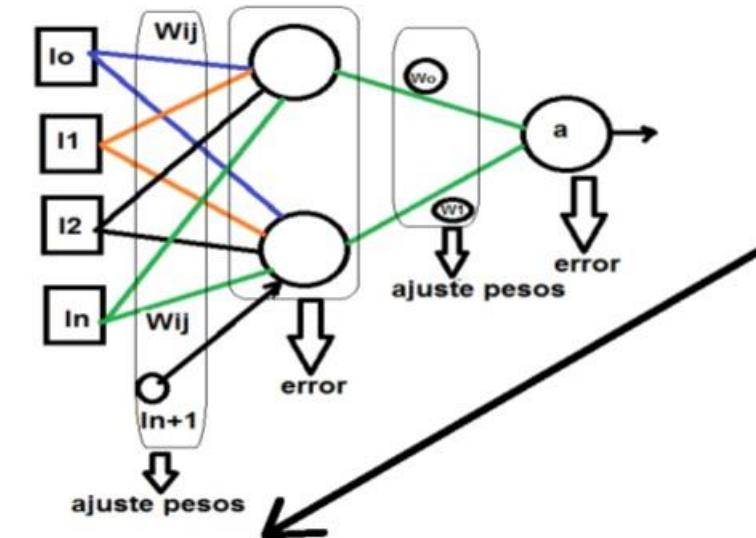
# FASES DE OPERACIÓN DE UNA ANN

## • Entrenamiento de una Red neuronal

Entrenar una red neuronal consiste en ajustar cada uno de los pesos de las entradas de todas las neuronas que forman parte de la red neuronal, para que las respuestas de la capa de salida se ajusten lo más posible a los datos que se conocen

## • Inferencia de una Red neuronal

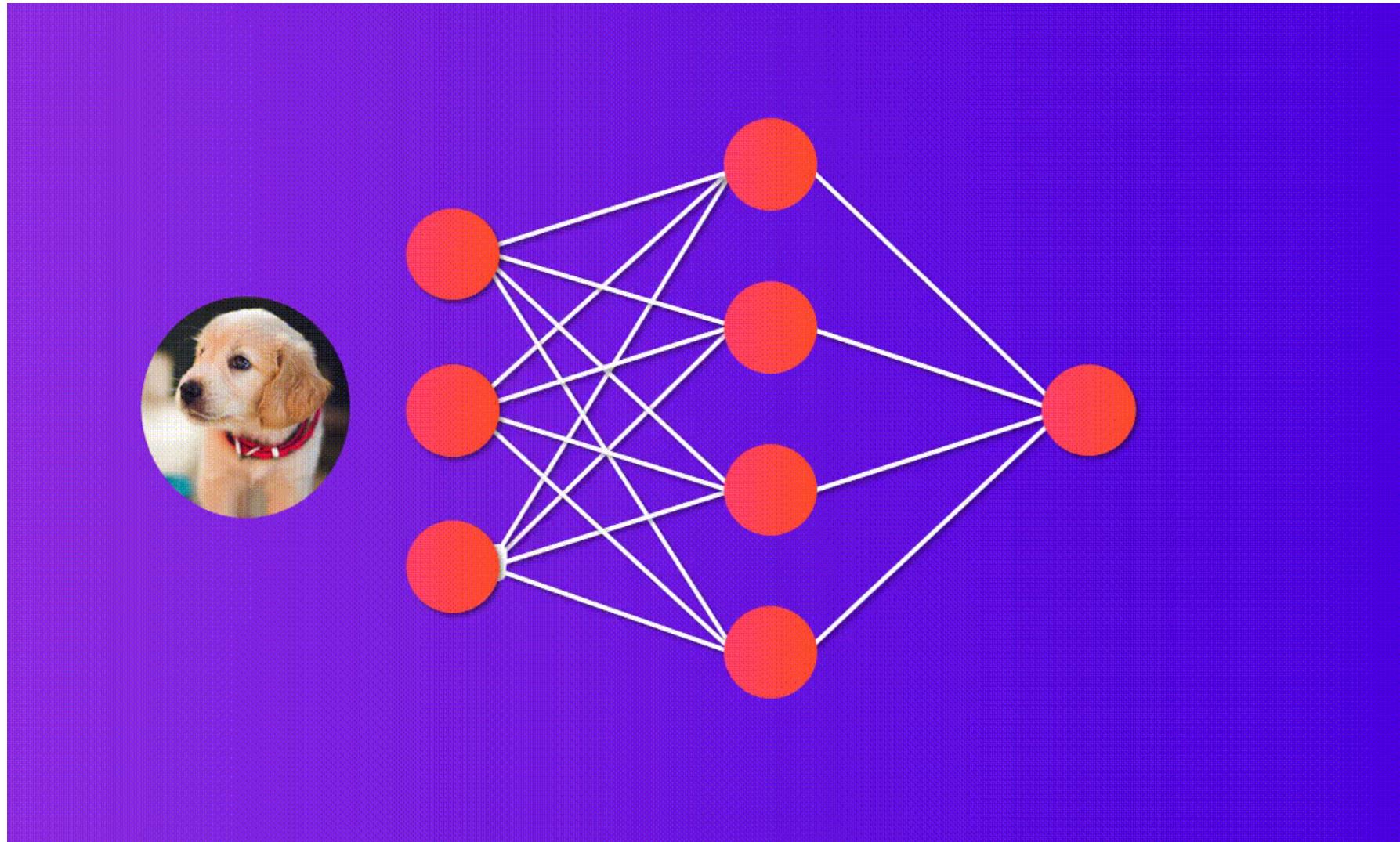
Inferencia es el término que describe el acto de usar una red neuronal para proporcionar insights después de que ha sido entrenada.





# FASES DE OPERACIÓN DE UNA ANN

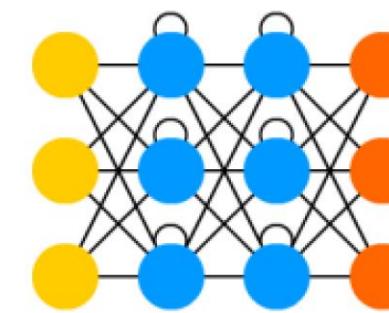
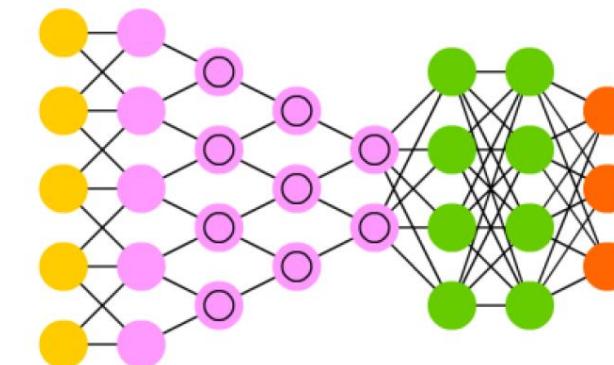
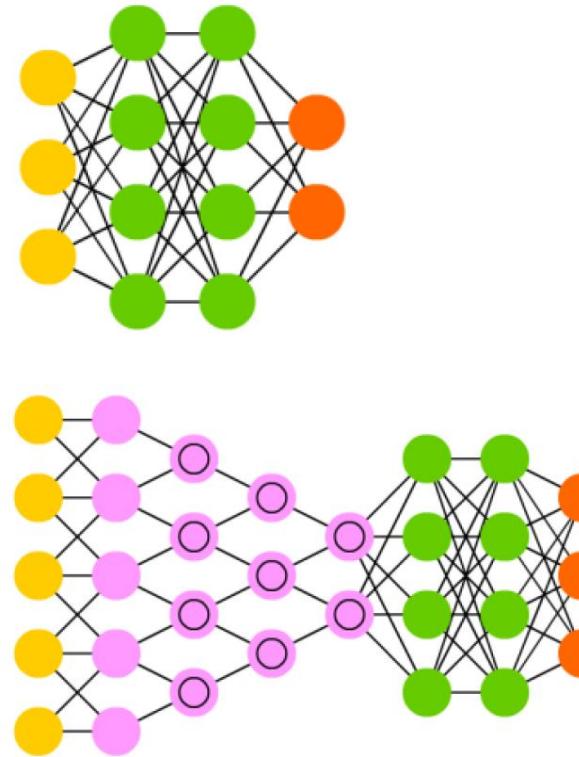
Red Neuronal- Inferencia



# TIPOS DE ANNs

Hoy en día son comunes tres tipos de NN:

1. **Perceptrones multicapa (MLP):** cada nueva capa es un conjunto de funciones no lineales de suma ponderada de todas las salidas (completamente conectadas) de una anterior, que reutiliza las ponderaciones.
2. **Redes Neuronales Convolucionales (CNN):** cada capa resultante es un conjunto de funciones no lineales de sumas ponderadas de subconjuntos de salidas espacialmente cercanos de la capa anterior, que también reutiliza las ponderaciones.
3. **Redes neuronales Recurrentes (RNN):** cada capa subsiguiente es una colección de funciones no lineales de sumas ponderadas de salidas y el estado anterior. El RNN más popular es Long Short-Term Memory (LSTM). El arte del LSTM consiste en decidir qué olvidar y qué pasar como estado a la siguiente capa. Los pesos se reutilizan a lo largo de los pasos de tiempo.



- Input Cell
- Hidden Cell
- Convolution
- Recurrent Cell
- Scanning Filter
- Output Cell

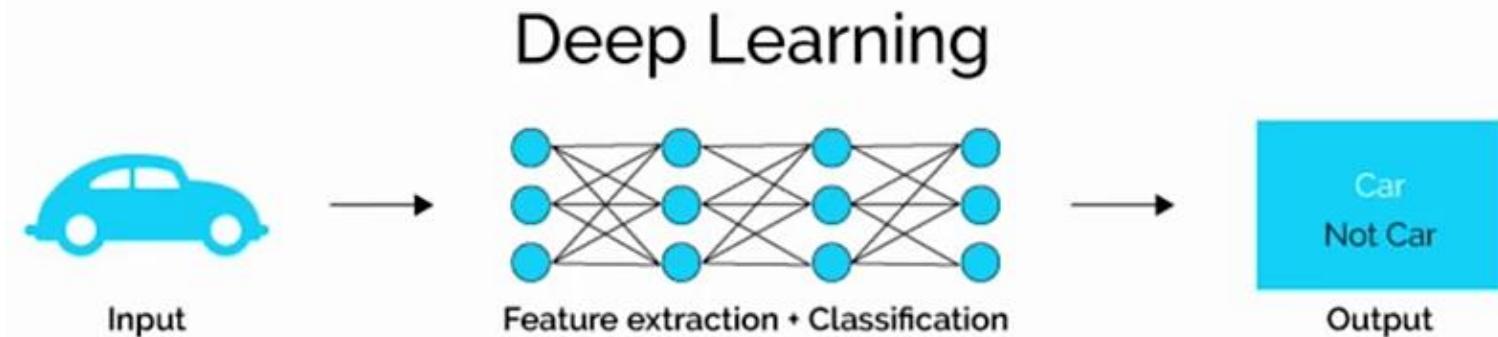
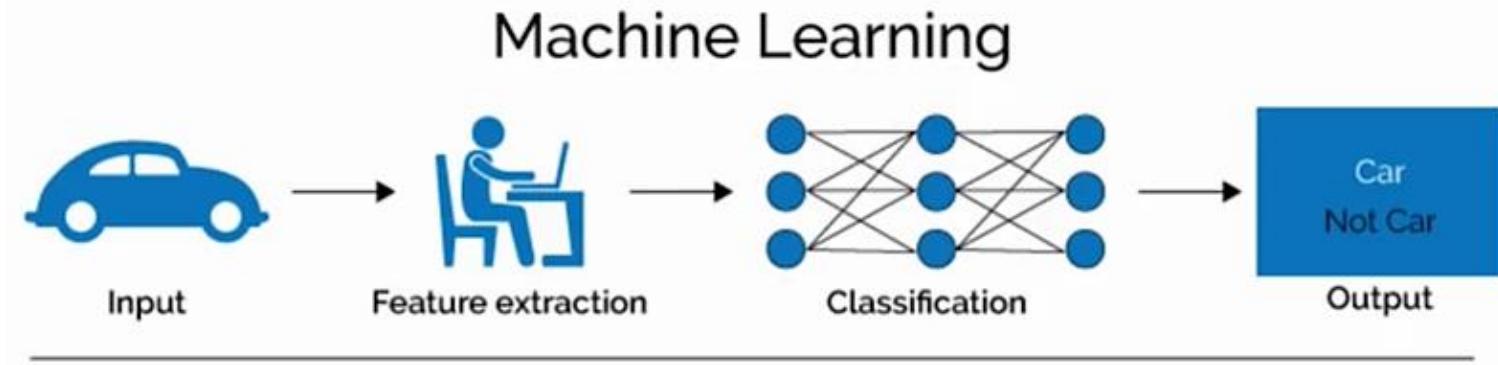


# DEEP LEARNING

Los modelos tradicionales de machine learning que utilizan características diseñadas a mano dominaron por mucho tiempo el campo de la inteligencia computacional. Sin embargo para obtener un desempeño adecuado se requiere un conocimiento profundo del problema.

**Deep Learning:** es un tipo de machine learning que entrena a una computadora para que realice tareas como las que hacen los seres humanos (reconocimiento del habla, la identificación de imágenes, hacer predicciones, etc.).

En lugar de organizar datos para que se ejecuten a través de ecuaciones predefinidas, el deep learning configura parámetros básicos acerca de los datos y entrena a la computadora para que aprenda por cuenta propia reconociendo patrones mediante el uso de muchas capas de procesamiento.





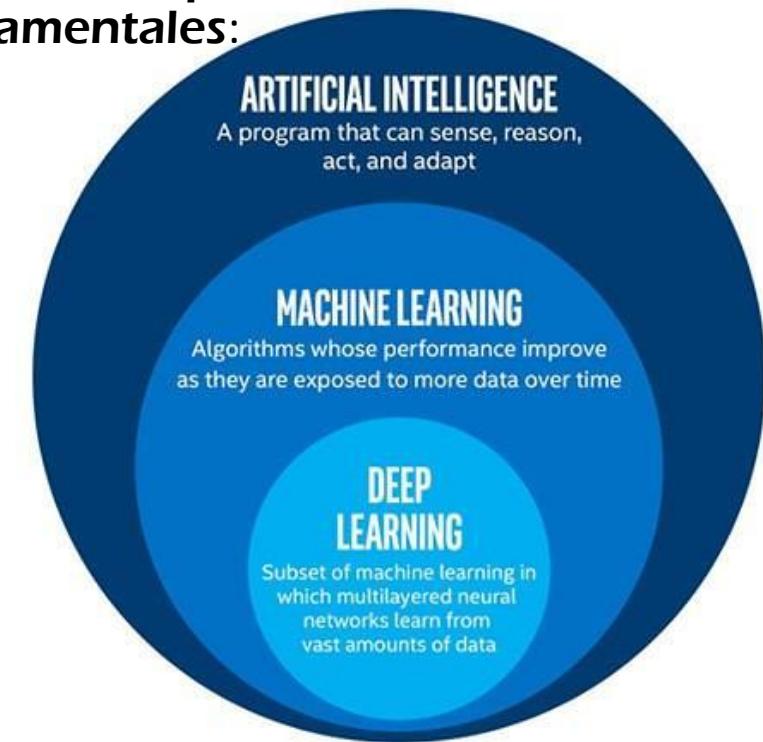
# DEEP LEARNING

Muchas de las ideas centrales como CNN y backpropagation ya eran bien comprendidas desde finales de los 80's. Sin embargo, se enfrentaron a tres problemas fundamentales:

- **Hardware**
- **Datos**
- **Algoritmos**

## Ventajas

- **Simplicidad**
- **Escalabilidad**
- **Versatilidad**



Las DNNs han cobrado mayor importancia en aplicaciones de altas especificaciones como el procesamiento del lenguaje natural (NLP), la comprensión del lenguaje natural (NLU), el reconocimiento de imágenes y el modelado predictivo(REF), utilizando particularmente para el caso, algunos tipos especializados de arquitecturas profundas como las Redes Neuronales Convolucionales (CNNs).

En este aspecto, las Deep Neural Networks (DNN) presentan la importante capacidad de abstraer una mayor cantidad de información debido a su arquitectura extensa y compleja, sin embargo demandan a la vez, mayores recursos para su implementación y entrenamiento.

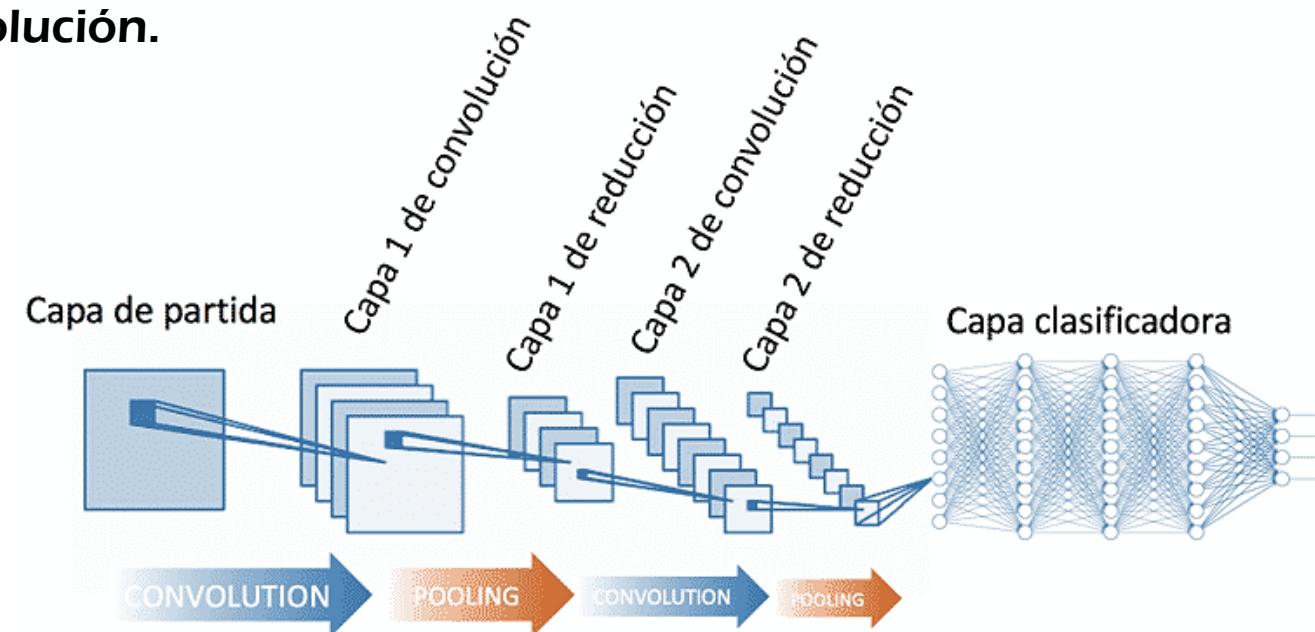


# REDES NEURONALES CONVOLUCIONALES-CNNs

La principal diferencia de la red CNN con el perceptrón multicapa viene en que cada neurona no se une con todas y cada una de las capas siguientes sino que solo con un subgrupo de ellas (se especializa), con esto se consigue reducir el número de neuronas necesarias y la complejidad computacional.

## CARACTERÍSTICAS

- Especializada en el procesamiento de datos que tengan topología de malla.
- Las CNNs han tenido gran éxito en aplicaciones de visión por computador.
- Utiliza la operación matemática de convolución.





# REDES NEURONALES CONVOLUCIONALES-CNNs

**Todo comienza con un Pixel...**



Una imagen...

	0.6	0.6			
0.6			0.6		
0.6	0.6	0.6	0.6		
0.6			0.6		

...es una matriz de píxeles.

El valor de los píxeles va de 0 a 255 pero se normaliza para la red neuronal de 0 a 1

## Píxeles y neuronas

Para comenzar, la red toma como entrada los píxeles de una imagen. Si tenemos una imagen con apenas  $28 \times 28$  píxeles de alto y ancho, esto equivale a utilizar 784 neuronas.

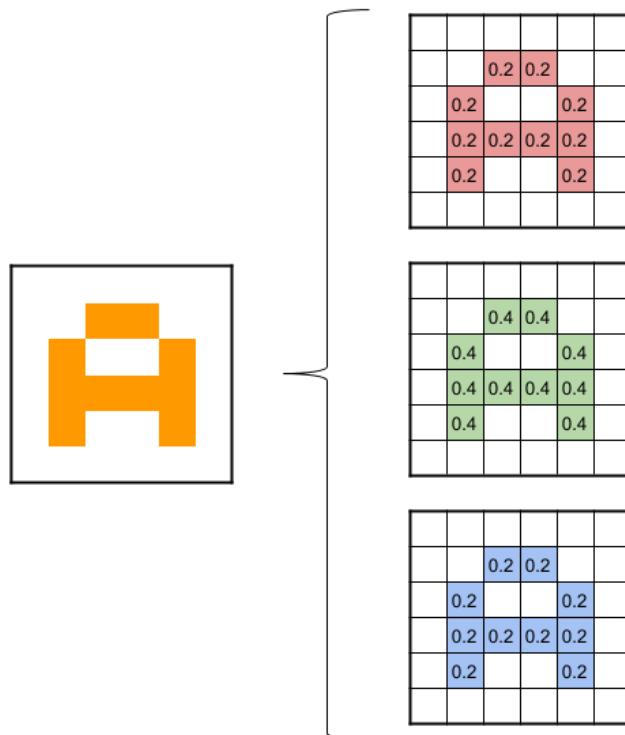
Eso es si sólo tenemos 1 color (escala de grises). Si tuviéramos una imagen a color, necesitaríamos 3 canales RGB (red, green, blue) y entonces usaríamos  $28 \times 28 \times 3 = 2352$  neuronas. Estas neuronas constituyen nuestra capa de entrada.



# REDES NEURONALES CONVOLUCIONALES-CNNs

## Pre-procesamiento

Antes de alimentar la red, como entrada conviene convertir los valores entre 0 y 1 y por tanto tendrá que dividirse todos entre 255 . Este 255 es por que los colores de los píxeles tienen valores que van del 0 al 255 .



Si la imagen es a color, estará compuesta de tres canales: rojo, verde, azul.

## Convoluciones

Ahora comienza el «procesado distintivo» de las CNNS... las llamadas «convoluciones»: Estas consisten en tomar «grupos de píxeles cercanos» de la imagen de entrada e ir operando matemáticamente (producto escalar) contra una pequeña matriz que se llama kernel. Ese kernel, suponiendo que tiene un tamaño de de  $3 \times 3$  pixels, con ese tamaño logra «visualizar» todas las neuronas de entrada (de izquierda-derecha, de arriba-abajo), generando una nueva matriz de salida, que en definitiva será nuestra nueva capa de neuronas ocultas.



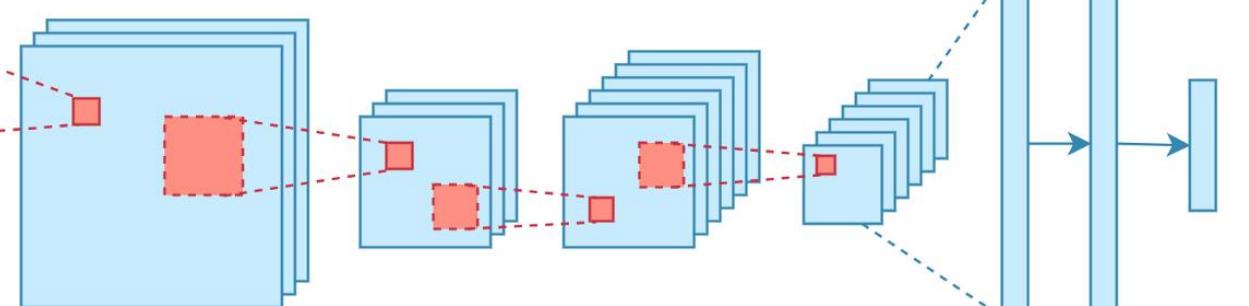
# REDES NEURONALES CONVOLUCIONALES-CNNs

## ARQUITECTURA DE LA RED.

Consta de capas convolucionales y de reducción alternadas, y al finalmente tiene capas de conexión total como una red perceptrón multicapa.

**Convolución:** En la convolución se realizan operaciones de productos y sumas entre la capa de partida y los n filtros (o kernels) que genera un mapa de características. Las características extraídas corresponden a cada posible ubicación del filtro en la imagen original.

La ventaja es que el mismo filtro (= neurona) sirve para extraer la misma característica en cualquier parte de la entrada, con esto que consigue reducir el número de conexiones y el número de parámetros a entrenar en comparación con una red multicapa de conexión total.



Input

Conv

Pool

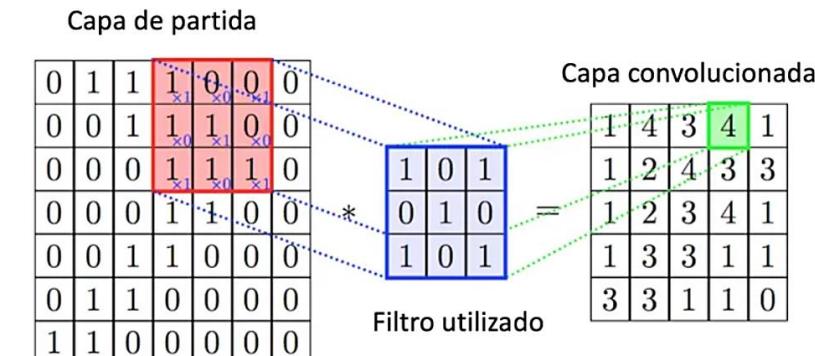
Conv

Pool

FC

FC

Softmax



El cálculo principal ocurre en esta capa y es el bloque esencial de construcción en una CNN. Aquí, una imagen de entrada se procesa con un kernel o filtro, donde se verifica si la característica de la imagen está presente. El kernel barre toda la imagen después de varias iteraciones. La salida se calcula entre la entrada y el filtro después de cada iteración. El resultado final de esta capa se conoce como característica convolucionada. La imagen se convierte en valores numéricos.

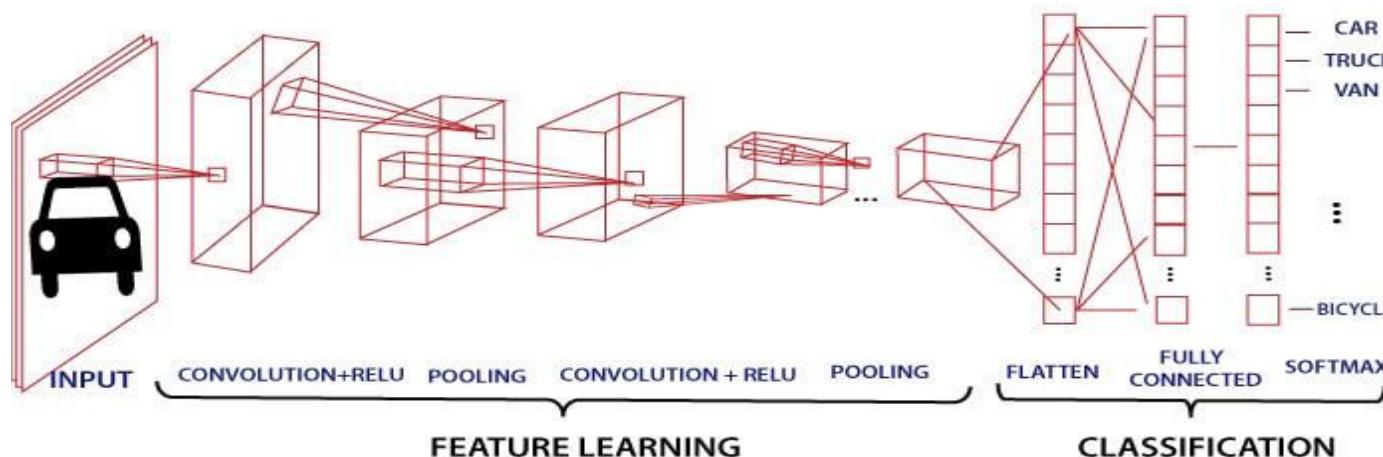
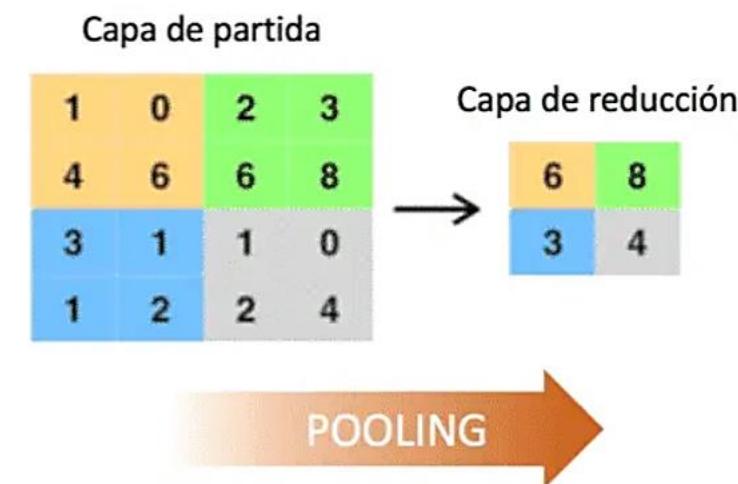


# REDES NEURONALES CONVOLUCIONALES-CNNs

## ARQUITECTURA DE LA RED.

**Reducción – pooling:** En la reducción se disminuye la cantidad de parámetros al quedarse con las características más comunes. La forma de reducir parámetros se realiza mediante la extracción de estadísticas como el promedio o el máximo de una región fija del mapa de características, al reducir características el método pierde precisión aunque mejora su compatibilidad.

**Clasificador – red perceptrón multicapa:** El final de las capas convolucionales y de reducción, se suele utilizar capas completamente conectadas en la que cada pixel se considera como una neurona separada al igual que en un perceptrón multicapa. La última capa de esta red es una capa clasificadora que tendrá tantas neuronas como el número de clases a predecir





# REDES NEURONALES CONVOLUCIONALES-CNNs

**CONVOLUCIÓN:** La matriz resultante se obtiene por medio de la sumatoria del producto elemento por elemento entre el kernel y la matriz de entrada.

3	0	1	2	7	4
1	5	8	9	3	1
2	7	2	5	1	3
0	1	3	1	7	8
4	2	1	6	2	8
2	4	5	2	3	9

$$\begin{array}{c}
 \begin{array}{|c|c|c|} \hline 3 & 0 & 1 \\ \hline 1 & 5 & 8 \\ \hline 2 & 7 & 2 \\ \hline 0 & 1 & 3 \\ \hline 4 & 2 & 1 \\ \hline 2 & 4 & 5 \\ \hline \end{array} * \begin{array}{|c|c|c|} \hline 1 & 0 & -1 \\ \hline 1 & 0 & -1 \\ \hline 1 & 0 & -1 \\ \hline \end{array} = \begin{array}{|c|c|c|c|} \hline -5 & -4 & 0 & 8 \\ \hline -10 & -2 & 2 & 3 \\ \hline 0 & -2 & -4 & -7 \\ \hline -3 & -2 & -3 & -16 \\ \hline \end{array} \\
 \text{Filtro/Kernel} \quad 3 \times 3
 \end{array}$$

$(3 \times 1) + (1 \times 1) + (2 \times 1) + (0 \times 0) + (5 \times 0) + (7 \times 0) + (1 \times -1) + (8 \times -1) + (2 \times -1)$

3	0	1	2	7	4
1	5	8	9	3	1
2	7	2	5	1	3
0	1	3	1	7	8
4	2	1	6	2	8
2	4	5	2	3	9

$$\begin{array}{c}
 \begin{array}{|c|c|c|} \hline 3 & 0 & 1 \\ \hline 1 & 5 & 8 \\ \hline 2 & 7 & 2 \\ \hline 0 & 1 & 3 \\ \hline 4 & 2 & 1 \\ \hline 2 & 4 & 5 \\ \hline \end{array} * \begin{array}{|c|c|c|} \hline 1 & 0 & -1 \\ \hline 1 & 0 & -1 \\ \hline 1 & 0 & -1 \\ \hline \end{array} = \begin{array}{|c|c|c|c|} \hline -5 & -4 & 0 & 8 \\ \hline -10 & -2 & 2 & 3 \\ \hline 0 & -2 & -4 & -7 \\ \hline -3 & -2 & -3 & -16 \\ \hline \end{array} \\
 \text{Filtro/Kernel} \quad 3 \times 3
 \end{array}$$

$(0 \times 1) + (5 \times 1) + (7 \times 1) + (1 \times 0) + (8 \times 0) + (2 \times 0) + (2 \times -1) +$

3	0	1	2	7	4
1	5	8	9	3	1
2	7	2	5	1	3
0	1	3	1	7	8
4	2	1	6	2	8
2	4	5	2	3	9

$$\begin{array}{c}
 \begin{array}{|c|c|c|} \hline 3 & 0 & 1 \\ \hline 1 & 5 & 8 \\ \hline 2 & 7 & 2 \\ \hline 0 & 1 & 3 \\ \hline 4 & 2 & 1 \\ \hline 2 & 4 & 5 \\ \hline \end{array} * \begin{array}{|c|c|c|} \hline 1 & 0 & -1 \\ \hline 1 & 0 & -1 \\ \hline 1 & 0 & -1 \\ \hline \end{array} = \begin{array}{|c|c|c|c|} \hline -5 & -4 & 0 & 8 \\ \hline -10 & -2 & 2 & 3 \\ \hline 0 & -2 & -4 & -7 \\ \hline -3 & -2 & -3 & -16 \\ \hline \end{array} \\
 \text{Filtro/Kernel} \quad 3 \times 3
 \end{array}$$

$(1 \times 1) + (8 \times 1) + (2 \times 1) + (2 \times 0) + (9 \times 0) + (5 \times 0) + (7 \times -1) + (3 \times -1) + (1 \times -1)$

3	0	1	2	7	4
1	5	8	9	3	1
2	7	2	5	1	3
0	1	3	1	7	8
4	2	1	6	2	8
2	4	5	2	3	9

$$\begin{array}{c}
 \begin{array}{|c|c|c|} \hline 3 & 0 & 1 \\ \hline 1 & 5 & 8 \\ \hline 2 & 7 & 2 \\ \hline 0 & 1 & 3 \\ \hline 4 & 2 & 1 \\ \hline 2 & 4 & 5 \\ \hline \end{array} * \begin{array}{|c|c|c|} \hline 1 & 0 & -1 \\ \hline 1 & 0 & -1 \\ \hline 1 & 0 & -1 \\ \hline \end{array} = \begin{array}{|c|c|c|c|} \hline -5 & -4 & 0 & 8 \\ \hline -10 & -2 & 2 & 3 \\ \hline 0 & -2 & -4 & -7 \\ \hline -3 & -2 & -3 & -16 \\ \hline \end{array} \\
 \text{Filtro/Kernel} \quad 3 \times 3
 \end{array}$$

$(2 \times 1) + (9 \times 1) + (5 \times 1) + (7 \times 0) + (3 \times 0) + (1 \times 0) + (4 \times -1) + (1 \times -1) + (3 \times -1)$

3	0	1	2	7	4
1	5	8	9	3	1
2	7	2	5	1	3
0	1	3	1	7	8
4	2	1	6	2	8
2	4	5	2	3	9

$$\begin{array}{c}
 \begin{array}{|c|c|c|} \hline 3 & 0 & 1 \\ \hline 1 & 5 & 8 \\ \hline 2 & 7 & 2 \\ \hline 0 & 1 & 3 \\ \hline 4 & 2 & 1 \\ \hline 2 & 4 & 5 \\ \hline \end{array} * \begin{array}{|c|c|c|} \hline 1 & 0 & -1 \\ \hline 1 & 0 & -1 \\ \hline 1 & 0 & -1 \\ \hline \end{array} = \begin{array}{|c|c|c|c|} \hline -5 & -4 & 0 & 8 \\ \hline -10 & -2 & 2 & 3 \\ \hline 0 & -2 & -4 & -7 \\ \hline -3 & -2 & -3 & -16 \\ \hline \end{array} \\
 \text{Filtro/Kernel} \quad 3 \times 3
 \end{array}$$

$(1 \times 1) + (2 \times 1) + (0 \times 1) + (5 \times 0) + (7 \times 0) + (1 \times 0) + (8 \times -1) + (2 \times -1) + (3 \times -1)$



# REDES NEURONALES CONVOLUCIONALES-CNNs

**CONVOLUCIÓN:** La matriz resultante se obtiene por medio de la sumatoria del producto elemento por elemento entre el kernel y la matriz de entrada.

1 <small>x1</small>	1 <small>x0</small>	1 <small>x1</small>	0	0
0 <small>x0</small>	1 <small>x1</small>	1 <small>x0</small>	1	0
0 <small>x1</small>	0 <small>x0</small>	1 <small>x1</small>	1	1
0	0	1	1	0
0	1	1	0	0

Image

4		

Convolved Feature

3	0	1	2	7	4
1	5	8	9	3	1
2	7	2	5	1	3
0	1	3	1	7	8
4	2	1	6	2	8
2	4	5	2	3	9

Entrada nxn

\*

1	0	-1
1	0	-1
1	0	-1

Filtro/Kernel  
fxf

=

-5	-4	0	8
-10	-2	2	3
0	-2	-4	-7
-3	-2	-3	-16

Salida  $n-f+1 \times n-f+1$

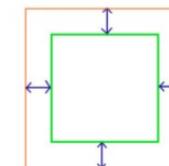
Se debe tener en cuenta dos elementos esenciales:

Padding y Stride

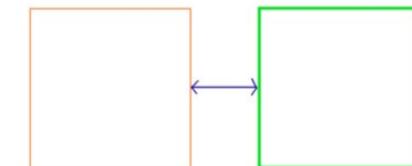
**PADDING (p): (relleno)**

- Cada vez que se aplica una convolución la imagen se reduce.
- Puede realizarse la convolución con los elementos válidos o conservando la dimensión original.

Padding  
 $p = (f-1)/2$



Padding



Margin



# REDES NEURONALES CONVOLUCIONALES-CNNs

## CONVOLUCIÓN:

**STRIDE (s)**: (paso) Distancia de desplazamiento del kernel

STRIDE: 1

3	0	1	2	7	4
1	5	8	9	3	1
2	7	2	5	1	3
0	1	3	1	7	8
4	2	1	6	2	8
2	4	5	2	3	9

Entrada nxn

$$\begin{array}{c} * \\ \text{Filtro/Kernel} \\ 3 \times 3 \end{array} = \begin{array}{c} \text{Salida } n-f+1 \times n-f+1 \end{array}$$

STRIDE: 3

3	0	1	2	7	4
1	5	8	9	3	1
2	7	2	5	1	3
0	1	3	1	7	8
4	2	1	6	2	8
2	4	5	2	3	9

Entrada nxn

$$\begin{array}{c} * \\ \text{Filtro/Kernel} \\ f \times f \end{array} = \begin{array}{c} \text{Salida } \frac{n+2p-f}{s} + 1 \times \frac{n+2p-f}{s} + 1 \end{array}$$



# REDES NEURONALES CONVOLUCIONALES-CNNs

## FILTROS O KERNELS:

Los filtros estáticos se enfocan en buscar características particulares de la imagen como líneas verticales, horizontales, diagonales, etc.

1	0	-1
1	0	-1
1	0	-1

Vertical

1	1	1
0	0	0
-1	-1	-1

Horizontal

1	0	-1
2	0	-2
1	0	-1

Sobel

3	0	-3
10	0	-10
3	0	-3

Scharr

Los parámetros del filtro pueden ser ajustados por medio del proceso de optimización

$w_1$	$w_2$	$w_2$
$w_4$	$w_5$	$w_6$
$w_7$	$w_8$	$w_9$



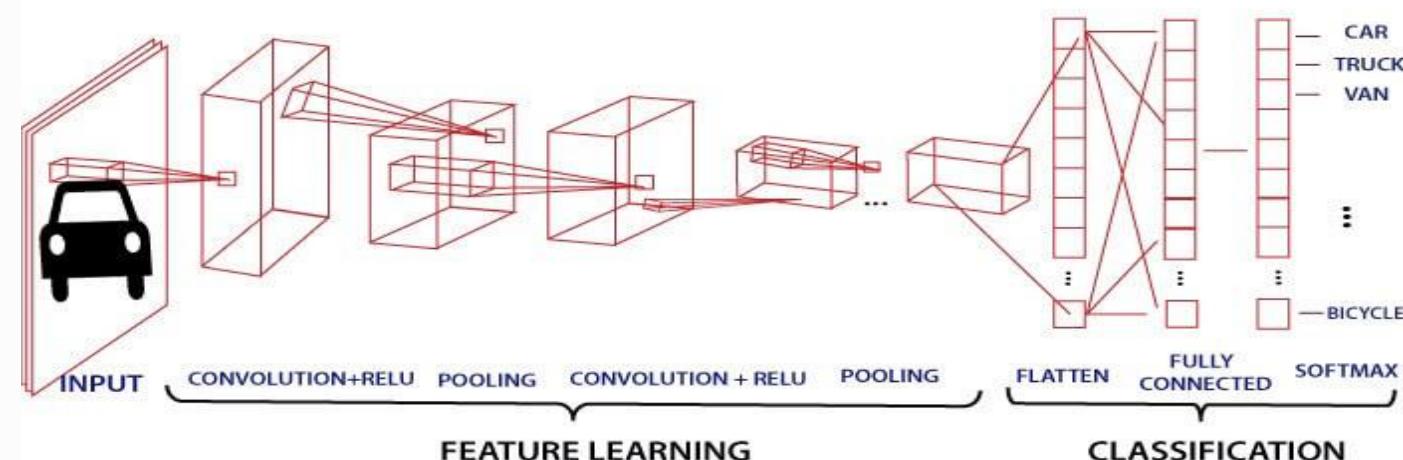
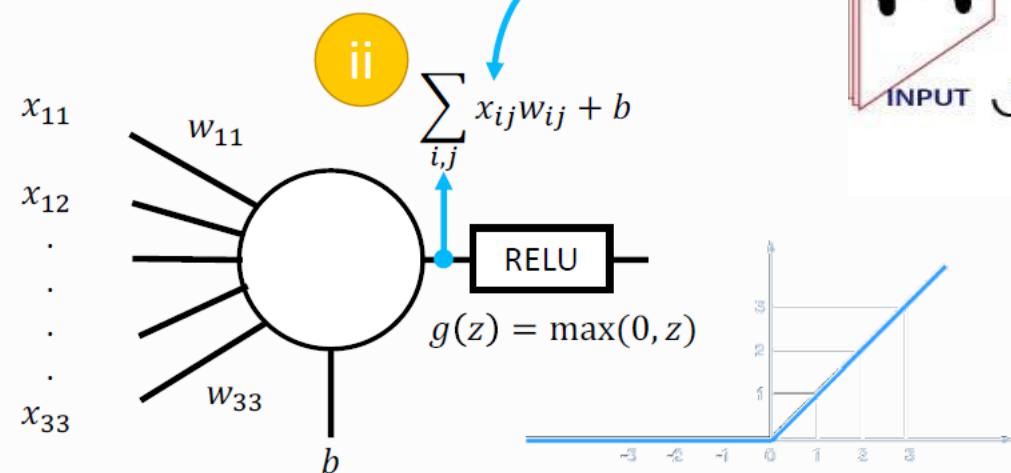
# REDES NEURONALES CONVOLUCIONALES-CNNs

## Entonces para una red CNN

Se considera entonces la convolución como una operación en la cual se tiene una serie de pesos que se multiplican por las entradas y finalmente se le aplica un proceso de no-linealidad.

Bajo este concepto la operación resultante es una suma ponderada de pesos.

Trozo de la Imagen	Filtro/Kernel	Resultado Convolución
$x_{11} \quad x_{12} \quad x_{13}$	$w_{11} \quad w_{12} \quad w_{13}$	
$x_{21} \quad x_{22} \quad x_{23}$	$w_{21} \quad w_{22} \quad w_{23}$	
$x_{31} \quad x_{32} \quad x_{33}$	$w_{31} \quad w_{32} \quad w_{33}$	





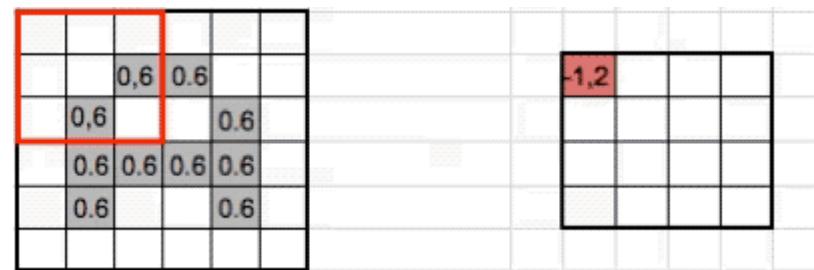
# REDES NEURONALES CONVOLUCIONALES-CNNs

## Filtro: conjunto de kernels

En la práctica no se aplica 1 sólo kernel, si no que se aplican muchos kernel (al conjunto de Kernels se les llama filtros).

### Ejemplo:

Si en una primer convolución se tienen 32 filtros, se obtendrían 32 matrices de salida (este conjunto se conoce como «feature mapping»), cada una de  $28 \times 28 \times 1$  dan un total del 25.088 neuronas para nuestra PRIMER CAPA OCULTA de neuronas, y que solo analiza una imagen cuadrada de apenas 28 píxeles.



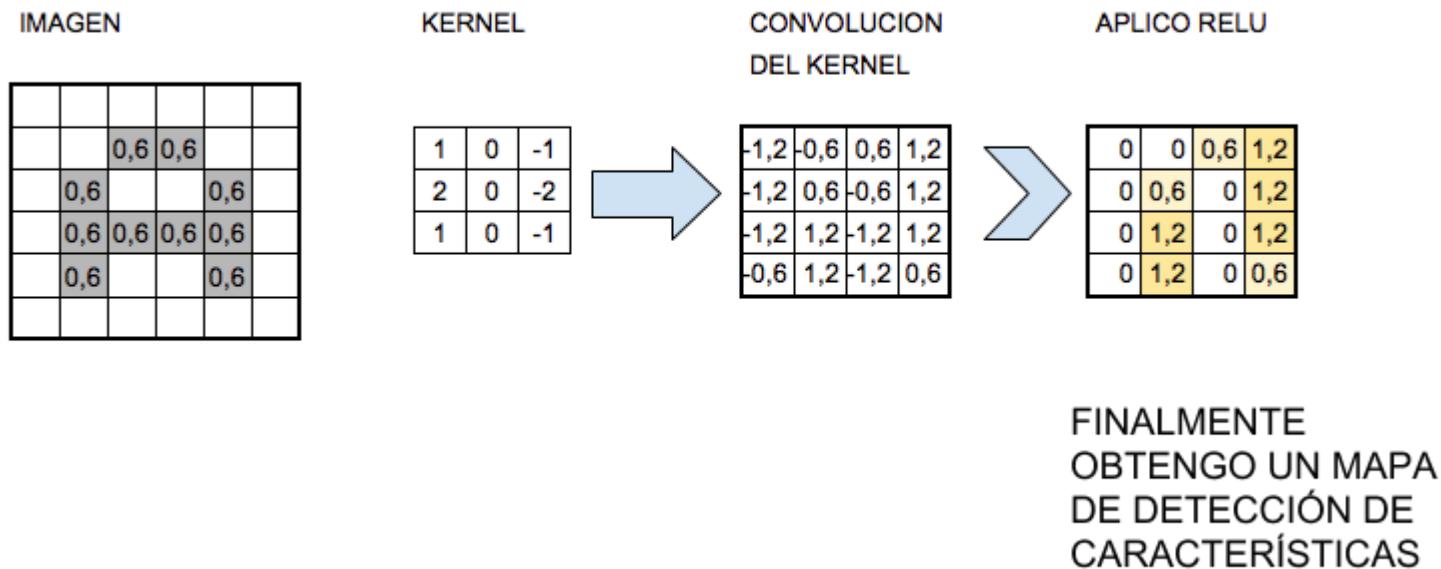
Aquí vemos al kernel realizando el producto matricial con la imagen de entrada y desplazando de a 1 pixel de izquierda a derecha y de arriba-abajo y va generando una nueva matriz que compone al mapa de features



# REDES NEURONALES CONVOLUCIONALES-CNNs

## Filtro: conjunto de kernels

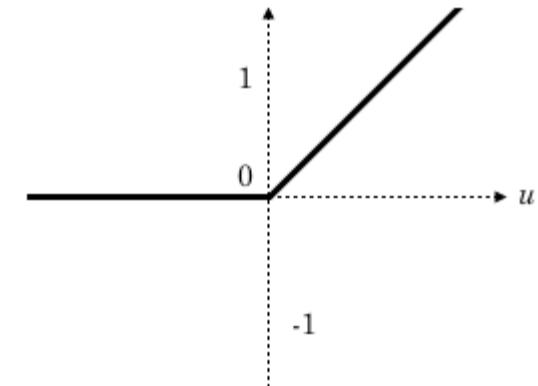
A medida se vamos desplaza el kernel, se va obteniendo una «nueva imagen» filtrada. En esta primer convolución y siguiendo con el ejemplo anterior, es como si obtuviéramos 32 «imágenes filtradas nuevas». Estas imágenes nuevas lo que están «dibujando» son ciertas características de la imagen original. Esto ayudará en el futuro a poder distinguir un objeto de otro (por ej. gato ó un perro).



La imagen realiza una convolución con un kernel y aplica la función de activación, en este caso ReLu

In the context of artificial neural networks, the rectifier or ReLU (Rectified Linear Unit) activation function[1][2] is an activation function defined as the positive part of its argument:

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



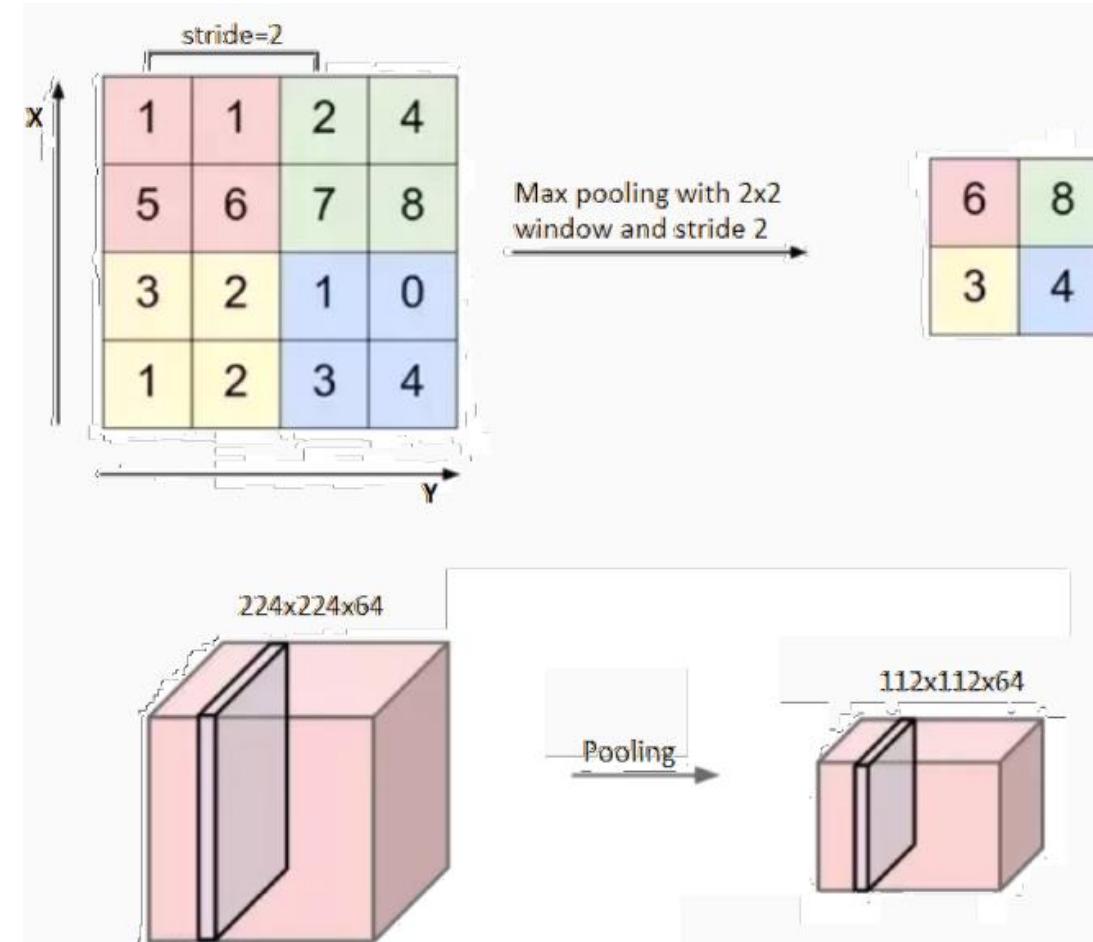


# REDES NEURONALES CONVOLUCIONALES-CNNs

## Subsampling (Pooling)

Es utilizado para realizar reducir el tamaño de la representación obtenida en la capa de convolución, acelera la velocidad de computo y robustece los mapas de características.

- Max-Pooling
- Min - Pooling
- Average-Pooling



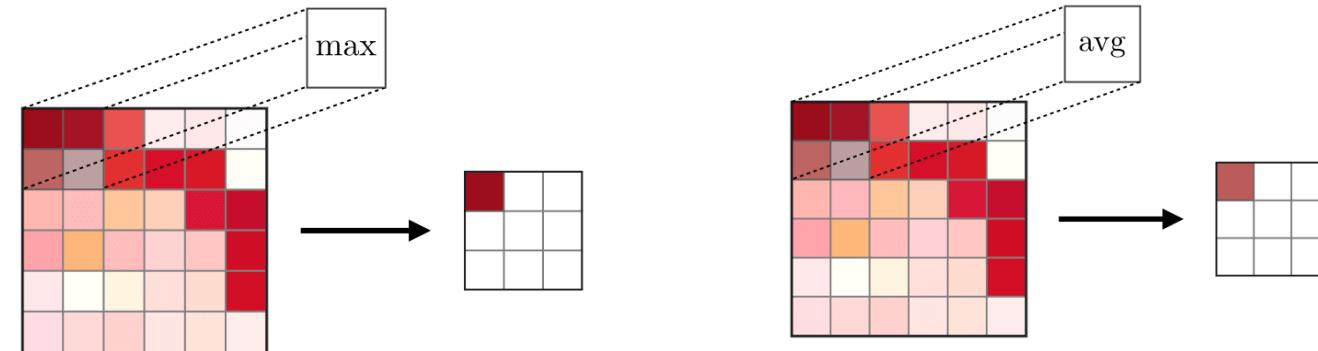


# REDES NEURONALES CONVOLUCIONALES-CNNs

## Pooling

Type	Max pooling	Average pooling
Purpose	Each pooling operation selects the maximum value of the current view	Each pooling operation averages the values of the current view
Comments	<ul style="list-style-type: none"><li>• Preserves detected features</li><li>• Most commonly used</li></ul>	<ul style="list-style-type: none"><li>• Downsamples feature map</li><li>• Used in LeNet</li></ul>

## Illustration





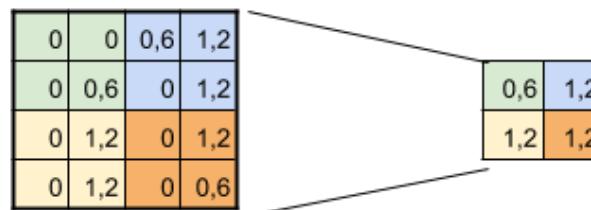
# REDES NEURONALES CONVOLUCIONALES-CNNs

## Pooling

### Ejemplo (continuación)

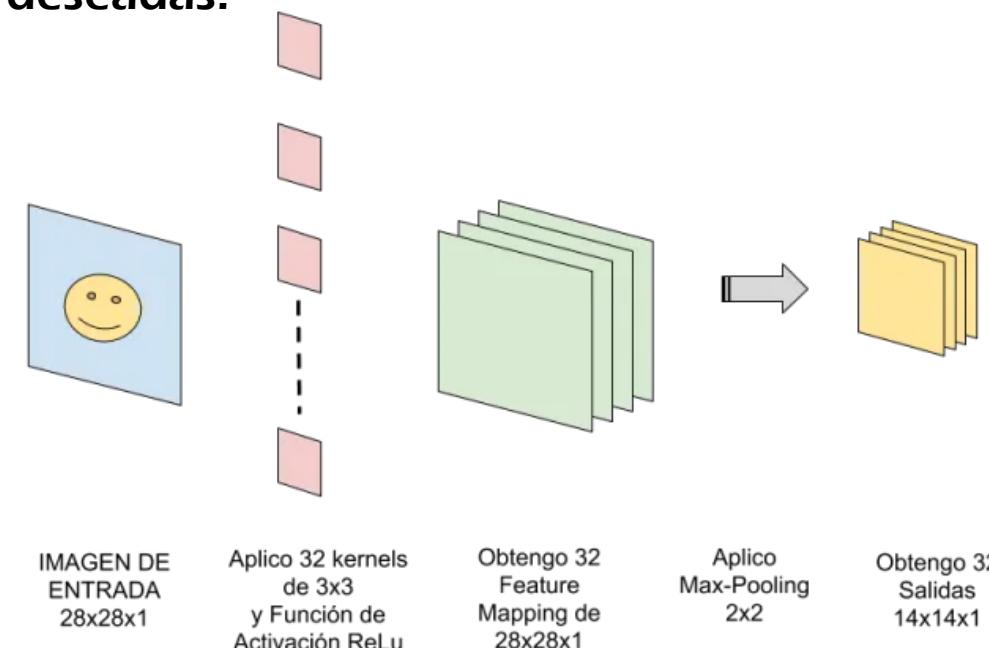
Utilizando la técnica de «Max-pooling» con un tamaño de  $2 \times 2$ , se recorre cada una de las 32 imágenes de características obtenidas anteriormente de  $28 \times 28$  píxeles de izquierda-derecha, arriba-abajo PERO en vez de tomar de a 1 pixel, se toma de « $2 \times 2$ » (2 de alto por 2 de ancho = 4 píxeles) y se irá preservando el valor «más alto» de entre esos 4 píxeles («Max»).

En este caso, usando  $2 \times 2$ , la imagen resultante es reducida «a la mitad» y quedará de  $14 \times 14$  píxeles. Luego de este proceso de submuestreo nos quedarán 32 imágenes de  $14 \times 14$ , pasando de 25.088 neuronas a 6272, las cuales son bastante menores y que -en teoría- deberían seguir almacenando la información más importante para detectar características deseadas.



SUBSAMPLING:

Aplico Max-Pooling de  $2 \times 2$   
y reduzco mi salida a la mitad





# REDES NEURONALES CONVOLUCIONALES-CNNs

## Pooling

### Ejemplo (continuación)

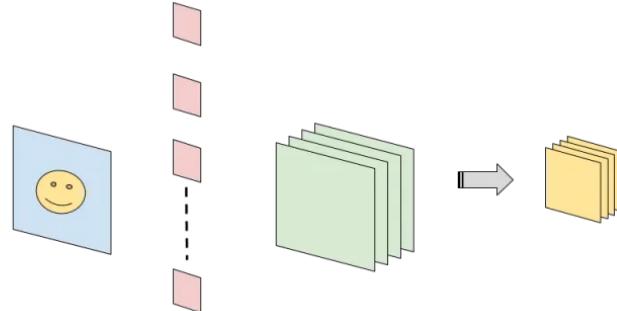
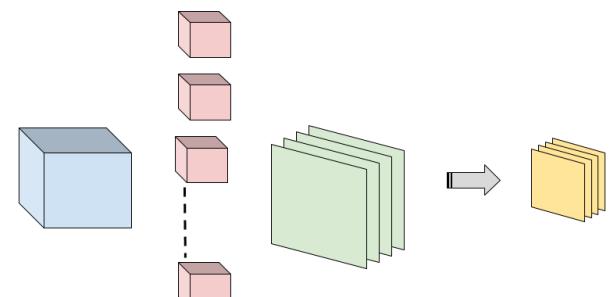


IMAGEN DE ENTRADA  
28x28x1      Aplico 32 kernels de 3x3 y Función de Activación ReLu      Obtengo 32 Feature Mapping de 28x28x1      Max-Pooling 2x2      Obtengo 32 Salidas 14x14x1

### SEGUNDA CONVOLUCIÓN (y sucesivas)



ENTRADA de convolución previa:  
14x14x32      Aplico 64 kernels de 3x3x32 y Función de Activación ReLu      Obtengo Feature Mapping de 14x14x64      Max-Pooling 2x2      Obtengo Salidas 7x7x64

PRIMERA CONVOLUCIÓN	2)Aplico Kernel	3)Obtengo Feature Mapping	4)Aplico Max-Pooling	5)Obtengo «Salida» de la 1era Convolución
<b>28x28x1 = 784 neuronas</b>	32 filtros de 3x3	<b>28x28x 32 kernel = 25.088 neuronas</b>	de 2x2	<b>14x14x32 = 6.272 neuronas</b>

SEGUNDA CONVOLUCIÓN	2)Aplico Kernel	3)Obtengo Feature Mapping	4)Aplico Max-Pooling	5)Obtengo «Salida» de la Convolución
<b>14x14x32= 6272 neuronas</b>	64 filtros de 3x3	<b>14x14x64 = 12544 neuronas</b>	de 2x2	<b>7x7x64 = 3136 neuronas</b>

TERCERA CONVOLUCIÓN	2)Aplico Kernel	3)Obtengo Feature Mapping	4)Aplico Max-Pooling	5)Obtengo «Salida» de la Convolución
<b>7x7x64= 3.136 neuronas</b>	128 filtros de 3x3	<b>7x7x128= 6272 neuronas</b>	de 2x2	<b>3x3x128 = 768 neuronas</b>

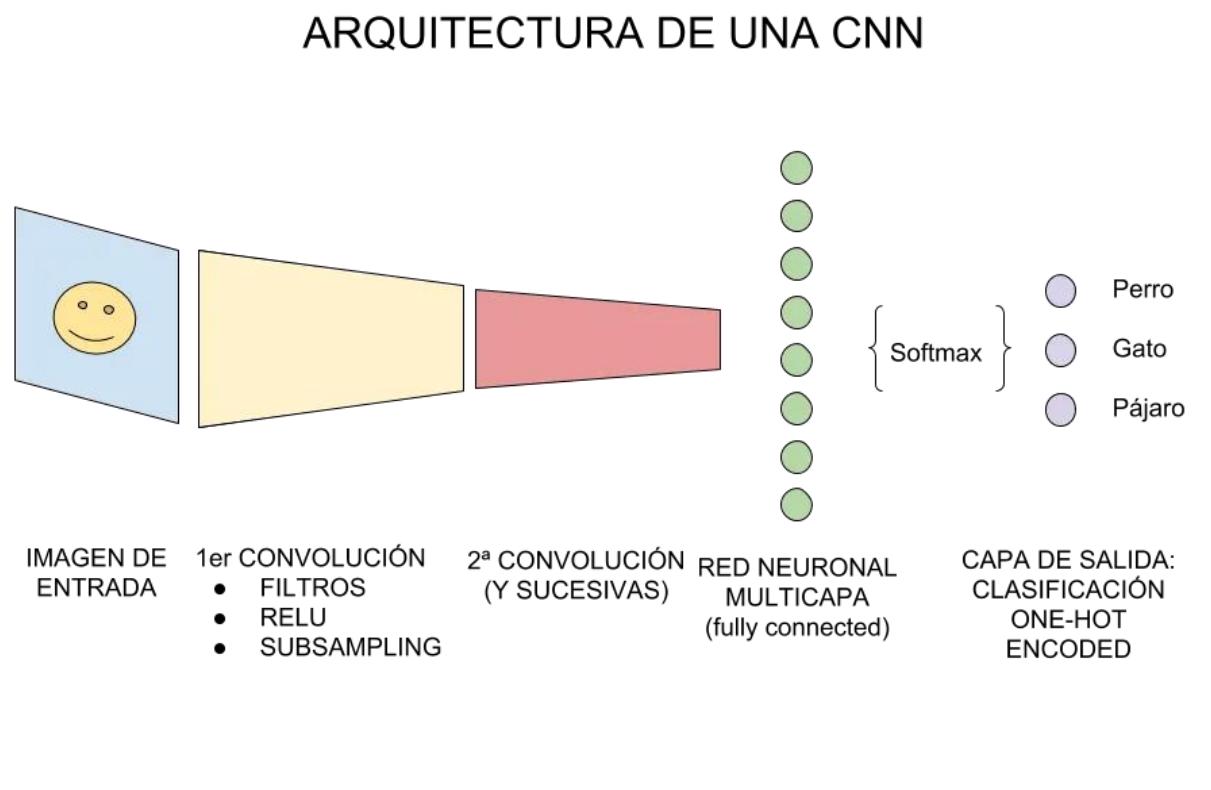


# REDES NEURONALES CONVOLUCIONALES-CNNs

## Ejemplo (continuación)

Coneectar con una red neuronal «tradicional».

Para terminar, se toma la última capa oculta a la que se hizo *subsampling*, que se dice que es «tridimensional» por tomar la forma -en nuestro ejemplo-  $3 \times 3 \times 128$  (alto,ancho,mapas) y la «aplanamos», esto es que deja de ser tridimensional, y pasa a ser una capa de neuronas «tradicionales», y con ello aplanamos (y conectamos) una nueva capa oculta de neuronas tipo feedforward.



Entonces, a esta nueva capa oculta «tradicional», le aplicamos una función llamada Softmax que conecta contra la capa de salida final que tendrá la cantidad de neuronas correspondientes con las clases que estamos clasificando. Si clasificamos perros y gatos, serán 2 neuronas. Si clasificamos coches, aviones ó barcos serán 3, etc.

Las salidas al momento del entrenamiento tendrán el formato conocido como «one-hot-encoding» en el que para perros y gatos sera: [1,0] y [0,1], para coches, aviones ó barcos será [1,0,0]; [0,1,0];[0,0,1].

Y la función de Softmax se encarga de pasar a probabilidad (entre 0 y 1) a las neuronas de salida. Por ejemplo una salida [0,2 0,8] indica 20% probabilidades de que sea perro y 80% de que sea gato, según este ejemplo.



# REDES NEURONALES CONVOLUCIONALES-CNNs

Con estos procesos apilados se logra un compresión progresiva de la información. La red debe conocer el número de filtros y su tamaño.

Entonces la red convolucional realiza la búsqueda de patrones (convolución), seleccionando los datos mas importantes (Max-pooling) y establece las relaciones entre los patrones y los objetos como tal (capa de salida - clasificador)

## Ejemplo 2:

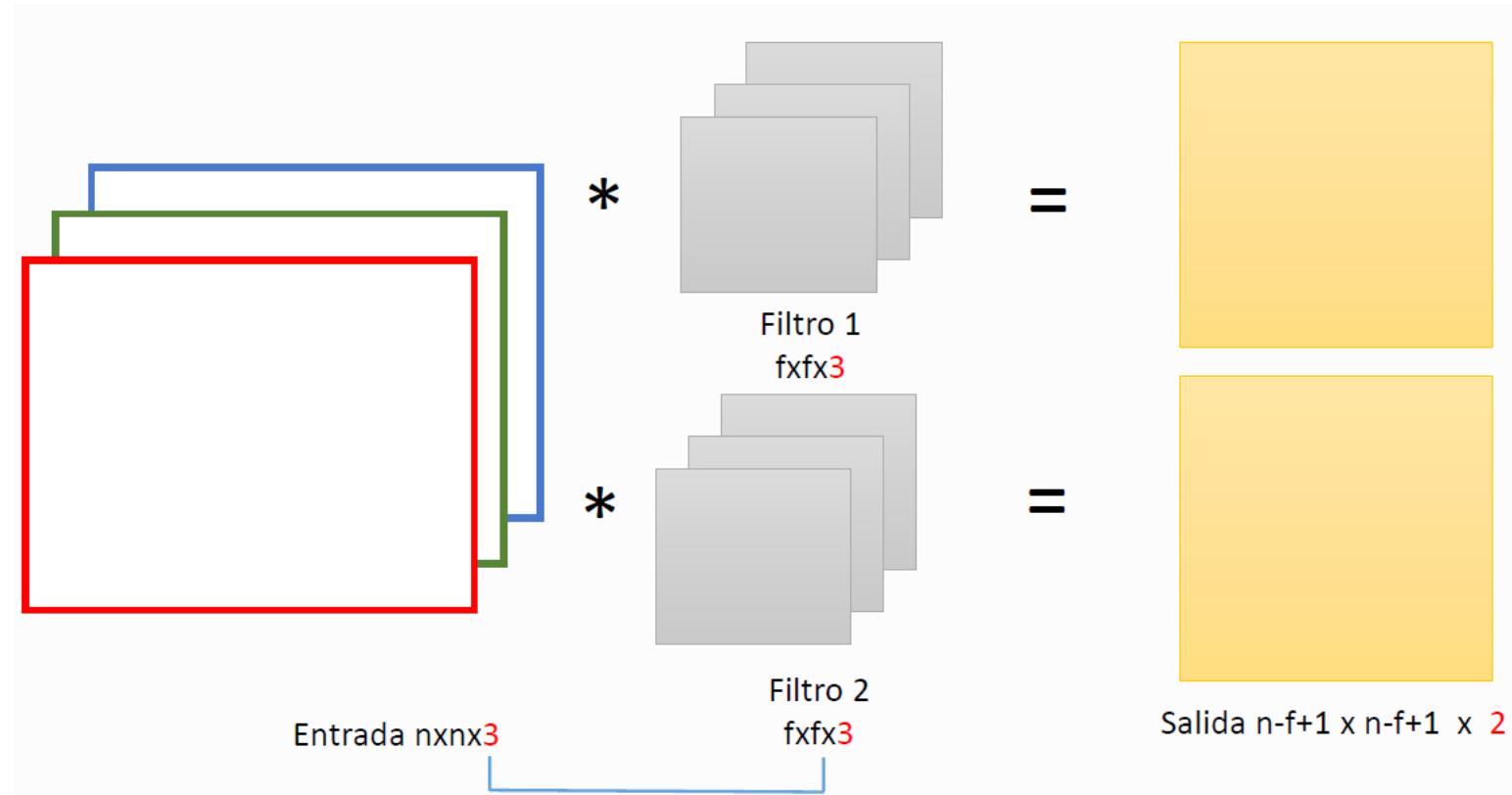
	Forma	Tamaño	# Parámetros
Entrada	(32,32,3)	3072	0
Conv1( $f=5, s=1, \#8$ )	(28,28,8)	6272	208
Pool1 (2x2, s=2)	(14,14,8)	1568	0
Conv2( $f=5, s=1, \#16$ )	(10,10,16)	1600	416
Pool2 (2x2, s=2)	(5,5,16)	400	0
FullCon3	(120,1)	120	48001
FullCon4	(84,1)	84	10081
Softmax	(10,1)	10	841



# REDES NEURONALES CONVOLUCIONALES-CNNs

## Convolución en imágenes de varios canales

Si la imagen es a color, el kernel realmente sería por ejemplo de  $3 \times 3 \times 3$  es decir: un filtro con 3 kernels de  $3 \times 3$ ; luego esos 3 filtros se suman (y se le suma una unidad bias) y conformarán 1 salida (como si fuera 1 solo canal).

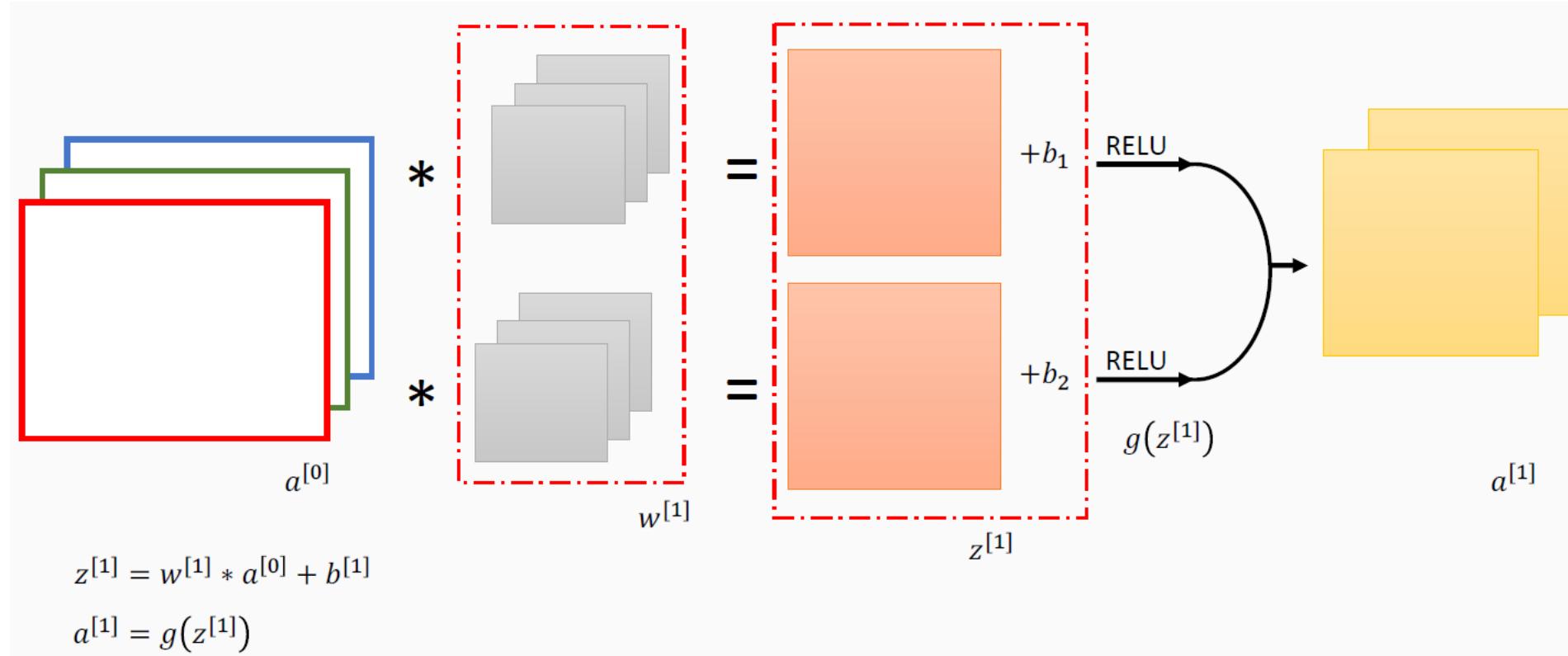




# REDES NEURONALES CONVOLUCIONALES-CNNs

Convolución en imágenes de varios canales

X.



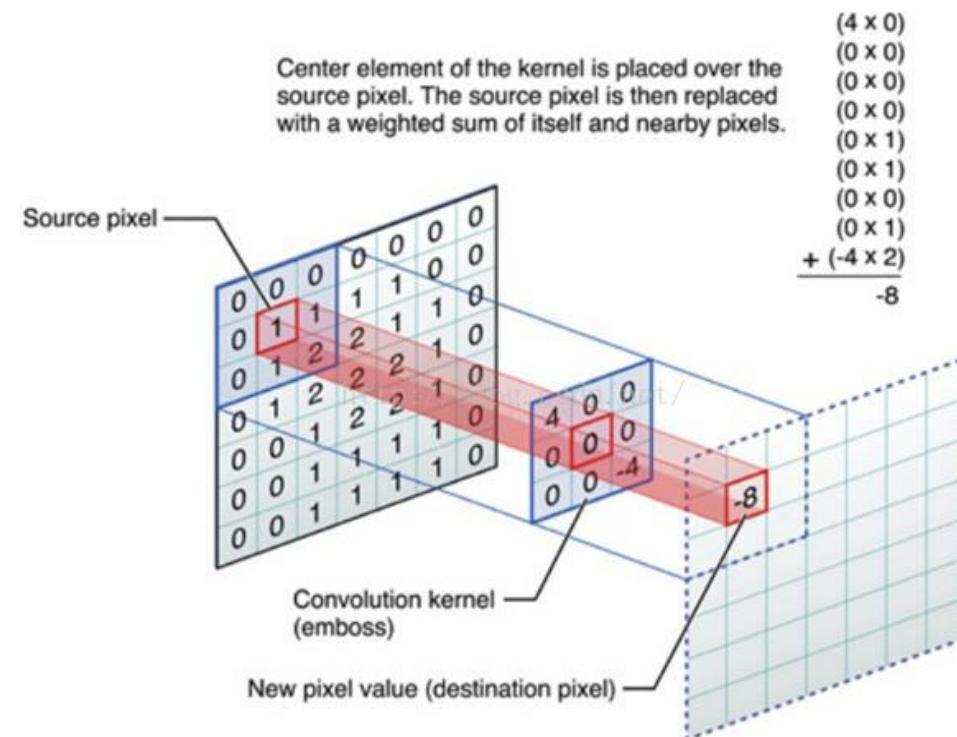


# REDES NEURONALES CONVOLUCIONALES-CNNs

## Proceso de convolución multicanal (como RGB de tres canales)

Para una imagen de un solo canal, si se utilizan 10 núcleos de convolución para el cálculo de convolución, se pueden obtener 10 mapas de características; si la entrada es una imagen multicanal, el número de mapas de características de salida sigue siendo el número de núcleos de convolución (10) .

### 1. Proceso de convolución de un núcleo de convolución de un solo canal



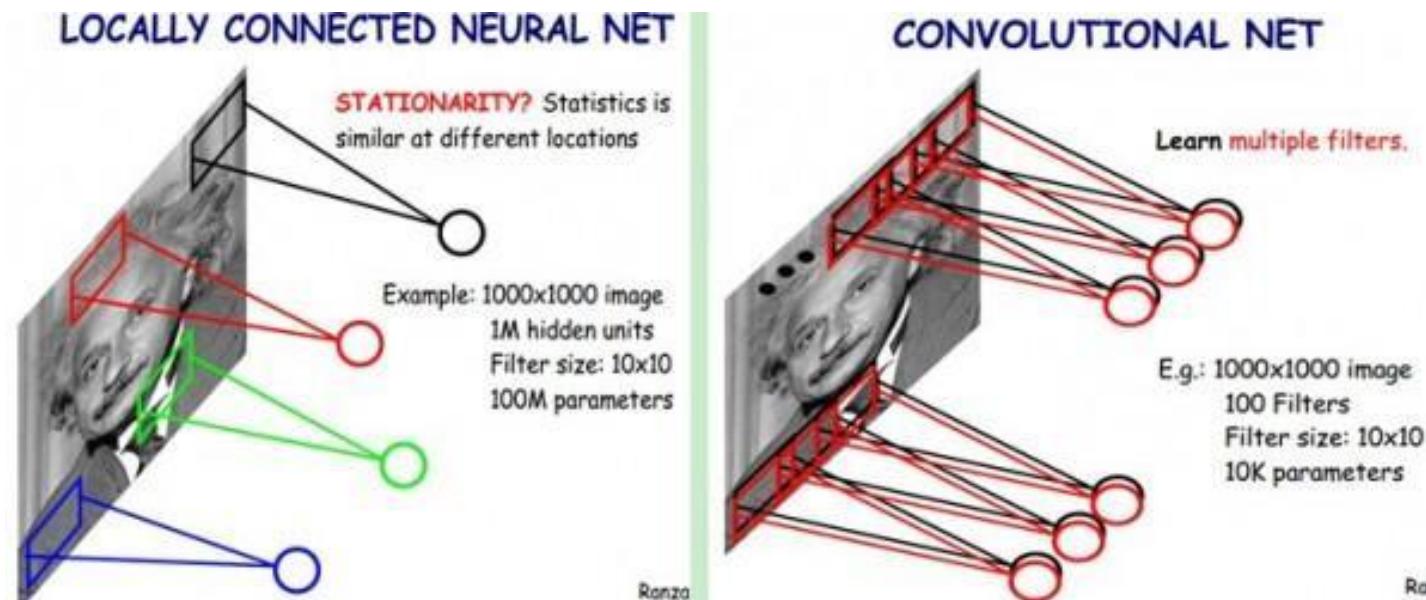


# REDES NEURONALES CONVOLUCIONALES-CNNs

Proceso de convolución multicanal (como RGB de tres canales)

## 2. Proceso de convolución del kernel de convolución múltiple de un canal

La extracción de características obtenida por un kernel de convolución no es suficiente. Podemos agregar múltiples kernels de convolución, como 32 núcleos de convolución, y podemos aprender 32 tipos de características. Cuando hay varios núcleos de convolución, como se muestra a continuación: la salida es 32 mapas de características





# REDES NEURONALES CONVOLUCIONALES-CNNs

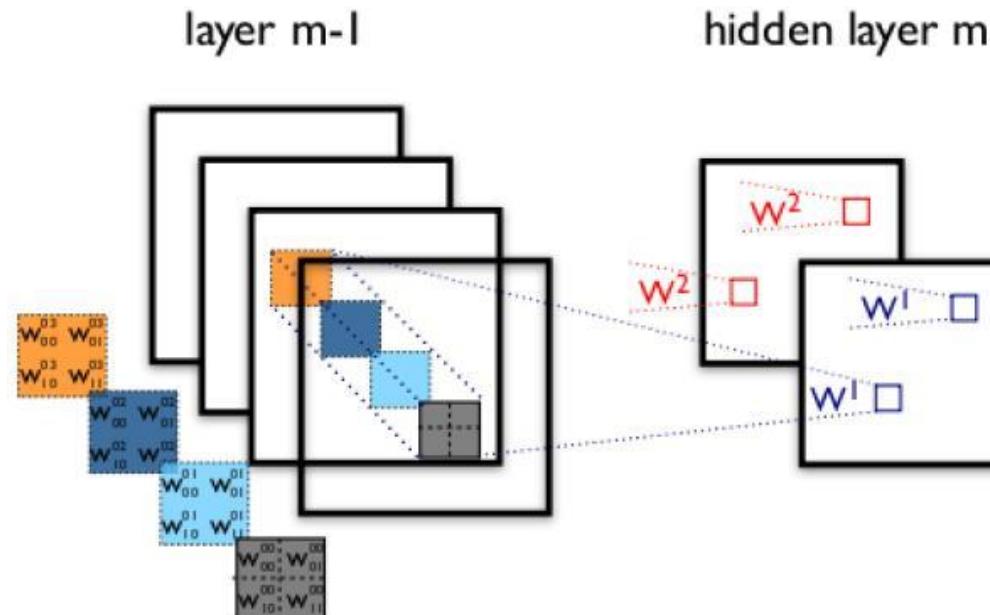
Proceso de convolución multicanal (como RGB de tres canales)

## 3. Múltiples núcleos de convolución con múltiples canales

La figura muestra la operación de convolución en cuatro canales. Hay dos núcleos de convolución para generar dos canales. Cada canal en los cuatro canales corresponde a un núcleo de convolución. Primero, ignore  $w_2$  y solo observe  $w_1$ . Luego, el valor en una determinada posición  $(i, j)$  de  $w_1$  está determinado por los cuatro canales (**Se obtiene sumando los resultados de la convolución en  $i, j$  y luego tomando el valor de la función de activación**. Entonces, finalmente se obtienen dos mapas de características, es decir, el número de núcleos de convolución en la capa de salida es el número de mapas de características.

$$h_{ij}^k = \tanh((W^k * x)_{ij} + b_k)$$

Por tanto, en el proceso de obtención de 2 canales por convolución de 4 canales en la figura anterior, el **número de parámetros** es  $4 \times 2 \times 2 \times 2$ , donde 4 significa 4 canales, los primeros 2 significa generar 2 canales, y finalmente El  $2 \times 2$  representa el tamaño del núcleo de convolución.





# REDES NEURONALES CONVOLUCIONALES-CNNs

## Proceso de convolución multicanal (como RGB de tres canales)

### 3. Múltiples núcleos de convolución con múltiples canales

- **Imagen:** Si el ancho de la imagen es ancho:  $W$ , la altura es alto:  $H$  y el número de canales de la imagen es  $D$ . Generalmente, en la actualidad se utiliza RGB de tres canales  $D = 3$ . El número de canales está representado por  $D$ ;
- **Kernel de convolución:** el tamaño del kernel de convolución es  $K * K$ . Dado que la imagen procesada es del canal  $D$ , el kernel de convolución tiene un tamaño de  $K * K * D$ . Por lo tanto, para la imagen RGB de tres canales, bajo la premisa de especificar `kernel_size`, el tamaño real del kernel de convolución es `kernel_size * kernel_size * 3`. A

Para cada canal de la imagen del canal  $D$ , la convolución bidimensional se realiza en cada canal por separado, y luego los canales  $D$  se suman para obtener la salida de convolución bidimensional en esa posición. Por lo tanto, si hay núcleos de convolución  $M$ , se pueden obtener resultados de salida de convolución bidimensional  $M$ . En el caso del relleno, el tamaño de la imagen de salida se puede mantener igual que el original, por lo que se emite  $(W, H, M)$ .

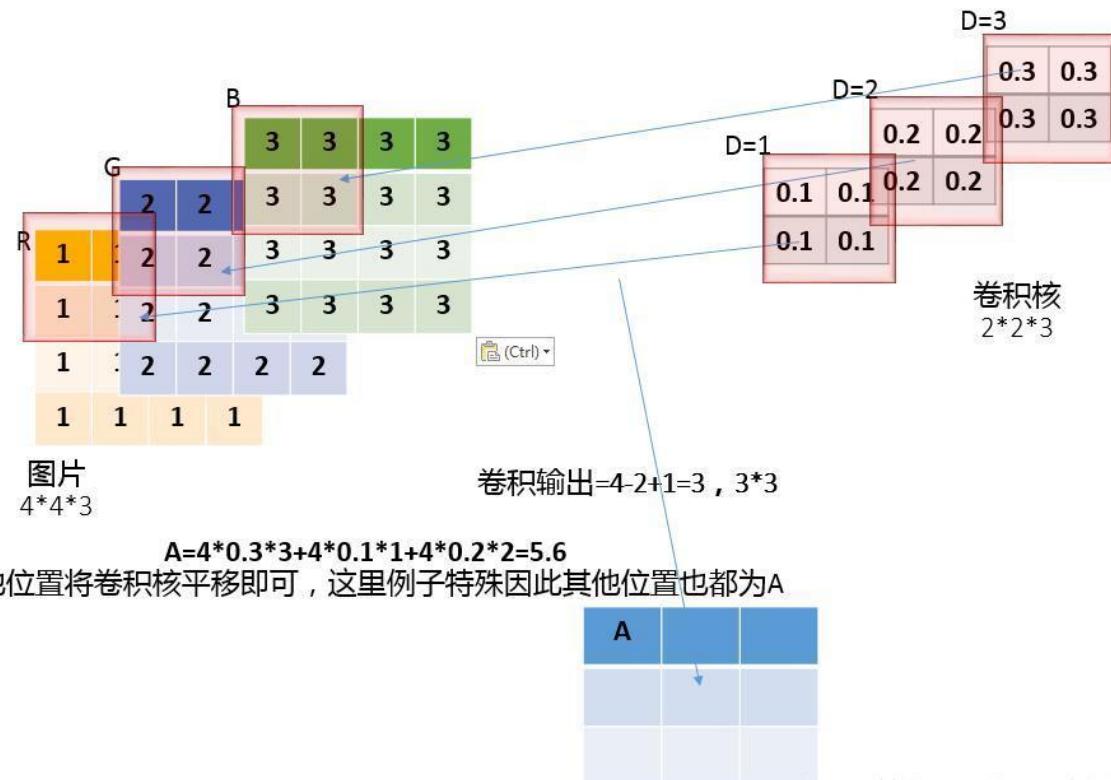
Para la imagen RGB de tres canales, se debe usar `kernel_size * kernel_size` size kernel en cada canal para deconvolver la imagen  $W * H$  en cada canal en los tres canales de R, G y B, y luego La salida obtenida por la convolución de los tres canales se suma para obtener el resultado de la salida de convolución bidimensional.



# REDES NEURONALES CONVOLUCIONALES-CNNs

Proceso de convolución multicanal (como RGB de tres canales)

## 3. Múltiples núcleos de convolución con múltiples canales



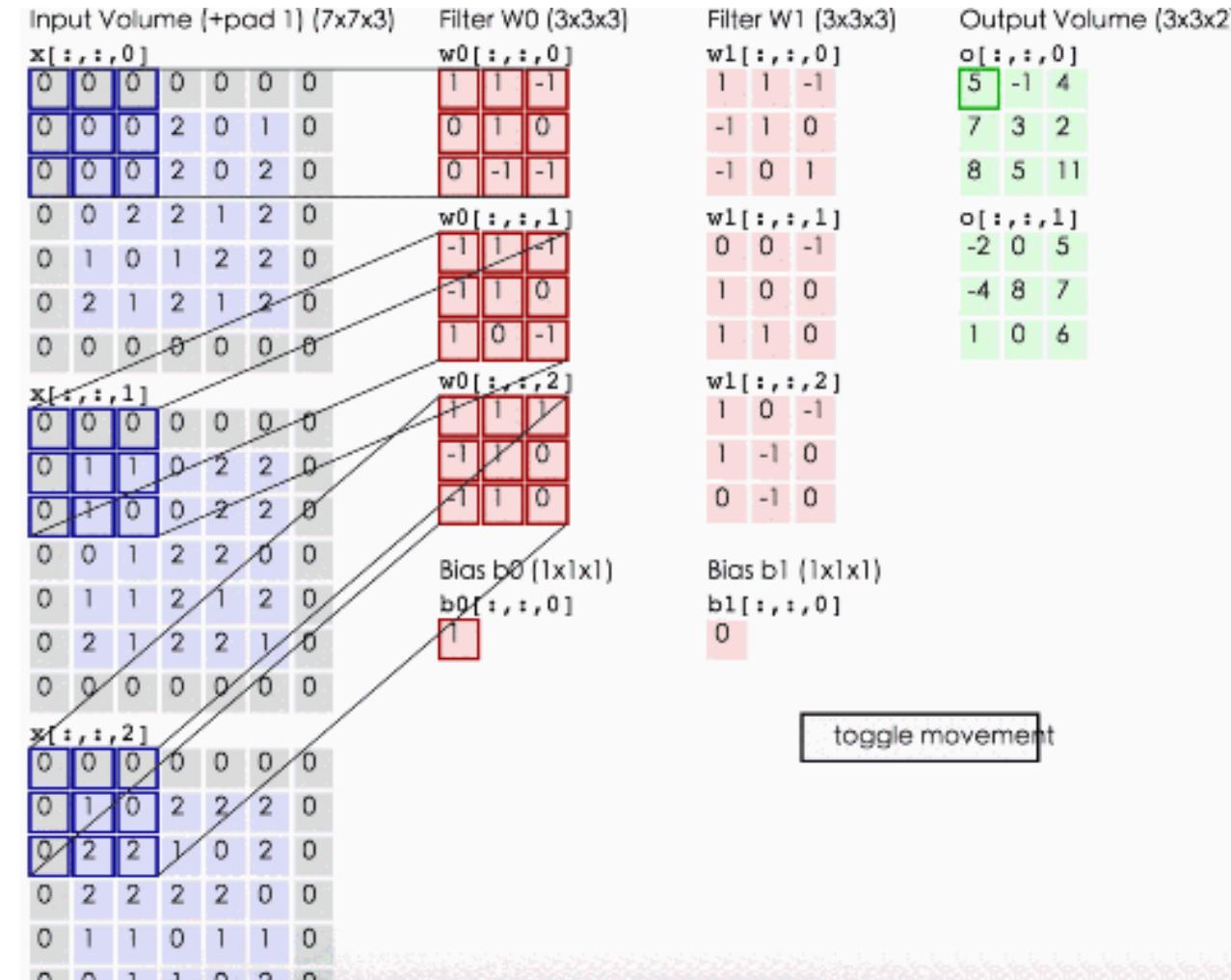


# REDES NEURONALES CONVOLUCIONALES-CNNs

Proceso de convolución multicanal (como RGB de tres canales)

## 3. Múltiples núcleos de convolución con múltiples canales

La siguiente figura muestra de forma dinámica y vívida el proceso de cálculo de la capa de convolución de imagen de tres canales:

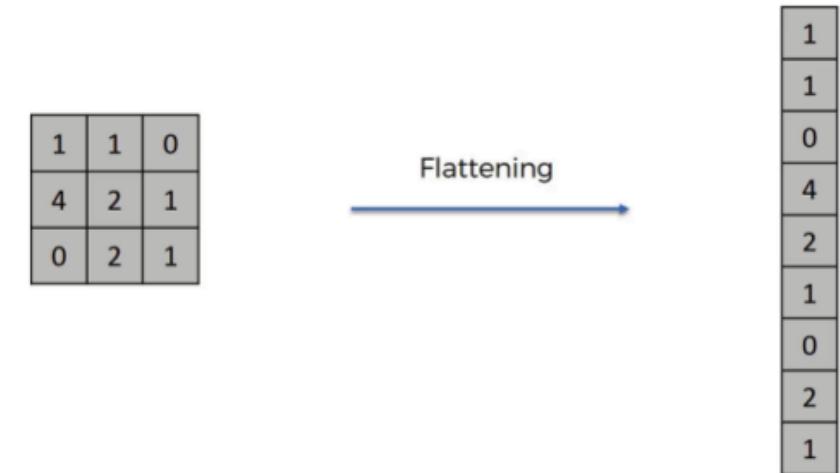




# REDES NEURONALES CONVOLUCIONALES-CNNs

## Capa Flatten

Capa que transforma un tensor o matriz de entrada en un vector unidimensional. Se suele utilizar al inicio de las redes neuronales que trabajan con imágenes para transformar la entrada a un vector unidimensional.





# REDES NEURONALES CONVOLUCIONALES-CNNs

**CNN Explainer- Learn Convolutional Neural Network (CNN) in your browser!**

**HERRAMIENTA INTERACTIVA**

<https://poloclub.github.io/cnn-explainer/>

**CNN Explainer: Learning Convolutional Neural Networks with Interactive Visualization**

<https://arxiv.org/abs/2004.15004>

**Demo Video "CNN Explainer: Learning Convolutional Neural Networks with Interactive Visualization"**

<https://www.youtube.com/watch?v=HnWIHWFbuUQ>



# REDES NEURONALES CONVOLUCIONALES-CNNs

Puede encontrarse información adicional en el siguiente link de la universidad de Stanford

<https://stanford.edu/~shervine/teaching/cs-230/cheatsheet-convolutional-neural-networks>

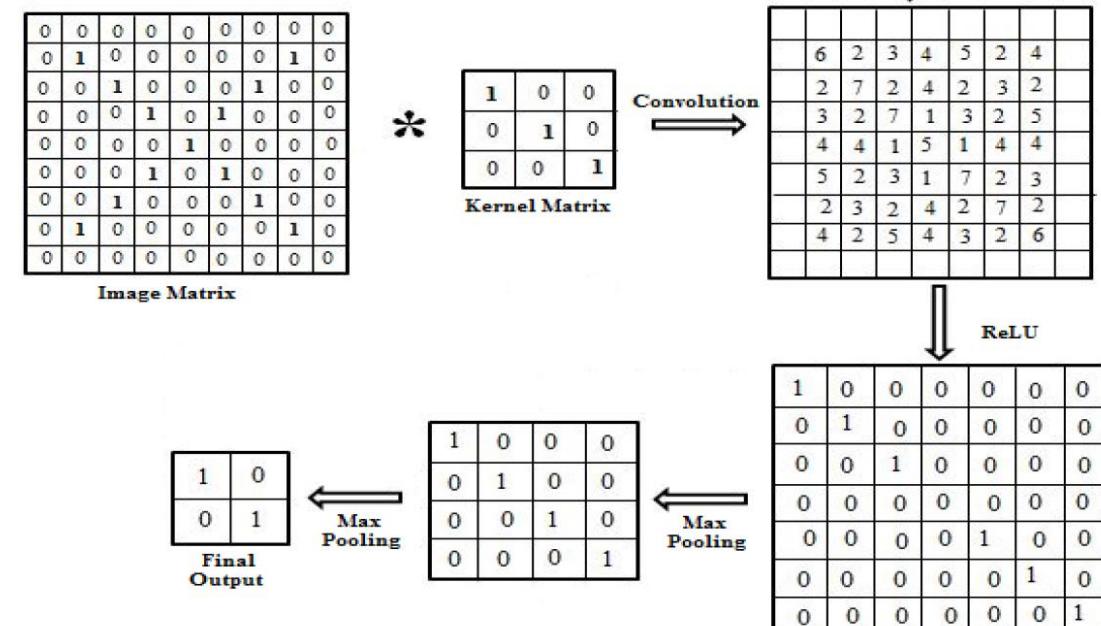


# REDES NEURONALES CONVOLUCIONALES-CNNs

## Capa Fully Connected

En esta capa, las salidas agrupadas y convolucionadas, utilizando diferentes núcleos, se recopilan para formar una única matriz. En la CNN, las características se extraen de las capas anteriores. Aquí, “fully connected” explica que todas las entradas o nodos de una capa están asociados a cada unidad de activación de la siguiente capa.

Una imagen de entrada se convoluciona con el kernel y la imagen, luego se realiza la función ReLU y la función de agrupación. Después de eso, sale el resultado. Esta imagen de entrada está convolucionada con otros núcleos que son bordes de la imagen de entrada. Después de estos procesos van saliendo los resultados.



# REDES NEURONALES CONVOLUCIONALES-CNNs



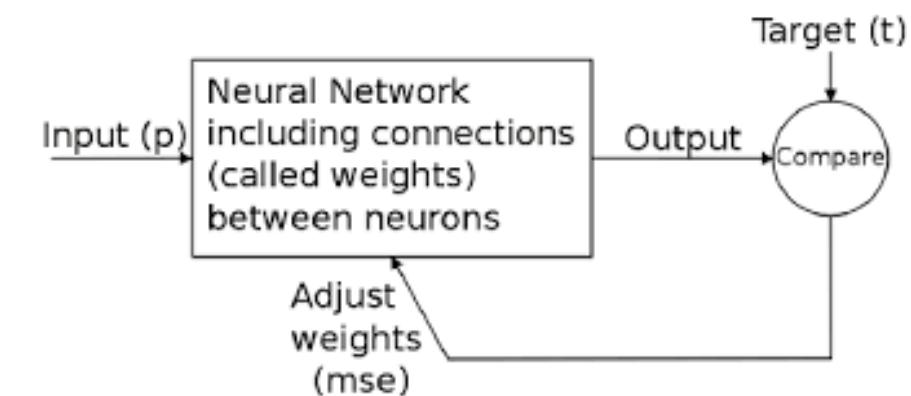
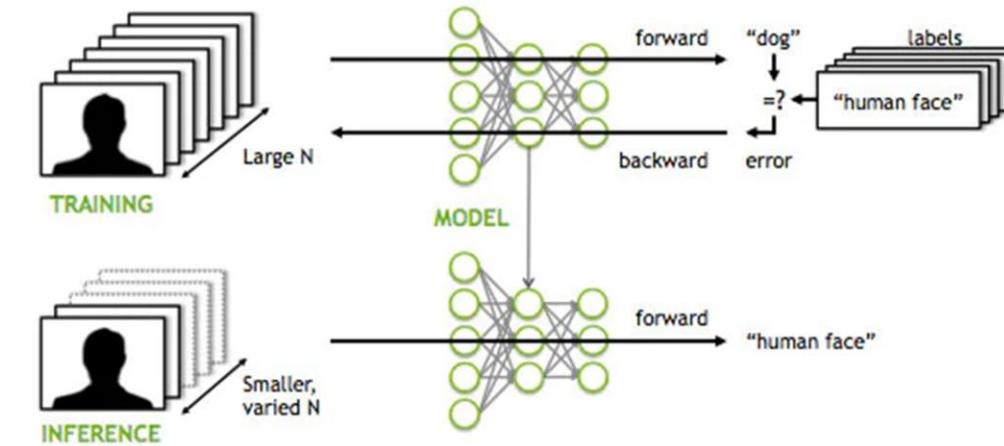
## ENTRENAMIENTO

El entrenamiento de una red neuronal es el proceso mediante el cual la red aprende a realizar la tarea para la cual ha sido diseñada. Consiste en iterar sobre los valores de los parámetros de la red (pesos y sesgos) de manera que el error de predicción en los datos de entrenamiento sea mínimo.

Se utiliza el algoritmo backpropagation y procesos de optimización basados **Gradiente Descendente Estocástico**.

De manera común a cualquier proceso de entrenamiento el conjunto de datos se divide en tres:

- **Datos de Entrenamiento:** Permiten realizar el ajuste de los parámetros en la red neuronal.
- **Datos de Validación:** Permite realizar el ajuste de los hiper-parámetros durante el proceso de entrenamiento.
- **Datos de Prueba:** Permite estimar el desempeño de la red frente a datos futuros.





# REDES NEURONALES CONVOLUCIONALES-CNNs

## Conceptos asociados al entrenamiento:

- **Numero de épocas**: es un hiperparámetro que representa el número de veces que se recorre el conjunto de datos de entrenamiento completo.
- **Batch size**: es un hiperparámetro que define el número de ejemplos del conjunto completo de datos que se van a usar en una actualización de los pesos de la red. Una de las principales ventajas de usar un batch size de tamaño pequeño es la menor necesidad de memoria durante el proceso de propagación hacia atrás.
- **Step size**: es un hiperparámetro que define la magnitud de actualización de los parámetros entrenables durante el proceso de back propagation. La elección correcta de este parámetro es determinante para lograr la convergencia del algoritmo.
- **Función de coste**: conocida en inglés como loss. Es la función que obtiene el valor del error entre la salida de la red y la salida que se esperaba. Es la función a optimizar durante el proceso de entrenamiento..

$$J(\theta) = \frac{1}{m} \sum_{n=1}^m \text{Coste}(h_\theta(x^{(i)}), y^{(i)})$$

Habitualmente, las redes neuronales, se entranan mediante el método Stochastic Gradient Descent, (SGD) el cual consiste en aplicar el método de descenso de gradiente, Gradient Descent, a cada batch, con el objetivo de reducir la función de coste.



# REDES NEURONALES CONVOLUCIONALES-CNNs

## ALGORITMO DE OPTIMIZACIÓN STOCHASTIC GRADIENT DESCENTE - SGD

Gradiente Descendente Estocástico.

La función objetivo a minimizar es  $J(f)$ , la cual permite medir el desempeño general del proceso de entrenamiento.

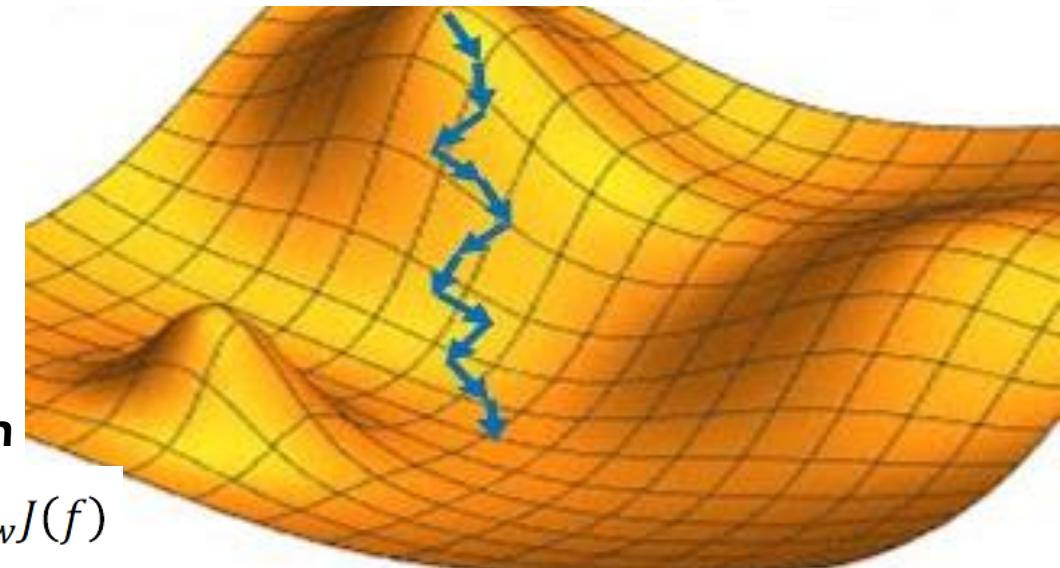
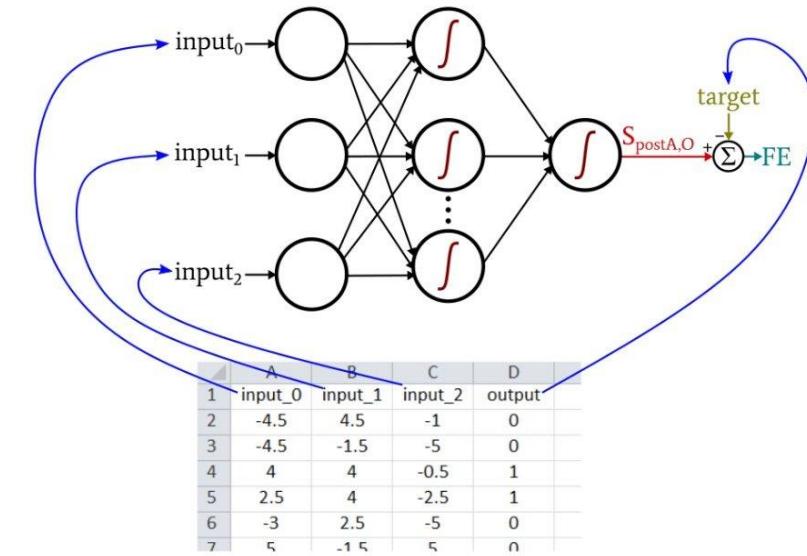
$$J(f) = \frac{1}{n} \sum_{i=1}^n \mathcal{L}(\hat{y}_i, y_i)$$

Donde  $\mathcal{L}(\hat{y}_i, y_i)$  se denomina función de pérdida y mide el costo de la predicción  $\hat{y}_i = f(x_i, w)$  cuando la salida actual es  $y_i$

$$J(f) = \frac{1}{n} \sum_{i=1}^n (f(x_i, w) - y_i)^2$$

Utilizando el método de gradiente descendente tradicional, en cada iteración se actualizan los parámetros  $w$  con base en la dirección descendente del gradiente.

$$w^{k+1} = w^k - \alpha^k \nabla_w J(f)$$





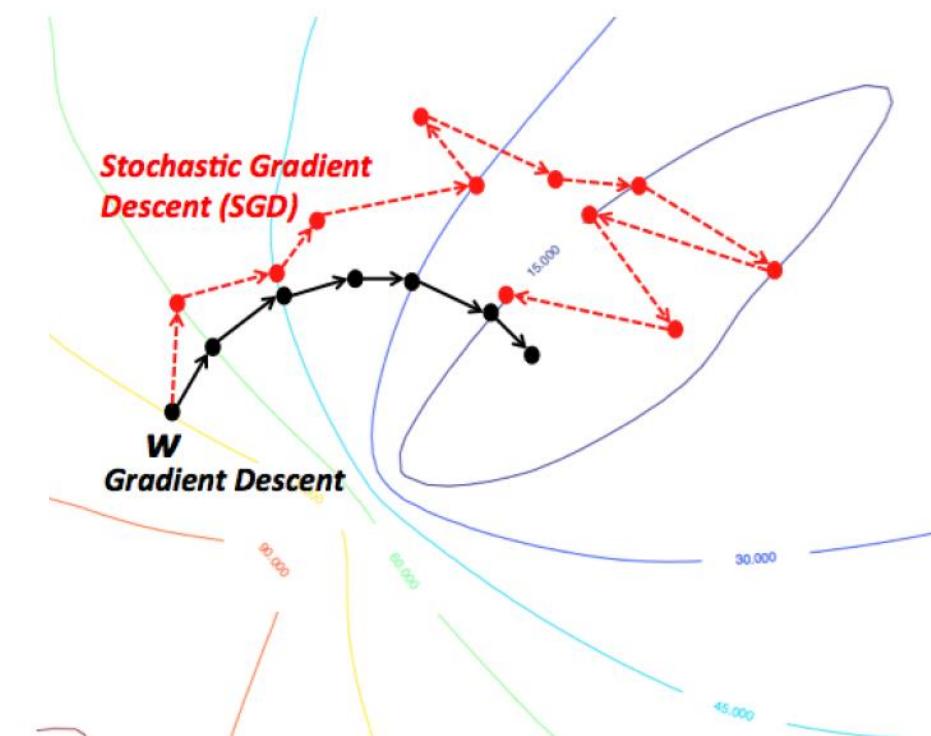
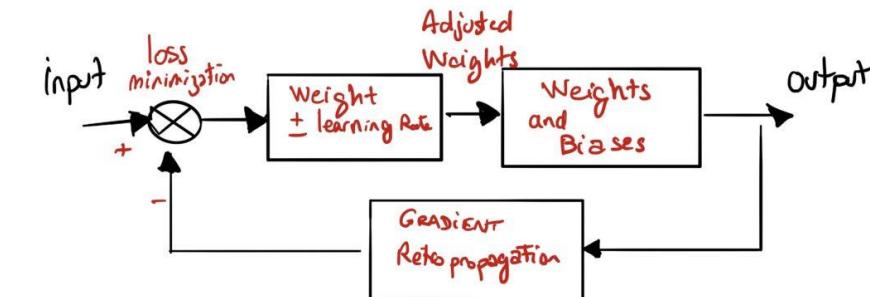
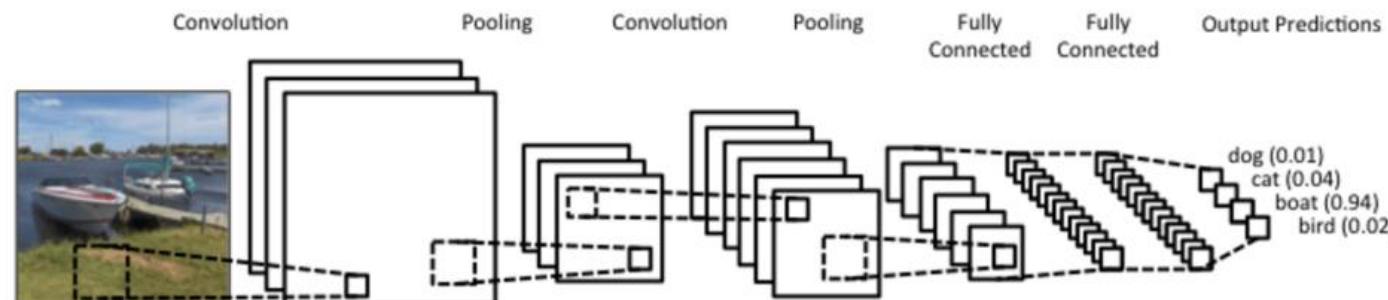
# REDES NEURONALES CONVOLUCIONALES-CNNs

## ALGORITMO DE OPTIMIZACIÓN STOCHASTIC GRADIENT DESCENTE - SGD

### Gradiente Descendente Estocástico.

El algoritmo de gradiente descendente estocástico SGD es una variante del gradiente descendente que busca simplificar de forma drástica el proceso de optimización.

No se calcula  $\nabla_w J(f)$  de forma exacta, sino que en cada iteración se realiza una estimación del gradiente en base a una única muestra  $[x_i, y_i]$  o un conjunto de  $m$  muestras seleccionadas de forma aleatoria.





# REDES NEURONALES CONVOLUCIONALES-CNNs

## Conceptos asociados al entrenamiento:

- El proceso para cada batch es el siguiente:

Cada imagen del batch es propagada hacia adelante, de tal manera que, atraviesa la red siendo multiplicada por los parámetros entrenables (**pesos y sesgos**) de cada capa. El resultado predicho por la red es comparado con el resultado deseado y se obtiene la función de perdida, la cual depende de todos los parámetros entrenables de la red. Para cada imagen en particular y con el objetivo de reducir la función de pérdida, se aplica el algoritmo de propagación hacia atrás (**back propagation**), en el cual se calcula la derivada de la función de coste con respecto a cada parámetro entrenable  $\nabla(w_1, w_2...w_n)$ . De aquí obtenemos un vector fila con tantas dimensiones como parámetros entrenables, el cual nos indica en qué dirección habría que variar cada parámetro para conseguir reducir la función de coste de esta imagen en concreto.

Ahora bien, no solo tenemos una imagen. En el entrenamiento, se suman las contribuciones de cada imagen en el batch y se aplica la actualización a los parámetros entrenables. El proceso se repite iterativamente para todos los batchs (SGD) y tantas veces como épocas haya,

$$W_{\text{new}} = W_{\text{old}} - \eta \nabla(w_1, w_2...w_n).$$

Al final se habrá reducido la función de coste total, es decir, de todos los datos, y, por tanto, el error entre la predicción de la red y la realidad será más pequeño. De esta manera la red aprende a hacer predicciones cada vez más acertadas en los datos de entrenamiento. Asumiendo que los datos de test son similares a los datos de entrenamiento, las predicciones también tendrán una alta precisión en ellos.

# REDES NEURONALES CONVOLUCIONALES-CNNs

## Conceptos asociados al entrenamiento:

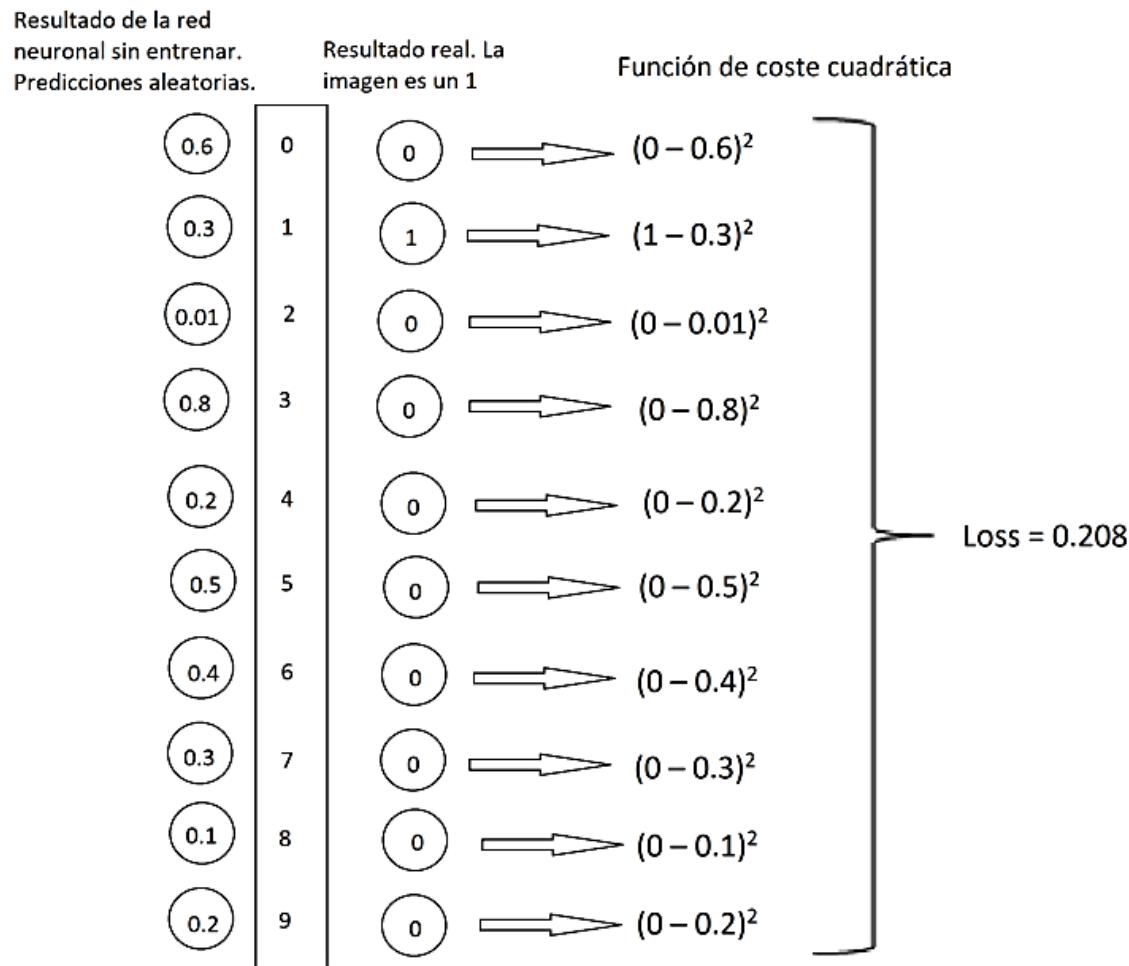
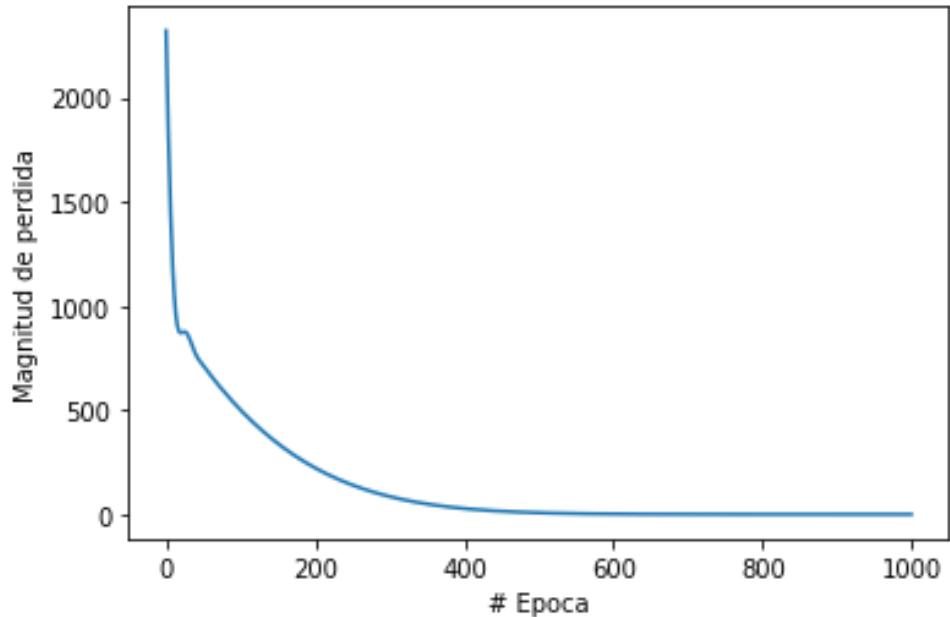


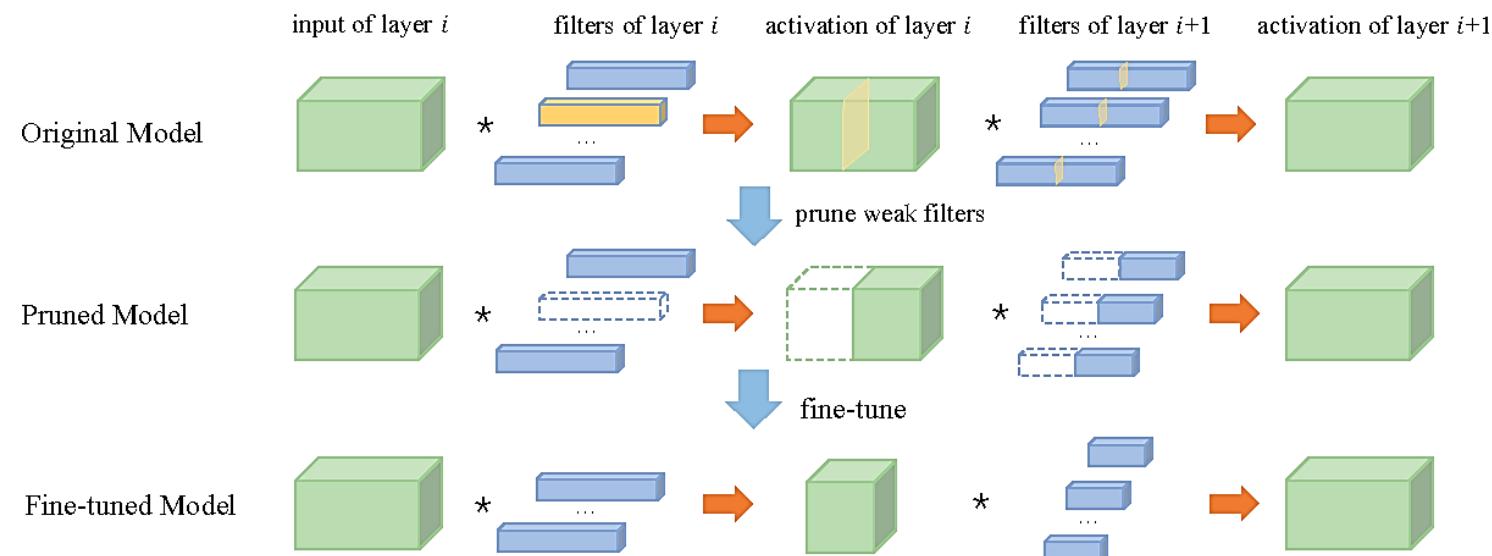
Figura 2.7: Ejemplo numérico de predicción de los dígitos del dataset MNIST. Tras atravesar la red neuronal sin entregar se puede apreciar que las predicciones no son correctas ( primera columnas de neuronas), la siguiente columna de neuronas ilustra lo que idealmente debería de resultar de la predicción. El error se calcula con una función de coste cuadrática.



# Pruning: Poda de redes neuronales convolucionales

La poda de un modelo entrenado consiste en la eliminación de parámetros considerados innecesarios para el modelo (inspirado en el artículo "Pruning Filters for Efficient ConvNets" de Hao Li et al). En una red neuronal completamente entrenada, muchos de sus pesos están muy cerca de cero. Esto indica que estos pesos no juegan un papel importante en la determinación de la salida de la red neuronal.

Con esta idea, se revisa las capas de CNN en el modelo, se verifica qué canales de filtro tienen pesos muy cercanos a cero y se eliminan por completo de la capa. A partir de entonces, se vuelve a conectar las capas individuales de la CNN de manera adecuada (usando una capa donde sea necesario para llenar un tensor de entrada/salida con ceros para obtener la forma correcta) para garantizar que la información aún pueda fluir correctamente a través del modelo. El resultado es un modelo mucho más delgado que aún puede realizar la clasificación de imágenes/detección de objetos para la que fue entrenado.





# Capacidad de una red

**Capacidad es un término similar a la complejidad del modelo. Un modelo con mayor capacidad puede modelar más relaciones entre más variables que un modelo con menor capacidad.**

**La capacidad de un modelo de red neuronal de aprendizaje profundo controla el alcance de los tipos de funciones de mapeo que puede aprender. Un modelo con muy poca capacidad no puede aprender el conjunto de datos de entrenamiento, lo que significa que no se ajustará bien, mientras que un modelo con demasiada capacidad puede memorizar el conjunto de datos de entrenamiento, lo que significa que se ajustará en exceso o se atascará o perderá durante el proceso de optimización.**

**La capacidad de un modelo de red neuronal se define configurando el número de nodos y el número de capas.**

**La capacidad del modelo de red neuronal se controla tanto por la cantidad de nodos como por la cantidad de capas en el modelo.**

**Un modelo con una sola capa oculta y un número suficiente de nodos tiene la capacidad de aprender cualquier función de mapeo, pero el algoritmo de aprendizaje elegido puede o no ser capaz de realizar esta capacidad.**

**Aumentar la cantidad de capas proporciona un atajo para aumentar la capacidad del modelo con menos recursos, y las técnicas modernas permiten que los algoritmos de aprendizaje entrenen con éxito modelos profundos.**



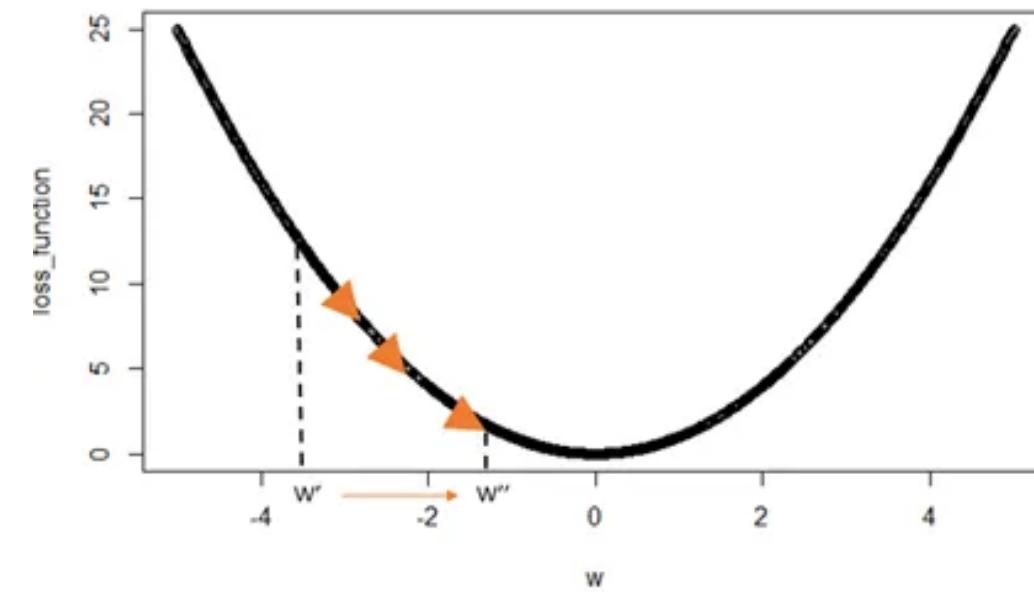
# Parámetros e hiperparámetros de una red.

- **Parámetros** : son los coeficientes del modelo, y son elegidos por el propio modelo. Significa que el algoritmo, mientras aprende, optimiza estos coeficientes (de acuerdo con una estrategia de optimización dada) y devuelve una serie de parámetros que minimizan el error. Requieren inicializarse en el entrenamiento de la red. Para una CNN puede consistir en los pesos y bias.
- **Hiperparámetros** : son elementos que, a diferencia de los anteriores, es necesario configurar. Además, el modelo no los actualizará de acuerdo con la estrategia de optimización: siempre será necesaria su intervención manual.



# Parámetros e hiperparámetros de una red.

- **Tipo de Hiperparámetros :**
- **Número de capas ocultas:** representa el numero de capas ocultas de la red. Se asocia un balance entre la simplicidad y la capacidad para clasificar adecuadamente los datos de entrada. Hay que tener en cuenta sin embargo que existe un numero limite de capas limite a partir del cual ya no se mejora la precisión de la red.
- **Tasa de aprendizaje:** se refiere al paso de retropropagación, cuando los parámetros se actualizan de acuerdo con una función de optimización. Los parámetros se actualizan para que puedan converger hacia el mínimo de la función de pérdida. Desea minimizar la pérdida, por lo que idealmente desea que el  $w$  actual se deslice hacia el mínimo.



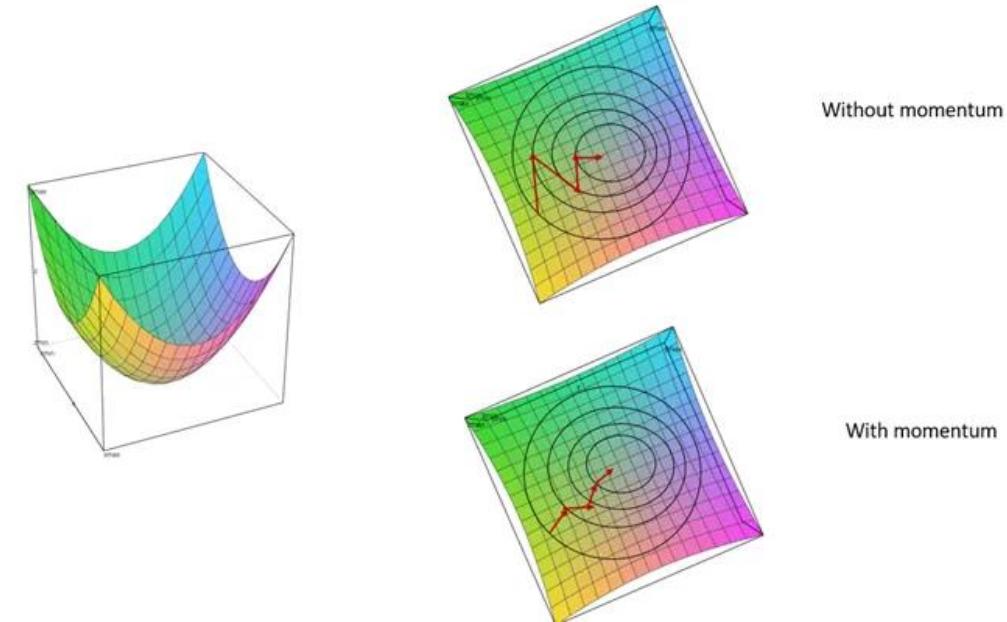


# Parámetros e hiperparámetros de una red.

## ■ **Tipo de Hiperparámetros :**

• **Impulso:** técnica utilizada durante la fase de retropropagación. Debido a que el proceso de la tasa de aprendizaje puede resultar demasiado largo y afectar la eficiencia del algoritmo, puede hacerse seguimiento de las direcciones anteriores (gradientes de la función de pérdida con respecto a los pesos). Esto se realiza para aumentar la velocidad de convergencia no en términos de tasa de aprendizaje(actualización de un peso cada vez) sino en términos de memoria integrada de recalibración pasada (el algoritmo sabe que la dirección anterior de ese peso era, digamos, correcta , y procederá directamente hacia esta dirección durante la próxima propagación).

Como puede verse, si se agrega el hiperparámetro de impulso, la fase descendente es más rápida, ya que el modelo conserva los rastros de las direcciones de gradiente pasadas.





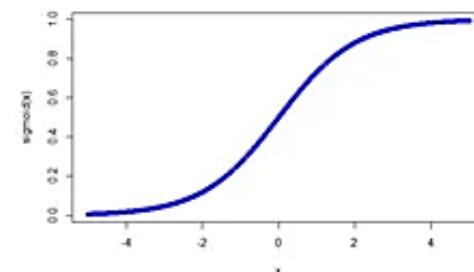
# Parámetros e hiperparámetros de una red.

## ■ Tipo de Hiperparámetros :

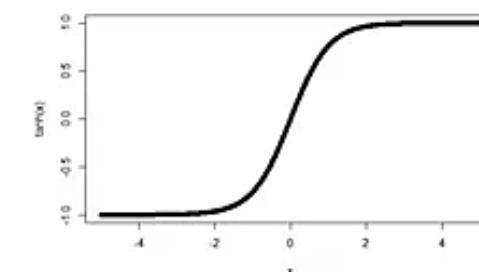
- **Función de activación** : es la función a través de la cual se pasa la suma ponderada, para tener una salida significativa, es decir, como un vector de probabilidad o una salida 0–1. Las principales funciones de activación son:

- **Tanh**
- **RELU**
- **Sigmoid** (para clasificación multiclase se utiliza una variante de esta función, llamada función SoftMax, la cual devuelve como salida un vector de probabilidad cuya suma es igual a uno).

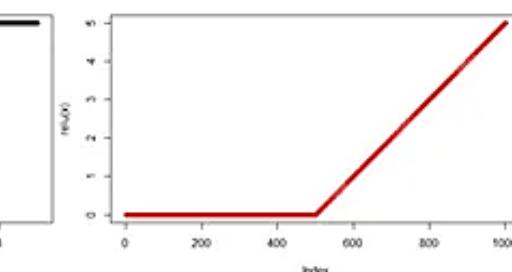
La función de activación se puede ubicar en cualquier punto de la NN, tantas veces como desee. Sin embargo, siempre hay que pensar en la eficiencia y la velocidad. Es decir, la función ReLU es muy rápida en términos de entrenamiento, mientras que la Sigmoid es más compleja y lleva más tiempo. Por lo tanto, una buena práctica podría ser usar ReLU para capas ocultas y luego, en la última capa, insertar su Sigmoid.



$$S(x) = \frac{1}{1 + e^{-x}}$$



$$T(x) = \frac{e^{2x} + 1}{e^{2x} - 1}$$



$$R(x) = \begin{cases} 0 & \text{if } x < 0 \\ x & \text{if } x \geq 0 \end{cases}$$



# Parámetros e hiperparámetros de una red.

- **Tipo de Hiperparámetros :**
  - **Tamaño de minibatch**: ya que resulta ineficiente y contraproducente alimentar una ANN con miles de millones de datos, mejor se alimenta con muestras más pequeñas, llamadas lotes. Al hacerlo, cada vez que el algoritmo se entrena a sí mismo, se entrenará en una muestra del mismo tamaño que el lote. El tamaño típico es 32 o más, sin embargo, debe tener en cuenta que, si el tamaño es demasiado grande, el riesgo es un modelo demasiado generalizado que no se ajustará bien a los nuevos datos.
- **Épocas** : representa el numero de veces que se desea entrenar el algoritmo en todo su conjunto de datos. Las épocas son diferentes de las iteraciones (iteraciones son la cantidad de lotes necesarios para completar una época).

La cantidad de épocas depende del tipo de datos y la tarea a la que se enfrenta. Durante el entrenamiento, puede imponerse condiciones para que las épocas se detengan cuando el error sea cercano a cero, o comenzar con un número relativamente bajo de épocas y luego aumentarlo progresivamente, rastreando algunas métricas de evaluación (como la precisión).



# Parámetros e hiperparámetros de una red.

- **Tipo de Hiperparámetros :**
- **Dropout:** esta técnica consiste en eliminar algunos nodos para que la NN no sea demasiado pesada. Esto se puede implementar durante la fase de entrenamiento. La idea es eliminar algunos nodos que pueden ser redundantes e inútiles. Entonces, mientras se construye el algoritmo, se puede decidir mantener, para cada etapa de entrenamiento, cada nodo con probabilidad  $p$  (llamada 'probabilidad de mantener') o eliminarlo con probabilidad  $1-p$  (llamada 'probabilidad de caída').

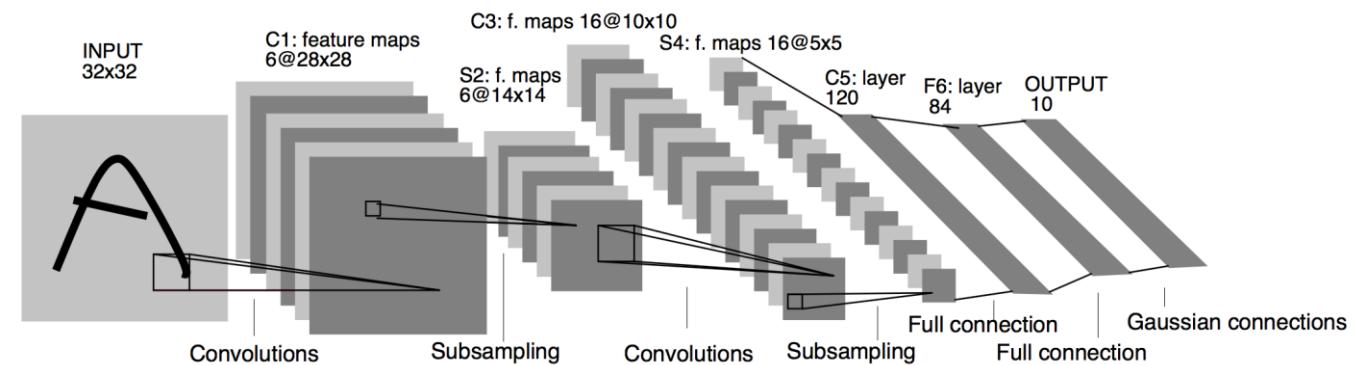


# REDES NEURONALES CONVOLUCIONALES-CNNs

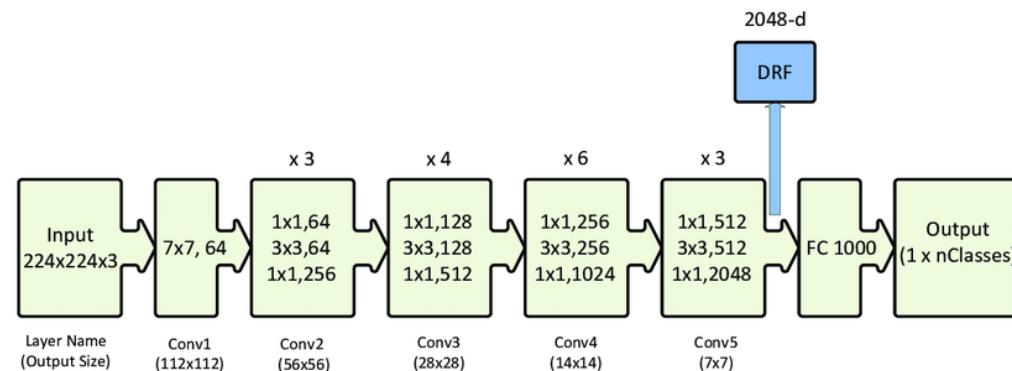
## Arquitectura de Redes Convolucionales

- LeNet-5
- AlexNet
- VGG
- Resnet
- Inception

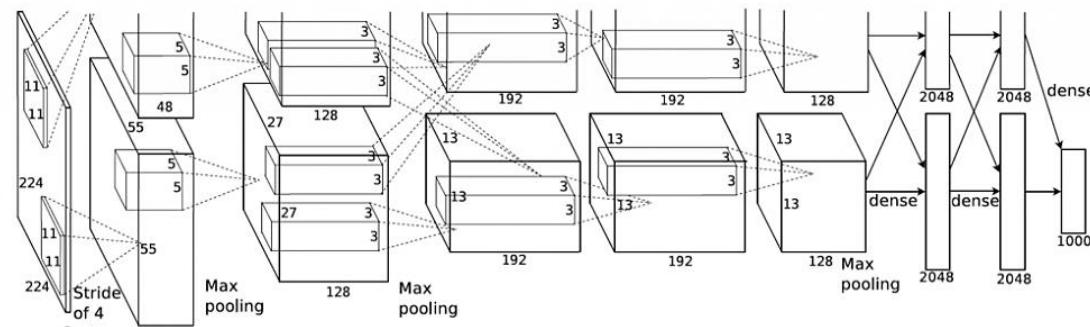
.....



LeNet-5 (60k parámetros) : clasificación automática de dígitos y procesamiento de imágenes



ResNet : Procesamiento de imágenes



AlexNet (60M parámetros):procesamiento de imágenes

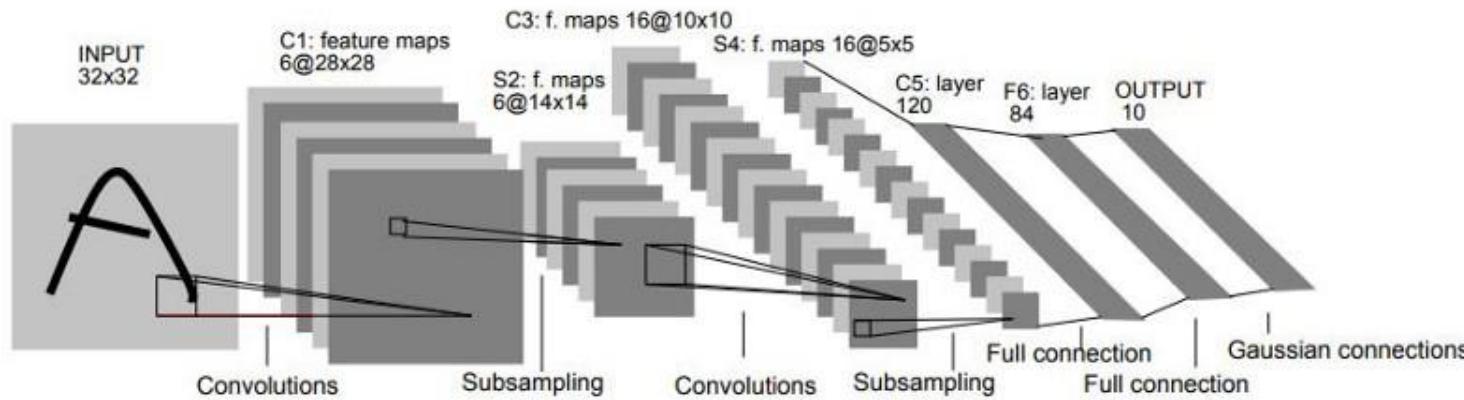
PAPERS DE DIFERENTES DNNS

<https://paperswithcode.com/methods/category/convolutional-neural-networks>



# REDES NEURONALES CONVOLUCIONALES-CNNs

- LeNet-5: (Yann LeCun , Leon Bottou , Yoshua Bengio y Patrick Haffner- 1998). Consta de dos conjuntos de capas de agrupación convolucional y promedio, seguidas de una capa convolucional de aplanamiento, luego dos capas completamente conectadas y finalmente un clasificador softmax.

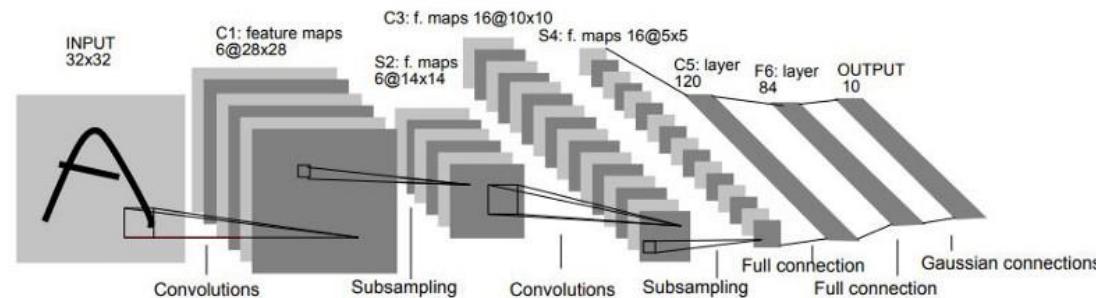


Layer		Feature Map	Size	Kernel Size	Stride	Activation
Input	Image	1	32x32	-	-	-
1	Convolution	6	28x28	5x5	1	tanh
2	Average Pooling	6	14x14	2x2	2	tanh
3	Convolution	16	10x10	5x5	1	tanh
4	Average Pooling	16	5x5	2x2	2	tanh
5	Convolution	120	1x1	5x5	1	tanh
6	FC	-	84	-	-	tanh
Output	FC	-	10	-	-	softmax



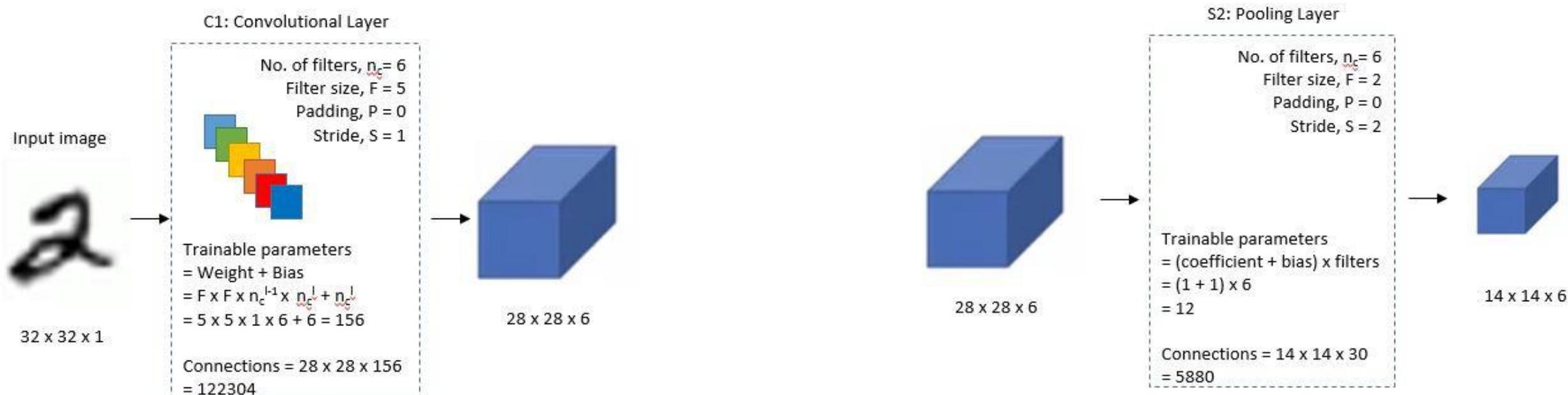
# REDES NEURONALES CONVOLUCIONALES-CNNs

- LeNet-5:



**Primera capa:** La entrada es una imagen en escala de grises de  $32 \times 32$  que pasa a través de la primera capa convolucional con 6 mapas de características o filtros que tienen un tamaño de  $5 \times 5$  y un paso de uno. Las dimensiones de la imagen cambian de  $32 \times 32 \times 1$  a  $28 \times 28 \times 6$ .

**Segunda capa:** Luego, se aplica una capa de agrupación promedio o una capa de submuestreo con un tamaño de filtro de  $2 \times 2$  y un paso de dos. Las dimensiones de la imagen resultante se reducirán a  $14 \times 14 \times 6$ .





# REDES NEURONALES CONVOLUCIONALES-CNNs

## • LeNet-5

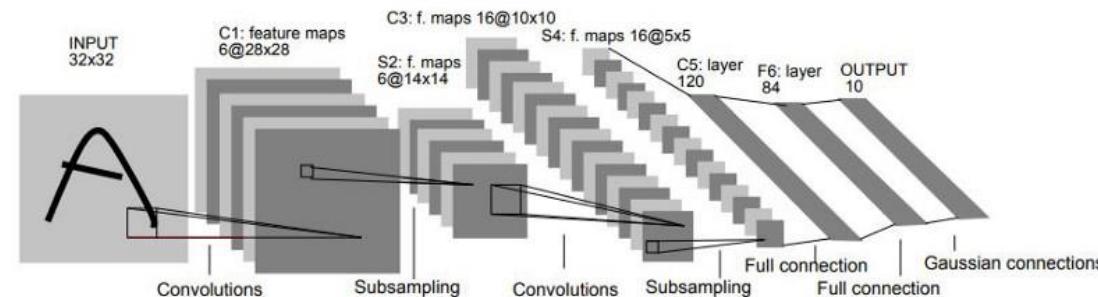
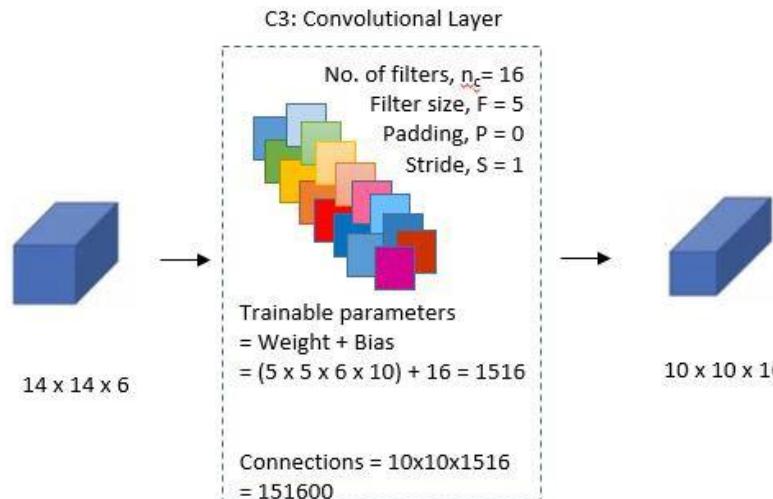
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	X			X	X	X			X	X	X	X		X	X	
1	X	X			X	X	X			X	X	X	X		X	
2	X	X	X			X	X	X		X		X	X	X		
3		X	X	X		X	X	X	X		X	X	X		X	
4		X	X	X		X	X	X	X		X	X		X		
5		X	X	X		X	X	X	X		X	X	X		X	

TABLE I

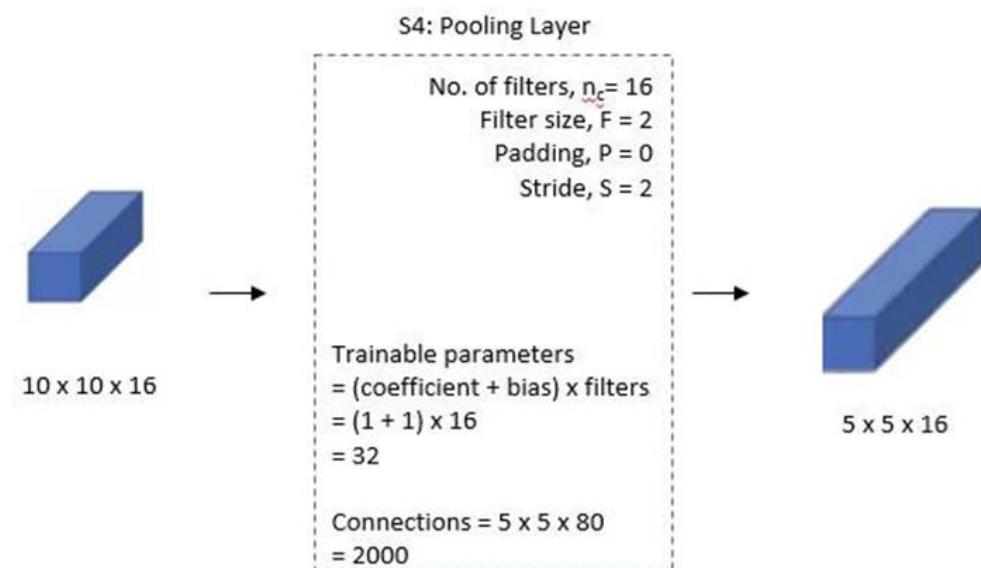
EACH COLUMN INDICATES WHICH FEATURE MAP IN S2 ARE COMBINED BY THE UNITS IN A PARTICULAR FEATURE MAP OF C3.

**Tercera capa:** Es una segunda capa convolucional con 16 mapas de características que tienen un tamaño de  $5 \times 5$  y un paso de 1. En esta capa, solo 10 de los 16 mapas de características están conectados a 6 mapas de características de la capa anterior ( Ver tabla).

La razón principal es romper la simetría en la red y mantener el número de conexiones dentro de límites razonables. Es por eso que la cantidad de parámetros de entrenamiento en estas capas es 1516 en lugar de 2400 y, de manera similar, la cantidad de conexiones es 151600 en lugar de 240000.



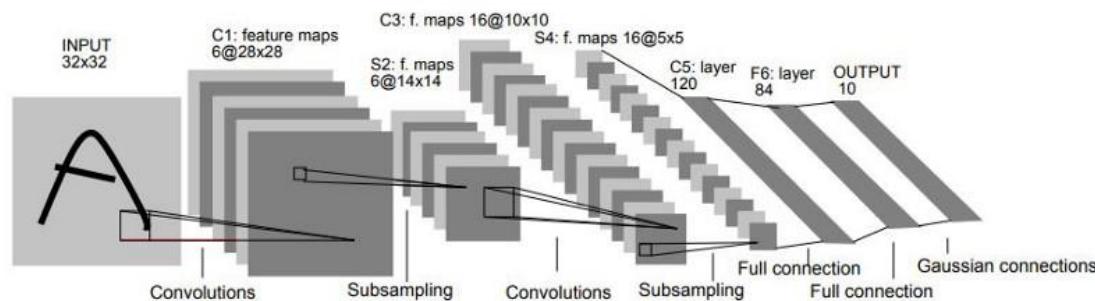
**Cuarta capa:** La cuarta capa (S4) es nuevamente una capa de agrupación promedio con un tamaño de filtro de  $2 \times 2$  y un paso de 2. Esta capa es igual que la segunda capa (S2), excepto que tiene 16 mapas de características, por lo que la salida se reducirá a  $5 \times 5 \times 16$ .





# REDES NEURONALES CONVOLUCIONALES-CNNs

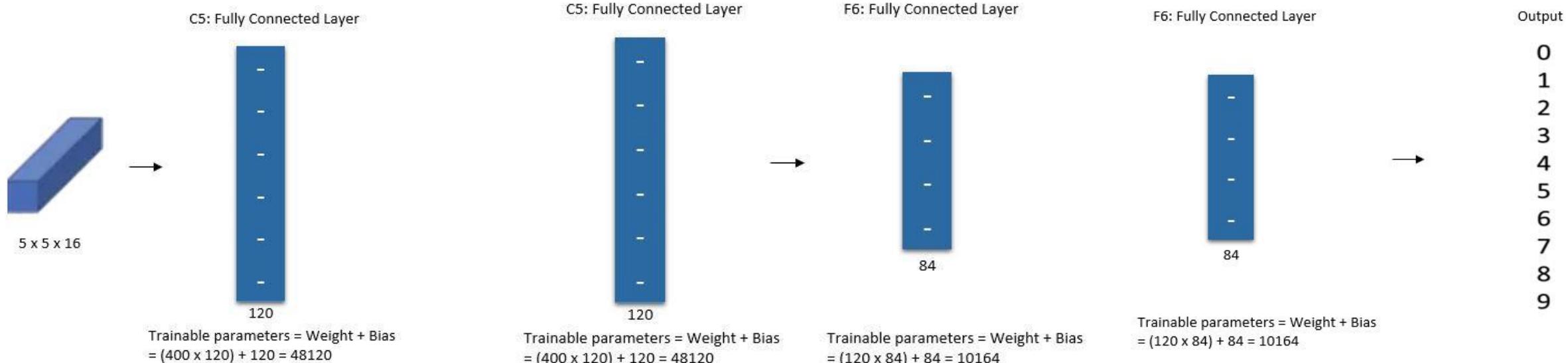
## • LeNet-5



**Quinta capa:** La quinta capa (C5) es una capa convolucional completamente conectada con 120 mapas de características, cada uno de tamaño  $1 \times 1$ . Cada una de las 120 unidades en C5 está conectada a todos los 400 nodos ( $5 \times 5 \times 16$ ) en la cuarta capa S4.

**Sexta Capa:** La sexta capa es una capa completamente conectada (F6) con 84 unidades.

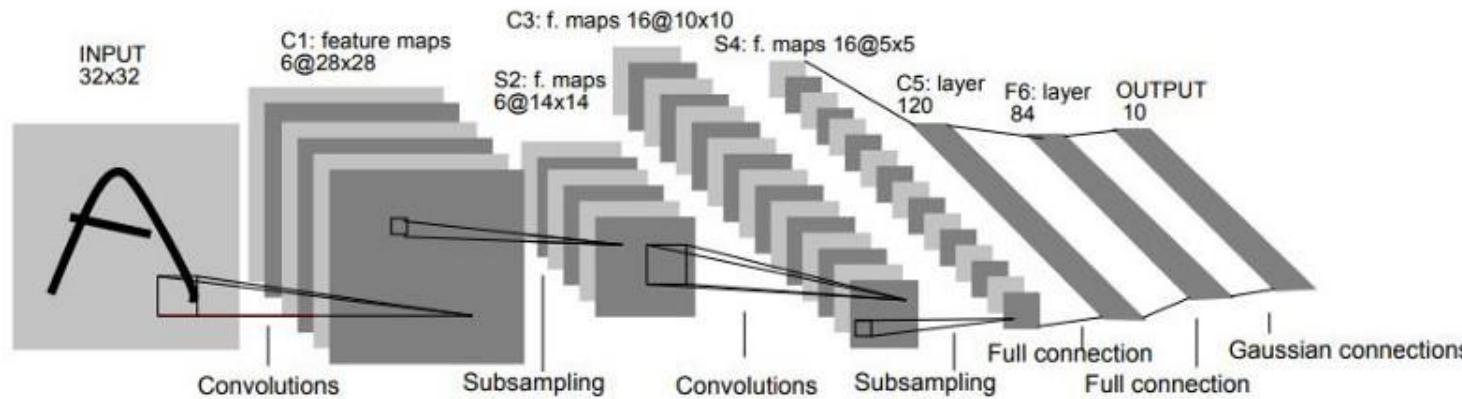
**Capa de salida:** Finalmente, hay una capa de salida softmax totalmente conectada con 10 valores posibles correspondientes a los dígitos del 0 al 9.



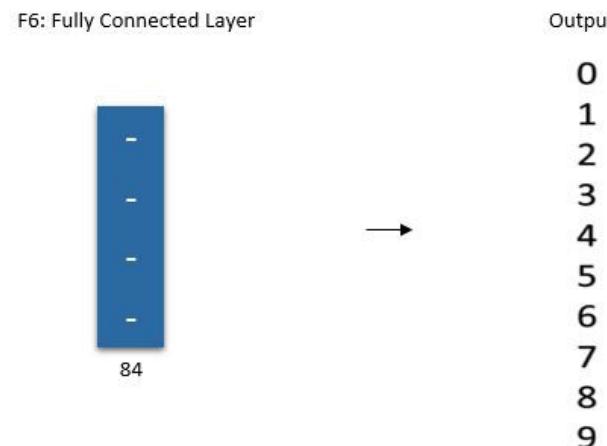


# REDES NEURONALES CONVOLUCIONALES-CNNs

- LeNet-5: por Yann LeCun , Leon Bottou , Yoshua Bengio y Patrick Haffner- 1998



**Capa de salida:** Finalmente, hay una capa de salida softmax y totalmente conectada con 10 valores posibles correspondientes a los dígitos del 0 al 9.





# REDES NEURONALES CONVOLUCIONALES-CNNs

- AlexNet

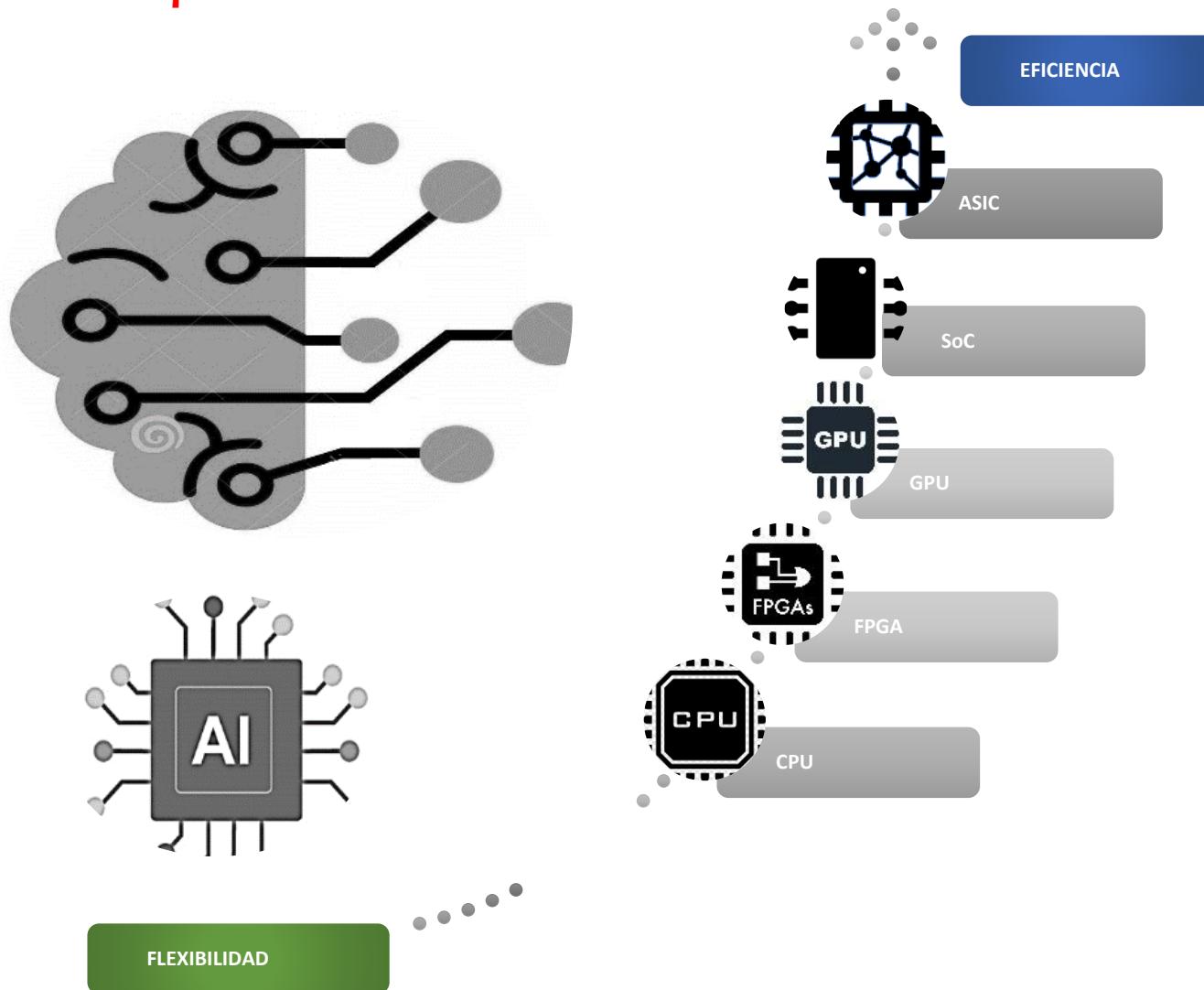
Layer	# filters / neurons	Filter size	Stride	Padding	Size of feature map	Activation function
Input	-	-	-	-	227 x 227 x 3	-
Conv 1	96	11 x 11	4	-	55 x 55 x 96	ReLU
Max Pool 1	-	3 x 3	2	-	27 x 27 x 96	-
Conv 2	256	5 x 5	1	2	27 x 27 x 256	ReLU
Max Pool 2	-	3 x 3	2	-	13 x 13 x 256	-
Conv 3	384	3 x 3	1	1	13 x 13 x 384	ReLU
Conv 4	384	3 x 3	1	1	13 x 13 x 384	ReLU
Conv 5	256	3 x 3	1	1	13 x 13 x 256	ReLU
Max Pool 3	-	3 x 3	2	-	6 x 6 x 256	-
Dropout 1	rate = 0.5	-	-	-	6 x 6 x 256	-
Fully Connected 1	-	-	-	-	4096	ReLU
Dropout 2	rate = 0.5	-	-	-	4096	-
Fully Connected 2	-	-	-	-	4096	ReLU
Fully Connected 3	-	-	-	-	1000	Softmax

Number of Parameters = 62.3 million



# REDES NEURONALES CONVOLUCIONALES-CNNs

Plataformas en hardware para Redes Convolucionales





# REDES NEURONALES CONVOLUCIONALES-CNNs

## Videos

<https://www.youtube.com/watch?v=-RpNI4ZrfIM>

edge tpu performance demo

<https://www.youtube.com/watch?v=lwM2epg6Ryw>

¿Qué es el GRADIENTE DESCENDENTE?

<https://www.youtube.com/watch?v=IKloEocn3Hw>

Clasificación de imágenes con REDES CONVOLUCIONALES en Python (tutorial)

<https://www.youtube.com/watch?v=SGoxsBgp3WM>

MIT 6.S191 (2020): Convolutional Neural Networks

<https://www.youtube.com/watch?v=iaSUYvmCekI>

How computers learn to recognize objects instantly | Joseph Redmon

<https://www.youtube.com/watch?v=Cgxsv1riJhl>



# SIMULACIÓN MONTECARLO

**MÉTODO DE MONTECARLO (1944):** Llamado así como alusión al Casino de Montecarlo (Mónaco) y los juegos de azar. La invención del método de Montecarlo se asigna a Stanislaw Ulam y a John Von Neumann, cuando se trabajaba en el proyecto Manhattan (desarrollo de la bomba atómica durante la Segunda Guerra Mundial en el Laboratorio Nacional de Los Álamos en EE. UU, basado en el comportamiento aleatorio de difusión de los neutrones).

Es un método no determinista o estadístico numérico, usado para aproximar expresiones matemáticas complejas y costosas de evaluar con exactitud. El método se mejoró a lo largo del desarrollo de la computación.



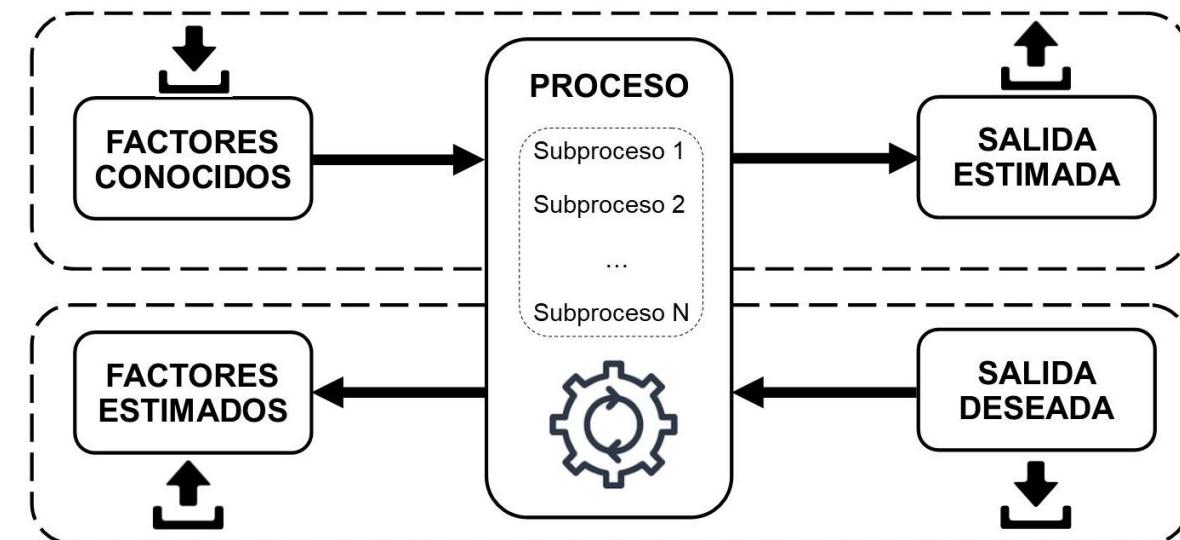


# SIMULACIÓN MONTECARLO

El método de Montecarlo proporciona soluciones aproximadas a una gran variedad de problemas matemáticos posibilitando la realización de experimentos con muestreos de números pseudoaleatorios en una computadora. El método es aplicable a cualquier tipo de problema, ya sea estocástico o determinista.

La clave de este método está en entender el término 'simulación'. Realizar una simulación consiste en repetir, o duplicar, las características y comportamientos de un sistema real. Así pues, el objetivo principal de la simulación de Montecarlo es intentar imitar el comportamiento de variables reales para, en la medida de lo posible, analizar o predecir cómo van a evolucionar.

**Simulación de Monte Carlo:** Técnica matemática que usa números aleatorios y probabilidad para entender el impacto del riesgo en un modelo de realidad. El modelo de simulación ayuda a entender como perturbaciones aleatorias se propagan al resto del modelo.



Requiere de herramientas computacionales como Excel, Python, R, entre otros.



# SIMULACIÓN MONTECARLO

## Elementos de una simulación:

1. Construcción de un modelo a simular ( es el elemento mas importante a considerar)
2. Identificar Insumos (Variables Aleatorias) y salidas dentro del modelo.
3. Configurar y correr la simulación (iteraciones, estadísticos , gráficos, etc.)
4. Analizar y tomar una decisión basados en los resultados.

Los insumos y salidas tiene relación con tres variables:

- Variables de decisión: sobre las que se tiene pleno control
- Variables aleatorias: Sobre las que no se tiene ningún control. Se requiere poder describir su comportamiento (función de distribución).
- Variables de salida o resultados: aquellas que dependen de funciones que involucran otras variables.



# SIMULACIÓN MONTECARLO

## Principales ventajas

La simulación de Monte Carlo tiene muchas ventajas respecto a otro tipo de análisis deterministas o de “estimación de un solo punto”. Entre ellos podemos destacar:

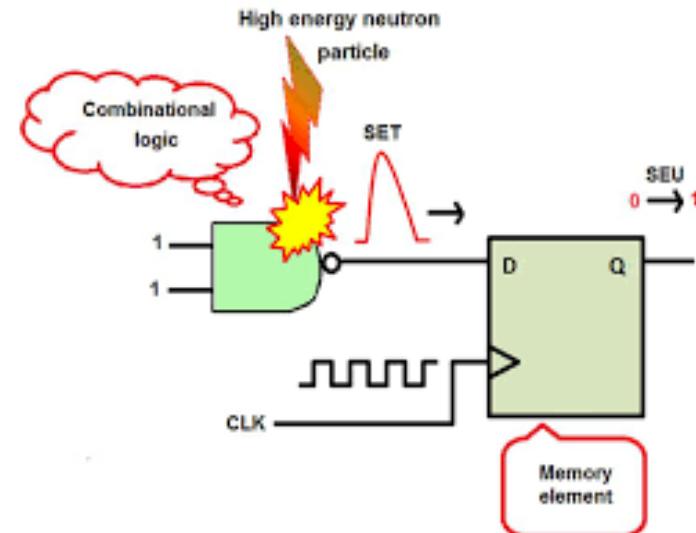
- **Ofrece resultados gráficos.** Gracias a los datos que genera una simulación Monte Carlo, es fácil crear gráficos de diferentes resultados y las posibilidades de que sucedan. Esto es importante para comunicar los resultados a otras personas interesadas.
- **Análisis de sensibilidad.** En este método resulta más fácil ver qué variables introducidas tienen mayor influencia sobre los resultados finales.
- **El análisis de escenario.** Usando la simulación Monte Carlo, los analistas pueden ver exactamente los valores que tienen cada variable cuando se producen ciertos resultados. Esto resulta muy valioso para profundizar en los análisis.
- **Correlación de variables de entrada.** También permite modelar relaciones interdependientes entre diferentes variables de entrada. Esto es importante para averiguar con precisión la razón real por la que, cuando algunos factores suben, otros suben o bajan paralelamente.



# SIMULACIÓN MONTECARLO

## Ejemplos:

1. Dispositivos electrónicos en el espacio y la probabilidad de impacto de partículas de alto nivel de energía.



Una alteración de evento único ( SEU ) es un cambio de estado causado por una sola partícula ionizante (iones, electrones, fotones ...) que golpea un nodo sensible en un dispositivo microelectrónico, como en un microporcesador , memoria semiconductora o transistores de potencia . El cambio de estado es el resultado de la carga gratuita creada por la ionización en o cerca de un nodo importante de un elemento lógico (por ejemplo, "bit" de memoria).

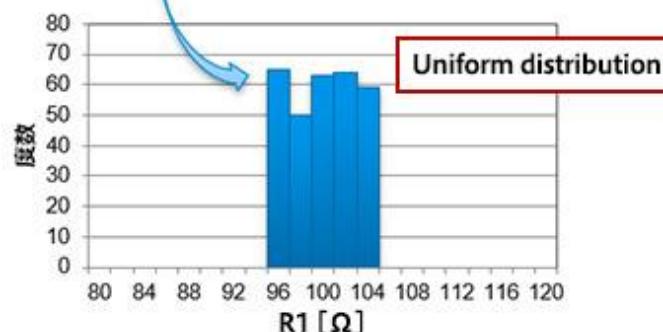
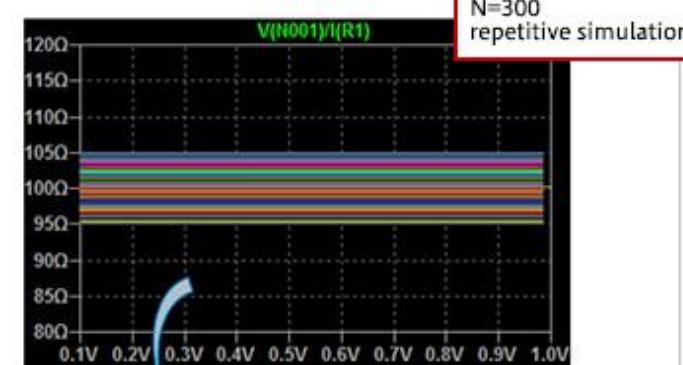
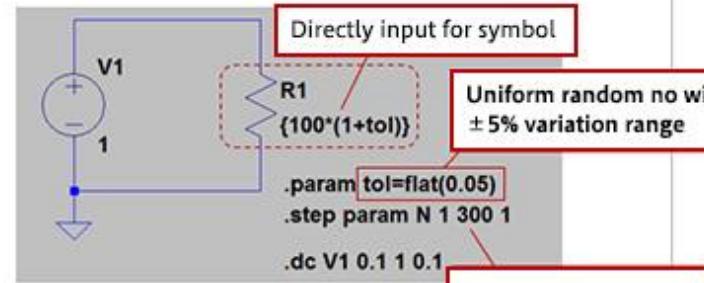
# SIMULACIÓN MONTECARLO

## Ejemplos:

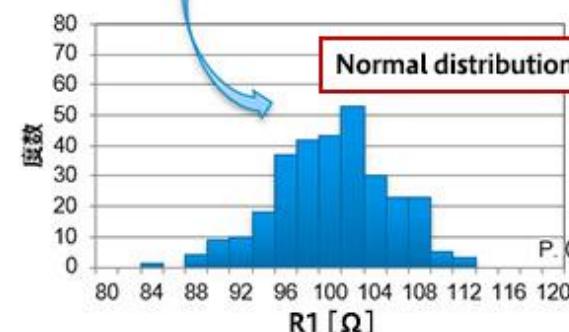
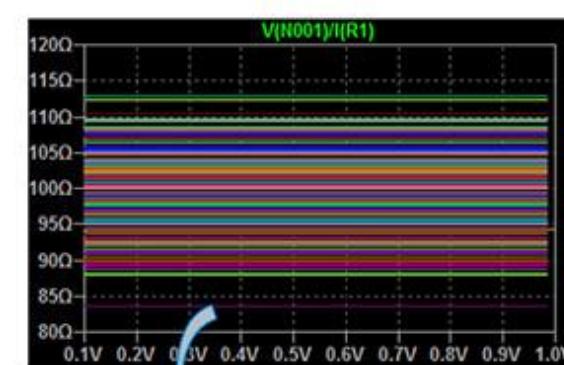
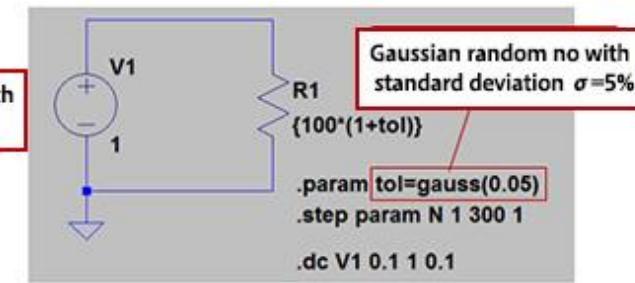
### 2. Simulación de un circuito eléctrico resistivo

#### ■ When a variation can be directly described for a symbol

① Example when uniform random no was used



② Example when Gaussian random no was used



# SIMULACIÓN MONTECARLO

Otros ejemplos:

Juego de dados

[https://www.youtube.com/watch?v=\\_5VAEHABKcY](https://www.youtube.com/watch?v=_5VAEHABKcY)

Calculo del numero pi

<https://www.youtube.com/watch?v=WJjDr67frtM>





# INFOGRAFÍA

- <https://www.diegocalvo.es/red-neuronal-convolucional/>
- [https://www.sas.com/es\\_co/insights/analytics/deep-learning.html#:~:text=El%20deep%20learning%20es%20un,de%20im%C3%A1genes%20o%20hacer%20predicciones.](https://www.sas.com/es_co/insights/analytics/deep-learning.html#:~:text=El%20deep%20learning%20es%20un,de%20im%C3%A1genes%20o%20hacer%20predicciones.)
- <https://medium.com/analytics-vidhya/convolution-padding-stride-and-pooling-in-cnn-13dc1f3ada26>
- <https://datascience.eu/es/programacion/un-recorrido-por-alexnet/>
- <https://medium.com/@pechyonkin/key-deep-learning-architectures-lenet-5-6fc3c59e6f4>
- <https://economipedia.com/definiciones/simulacion-de-montecarlo.html>
- <https://www.fundacionctic.org/es/actualidad/simulacion-de-procesos-mediante-el-metodo-de-montecarlo>
- <https://www.ealde.es/metodo-simulacion-monte-carlo/>
- <https://economipedia.com/definiciones/simulacion-de-montecarlo.html>
- Redes Neuronales Artificiales Claudio Javier Tablada – German Ariel Torres

# FIN!!!

## PREGUNTAS ???

