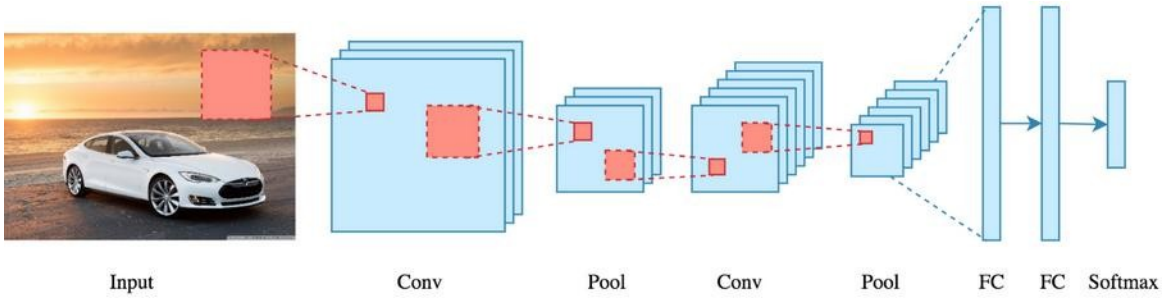
Redes Convolucionales

Índice de contenidos

1. Introducción

Las redes neuronales convolucionales (RNC) son un algoritmo de Deep Learning que está **diseñado para trabajar con imágenes**, tomando estas como input, asignándole importancias (pesos) a ciertos elementos en la imagen para así poder diferenciar unos de otros.

Las RNC son muy recientes (1998) y se basan en descubrimientos sobre el funcionamiento del cerebro de los mamíferos (concretamente gatos).

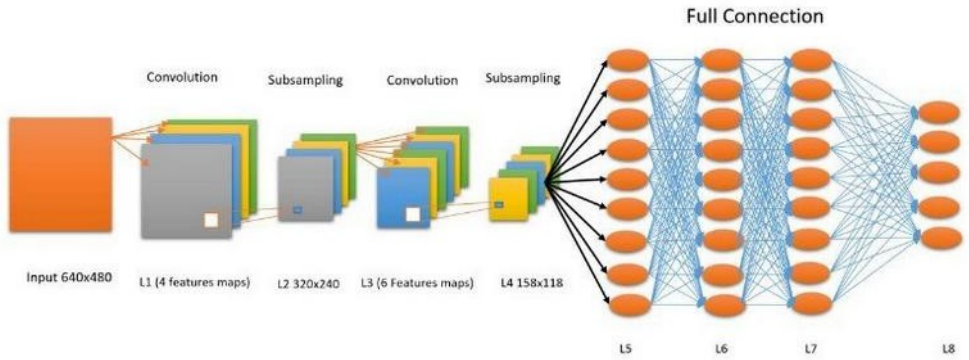


1. Introducción

Las RNC han contribuído definitivamente al desarrollo y perfeccionamiento del campo de visión de los ordenadores (reconomiento de caras, objetos, estructuras, coches autónomos, …).

Las redes convolucionales contienen varias *hidden layers.*

- Cada capa se especializa en una tarea, las primeras pueden detectar líneas, curvas y así se van especializando hasta llegar a capas más profundas que reconocen formas complejas como rostros, siluetas, edificios,... dentro de un contexto complejo.



1. Introducción

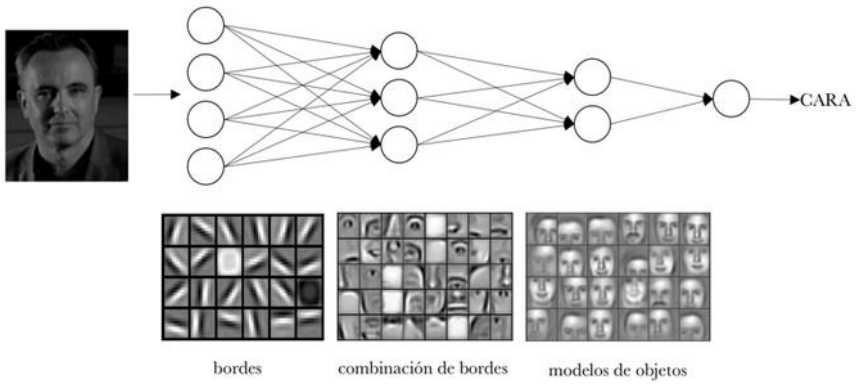
- Las RNC se basan en las RNA.
- Su aprendizaje es **supervisado**.
- Necesitaremos por tanto de imagenes para entrenar y testear (gran cantidad).
- **La red debe ser capaz de aprender no sólo los objetos, sino sus características asociadas cuando cambian aspectos como el ángulo, luz, sombras, intesidad del color, fondo, …**

- Y ésto, no es trivial. No es tan fácil disponer de conjuntos de datos que incluyan todas las características posibles para entrenar.

1. Introducción

- Consisten en múltiples capas de filtros convolucionales de una o más dimensiones.
- Después de cada capa, generalmente, se añade una función para realizar un mapeo causal no-lineal.
- Como cualquier red empleada para clasificación, al principio estas redes tienen una **fase de extracción de características**, compuesta de neuronas convolucionales, luego hay una **reducción por muestreo** y al final tendremos **neuronas de perceptrón** mas sencillas para **realizar la clasificación final** sobre las características extraídas.

- A medida que profundizamos en la red, **disminuimos la dimensionalidad** de los datos → Ésto ayuda a que las neuronas de las capas interiores sean **menos sensibles a pertubaciones** en los datos de entr por características complejas.



2. Arquitectura básica

- **Entrada:**
  Serán los pixeles de la imagen. Alto, ancho y profundidad para imágenes de 1 sólo color ó x3 para los canales rojo, verde y azul en imágenes a color.

- **Capa De Convolución:**
  Procesará la salida de neuronas que están conectadas en «regiones locales» de entrada (es decir pixeles cercanos), calculando el producto escalar entre sus pesos (valor de pixel) y una pequeña región a la que están conectados en el volumen de entrada. Aquí usaremos por ejemplo 32 filtros o la cantidad que decidamos y ése será el volumen de salida.

- **Capa RELU**
  Aplicará la función de activación ReLU en los elementos de la matriz.

- **MUESTREO ó SUBSAMPLING:**
  Hará una reducción en las dimensiones alto y ancho, pero mantiene la profundidad.

- **CAPA «TRADICIONAL»**
  Red de neuronas feedforward que conectará con la última capa de subsampling y finalizará con la cantidad de neuronas que queremos clasificar.

2. Arquitectura básica

- **Capas densamente conectadas vs. Capas Convolucionales**

Las capas densamente conectadas, aprenden patrones globales, mientras que las convolucionales aprenden patrones locales en pequeñas ventanas de 2 dimensiones.

El propósito de las capas convolucionales es detectar características y rasgos visuales como aristas, líneas, etc.

De esta manera cuando aprende esta característica en un punto concreto de la imagen, la puede reconocer en cualquier parte de la misma. Mientras que una RNA tiene que aprender el patrón nuevamente SI éste aparece en una nueva localización de la imagen.

2. Arquitectura básica

- **Capas Convolucionales**

Operan sobre tensores 3D, llamados FEATURE MAPS de 3 dimensiones (altura, anchura y profundidad).

La profundidad está representada por los canales de color (rojo, azul, verde ó gris)

2. Arquitectura básica

- **Filtros o Kernels**
  - Son matrices de NxN con valores iniciales aleatorios que se van ajustando a través de Backpropagation.
  - Podemos verlas como ventanas que nos permiten reducir la dimensionalidad de la imagen original.
  - Si la imagen original tiene unas dimensiones de 28x28 y le aplicamos un kernel de 5x5, obtendremos como resultado un Feature Map de 24x24 → 28-5+1 = 24. Sólo podemos mover la ventana de 5 píxeles a lo largo de 23 píxeles en cada eje (en el caso de aplicar el kernel en pasos de 1 pixel tanto horizontal, como verticalmente).

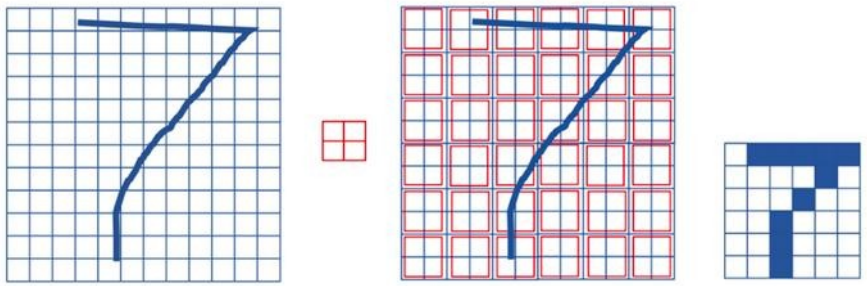
2. Arquitectura básica

- **Pooling (Muestreo o subsampling)**

  - Permite reducir la dimensionalidad de los datos a la salida obtenida de una capa convolucional.
  - **Básicamente simplifican la información recogida por la capa convolucional y crean una versión condensada de la información de dicha capa.**

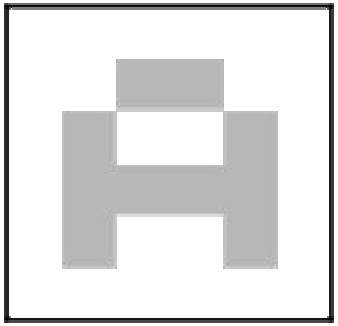
  - En el caso de que nuestra convolución nos devuelva feature mappings de 24x24, tras aplicar Pooling (Max Pooling 2x2) tendríamos una salida de 12x12 → 24/2 filas x 24/2 columnas.

  - Este tipo de transformación, mantiene la estructura básica de la imagen



3. Funcionamiento

- Supongamos la siguiente imagen:



- Podemos descomponerla en una matriz de píxeles. La imágenes habitualmente incluyen hasta 255 niveles de intensidad.

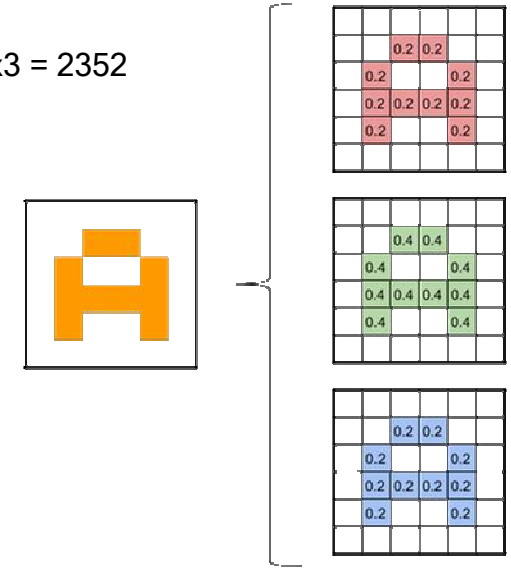
- Al igual que en las RNA y en los modelos de ML, es conveniente normalizar los datos (0,1). Esta imagen puede convertirse entonces en la siguiente matriz:

- De esta manera nuestra imagen estaría preprocesada.

		0.6	0.6		
	0.6			0.6	
	0.6	0.6	0.6	0.6	
	0.6			0.6	

3. Funcionamiento

- Píxeles = neuronas
  - Si tenemos una imagen de 28x28 píxeles = 768 píxeles = 784 neuronas
    Pero no es tan fácil. Si la imagen es en color, debemos añadir 3 canales (Rojo, Verde, Azul)
  - Por tanto una simple imagen de 28x28 en color implica manejar 28x28x3 = 2352 neuronas de entrada
    Cada neurona se encarga de procesar un pixel.
  - Habríamos definido nuestra capa de entrada, con la imagen preprocesada. El paso de preprocesamiento es muy importante.
  - El preprocesado incluye también al normalización de las imágenes en dimensiones.



3. Funcionamiento

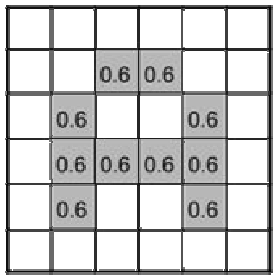


Imagen de entrada

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

kernel

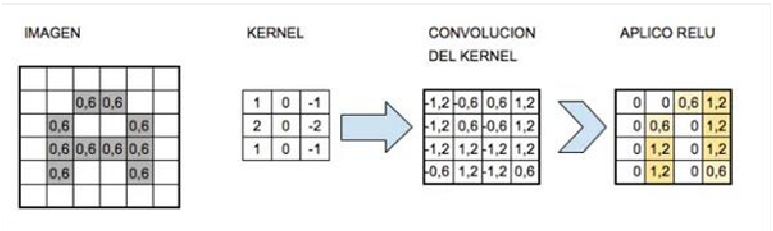
- Convoluciones

  - Es donde comienza la "magia"
  - El proceso consiste en tomar grupos de píxeles cercanos de la imagen de entrada e ir operando matemáticamente (producto escalar) contra una pequeña matriz llamada **Kernel.**
  - Ver GiFs incluídos en el material.
  - El Kernel se aplica a la imagen de entrada y genera una matriz de salida.
  - Esa matriz de salida, se convierte en nuestra **capa de neuronas ocultas** (la primera).
  - En el caso de imágenes en color, tendríamos 3 kernels (uno por canal). Esos filtros se suman y conformarán una salida (como si sólo hubiera un canal).

3. Funcionamiento

- Conjunto de Kernels
  - No sólo aplicamos un kernel, sino múltiples. Cada kernel toma valores iniciales aleatorios.

    - Supongamos una primera convolución con 32 kernels distintos. Tendríamos 32 matrices de salida cada una de 28x28x1 = 25088 neuronas SÓLO para nuestra primera capa oculta y para una imagen de 28x28.
    - Ahora pensad, lo que hace falta para entrenar una red convolucional con imágenes FullHD (1920x1080) ó 4K (3840x2160).



FINALMENTE OBTENGO UN MAPA DE DETECCIÓN DE CARACTERÍSTICAS

3. Funcionamiento

- Función de activación.
  - Funciona de manera similar a la función de activación en RNA.
  - La función de activación más habitual en RNC es la ReLU (Rectifier Linear Unit)

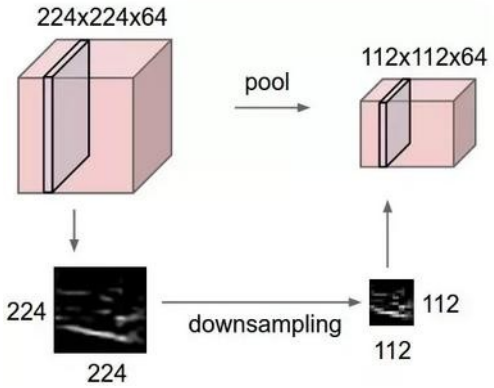
Características de la función ReLU:

- Activación Sparse – solo se activa si son
- positivos. No está acotada.
- Se pueden morir demasiadas neuronas. Se
- comporta bien con imágenes.
- Buen desempeño en redes convolucionales.

$$f(x) = \max(0, x) = \begin{cases} 0 \; for \; x < 0 \\ x \; for \; x \geq 0 \end{cases}$$

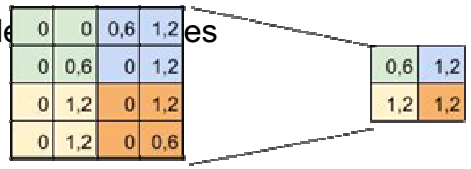
3. Funcionamiento

- Muestreo
  (subsampling)
  - Se aplica para poder mantener nuestra red dentro de unos límites computacionales aceptables.
  - Aun después de haber realizado la convolución el número de neuronas necesarias es muy alto.
  - Por ello se realiza un submuestreo de los datos, quedándonos sólo con las características más importantes que ha detectado cada filtro.
  - El muestreo más habitual es el Max-Pooling



3. Funcionamiento

- Muestreo Max-Pooling
  - Recorreremos cada una de las 32 imágenes obtenidas a través de los kernels y tras aplicarles ReLU, se les aplicá Max-Pooling.
  - Un Max-Pooling de 2x2, nos permite por ejemplo reducir el tamaño de las imágenes a la mitad (por cada 2 píxeles originales, me quedo con 1).
  - Cogeríamos (en este caso) matrices de 2x2 en las imágenes ReLU y nos quedamos con el valor máximo.
  - Pasamos de 32 imágenes de 28x28 (**25088 neuronas**) a 32 de 12x12 (**4608**
  - **neuronas**).
    Esta red, todavía conserva las características más importantes de es (patrones como líneas o curvas).



SUBSAMPLING:
Aplico Max-Pooling de 2x2
y reduzco mi salida a la mitad

3. Funcionamiento

PRIMERA CONVOLUCIÓN

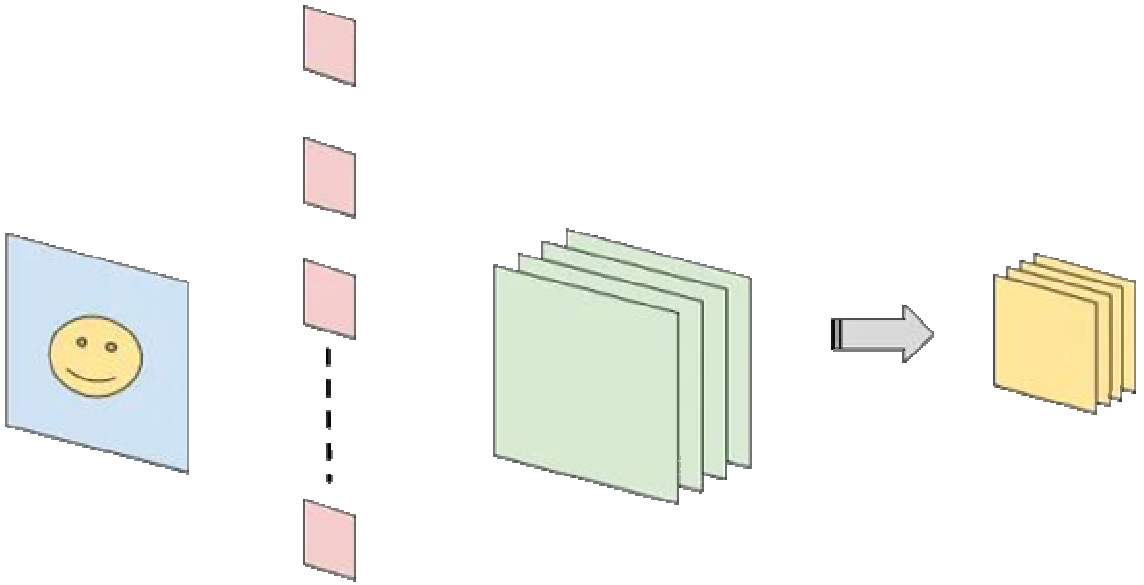


Imagen de entrada de 28x28

Aplico 32 kernels de 3x3 y Función de Activación ReLu

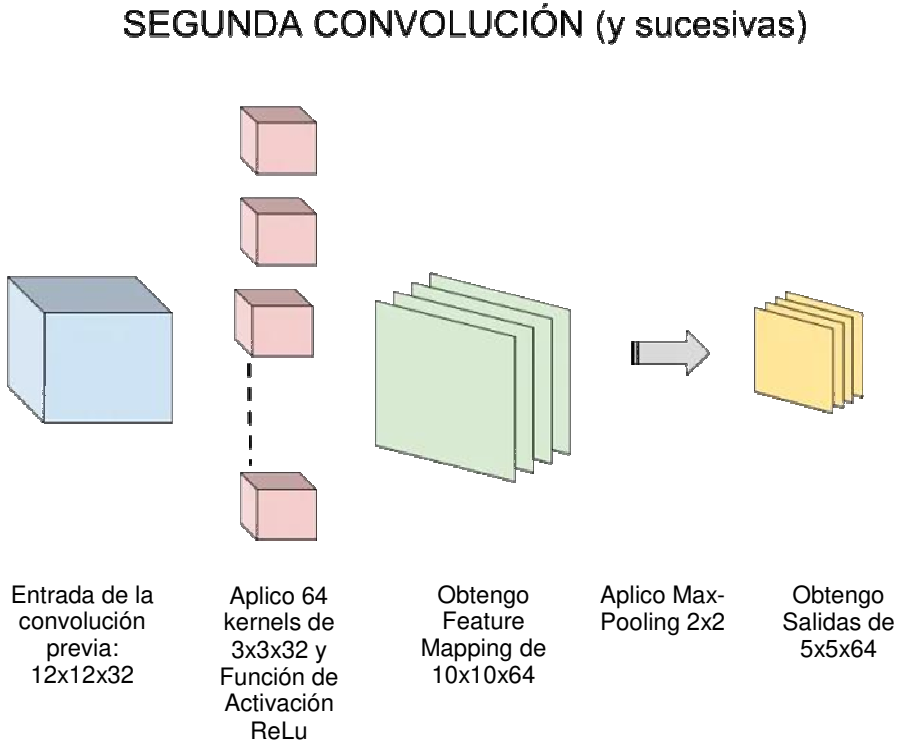
Obtengo 32 Feature Mapping de 24x24x1

Aplico Max-Pooling 2x2

Obtengo 32 salidas de 12x12x1

3. Funcionamiento

- Segunda convolución
  - Partimos de 12x12x32 = **4608**
  - **neuronas** Aplicamos 64 kernels de 3x3 y obtenemos un feature mapping de **6400 neuronas**
  - Aplicamos el muestreo Max-Pooling de 2x2 → 5x5x64 = **1600 neuronas**
  - La salida de la segunda convolución nos devuelve resultados de 5x5 píxeles
  - Estamos consiguiendo el objetivo de simplificar la complejidad del problema a resolver.



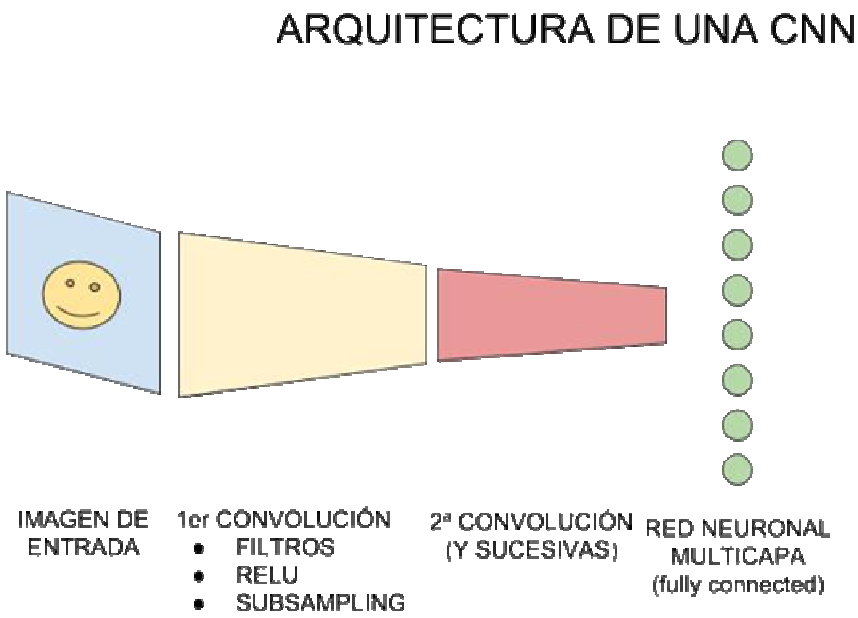
3. Funcionamiento

- Tercera convolución
  - Partimos de 5x5x64 = **1600 neuronas**
  - Aplicamos 128 kernels de 2x2 y obtenemos un feature mapping de **2048 neuronas** (4x4x128)
  - Aplicamos el muestreo Max-Pooling de 2x2 → **1024 neuronas**

  - Aquí habríamos llegado al **última convolución** (el resultado es del mismo tamaño que los kernels).
  - Esta es la última capa oculta que además es una **capa 'tridimensional'**. Tiene unas dimensiones de 2x2x128.
  - **Necesitamos 'aplanarla'** para poder usarla en una RNA típica.

3. Funcionamiento

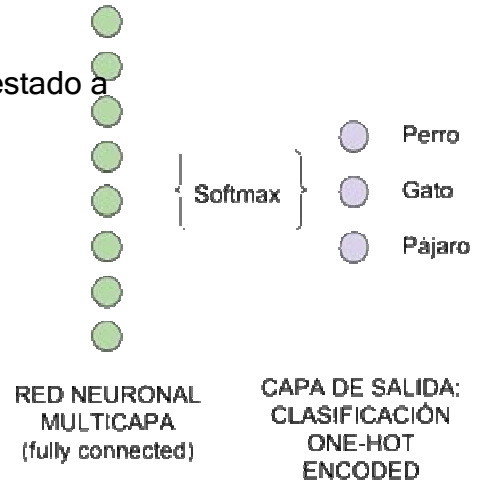
- Objetivo



3. Funcionamiento

- Arquitectura FeedForward

  - A la nueva capa de entrada ‘plana’ le aplicamos la **función de activación SoftMax.**
  - Como salida, tendremos tantas neuronas como clases a clasificar.
  - Si tenemos gatos y perros, necesitaremos 2 neuronas. Si clasificamos, gatos, perros o pájaros, necesitaremos 3, …
  - Las **salidas** del modelo son del tipo **One-Hot-Encoding.**
  - SoftMax se encarga de devolvernos la probabilidad de ocurrencia de cada estado a las neuronas de salida.



4. Backprogationen RNC

- Ajuste de los Kernels

  Los kernels se inicializan de manera aleatoria (de manera similar a los pesos en un RNA).

  La ventaja de este método es que el total de parámetros a ajustar es mucho menor del que tendríamos en una RNA tradicional.

  Si tenemos 32 kernels de 3x3 → 3x3x32 = **288 parámetros.**
  Sin embargo en una RNA de 768 y 6272 neuronas (pej.) tendríamos **4,5 millones de pesos** (todas las neuronas interconectadas con todas)

5. Comparativa RNA y RNC

Características	RNA Multicapa	RNC
Datos de entrada en la Capa Inicial	Las características que analizamos. Por ejemplo: ancho, alto, grosor, etc.	Pixeles de una imagen. Si es color, serán 3 capas para rojo,verde,azul
Capas ocultas	Elegimos una cantidad de neuronas para las capas ocultas.	Tenemos de tipo: * Convolución (con un tamaño de kernel y una cantidad de filtros) * Subsampling
Capa de Salida	La cantidad de neuronas que queremos clasificar. Para estados binarios, 2 neuronas de salida.	Debemos «aplanar» la última convolución con una (ó más) capas de neuronas ocultas «tradicionales» y hacer una salida mediante SoftMax en la capa de salida. Devuelve el resultado en formato One-Hot-Encoding
Aprendizaje	Supervisado	Supervisado
Interconexiones	Entre capas, todas las neuronas de una capa con la siguiente.	Son muchas menos conexiones necesarias, pues realmente los pesos que ajustamos serán los de los filtros/kernels que usamos.
Significado de la cantidad de capas ocultas	Realmente es algo desconocido y no representa algo en sí mismo.	Las capas ocultas son mapas de detección de características de la imagen y tienen jerarquía: primeras capas detectan lineas, luego curvas y formas cada vez más elaboradas.
Backpropagation	Se utiliza para ajustar los pesos de todas las interconexiones de las capas	Se utiliza para ajustar los pesos de los kernels.