

Clase 5 - Redes Neuronales Convolucionales: CNNs



Deep Learning

Episodio 5: Redes Neuronales Convolucionales

Fernando Gama

Escuela de Graduados en Ingeniería Informática y Sistemas, Facultad de Ingeniería, UBA

4 de Agosto de 2022

Escenas del capítulo anterior...



- ▶ A veces, los **datos tienen estructuras** que se pueden explotar para mejorar el algoritmo
 - ⇒ Señales en el tiempo, sistemas para procesarlos
- ▶ Operación de **convolución** ⇒ Operación lineal que **explota la estructura**
- ▶ Respuesta en frecuencia ⇒ Representación alternativa de señales con estructura
 - ⇒ Permite encontrar una **manera rápida y fácil de calcular la convolución**
- ▶ Convolución en **imágenes** ⇒ Conceptualmente igual que la convolución en el tiempo

> Como bien hablamos visto ya, una red neuronal es una sucesión de capas de neuronas, donde cada capa aplica una función lineal, seguido de una función de activación.



- ▶ Las redes neuronales son una cascada de capas \Rightarrow Función lineal seguido de activación

$$\mathbf{x}_\ell = \sigma_\ell(\mathbf{W}_\ell \mathbf{x}_{\ell-1} + \mathbf{b}_\ell)$$

- ▶ Si la dimensión N de los datos es muy grande $\Rightarrow \mathbf{W}_0$ va a ser muy grande
 \Rightarrow Más parámetros \Rightarrow Es más difícil de aprender \Rightarrow Necesito más datos

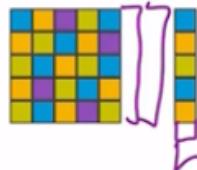


- ▶ Si los datos no tienen estructura, no queda otra que conseguir más datos
 \Rightarrow Pero si tienen estructura, podemos elegir la operación lineal de manera más inteligente

Ahora, esta función lineal, descripta por la matriz w , es lo que yo voy a aprender, y la función de activación se decide por diseño y suele ser fija.

Ahora que pasa? Imaginémonos que la dimensión de los datos de entrada es muy grande. A medida que estos datos van creciendo, yo voy a tener que ir agregando cada vez más columnas a mi matriz, ya que va aumentando la cantidad de datos. Se hace cada vez más difícil de aprender —> más parámetros para aprender —> más grande el espacio que mi algoritmo de optimización va a recorrer —> complica la optimización, necesito más datos, tarda más tiempo

- ▶ Si la dimensión N de los datos es muy grande $\Rightarrow \mathbf{W}_0$ va a ser muy grande
 \Rightarrow Más parámetros \Rightarrow Es más difícil de aprender \Rightarrow Necesito más datos



Si queremos pensar en imágenes, por ejemplo, incluso para mini imágenes como 28x28, eso ya te da un vector de 700 y pico de tamaño. Esto va a hacer que el vector sea cada vez más grande y muy rápido. Si yo quiero procesar datos de gran tamaño, con una red neuronal se me va a hacer cada vez más difícil ya que la matriz no parará de crecer.

Aca es donde se abren las opciones:

- conseguir más datos, que siempre es cada vez más difícil y más costoso.
- si los datos tienen estructura, como en las imágenes, porque no elegimos una operación lineal que logre explotarla? Si mis datos ya tienen una banda de info dada por la estructura espacial de la imagen, porque no acotamos una

operación lineal restringida a tenerla en cuenta?

Estructura de los Datos



- Si los datos tienen una **estructura regular** (datos contiguos están relacionados)
 - ⇒ Podemos usar la **operación de convolución** (es una operación lineal) en lugar de un \mathbf{W}_ℓ genérico



- Observar que, ahora, la transformación lineal **sólo relaciona elementos contiguos**
 - ⇒ Es lo mismo que hace la convolución: combinación lineal de pocos elementos y desplazamiento

Esto es basicamente lo que hace una operación de convolucion. Pasa a solo aprender los dos valores amarillo y violeta. Estas imponiendo una estructura especifica para la matriz de transformación lineal, ahorrándote puntos a aprender. Esto hace que la cantidad de parametros que tengo que aprender ya no dependa mas de los datos de entrada, solo aprendo los del filtro de tamaño k. (Es independiente el tamaño k del tamaño de los datos)

Cuando yo agarro el perceptron multicapa y reemplazo la operación lineal por una convolución, eso da lugar a una red neuronal convolucional. :)

> Hoy por hoy, cuando se habla de lo bien que andan las redes neuronales, se esta hablando de lo bien que andan las CNN para procesar imágenes, típicamente para clasificación.

Conceptualmente, lo que sabemos de las RN normales sigue valiendo, pero agregamos el acote de la operación de convolucion.

MLP = multilayer perceptron

Redes Neuronales Convolucionales



- ▶ Las CNNs son un subconjunto de las MLPs \Rightarrow Sin embargo funcionan mejor (en imágenes)



\Rightarrow

- ▶ Son más fáciles de optimizar \Rightarrow Menos parámetros para aprender, menos datos
- ▶ Son más computacionalmente eficientes \Rightarrow Requieren $O(KN)$ operaciones en vez de $O(N^2)$
- ▶ Comparten parámetros \Rightarrow Reducen el almacenamiento en memoria: $O(K)$ vs. $O(N^2)$
- ▶ Usan la estructura de los datos para generalizar mejor \Rightarrow Equivarianza a traslaciones



- ▶ Si los datos se repiten \Rightarrow La salida se repite
- ▶ La CNN captura información independientemente de su ubicación

Las CNN tienden a funcionar mejor porque al saber que ya son imágenes, el espacio que tenes que recorrer para aprender es mucho mas chico. Son computacionalmente mas eficientes y reducen el almacenamiento en memoria.

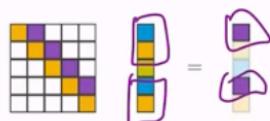
W = matriz de pesos a aprender.

X = datos de imágenes puestos como vectores.



> Elegir la operación de convolución hace que yo generalize mejor. Porque tiene lo que se llama equivarianza a traslaciones:

- ▶ Usan la estructura de los datos para generalizar mejor \Rightarrow Equivarianza a traslaciones



- ▶ Si los datos se repiten \Rightarrow La salida se repite
- ▶ La CNN captura información independientemente de su ubicación

ejemplo: para una sucesión de valores azul y naranja el output va a ser siempre violeta. Independientemente de que lugar esta, voy a tener siempre el mismo resultado. Explota la estructura de grupo que tiene la convolucion.

Same para variable temporales. Por ejemplo un soplo en un electrocardiograma, es independiente de cuando ocurra. Si yo en vez de eso aprendiera una matriz arbitraria no convolucional, tendría que aprenderme todas la combinaciones posibles de esas traslaciones. Por esta razón las CNN

generalizan mucho mejor. Con una sola muestra aprenden todas las variantes.

Redes Neuronales Convolucionales



- Informalmente, una red neuronal convolucional (CNN) es una perceptrón multicapa (MLP)
 - ⇒ Se reemplaza la transformación lineal por una convolución (que también es lineal)
 - ⇒ La operación de convolución se escribe como una matriz con estructura Toeplitz



- Ahora no hay que aprender $N \times N_1$ parámetros en \mathbf{W}_0 , sino que **hay que aprender $K \ll N$**
- Las redes neuronales convolucionales han sido tremadamente exitosas
 - ⇒ Fundamentalmente para procesar imágenes (clasificación, detección, etc.)

Puedo elegir un tamaño de filtro independientemente de la cantidad de datos, top.

> Cuando yo tengo una matriz arbitraria, la dimensión de salida, también la puedo elegir de forma arbitraria, puedo mapear de cualquier dimensión a cualquier dimensión

Dimensionalidad y Capacidad de Representación



- Una transformación lineal arbitraria puede cambiar las dimensiones
 - ⇒ La matriz $\mathbf{W}_\ell \in \mathbb{R}^{N_\ell \times N_{\ell-1}}$ toma un vector en $N_{\ell-1}$ y devuelve un vector en N_ℓ
- Las convoluciones toman una señal de entrada, y devuelven una señal de salida
 - ⇒ La convolución toma un vector de dimensión N y devuelve un vector en $N-K+1$
- Al utilizar una **convolución**, perdemos la capacidad de ajustar las dimensiones
 - ⇒ Perdimos la posibilidad de ajustar la capacidad de representación del algoritmo

De alguna forma, en el momento que elegí la convolución, estoy perdiendo la capacidad de determinar las dimensiones que tiene la representación. Ya no puedo elegir la dimensión de manera arbitraria, mi filtro va siempre con la misma. En el perceptron multicapa podamos ir variando las dimensiones pero aca ya no, el tamaño del vector de salida esta determinado por el tamaño del vector de entrada, capa a capa. De alguna forma, nos gustaría poder recuperar esta capacidad que teníamos antes... Esto es lo que hacemos con el concepto de **features o channels (tensores)**.

Características (Features) o Canales (Channels)



- Se puede recuperar el control de la capacidad de representación usando muchas convoluciones
⇒ Entra la señal \mathbf{x} y la filtramos con F_1 convoluciones h_1^f con K_1 coeficientes

$$\mathbf{x} \mapsto \{h_1^1 * \mathbf{x} + b_1^1, h_1^2 * \mathbf{x} + b_1^2, \dots, h_1^{F_1} * \mathbf{x} + b_1^{F_1}\} = \{\mathbf{z}_1^1, \dots, \mathbf{z}_1^{F_1}\}$$

- Las podemos juntar en una matriz \mathbf{Z} de tamaño $N - K_1 + 1 \times F_1$

$$\mathbf{Z} = [\mathbf{z}^1 \quad \mathbf{z}^2 \quad \dots \quad \mathbf{z}^{F_1}]$$

En términos de imágenes, tenemos 3 canales RGB por ejemplo, luego me gustaría poder representar esto en 512 dimensiones distintas.. me gustaría pasar de 3 imágenes a 512 imágenes distintas. Como puedo hacer estos? Aprendiendo muchos filtros distintos → en cada capa no tengo un solo filtro sino que tengo un *banco de filtros*. Voy a aprender cada uno de esos bancos de filtros. En el slide ponele, arranco con una señal de dimension 1, ($N \times 1$) y me interesa llevar esto a una dimension mas grande, por ejemplo $f_1 = 512$. Lo que voy a hacer entonces es aprender 512 filtros desde h_{11} hasta h_{1F_1} , y donde cada uno de esos si quieras te va a aprender un aspecto distinto de la señal de entrada. De esta forma, usando un banco de filtros logramos poder recuperar la capacidad de ir a espacios de mayor dimensionalidad. Obviamente lo que voy a tener a la salida, van a ser F_1 512 señales distintas.

~~Entrar \mathbf{x} y la filtramos con F_1 convoluciones h_1^f con K_1 coeficientes~~

$N \times 1$

$$\mathbf{x} \mapsto \{h_1^1 * \mathbf{x} + b_1^1, h_1^2 * \mathbf{x} + b_1^2, \dots, h_1^{F_1} * \mathbf{x} + b_1^{F_1}\} = \{\mathbf{z}_1^1, \dots, \mathbf{z}_1^{F_1}\}$$

~~luego las podemos juntar en una matriz \mathbf{Z} de tamaño $N - K_1 + 1 \times F_1$~~

Yo las puedo acomodar en una matriz que va a tener de tamaño, el tamaño de la señal X Cantidad de filtros

- Las podemos juntar en una matriz \mathbf{Z} de tamaño $N - K_1 + 1 \times F_1$

$$\mathbf{Z} = [\mathbf{z}^1 \quad \mathbf{z}^2 \quad \dots \quad \mathbf{z}^{F_1}]$$

Una vez que tenemos esto, le puedes aplicar la función de activación, ya que es siempre puntual

Características (*Features*) o Canales (*Channels*)



- Se puede recuperar el control de la capacidad de representación usando muchas convoluciones
 - ⇒ Entra la señal \mathbf{x} y la filtramos con F_1 convoluciones h_1^f con K_1 coeficientes
$$\mathbf{x} \mapsto \{h_1^1 * \mathbf{x} + b_1^1, h_1^2 * \mathbf{x} + b_1^2, \dots, h_1^{F_1} * \mathbf{x} + b_1^{F_1}\} = \{\mathbf{z}_1^1, \dots, \mathbf{z}_1^{F_1}\}$$
- Las podemos juntar en una matriz \mathbf{Z} de tamaño $N - K_1 + 1 \times F_1$
$$\mathbf{Z} = [\mathbf{z}^1 \quad \mathbf{z}^2 \quad \dots \quad \mathbf{z}^{F_1}]$$
- Ahora aplicamos la función de activación $\mathbf{X}_1 = \sigma_1(\mathbf{Z})$
 - ⇒ La salida de la primer capa tiene tamaño $N - K_1 + 1 \times F_1$
 - ⇒ El valor de F_1 lo controlamos (se elige por diseño) ⇒ Controla la capacidad de representación

Características (*Features*) o Canales (*Channels*)



- Se puede recuperar el control de la capacidad de representación usando muchas convoluciones
 - ⇒ Entra la señal \mathbf{x} y la filtramos con F_1 convoluciones h_1^f con K_1 coeficientes
$$\mathbf{x} \mapsto \{h_1^1 * \mathbf{x} + b_1^1, h_1^2 * \mathbf{x} + b_1^2, \dots, h_1^{F_1} * \mathbf{x} + b_1^{F_1}\} = \{\mathbf{z}_1^1, \dots, \mathbf{z}_1^{F_1}\}$$
- Las podemos juntar en una matriz \mathbf{Z} de tamaño $N - K_1 + 1 \times F_1$
$$\mathbf{Z} = [\mathbf{z}^1 \quad \mathbf{z}^2 \quad \dots \quad \mathbf{z}^{F_1}]$$
- Ahora aplicamos la función de activación $\mathbf{X}_1 = \sigma_1(\mathbf{Z})$
 - ⇒ La salida de la primer capa tiene tamaño $N - K_1 + 1 \times F_1$
 - ⇒ El valor de F_1 lo controlamos (se elige por diseño) ⇒ Controla la capacidad de representación
- Cada vector \mathbf{x}_1^f (cada columna de \mathbf{X}_1) es un *feature* (o *channel*)
 - ⇒ F_1 es la cantidad de features (o la cantidad de channels) de la capa $\ell = 1$
 - ⇒ La cantidad de parámetros a aprender ahora es $K_1 F_1$ (indep. de N)

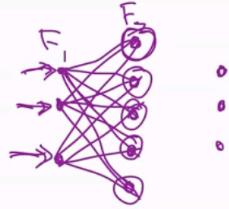
Bueno pero, cuando salgo de la primera capa que ya tengo las F_1 señales, como consigo F_2 a la salida de mi segunda capa? ahí vamos a pensar que vamos a hacer lo mismo que venimos haciendo en el MLP.

Haces el producto punto (convolucion en realidad) de cada una de las señales de entrada (F_1) con el filtro F_2 y los sumas:

Características (Features) o Canales (Channels)



- A la salida de la primer capa, tenemos F_1 features \mathbf{x}^f en la matriz $\mathbf{X}_1 \in \mathbb{R}^{N-K_1+1 \times F_1}$
- ⇒ Si ahora quiero F_2 features a la salida de la segunda capa, ¿cómo hacemos?
- ⇒ Recordar que en un MLP la dimensión N_ℓ (features) de cada capa era elección del diseñador



Como si fuera una producto de Matrices pero en vez de escalares tenes filtros y en vez de multiplicar haces la convolucion.

- En la capa 2 se usan $F_1 \times F_2$ convoluciones con filtros h_2^{fg} con K_2 coeficientes ⇒ Banco de filtros

$$\{\mathbf{x}_1^1, \dots, \mathbf{x}_1^{F_1}\} \mapsto \left\{ \begin{array}{l} h_2^{11} * \mathbf{x}_1^1, \quad h_2^{12} * \mathbf{x}_1^1, \quad \dots, \quad h_2^{1F_2} * \mathbf{x}_1^1, \\ h_2^{21} * \mathbf{x}_1^2, \quad h_2^{22} * \mathbf{x}_1^2, \quad \dots, \quad h_2^{2F_2} * \mathbf{x}_1^2, \\ \dots, \\ h_2^{F_11} * \mathbf{x}_1^{F_1}, \quad h_2^{F_12} * \mathbf{x}_1^{F_1}, \quad \dots, \quad h_2^{F_1F_2} * \mathbf{x}_1^{F_1} \end{array} \right\}$$

$F_1 \times F_2$

- En la capa 2 se usan $F_1 \times F_2$ convoluciones con filtros h_2^{fg} con K_2 coeficientes ⇒ Banco de filtros

Convolucion entre el filtro H11 y la señal 11

$$\{\mathbf{x}_1^1, \dots, \mathbf{x}_1^{F_1}\} \mapsto \left\{ \begin{array}{l} h_2^{11} * \mathbf{x}_1^1, \quad h_2^{12} * \mathbf{x}_1^1, \quad \dots, \quad h_2^{1F_2} * \mathbf{x}_1^1, \\ h_2^{21} * \mathbf{x}_1^2, \quad h_2^{22} * \mathbf{x}_1^2, \quad \dots, \quad h_2^{2F_2} * \mathbf{x}_1^2, \\ \dots, \\ h_2^{F_11} * \mathbf{x}_1^{F_1}, \quad h_2^{F_12} * \mathbf{x}_1^{F_1}, \quad \dots, \quad h_2^{F_1F_2} * \mathbf{x}_1^{F_1} \end{array} \right\}$$

señales de tamaño N

- Esto da $F_1 F_2$ señales a la salida ⇒ Cada señal de tamaño $N - K_1 - K_2 + 2$

$$\{\mathbf{x}_1^1, \dots, \mathbf{x}_1^{F_1}\} \mapsto \left\{ \begin{array}{l} z_2^{11}, \quad z_2^{12}, \quad \dots, \quad z_2^{1F_2}, \\ z_2^{21}, \quad z_2^{22}, \quad \dots, \quad z_2^{2F_2}, \\ \dots, \\ z_2^{F_11}, \quad z_2^{F_12}, \quad \dots, \quad z_2^{F_1F_2} \end{array} \right\}$$

⇒ Pero queremos sólo F_2 señales, no $F_1 F_2$ señales ⇒ ¿Qué hacemos?

Ojo porque los filtros que le aplicas a cada señal son distintos! F_2 es la cantidad de filtros. Tenés F_2 filtros PARA CADA FILA distintos. Le estas dando la libertad de aprender distintos aspectos para cada una de las señales que tuve a la entrada. Cada filtro aprende un aspecto distinto de cada señal de entrada. Recordemos que en el caso de las imágenes podia ser 512 para cada uno de los 3 canales.

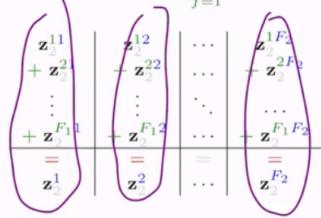
Como nos quedaron $f_1 \times f_2$ filtro y solo queremos F_2 , lo que hacemos es sumarlas para obtener F_2 . Tenemos $F_1 \times F_2$ filtros.

Características (*Features*) o Canales (*Channels*)



- Tenemos $F_1 F_2$ señales \mathbf{z}_2^{fg} , pero queremos sólo F_2 señales

$$\Rightarrow \text{Sumamos las señales en la dimensión } F_1 \Rightarrow \mathbf{z}_2^g = \sum_{f=1}^{F_1} \mathbf{z}_2^{fg}$$



- Operación **lineal**, que explota la **estructura** y que ofrece **control sobre la representación**
- \Rightarrow Convolución = Banco de filtros + Suma \Rightarrow Mapea F_1 features en F_2

De esta forma, recuperamos el control sobre la capacidad de representación de la arquitectura:

Características (*Features*) o Canales (*Channels*)

fgan



- Recuperamos el **control sobre la capacidad de representación** de la arquitectura
- \Rightarrow Convolución = Banco de filtros + Suma sobre la dimensión de entrada
- \Rightarrow El **diseñador** elige la cantidad de **features** en cada capa F_ℓ

Red Neuronal Convolucional

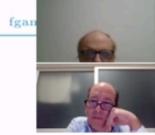
$$f(\mathbf{X}; \boldsymbol{\Theta}) = \{\mathbf{x}_L^g\}_{g=1}^{F_L} \quad \text{con} \quad \mathbf{x}_\ell^g = \sigma_\ell \left(\sum_{f=1}^{F_{\ell-1}} \mathbf{h}_\ell^{fg} * \mathbf{x}_{\ell-1}^f + \mathbf{b}_\ell^g \right), \quad \ell = 1, \dots, L \quad \text{y} \quad \mathbf{X}_0 = \mathbf{X}$$

- La señal de salida \mathbf{X}_ℓ en cada capa tiene dimensión $N - \sum_{\ell'=1}^\ell K_{\ell'} + \ell \times F_\ell$
- Los **parámetros a aprender** son los coef. de todos los filtros $\sum_{\ell=1}^L K_\ell F_\ell F_{\ell-1} \Rightarrow$ Independiente de N
- Los hiperparámetros son el **tamaño del filtro** en cada capa K_ℓ y la **cant. de features** F_ℓ (y L y σ_ℓ)
- La convolución es lineal \Rightarrow Puedo usar **backpropagation** como siempre

> Ahora bien, por lo general a medida que uno va mas profundo en las capas, la cantidad de features aumenta. Si yo empiezo a aumentar mucho FL, tengo cada vez mas datos. Por ejemplo si tengo imágenes en RGB, tengo 3 imágenes a la entrada y si quiero irme a alta dimension me voy a 512. si esas imágenes son muy grandes, por mas de que la convolucion sea muy eficiente, a la salida voy a tener 512 imágenes que voy a tener que guardar en algun lado, ni hablar si sigo subiendo la dimension.

Entonces, de alguna forma lo que vamos a buscar hacer es achicar el tamaño de la imagen. Aumentar la cantidad de canales, achicando el tamaño de la imagen. Esto es lo que se conoce com **pooling**.

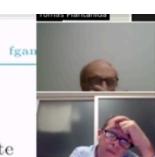
Pooling



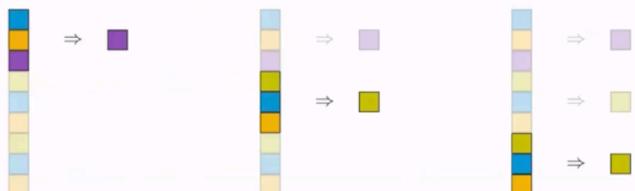
- ▶ La señal de salida \mathbf{X}_ℓ en cada capa tiene dimensión $N - \sum_{\ell'=1}^{\ell} K_{\ell'} + \ell \times F_\ell$
- ▶ Con F_ℓ controlamos la capacidad de representación \Rightarrow Mayor F_ℓ , mayor capacidad
- ▶ Pero a medida que F_ℓ se hace más grande, el tamaño de las señales \mathbf{X} se hace más grande
 \Rightarrow El costo computacional se incrementa, la necesidad de almacenamiento en memoria también
- ▶ La técnica de *pooling* se utiliza para construir resúmenes y reducir la dimensión de la señal

Básicamente construimos "resúmenes" de ciertas regiones de mis datos, me quedo solo con estos antes de pasarlos a la siguiente capa. Empieza a ver un tradeoff en el aumento de la cantidad de features mientras achico el tamaño de mi señal. Por ejemplo, si tenemos esta señal de dimension 9, agarro señales de 3 y por cada una de ellas me construyo un resumen. Por ejemplo el máximo o el promedio de los valores (cualquier otra función también puede valer pero es lo que mas se usa)

Pooling



- ▶ Pooling \Rightarrow Construir un resumen de una región contigua de valores de la señal
 \Rightarrow Tomar un representante de la región, descartar los demás, y moverse a la región siguiente



De esta manera voy achicando la dimensión a cambio de tener cada vez más features a medida que avanzan las capas.

En la imagen la idea es lo mismo:

Pooling



- En el caso de imágenes, la idea es la misma: construir un resumen, tomar un representante



- El resultado de resumir las regiones es otra imagen, pero más chica



Entonces, en resumen, pooling lo que hace es construir resúmenes espaciales o temporales con el objetivo de reducir la dimensionalidad de la imagen. Mientras ganamos dimensionalidad en cantidad de features.

Luego lo que ocurre a medida que vamos en profundidad de la red, vamos achicando la dimension espacial mientras aumentamos la dimension de features.

Resúmenes de las Regiones



- Para construir los resúmenes, aplicamos una determinada función a la región
- Las elecciones típicas son el máximo o el promedio de los valores en la región
 - ⇒ Pero puede haber muchas alternativas, como la norma o algún otro momento
- La elección de la función de resumen impacta en el desempeño del algoritmo
 - ⇒ Tanto la elección de la función como el tamaño de la región corren por cuenta del diseñador

Esto me da una invarianza local dentro de la region:



- ▶ El uso de pooling le otorga a la arquitectura la propiedad de **invarianza local** (en la región)
 - ⇒ Sin importar dónde esté la información dentro de la región, igual la vamos a procesar
 - ⇒ Importante en problemas de clasificación donde la ubicación del concepto no es importante

- ▶ El uso de pooling transforma la equivarianza a traslaciones de la convolución
 - ⇒ En invarianza local a traslaciones
- ▶ Es otra forma de **explotar la estructura de los datos para mejorar la generalización**

El usar la convolucion me daba una equivarianza (encontraba lo mismo en cualquier parte de la region), al achicar esa region a un único valor, toda esa region, sin importar en donde estaba esa region, el resultado del poling va a ser un valor. DE alguna forma el pooling convierte la equivarianza de la convolucion en invarianza dentro de la region. Lo que me ayuda tambien a explotar la estructura de los datos para mejorar la generalización.

> Observaciones sobre el pooling

Nos agrega nuevas decisiones como diseñadores, ya que ahora tenemos que decidir que tan grande es la region que vamos a resumir y como la vamos a resumir

Observaciones sobre el Pooling



- ▶ Pooling ⇒ Nuevas decisiones para el diseñador: el tamaño de la región y la función de resumen
- ▶ El pooling aplica sólo a **estructuras de datos** donde la noción de región está bien definida
 - ⇒ Con la convolución son formas **explotar la estructura de los datos para mejorar la generalización**
- ▶ Usar pooling sirve para **intercambiar información espacial** (en la dimensión N)
 - ⇒ Por **información aprendida** en forma de features (en la dimensión F_ℓ)
 - ⇒ La idea es que $N_L F_L < NF_0$ ⇒ Más features en imágenes más chicas
- ▶ Hay alternativas donde se hace pooling sobre la dimensión F ⇒ Pero no tienen estructura regular
- ▶ Hacer pooling **permite trabajar con señales de distinto tamaño**

Solo se aplica a estructuras de datos donde la noción de de region esta bien definida. Si yo puedo aplicar la convolución, por estructura, puedo aplicar pooling.

El acote es poder controlar que la dimension no se me vaya a la goma por querer aprender mas features.

Permite ademas adaptar una misma red neuronal para imágenes o señales de distinto tamaño.

> Arquitecturas

Una CNN es una arquitectura por capas, donde cada capa es una convolución, en vez de una transformación lineal arbitraria, seguida de una función de activación.

Redes Neuronales Convolucionales



- ▶ Una red neuronal convolucional (CNN) es una arquitectura por capas
 - ⇒ Donde **cada capa es una convolución seguida de una función de activación**
 - ⇒ La convolución se realiza con un banco de filtros para controlar la capacidad de representación
 - ⇒ Se puede incluir **pooling para controlar dimensionalidad de las señales**
- ▶ Se usan con **datos de estructura regular** ⇒ Elementos contiguos están relacionados

- ▶ Los **hiperparámetros** a elegir por el diseñador son
 - ⇒ La cantidad de capas L y las funciones de activación σ_ℓ
 - ⇒ La **cantidad de features** a la salida de cada capa F_ℓ
 - ⇒ El **tamaño de los filtros** K_ℓ
 - ⇒ Si se incluye pooling: el tamaño de la región y la función de resumen
- ▶ La cantidad de parámetros a aprender es $\sum_\ell K_\ell F_\ell F_{\ell-1}$ ⇒ **Independiente de N**

La cantidad de parametros que yo quiero aprender es independiente del tamaño de las imágenes, puedo aprender los mismos param sobre cualquier imagen de entrada que le pinte.



- ▶ Las CNNs han sido sumamente exitosas en procesamiento de imágenes
 - ⇒ Problemas de clasificación: la entrada es una imagen, la salida una clase
 - ⇒ Problemas de detección: la entrada es una imagen, la salida es una imagen

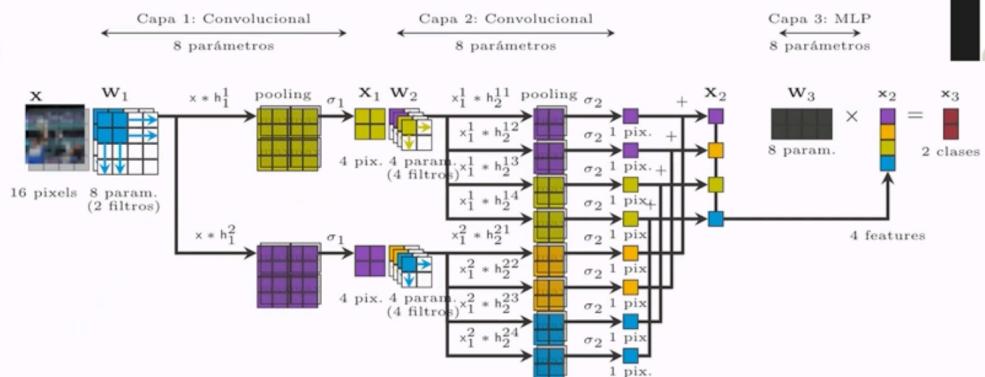
- ▶ En problemas donde la entrada es una imagen, pero la salida no (clasificación)
 - ⇒ Se suele incluir MLPs luego de varias capas convolucionales
- ▶ Las capas convolucionales actúan como 'extractores de features'
- ▶ Las capas de MLPs actúan como 'clasicificadores'
- ▶ El intercambio de dimensiones N_ℓ con features F_ℓ es importante

- ▶ **Stride** ⇒ Salto en el cálculo de la convolución en una posición ⇒ $h * x = \sum_{k=0}^{K-1} h_k \times (n - s k)$

Buena data la de cuando quieras clasificar imágenes, se usan capaz convolucionales en serie con MLPS. Las CNN actúan como extractoras de features y el perceptron las clasifica.

Stride: cuanto muevo yo el filtro cada vez que hago la convolucion. Hasta ahora lo movíamos una sola vez por convolucion, pero hay muchas arquitecturas que lo mueven mas de una vez. Es asumir que no hace falta moverme uno solo porque la info es redundante entonces me muevo mas. Esto tambien hace que la operación de convolucion sea mucho mas rápida.

Ejemplo de arquitectura:

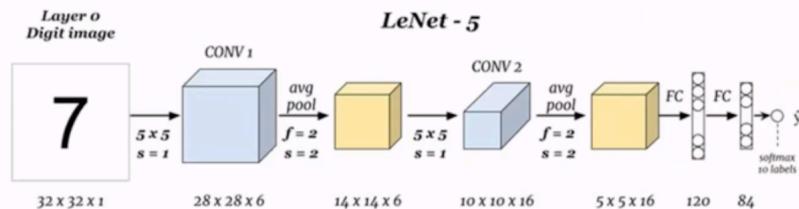


Esta es una medio de juguete, a modo de ejemplo. Para ver una estructura un poco mas realista:

Arquitecturas: LeNet (LeCun, 1998)



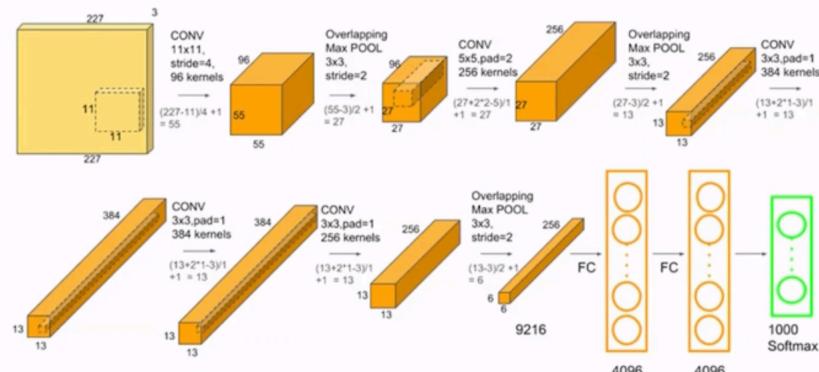
- ▶ Una de las primeras en lograr un desempeño extraordinario en MNIST
- ▶ Usa pooling en promedio en vez de max pooling (en aquella época no se usaba max pooling)
- ▶ Usa tanh como la función de activación (todavía las ReLU no se habían popularizado)
- ▶ Hay diez dígitos posibles \Rightarrow La última capa es un vector de tamaño 10



Arquitecturas: AlexNet (Krizhevsky, Sutskever, Hinton, 2012)



- ▶ Competición ImageNet \Rightarrow Clasificación de imágenes en 1000 categorías \Rightarrow 83 % en top-5

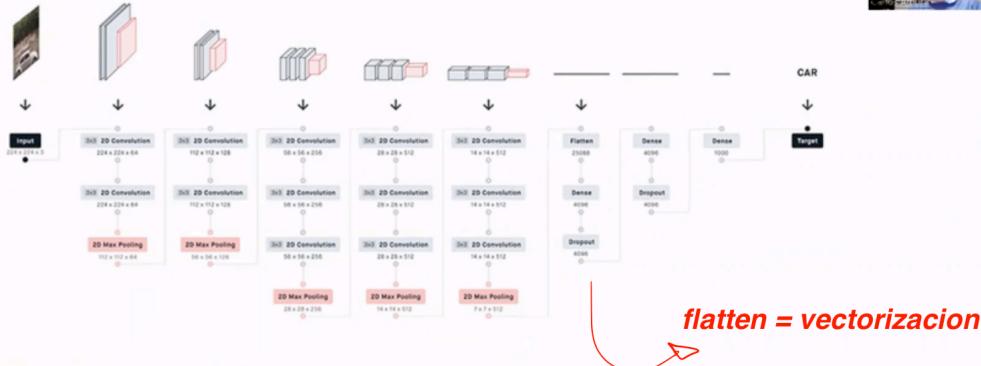


kernels = filtros \rightarrow cada uno representa un feature.

Nota: existe también el pooling con overlapping (solapamiento) como en el ejemplo de AlexNet

Arquitecturas: VGG (Simonyan, Zisserman, 2014)

- ▶ Competición ImageNet ⇒ Clasificación de imágenes en 1000 categorías ⇒ 92,7% en top-5



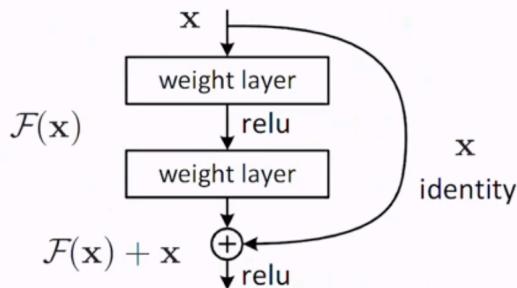
.UBAfiuba
FACULTAD DE INGENIERÍA

26/30

Una contribución interesante que se dio a las arquitecturas en 2015 fue la de skip connections:

Arquitecturas: *Skip Connections*

- ▶ Luego de una determinada cantidad de capas, **volver a incluir la entrada**
 - ⇒ Evita que la información se diluya a través de las capas y los gradientes
 - ⇒ Particularmente importante para arquitecturas profundas



.UBAfiuba
FACULTAD DE INGENIERIA

27 / 30

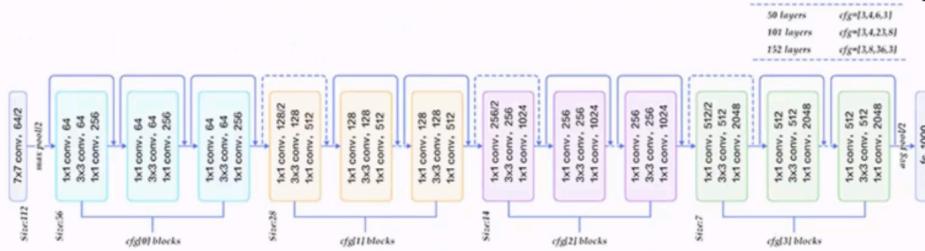
Es volver a repetir la entrada, pasadas distintas capas. Esto se hace ya que a medida que las NN se empiezan a hacer mas profundas, la información de entrada de la imagen "quedo muy atrás". Se va diluyendo la información. Si tengo NN extremadamente profundas, se propone volver a enchufar la imagen cada determinada cantidad de capas. Esto hace que funcione mucho mejor. Se le llama skip connections porque hay conexiones entre capas que justamente saltean capas.

En particular esto se uso mucho en ResNets:

Arquitecturas: ResNets (He et al, 2015)



- Muy profunda ($L = 152$ capas) \Rightarrow Resultado en ImageNet 96,4 % en top-5



Se llaman ResNets por "residuals", se entienden a estas skip connections como conexiones residuales que voy haciendo. Vamos a ver que van a aparecer un montón de cosas como estas que no quedan muy claras porque se hacen pero que se fue demostrando en la práctica que andan jajaja.

Esto depende más de la arquitectura, pero en el caso de arriba de ResNet no se enchufa la entrada propiamente dicha desde cero sino que se enchufa la salida de tres bloques anteriores. Veremos luego que en grafos eso si se hace de la señal de entrada. Un poco la noción acá es que puedes reenchufar la entrada cuando tengas ganas.

Notar como los requerimientos de hardware y memoria se te van a la goma. Nosotros no vamos a correr algo tan pesado o grande ya que necesitaríamos un servidor profesional para hacerlo.

Resumiendo la clase de hoy:

Resumen del Capítulo

fgan



- ▶ Redes Neuronales Convolucionales ⇒ Reemplazar la transformación lineal por una convolución
 - ⇒ Se pierde el control sobre la capacidad de representación
 - ⇒ Banco de Filtros + Suma
- ▶ Pooling ⇒ Construir resúmenes regionales y luego quedarse sólo con un representante
 - ⇒ Controlar el tamaño de las señales ⇒ Invarianza local a traslaciones
- ▶ Arquitecturas típicas ⇒ Incluir MLPs al final, conexiones residuales, etc.

Para la clase que viene:

Escenas del Próximo Capítulo...

fgan



- ▶ Queremos procesar secuencias de datos
 - ⇒ Adaptamos la red neuronal para generalizar en datos secuenciales
 - ⇒ Compartir parámetros ⇒ Mismos valores durante toda la secuencia
- ▶ Redes neuronales recurrentes ⇒ Aprenden un estado oculto
 - ⇒ Este estado es capaz de capturar la información temporal relevante
- ▶ Área activa de investigación ⇒ Muchas extensiones
 - ⇒ Redes neuronales bidireccionales ⇒ El contexto importa
 - ⇒ LSTMs, GRUs ⇒ Poder aprender dependencias de largo alcance