

CONVOLUTIONAL NEURAL NETWORK

Le CNN sono ottime per svolgere i seguenti compiti:

- 1) Classificazione di immagini
- 2) Rilevamento di oggetti
- 3) Semantic segmentation: predire una classe per ogni pixel.

Le CNN utilizzano filtri.

CONVOLUZIONI e CROSS-CORRELATION

Assumiamo di aver due funzioni $f[x]$ e $g[k]$:

$$f: \mathbb{Z}^d \rightarrow \mathbb{R}$$

$$g: \mathbb{Z}^d \rightarrow \mathbb{R}$$

Generalizziamo la definizione di convoluzione discreta:

$$f[u] * g[u] = \sum_{v \in \mathbb{Z}^d} f[v] \cdot g[u-v] \quad \text{con } u \in \mathbb{Z}^d$$

Per $d \geq 2$ stiamo facendo la convoluzione fra due tensori.

Generalizziamo ora la definizione di cross-correlation:

$$f[u] \odot g[u] = \sum_{v \in \mathbb{Z}^d} f[v] \cdot g[u+v] \quad \text{con } u \in \mathbb{Z}^d$$

Dove, rispetto alla convoluzione non si capovolge il filtro

Proprietà importanti:

1) $f[u] \circledast g[u] = f[-u] * g[u]$

Poiché:

$$\begin{aligned} f[u] \circledast g[u] &= \sum_{v \in \mathbb{Z}^d} f[v] \cdot g[u+v] = \text{sostituendo } v = -w \\ &= \sum_{w \in \mathbb{Z}^d} f[-w] \cdot g[u-w] = f[-u] * g[u] \end{aligned}$$

2) $f[u] \circledast g[u] \neq g[u] \circledast f[u]$: NON COMMUTATIVA

Poiché:

$$f[u] \circledast g[u] = f[-u] * g[u]$$

$$\neq$$

$$f[u] * g[-u] = g[u] \circledast f[u]$$

Da qui possiamo certamente dire che se $f[u] = f[-u]$
allora:

$$f[u] \circledast g[u] = f[u] * g[u]$$

Questo ci fa intuire che, uno dei filtri simmetrici è proprio il filtro gaussiano. Questo è molto importante perché in realtà le CNN fanno la cross-validation e non la convoluzione. Ma per i filtri simmetrici è la stessa cosa.

Dimensioni di $*$ e \otimes

Pondiamo:

f con $K_h \times K_v$ pixel

g con $M_h \times M_v$ pixel

Allora sia la convoluzione che la cross-correlation fra le due funzioni hanno:

$$(M_h - K_h + 1) \times (M_v - K_v + 1) \text{ pixel}$$

O ancora in 3D:

$$\begin{cases} f \text{ con } K_c \times K_h \times K_v \text{ pixel} \\ g \text{ con } M_c \times M_h \times M_v \text{ pixel} \end{cases} \Rightarrow \otimes \circ * \text{ con } (M_c - M_k + 1) \times (M_h - M_k + 1) \times (M_v - M_k + 1)$$

Ogni volta che applichiamo $\otimes \circ *$ riduciamo la dimensione dell'immagine. Per evitare questo possiamo aggiungere un PADDING all'ingresso. Questo bordo può essere

1) Costante

2) Intensità uguale al 1-Nearest-Neighbor

Per preservare la dimensione dei canali si usano più di un filtro.

MULTI LAYER PERCEPTRONS (MLP)

L'obiettivo di un MLP è quello di approssimare una funzione $H: \mathbb{R}^m \rightarrow \mathbb{R}^n$ con una funzione $H_\Theta: \mathbb{R}^m \rightarrow \mathbb{R}^n$ che dipende dal parametro Θ :

$$H(x) = y \longrightarrow H_\Theta(x) = \hat{y}$$

Come è fatta $H_\Theta(x)$?

È una funzione composta da L (numero di livelli della NN) funzioni $H_\Theta^1 \dots H_\Theta^L$:

$$\left\{ \begin{array}{l} h_\Theta = f \xrightarrow{\text{ingresso}} \\ h_1 = H_\Theta^1(h_\Theta) \\ h_2 = H_\Theta^2(h_1) \\ \vdots \\ h_L = H_\Theta^L(h_{L-1}) \xrightarrow{\text{uscita}} \end{array} \right. \Rightarrow g = H_\Theta^L \left(H_\Theta^{L-1} \left(\dots \left(H_\Theta^1(f) \right) \dots \right) \right)$$

Generalizzando $h_\Theta^i (i-1)$ si presenta:

$$z_i = w_i h_{i-1} + b_i$$

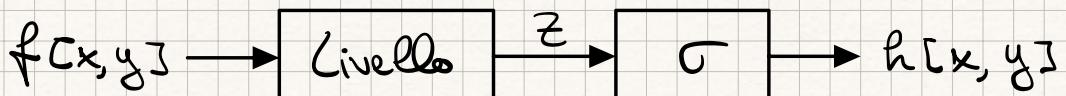
$$h_i = \sigma_i(z_i)$$

Dove:

w_i : Matrice dei pesi
 b_i : vettore dei bias

σ_i : funzione di attivazione (non lineare)

Concentriamoci su un singolo layer, w.l.o.g.:



In questo caso lo scrivo $z = W \cdot f + \beta$:

$$z[x, y] = \sum_k \sum_e W[x, y, k, e] \cdot f[k, e] + \beta[x, y]$$

Note: Questa è una versione 2D di $z = W \cdot f + \beta$, dove f è una funzione in 2D: $M \times N$. Non è un vettore in 1D con dimensione $M \cdot N$.

Sostituiamo $k = x + a$ e $e = y + b$:

$$z[x, y] = \sum_a \sum_b W[x, y, a, b] \cdot f[x+a, y+b] + \beta[x, y]$$

Abbiamo un tensore in 4D che moltiplica un'immagine in 2D. Se prendiamo $M=N=1024$ abbiamo che il tensore \hat{W} ha dimensione $1024^4 = 2^{40} = 1\text{ TERA di parametri da imparare.}$

LIVELLO CONVOLUZIONALE

L'unico modo per calcolare $Z[x, y]$ è quello di aggiungere dei vincoli. Questi vincoli sono validi solo per le immagini.

1) Translation Invariant

Questo vuol dire che \hat{W} e β non dipendono dalla posizione corrente $[x, y]$, per questo possiamo semplificare la formula:

$$Z[x, y] = \sum_a \sum_b \hat{W}[a, b] \cdot f[x+a, y+b] + \beta$$

2) Spatial Locality

Non bisogna cercare ovunque nell'immagine, ma basta cercare localmente. Abbiamo bisogno di informazioni locali nel punto $[x, y]$, ovvero per $a, b \in \mathcal{P}$, che è un patch di dimensione $D \times D$ pixel:

$$Z[x, y] = \sum_{a, b \in \mathcal{P}} \hat{W}[a, b] \cdot f[x+a, y+b] + \beta$$

Quanti parametri dobbiamo settare? Considerando \hat{W} con D^2 elementi, β scalare e con $D=3$ abbiamo $3^2+1 = 10$ parametri. Molto meglio di 2^{40} .

Equazioni livello convoluzionale:

$$Z[x, y] = \hat{W}[x, y] \otimes f[x, y] + \beta$$

$$h[x, y] = \sigma(Z[x, y])$$

MULTI-SPECTRAL IMAGES (Immagini colorate)

Semplicemente si usa una cross-correlation generalizzata

$$z[x, y] = \sum_c \sum_{\alpha, \beta \in P} \hat{W}[c, \alpha, \beta] \cdot f[c, x+\alpha, y+\beta] + \beta[c] = \\ = \sum_c \hat{W}[c, x, y] \otimes f[c, x, y] + \beta[c]$$

C sta per canali. Per $c=3$ abbiamo 3 Kernel convoluzionali \hat{W} ! E se generiamo più di un canale d'uscita?

$$z[c_0, x, y] = \sum_{c_i} \sum_{\alpha, \beta \in P} \hat{W}[c_0, c_i, \alpha, \beta] \cdot f[c_i, x+\alpha, y+\beta] + \beta[c_0, c_i] \\ = \sum_{c_i} \hat{W}[c_0, c_i, x, y] \otimes f[c_i, x, y] + \beta[c_0, c_i]$$

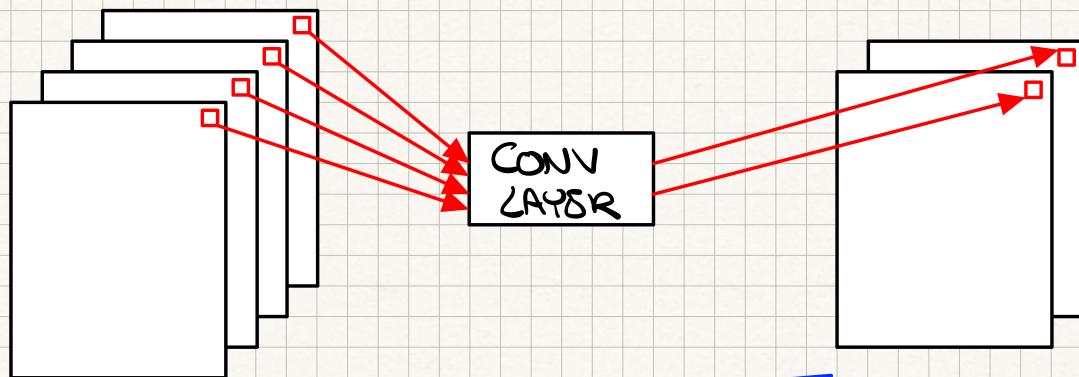


Immagine con
 $C_i = 4$

Immagine con
 $C_0 = 2$

Stride

Invece di usare il kernel pixel per pixel, possiamo fare passi più grandi:

Esempio

Da un'immagine f :

$$\begin{cases} f: 100 \times 100 \text{ pixel} \\ \hat{w}: 3 \times 3 \text{ pixel} \end{cases}$$

1) Con Stride 1:

$$g = \hat{w} \otimes f : 100 - 3 + 1 \Rightarrow 98 \times 98 \text{ pixel}$$

2) Con Stride 2

$$g = \hat{w} \otimes f : \frac{100 - 3 + 1}{2} \Rightarrow 49 \times 49 \text{ pixel}$$

Lo stride ci permette di ridurre la complessità globale della rete.

POOLING

Per scalare le dimensioni, viene usato un meccanismo di tipo **non-trainable**, detto **pooling layer**:

$$f[c, x, y] \rightarrow f[c, x', y']$$

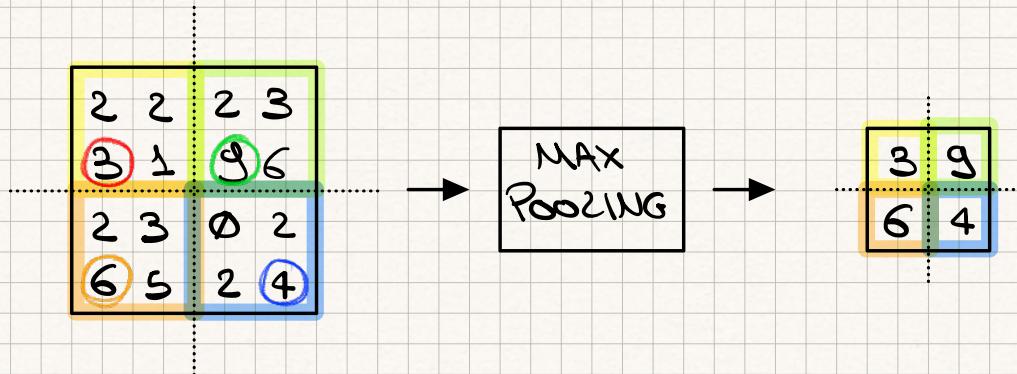
Fissato un canale c , il livello di pooling π modifica un patch $P(x', y')$ dell'immagine di ingresso che dipende dalla posizione dell'uscita $[x', y']$:

$$g[x', y'] = \pi_{x, y \in P(x', y')} [x', y']$$

I più comuni livelli di Pooling sono:

- 1) Max Pooling
- 2) Average Pooling

Esempio



CNNs

Sono composte da più livelli convoluzionali, più hidden layer e più livelli di Pooling.

OBJECT DETECTION

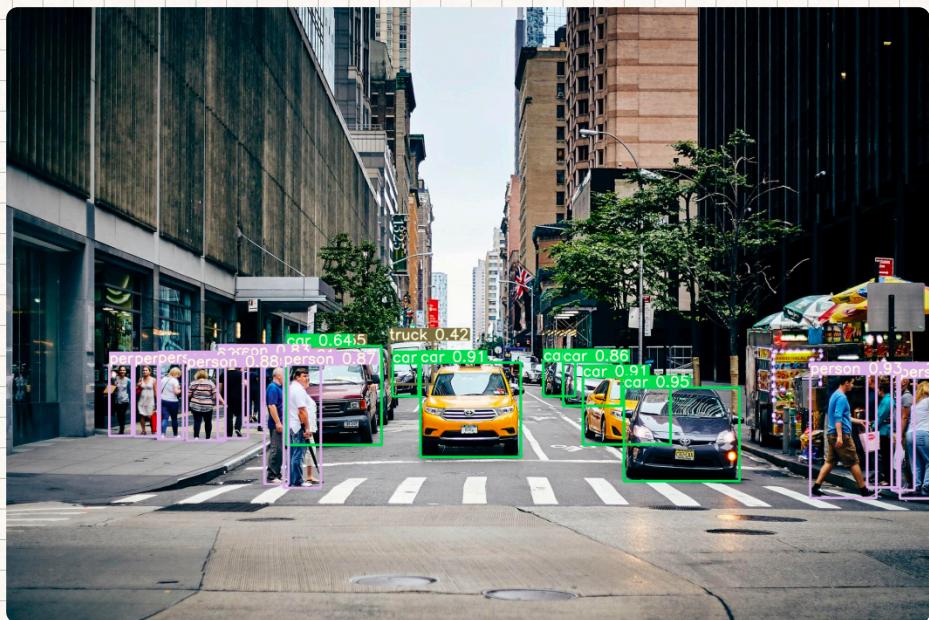
È il compito di predire classe e posizione di tutti gli oggetti visibili.

Un oggetto viene rappresentato tramite:

1) **Classe** (Macchina, fiore, persona ...)

2) **Bounding box** (x, y, w, h)

Dove (x, y) è il pixel centrale del box, mentre (w, h) indicano larghezza e altezza del box.



La più famosa CNN è **YOLO** (You only look once). Questo divide l'immagine in una griglia 5×5 , con quindi 5^2 celle:

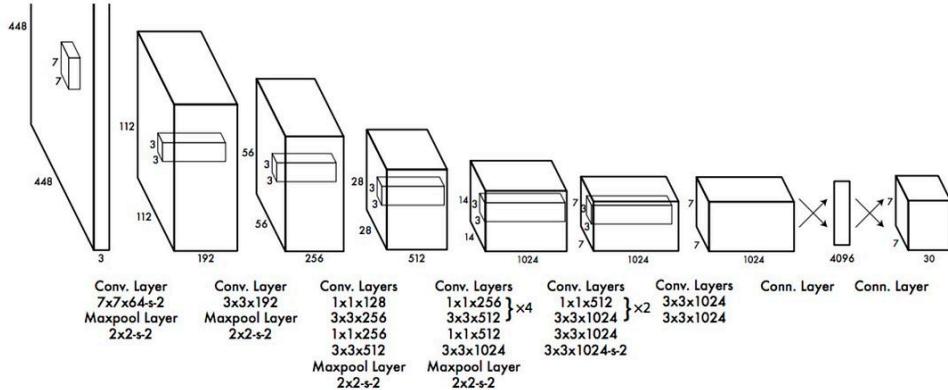
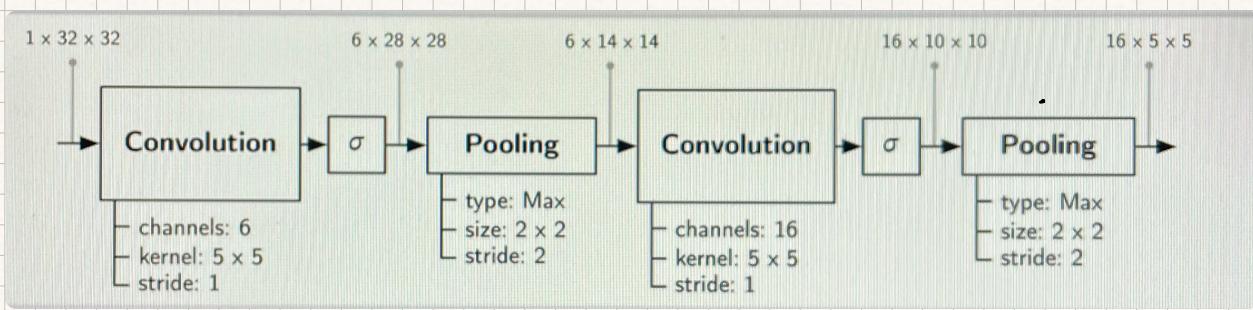
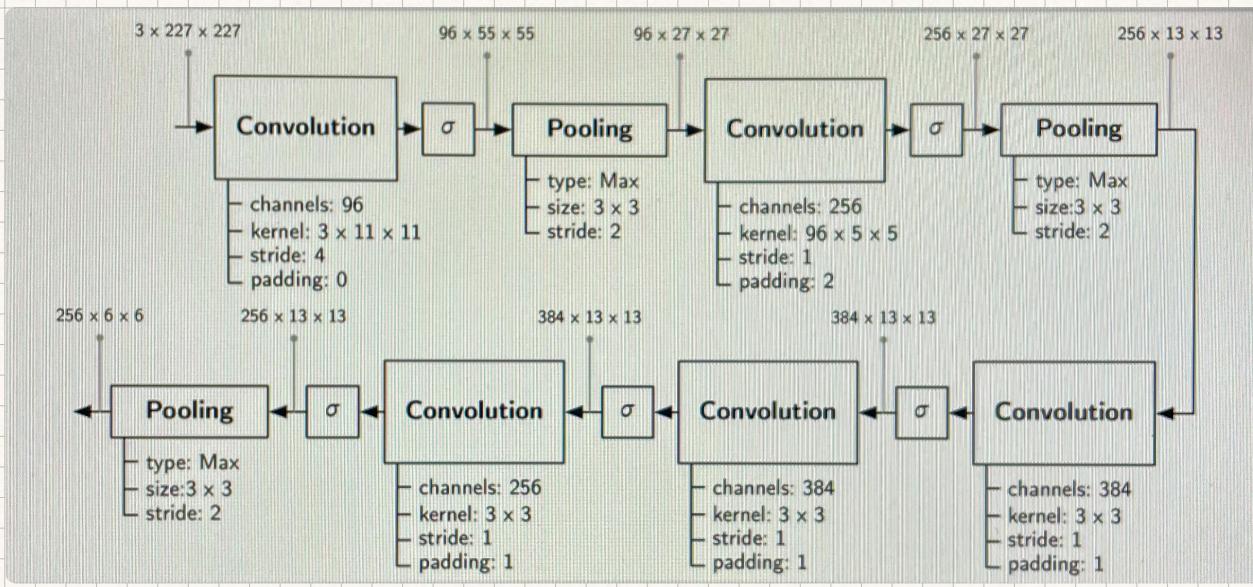


Figure 3: The Architecture. Our detection network has 24 convolutional layers followed by 2 fully connected layers. Alternating 1×1 convolutional layers reduce the features space from preceding layers. We pretrain the convolutional layers on the ImageNet classification task at half the resolution (224×224 input image) and then double the resolution for detection.

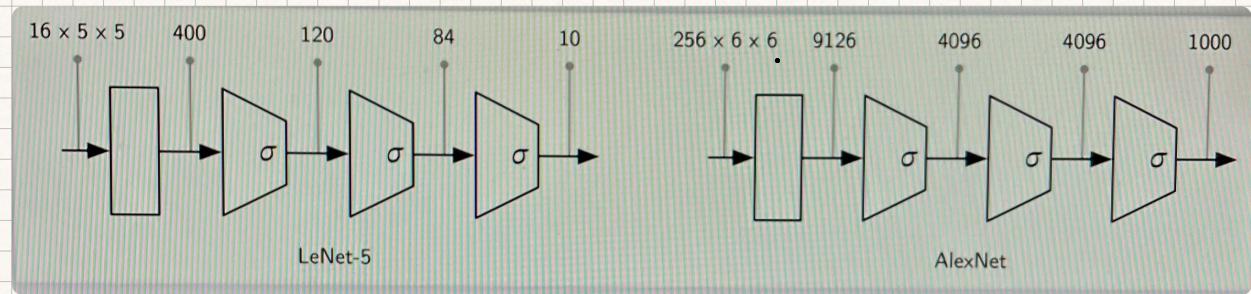
LINST 5 (1998)



ALEXNET



TASK DI IMAGE CLASSIFICATION



DECONVOLUTION NETWORK

