Indice

- Richiami storici
- Layer convolutivi
- Caratteristiche delle CNN
  - Local Receptive Fields
  - Weights Sharing
  - Upsampling/Downsampling
- Tipi di Layer

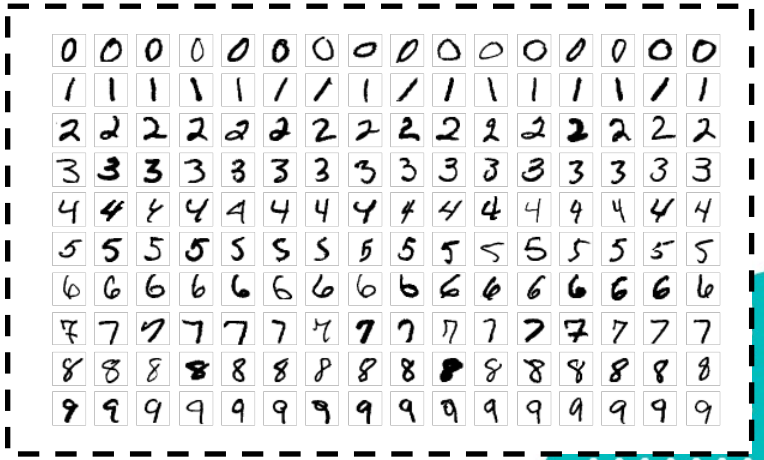
Richiami Storici

1990

La **prima implementazione** di una CNN risale al 1989 ad opera di Yann LeCun et al
Il loro modello aggregava feature inizialmente semplici in feature via via più complesse sul database delle cifre scritte a mano noto come MNIST

1998

Nel 1998 LeCun crea **LeNet-5**



2012

Nel 2012 Alex Krizhevsky et al. creano **AlexNet**

...

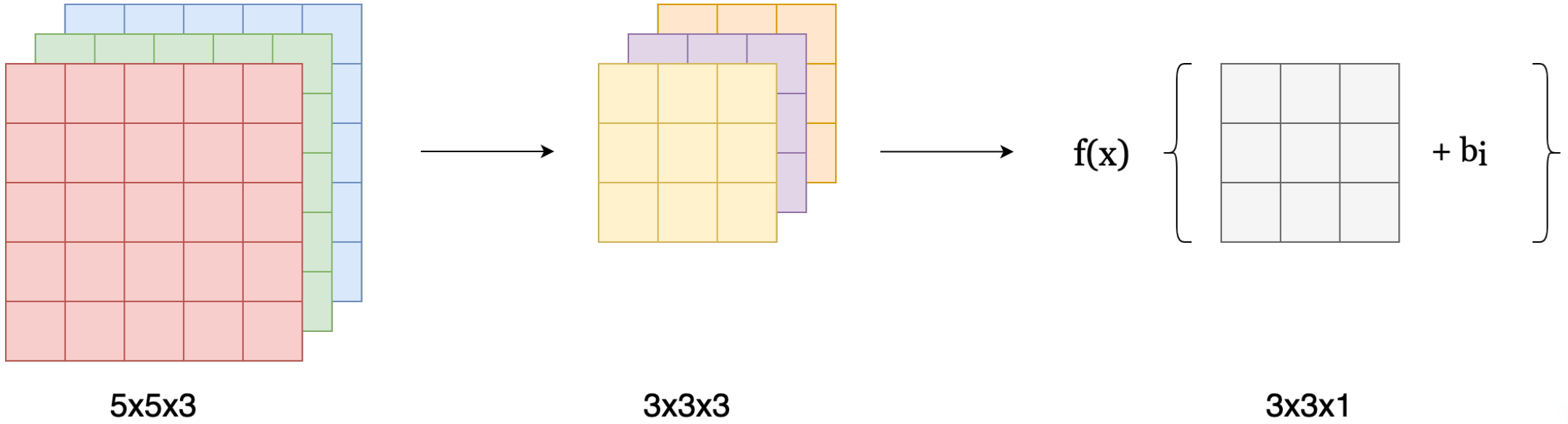
AlexNet si aggiudica il contest di **visual recognition ImageNet**
Negli anni successivi anche grazie all'aumento della capacità di calcolo dei computer e delle GPU si arriva alla stesura di molte nuove architetture *(U-Net, Inception, etc)*

Livelli convolutivi

Livelli convolutivi

- Una convoluzione non è l'unica operazione effettuata da una CNN convolutiva
- Una CNN è innanzitutto una **rete neurale**
- Al risultato di una convoluzione bisogna:
  - *Aggiungere un bias*
  - *Aggiungere una funzione di attivazione*
- Il risultato è un **livello convolutivo**

Livelli convolutivi



Livelli convolutivi

- I parametri dei singoli kernel di convoluzione sono i **parametri** del livello
- Per ogni filtro viene aggiunto un **bias**
- I risultati della convoluzione e il bias attraversano una funzione di attivazione *(es. ReLU, sigmoide, etc)* per aggiungere la non linearità al livello

Livelli convolutivi

Ricordiamo che in una Rete Neurale:

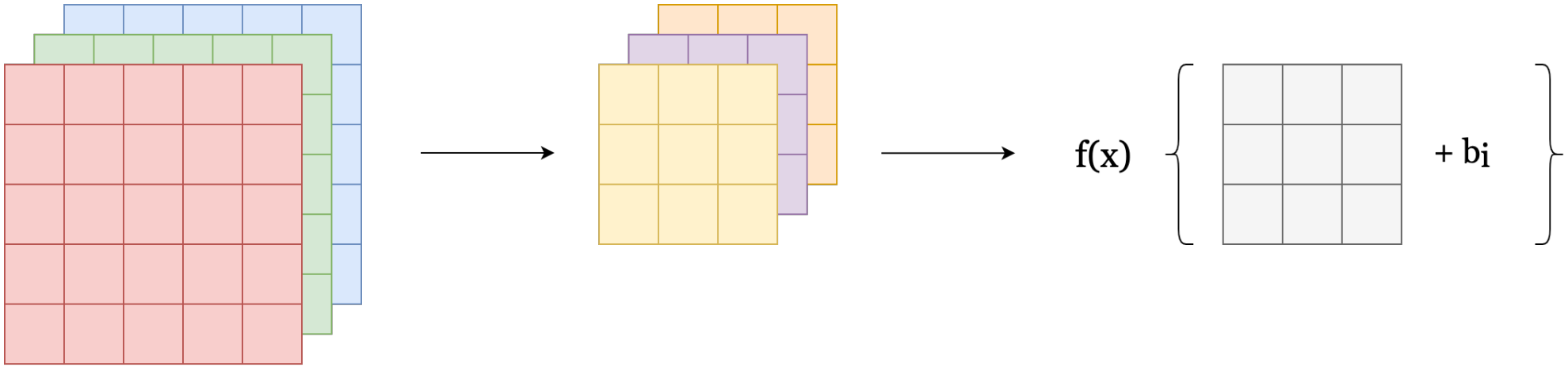
$$z^{[i]} = W^{[i]} * a^{[i-1]} + b^{[i]}$$

E che l'**output** di un livello in una rete neurale

è dato dall'applicazione della funzione di attivazione

$$a^{[i]} = g(z^{[i]})$$

Livelli convolutivi



$a^{[i-1]}$

$W^{[i]}$

$W^{[i]} * a^{[i-1]}$

Livelli convolutivi

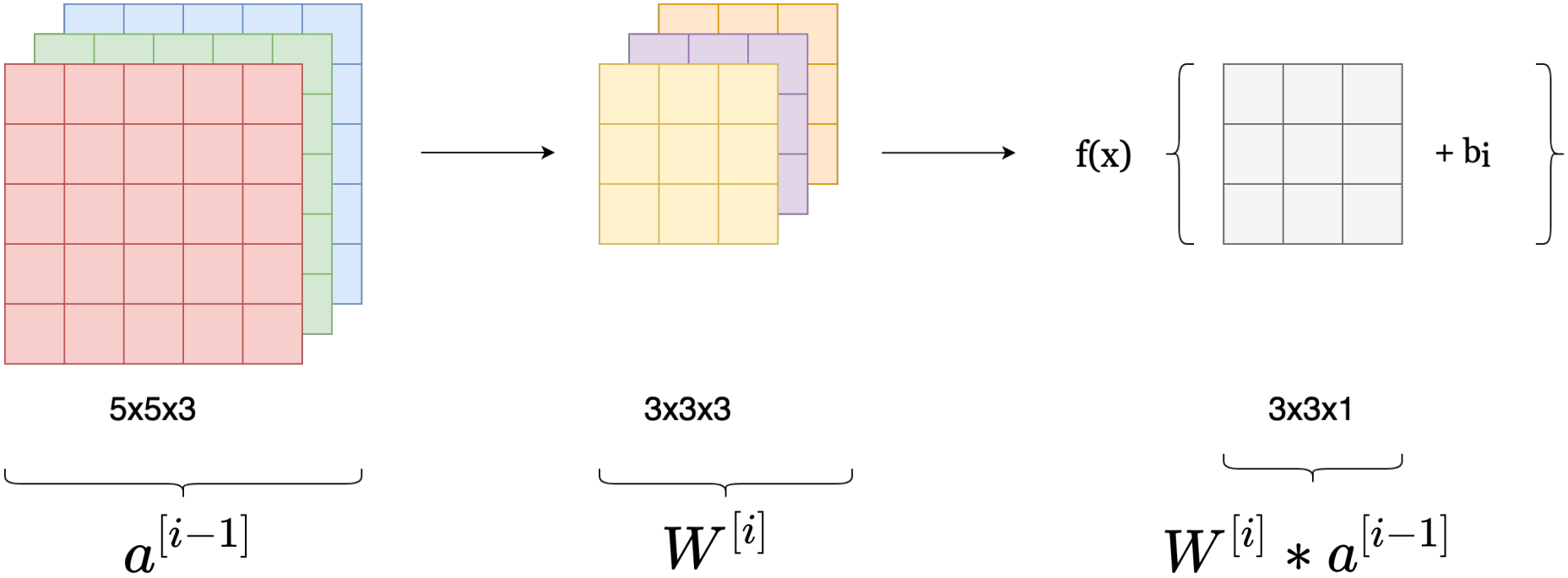
Dim. Filtro 3x3x3 = 27

N. filtri = 1

Bias = 1

Num. di parametri:

N. Filtri(Dim. Filtro + Bias) = 28



Livelli convolutivi

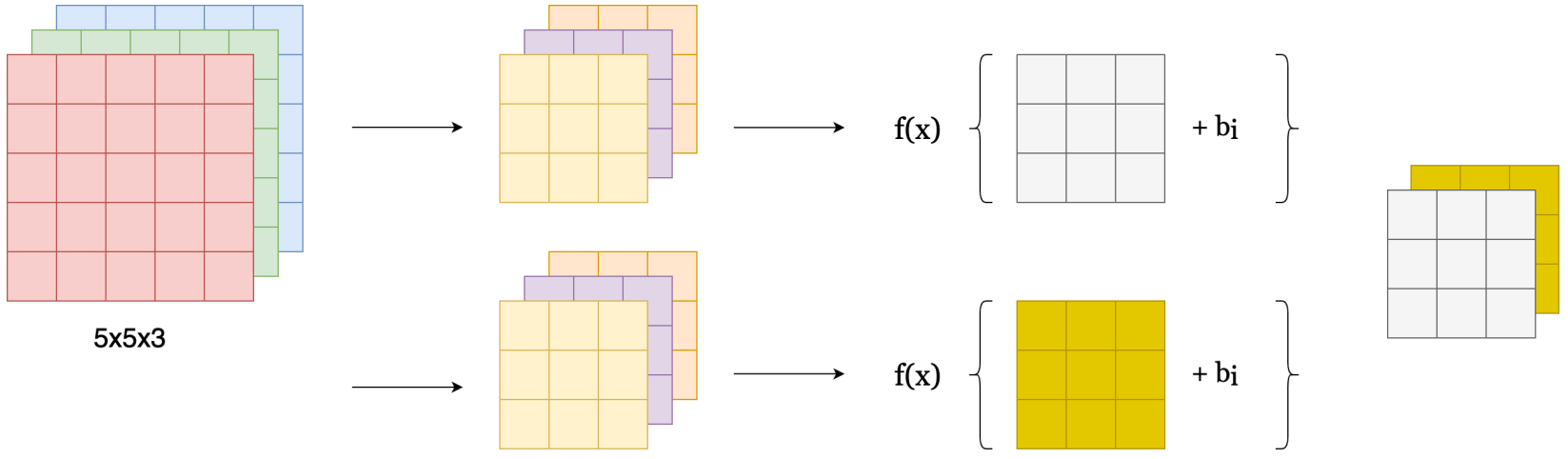
Numero di parametri:

2x(3x3x3 + 1) = 56

3x3x3 -> dimensione del filtro

2 -> numero di filtri

1 -> bias per kernel



$a^{[i-1]}$ $W^{[i]}$ $W^{[i]} * a^{[i-1]}$ $a^{[i]}$

Livelli convolutivi

$f^{[l]} = dim.\, filtro$
$p^{[l]} = dim.\, padding$
$s^{[l]} = dim.\, stride$
$n^{[l-1]} = dim.\, input$
$n^{[l]} = dim.\, output$
$n_c^{[l]} = num.\, filtri$

Ogni filtro ha dimensioni:

$f^{[l]} * f^{[l]} * n_c^{[l-1]}$

Dimensioni output livello convolutivo:

$$n^{[l]} = \left\lfloor \frac{n^{[l-1]} + 2p^{[l]} - f^{[l]}}{s^{[l]}} + 1 \right\rfloor$$

Numero di parametri:

$$(f_w^{[l]} * f_h^{[l]} * n_c^{[l-1]} + 1) * n_c^{[l]}$$

N.B. larghezza e altezza del filtro potrebbero essere diversi fra loro

Glossario

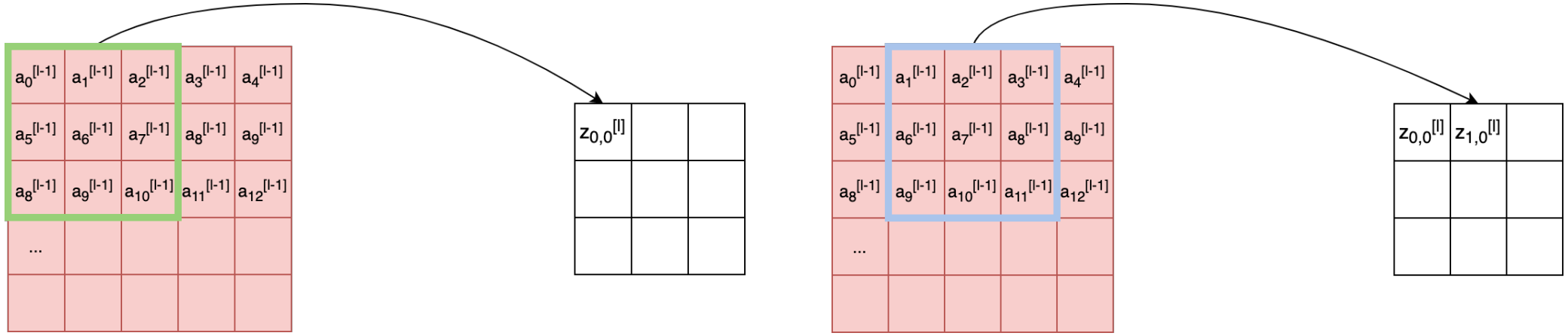
- **Kernel:** una matrice (solitamente quadrata) che performa l'operazione di convoluzione
- **Feature Map:** il risultato di un'operazione di convoluzione + funzione di attivazione + bias
- **Neurone**: È **un singolo elemento di una feature map**
- **Filtro:** È una serie di kernel
- **Receptive Field**: È la porzione dell'input di un livello convolutivo «collegato» ad un certo neurone di uscita

Local Perceptive Fields

- In una Rete Neurale i neuroni di un livello ricevono gli input da **tutti** i neuroni del livello precedente
- In una CNN un neurone è connesso solo **ad una parte dell'input**:
  - Ogni neurone è "responsabile" solo per una definita porzione dei dati di input
  - Questa porzione prende il nome di **Local Perceptive Field**

Un **Local Receptive Field** è l'area ben precisa e definita dell'input con cui un neurone in un livello convolutivo è moltiplicato durante il processo di convoluzione.

Local Perceptive Fields



- I due neuroni $z_{0,0}$ e $z_{1,0}$ sono connessi solo ai propri Local Receptive Field (bordi verdi e blu), non a tutto l'input
- Questo contribuisce in maniera determinante all'**invarianza**

Caratteristiche delle CNN

- **Invariance:** È l'abilità delle reti di **riconoscere** pattern comuni **indipendentemente** da trasformazioni all'interno dell'input
- Invariance to:
  - **Translation**
  - **Rotation**
  - **Scale**
  - **Deformation**
  - **Noise**

Caratteristiche delle CNN

- Come si raggiunge l'invarianza nelle CNN?
  - Weight Sharing
  - Pooling Operations
  - Hierarchical Feature Learning
  - Data augmentation
  - Regularization
  - Transfer Learning

Weights Sharing

- Il concetto di base è: *un filtro capace di estrarre una feature (es. un bordo verticale) in un particolare local receptive field potrebbe essere utile anche in* ***un'altra parte dell'immagine***

- Un filtro scorre sull'input generando una **feature map**. Lo stesso filtro viene usato in differenti parti dell'immagine e i suoi parametri sono sempre gli stessi.

- I filtri che generano le stesse feature map in un livello sono **condivisi**

- Questo abbatte il numero di parametri che la rete deve imparare

Weights Sharing

- Il numero di parametri in una CNN quindi **non dipende dalla dimensione dell'input**

- Il numero di parametri dipende **esclusivamente**:
  - Dalla dimensione dei filtri
  - Dal numero di filtri

- Questo genera meno connessioni e meno calcoli e garantisce una maggiore velocità

- I parametri appresi sono sfruttati in altre parti dell'immagine

Weights Sharing

- Calcoliamo il **numero di parametri** in un livello convolutivo di una CNN
  - s = 4
  - p = 0
  - input = 227x227x3
  - n. filtri = 96
  - dim. filtri = 11x11x3

- Calcoliamo innanzitutto la dimensione dell'output:
  - (227 – 2*0 – 11)/4 + 1 = **55**
  - Avremo un output di **55x55x96**

Weights Sharing

- Layer convolutivo
  - Dim. Filtri: 11x11x3
  - N. Filtri: 96
  - Num. Parametri da addestrare: (11x11x3 + 1)x96 = **34 944** parametri

Downsampling/Upsampling

- Servono a **modificare** le dimensioni delle feature map in una CNN
- Aiutano a creare una rappresentazione **gerarchica** delle feature estratte da un input
- Downsampling: **cattura informazioni di alto livello**
- Upsampling: **ripristina** dettagli e informazioni locali

Livelli di Pooling

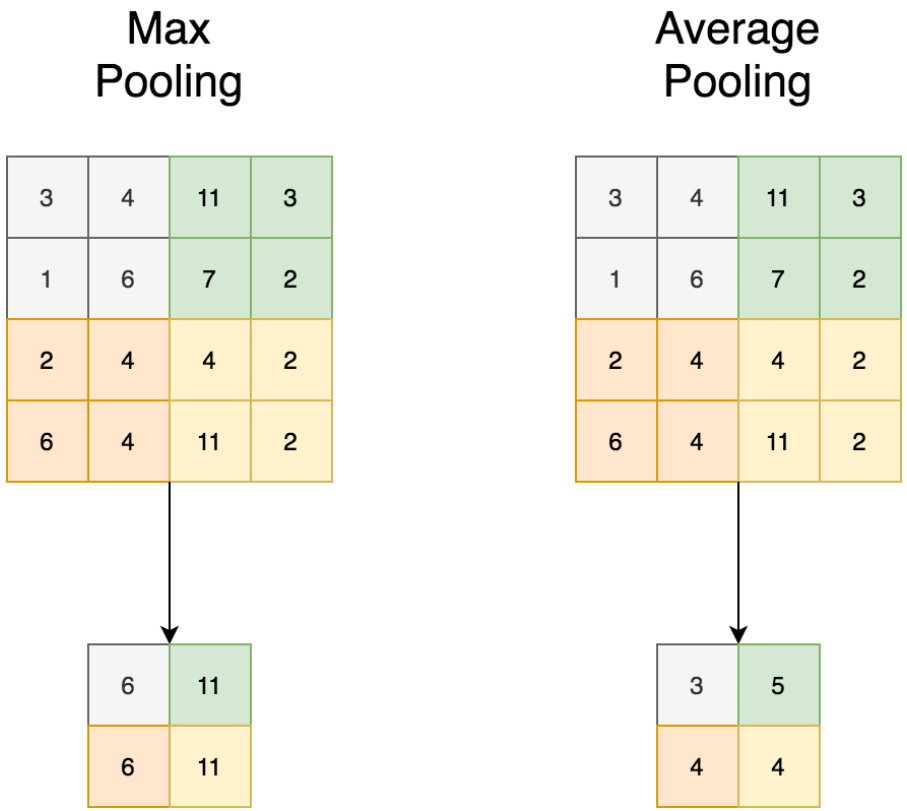
- Il pooling aiuta a rendere le feature map **invarianti** a piccole traslazioni spaziali nell'input
  - Se una feature nell'input si sposta di una quantità *ragionevole (es. Un pattern che vogliamo riconoscere, un volto, un bordo)* il valore di gran parte degli output non cambierà

- L'invarianza del modello ai piccoli spostamenti è molto utile se siamo interessati alla **presenza** di una feature più che alla sua esatta posizione

- Consiste nella suddivisione in gruppi di pixel e in un'operazione matematica *(media, max, min, etc)* al fine di associare 1 pixel in uscita ad un gruppi di pixel

- Il risultato è una diminuzione del volume di uscita al livello, e una conseguente **diminuzione dei parametri** da apprendere

Max pooling/Average Pooling



Upsampling

- Consiste nel processo inverso al downsampling
- Consente di avere un volume di output **maggiore** del volume di input
- Si può ottenere con dei **Transposed Layer Convolution**
- È una convoluzione con un volume di uscita **maggiore** dell'input
- **Non** è l'inverso di una convoluzione, ma piuttosto una convoluzione su un input "modificato"

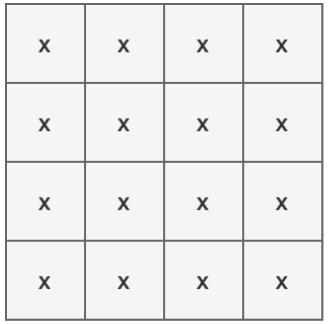
Upsampling

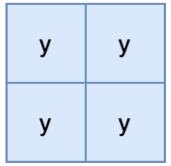
- Partendo da valori definiti di stride e padding:
  - Si "**allargano**" fra loro i pixel dell'immagine di una quantità pari ad (s – 1)

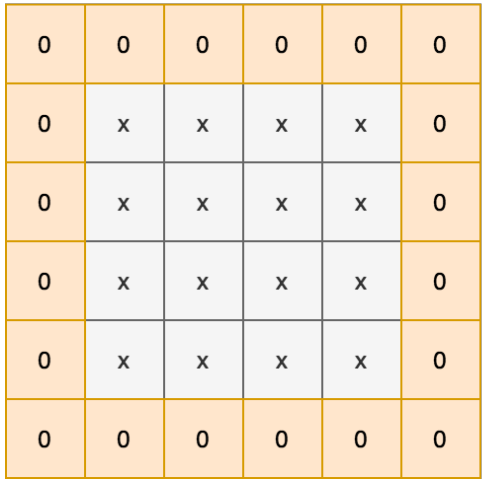
    (i pixel nuovi si settano a 0)
  - Si aggiunge una cornice di zeri di profondità pari a (f – p – 1)
  - Si fa la convoluzione con stride s' pari a 1 (si considera ogni pixel)
  - Dim. Output $(n_c - 1) * s + f - 2p$

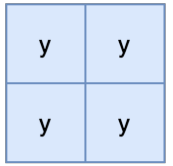
Upsampling – Transposed Convolutional Layer

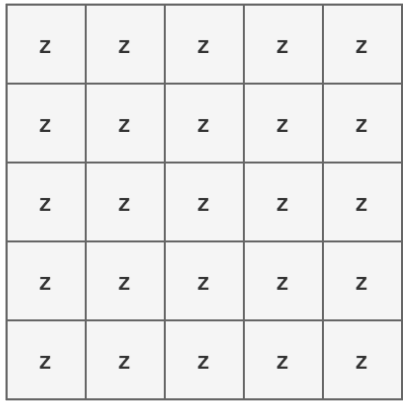
- Dim. Immagine: 4x4
- Dim. kernel: 2x2
- Padding: 0
- Stride: 1
- Dim. Output : 5x5





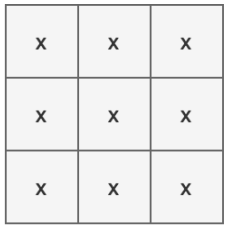


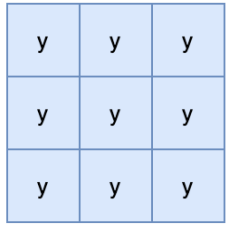


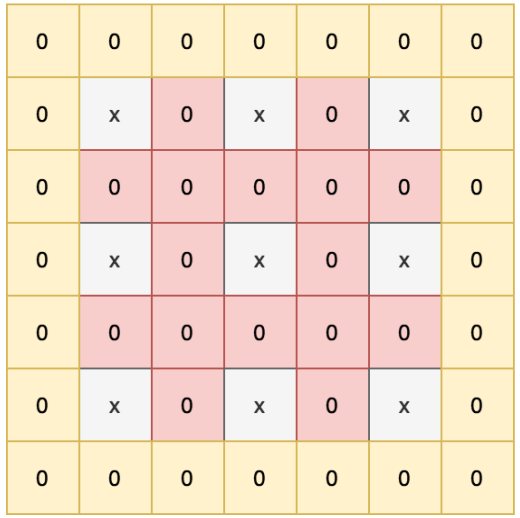


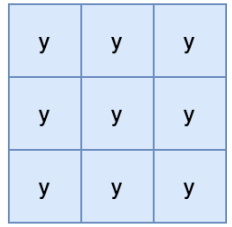
Upsampling – Transposed Convolutional Layer

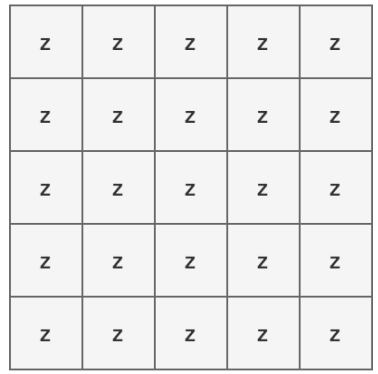
- Dim. Immagine: 3x3
- Dim. kernel: 3x3
- Padding: 1
- Stride: 2
- Dim. Output : 5x5



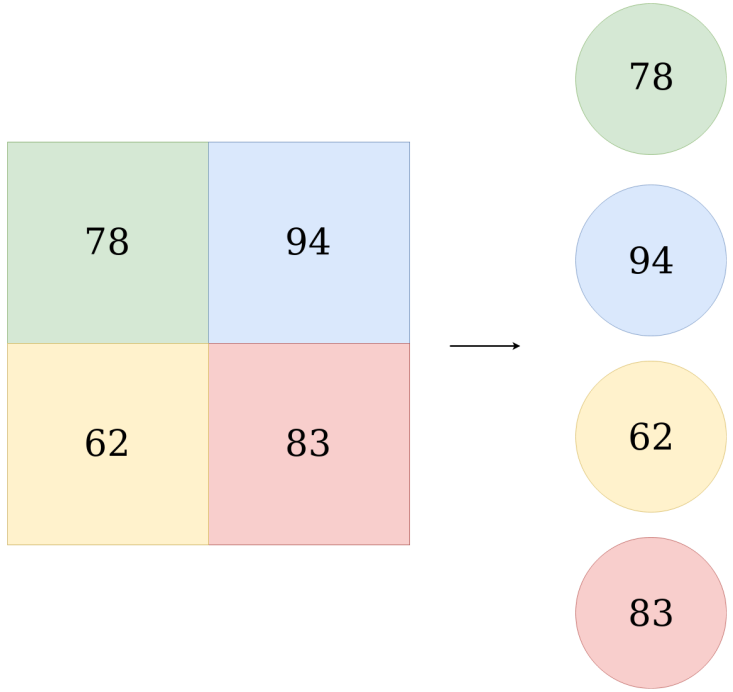








Livelli FC



I livelli **Fully Connected** sono i tradizionali livelli che abbiamo già visto nelle reti neurali

L'input di un livello FC va ***flattened***, perché in uscita da un livello di pooling o da un livello convolutivo abbiamo dei volumi

Cosa apprende una CNN?

- Durante l’addestramento una CNN apprende:
  - **Parametri** dei Kernel/filtri
  - **Pesi** dei livelli FC
  - **Bias**
  - **Parametri di Normalizzazione/Regolarizzazione**

Come apprende una CNN?

- Algoritmo è sempre la **discesa del gradiente** (in pratica però vengono usate delle varianti)
  - **Step 1:** Setup architettura (es. 1 livello convolutivo con 1 filtro con kernel 3x3)
  - **Step 2**: Inizializzazione (es. 9 pesi del kernel casuali, 1 bias pari a 0)
  - **Step 3**: Scelta loss function (es. cross entropy)
  - **Step 4**: Forward pass (es. applicazione del filtro, supponiamo di avere feature map 30x30)
  - **Step 5**: Calcolo dell'errore
  - **Step 6:** Backpropagation (calcolo dei gradienti per tutti i 10 parametri)
  - **Step 7: Aggiornamento dei parametri**
  - Ripetere passaggi 4-7 fino a convergenza
  - **Step 8:** Valutazione del modello su un test set mai visto dalla rete