

# Analisi di Immagini e Video (Computer Vision)

Giuseppe Manco

# Outline

- Segmentation
- Approcci classici
- Deep Learning for Segmentation

# Crediti

- Slides adattate da vari corsi e libri
  - Computational Visual Recognition (V. Ordonez), CS Virginia Edu
  - Computer Vision (S. Lazebnik), CS Illinois Edu

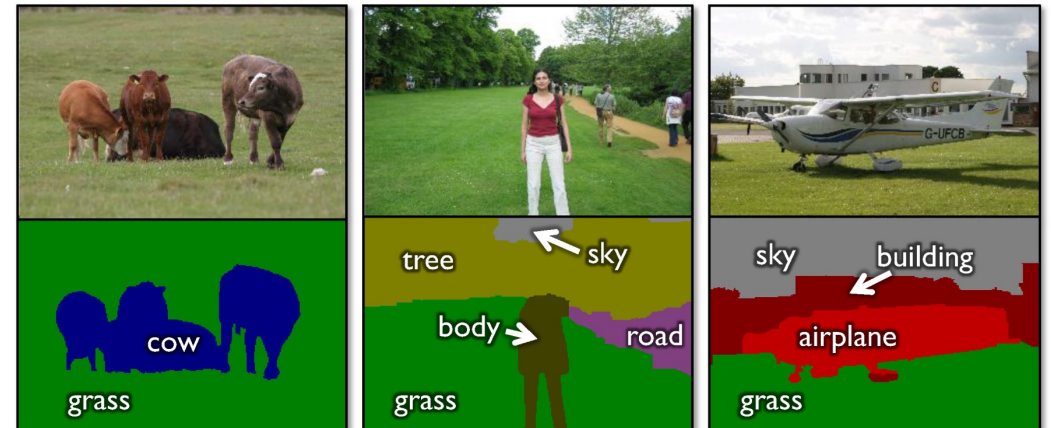
# Approcci supervisionati

- L'approccio basato su CRF è semi-supervisionato
- Possiamo renderlo supervisionato?
  - Parametrizziamo gli unary e binary potentials
    - E.g.,  $p(y_i|x_i; \theta) = \frac{1}{Z} \exp(w_{y_i} \cdot F(x_i))$
  - Apprendiamo i parametri che minimizzano l'energia media su tutti gli esempi



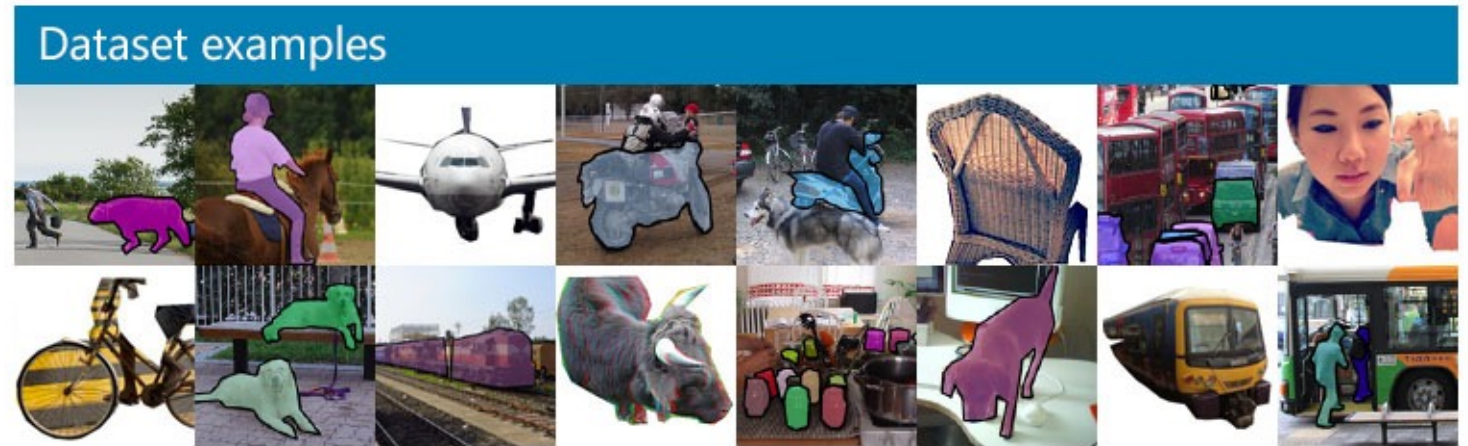
# Semantic segmentation, object detection

- Problema
  - Etichettare ogni pixel con una classe
  - Multi-class problem
- Utilizzo di dati già etichettati
  - Pascal VOC
  - MS COCO



object classes	building	grass	tree	cow	sheep	sky	airplane	water	face	car
bicycle	flower	sign	bird	book	chair	road	cat	dog	body	boat

# MS-COCO



- Large-scale dataset for object detection, segmentation and captioning
  - 330K images (>200K labeled)
  - 1.5 million object instances
  - 80 object categories
  - 91 stuff categories
  - 5 captions per image
  - 250,000 people with keypoints

# Perché Deep Learning?

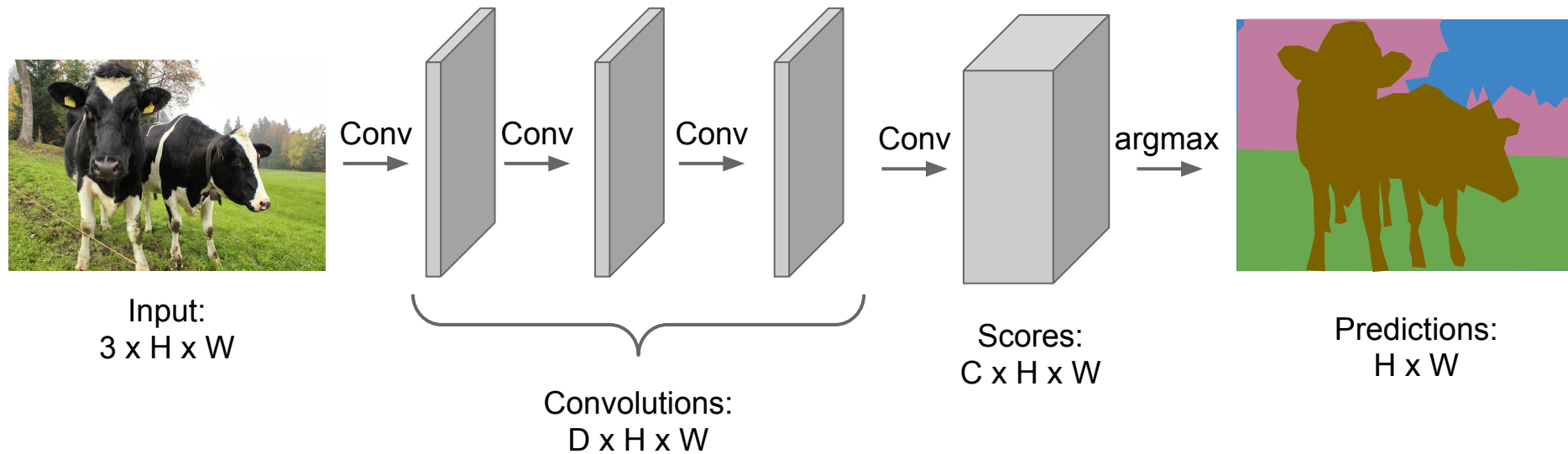
- Stesso principio dell'object detection
  - Convolutional features, learned from training data
- Accuratezza
- Velocità

# Approcci

- Approcci downsampling-upsampling
- Metodi multi-scala

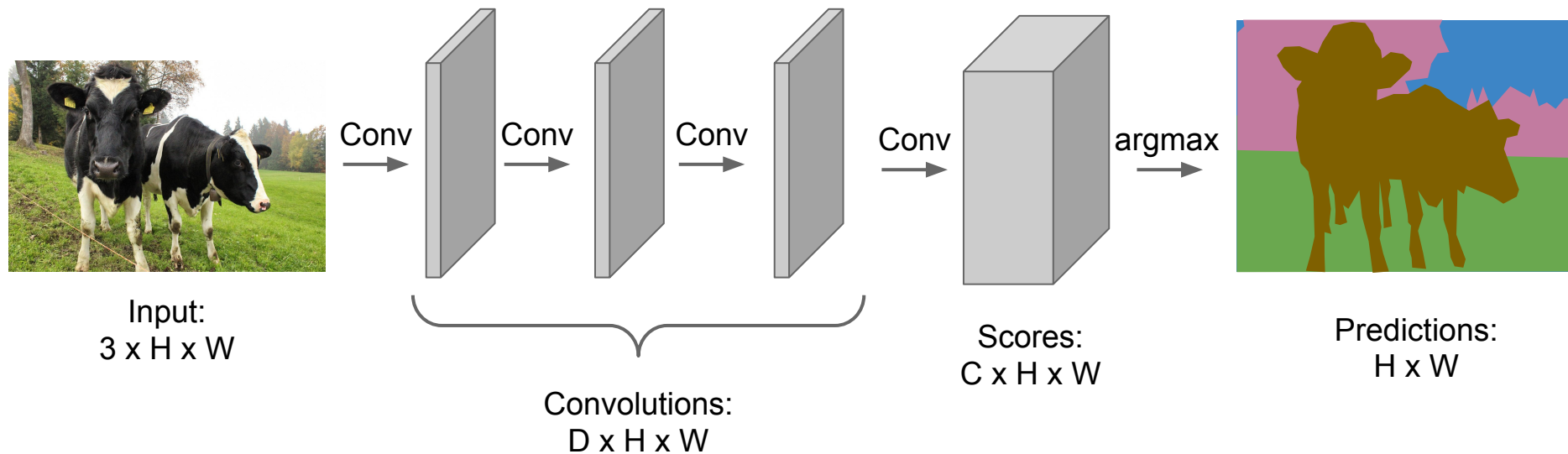
# Fully Convolutional Networks

- Utilizziamo i layer convoluzionali per fare le predizioni sui vari pixel



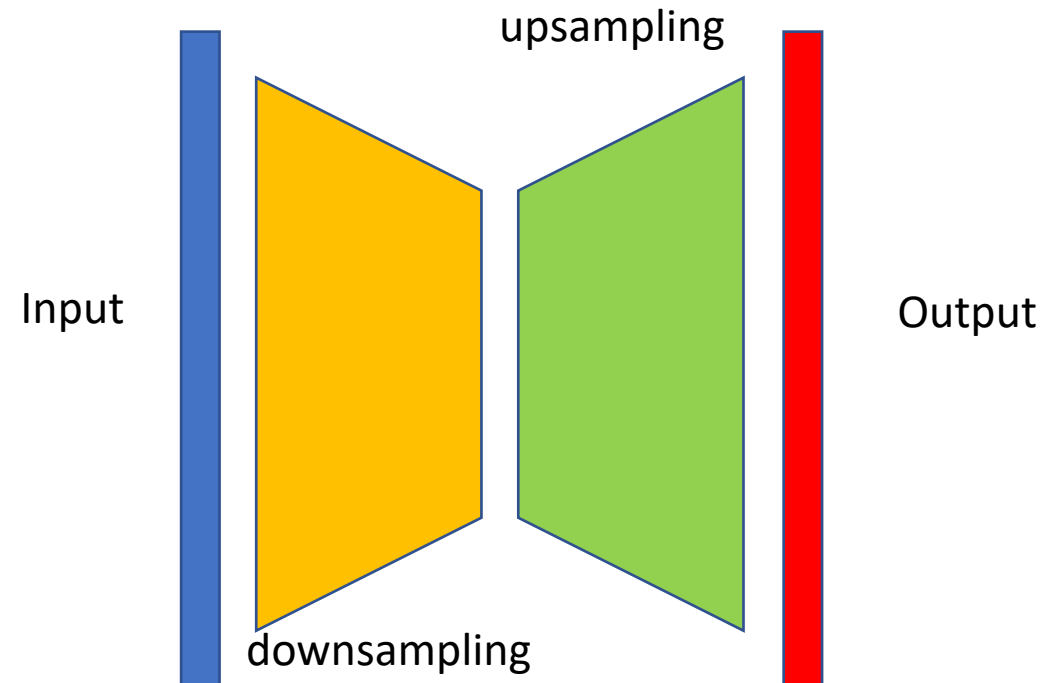
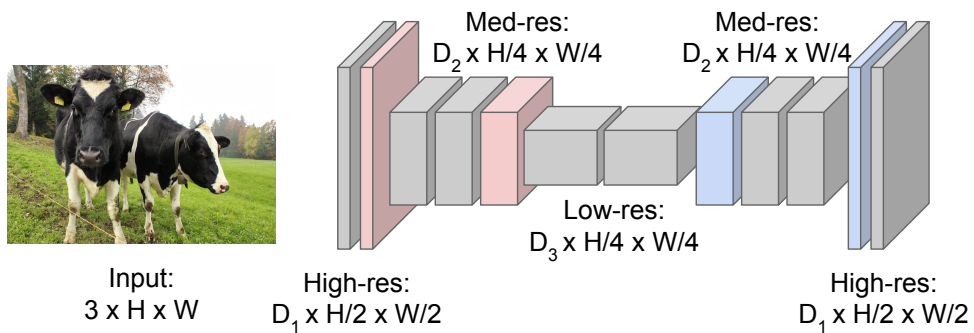
# Fully Convolutional Networks

- Utilizziamo i layer convoluzionali per fare le predizioni sui vari pixel
  - Ma fare convoluzioni su feature map grandi è costoso



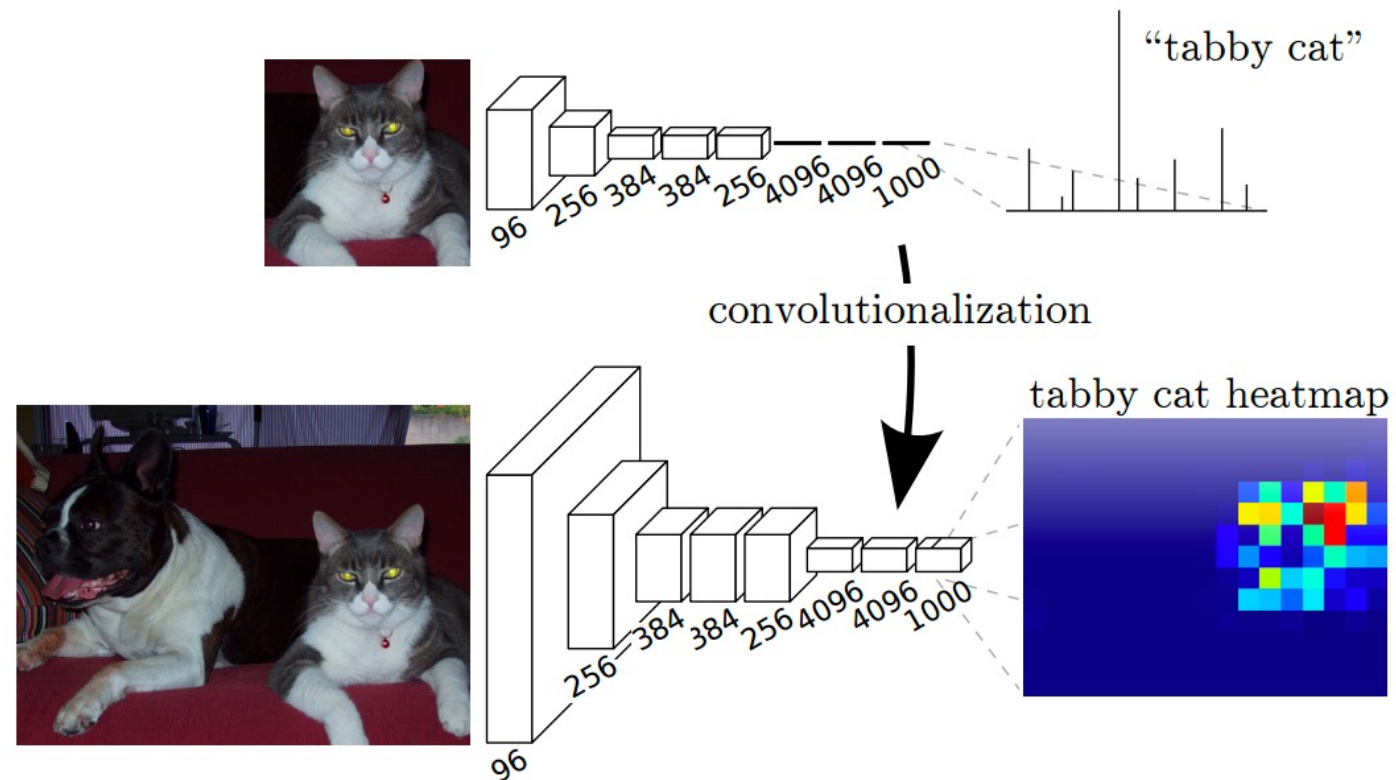
# Fully Convolutional Networks

- Soluzione
  - Architettura Encoder-Decoder



# Convolutionalization

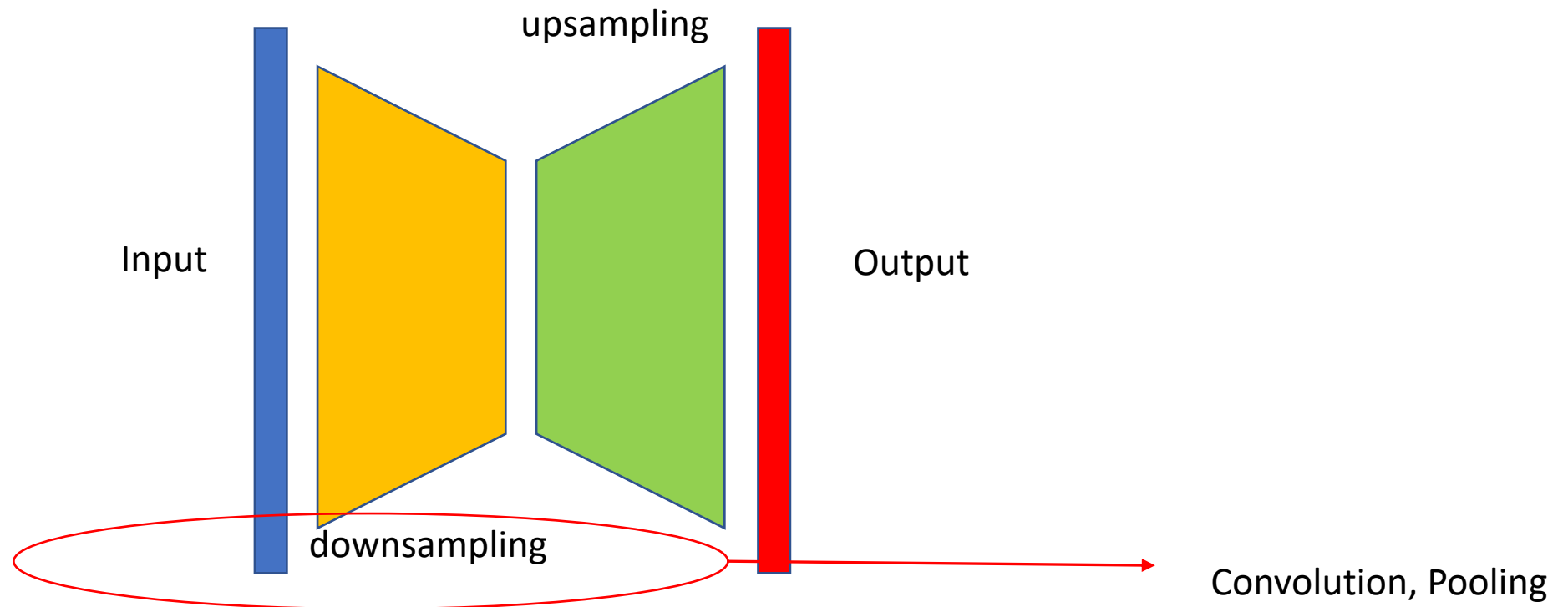
- Fully Convolutional Layers
- Faster-RCNN, SSD





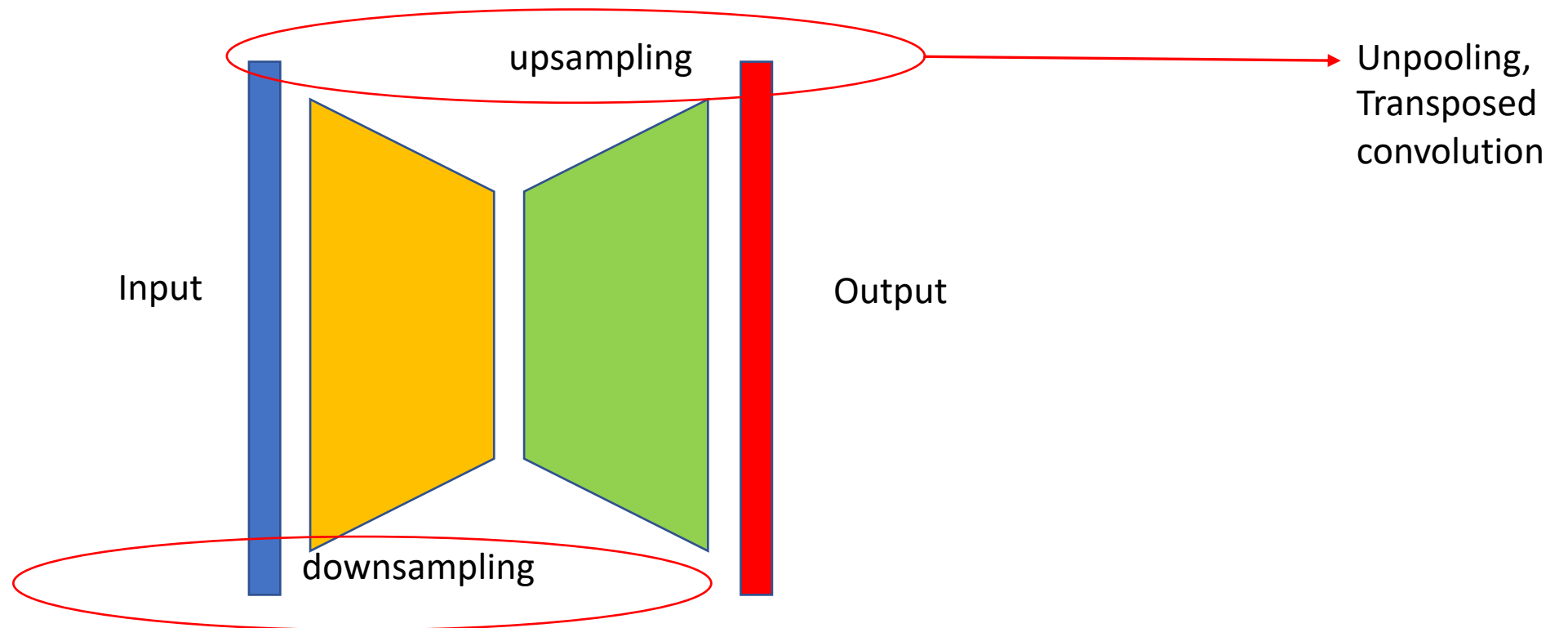
# Fully Convolutional Networks

- Architettura Encoder-Decoder



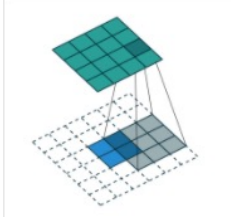
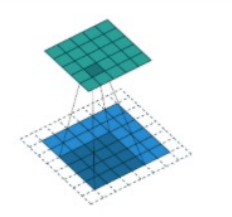
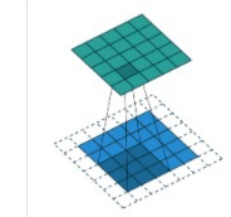
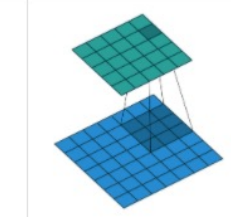
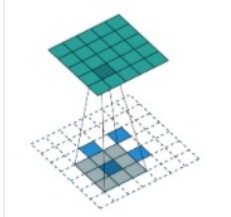
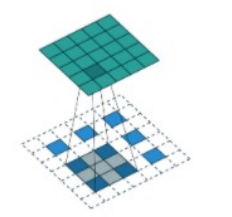
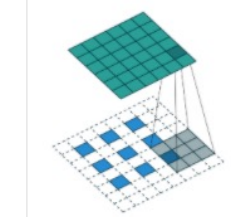
# Fully Convolutional Networks

- Architettura Encoder-Decoder



# Up-sampling Convolutions

- Upsampling
  - Da un'input a bassa risoluzione si passa ad uno a più alta risoluzione
- Transposed Convolution
  - Qual è la relazione?
    - Suggerimento: invertiamo le relazioni originarie

			
No padding, no strides, transposed	Arbitrary padding, no strides, transposed	Half padding, no strides, transposed	Full padding, no strides, transposed
			
No padding, strides, transposed	Padding, strides, transposed	Padding, strides, transposed (odd)	

[https://github.com/vdumoulin/conv\\_arithmetic](https://github.com/vdumoulin/conv_arithmetic)

# Transposed Convolution

Input

0	1
2	3

# Transposed Convolution

Input

0	1
2	3

Kernel

0	1
2	3

# Transposed Convolution

Input

0	1
2	3

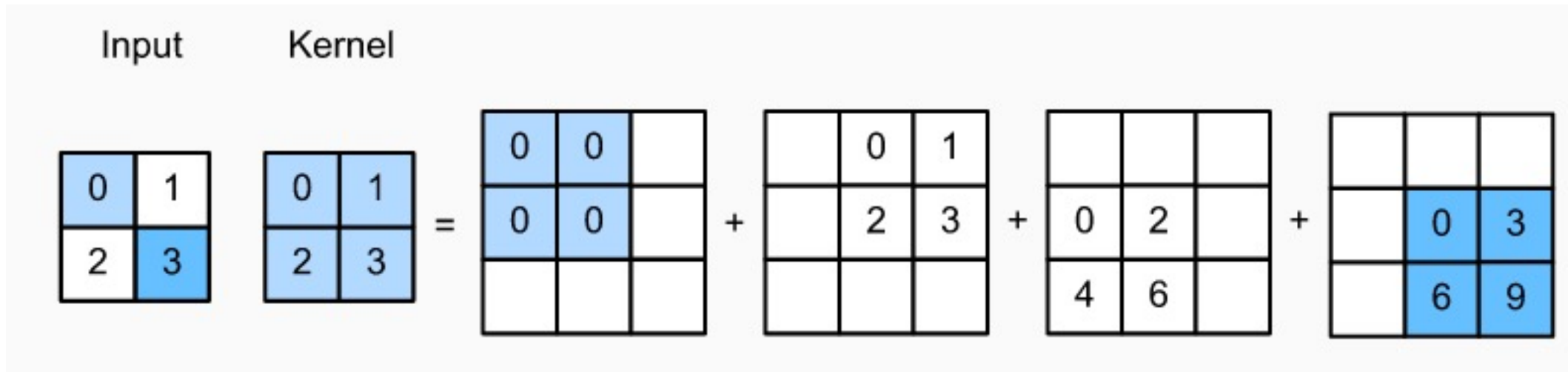
Kernel

0	1
2	3

**Output**


# Transposed Convolution

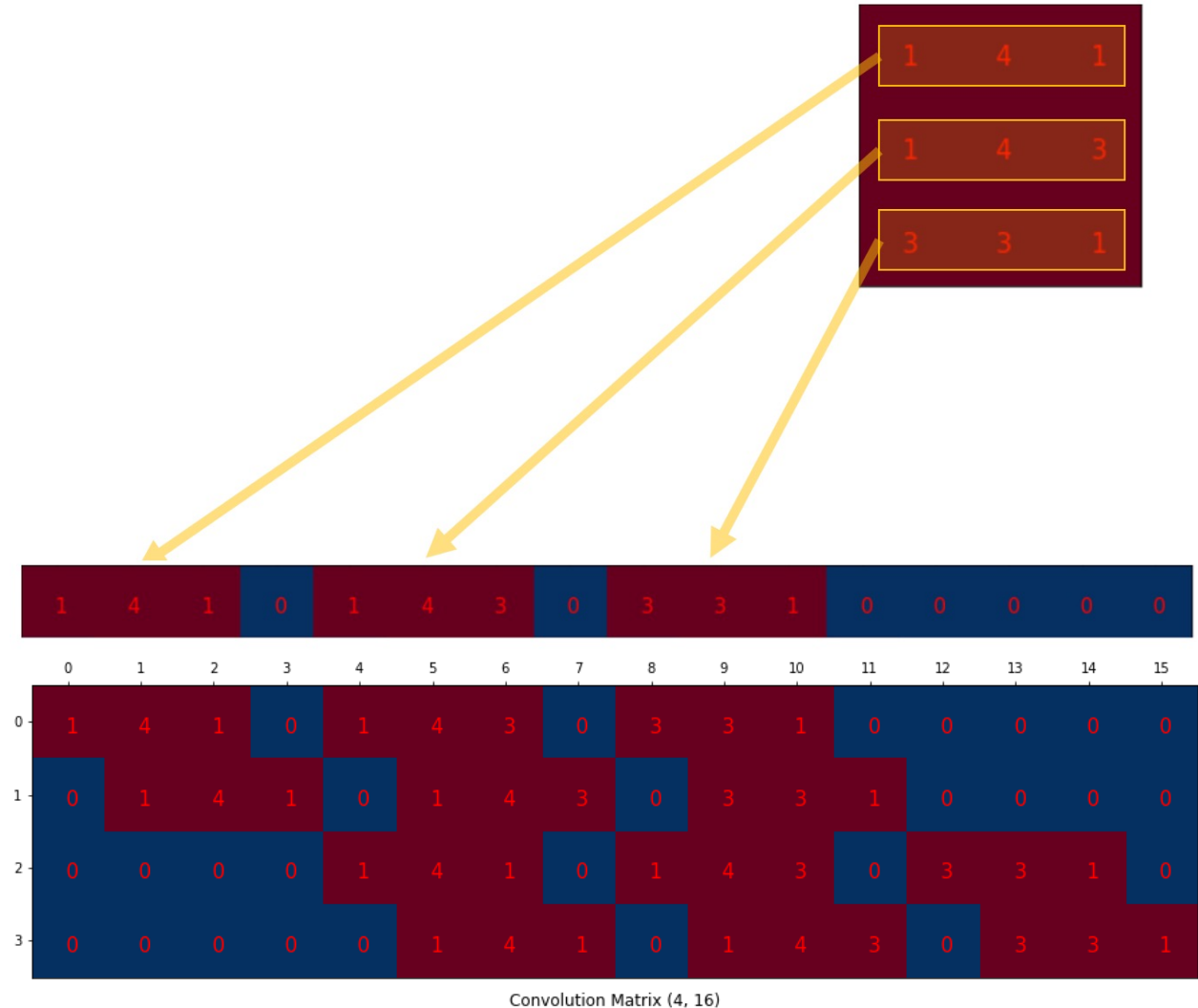
- Ogni valore si distribuisce su un intorno dell'output in base al kernel.



- La distribuzione viene guidata da padding e stride

# Convolution e transposed convolution

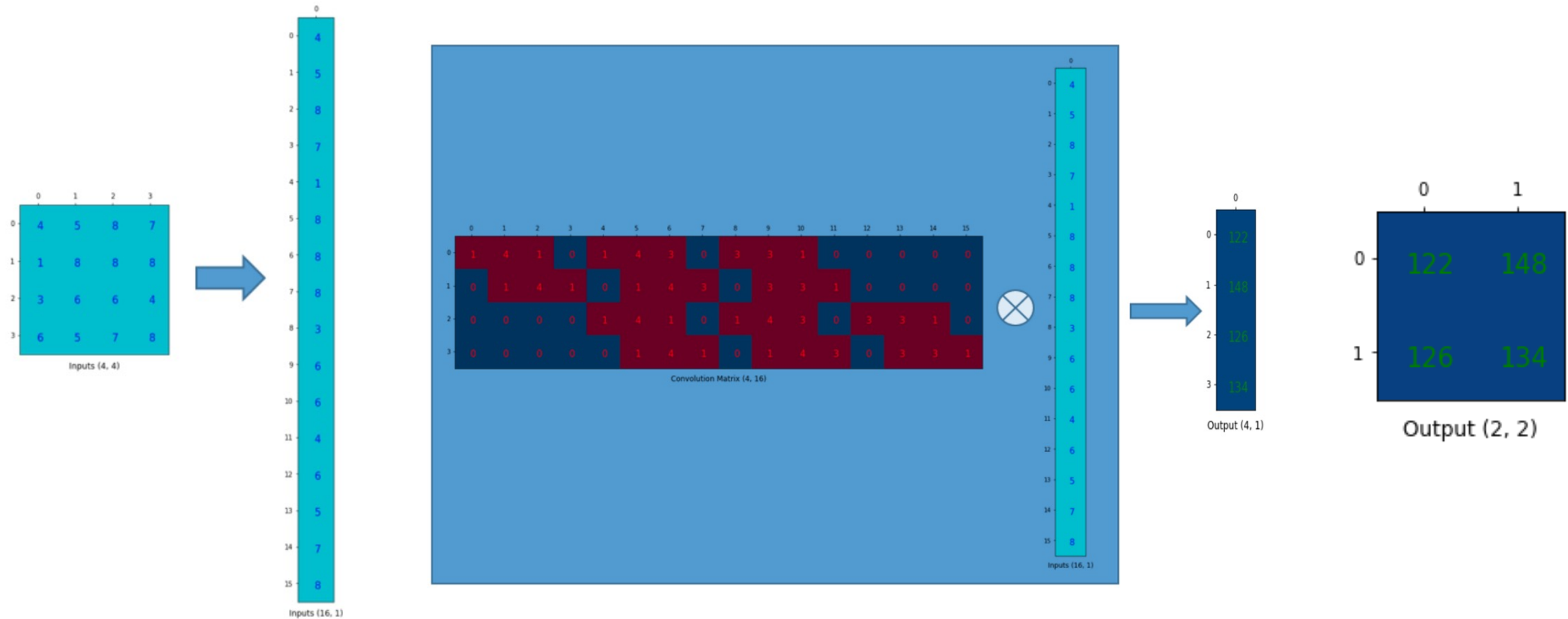
- Ogni riga definisce un'operazione di convoluzione
  - Filtro 3x3, input 4x4
    - No padding, no strides, no dilation





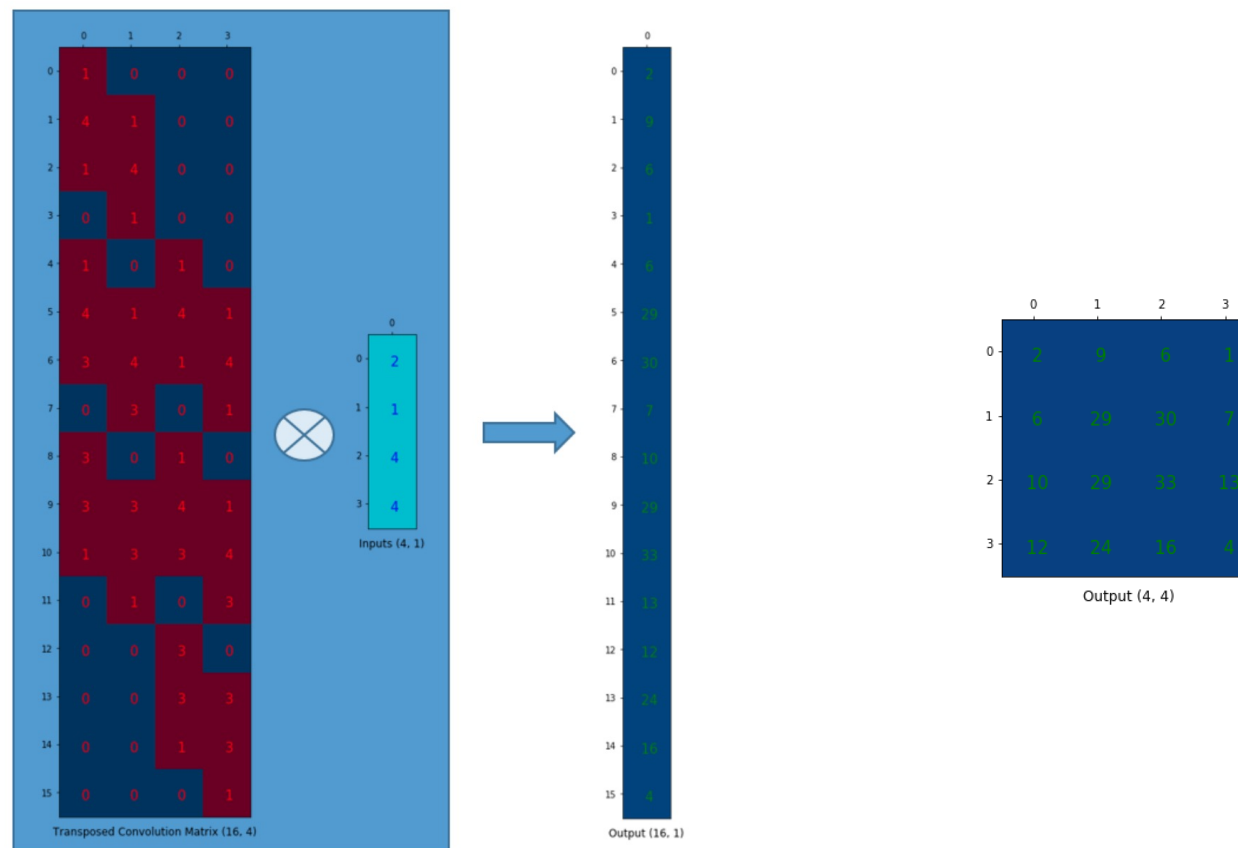
# Convolution e transposed convolution

- Ogni riga definisce un'operazione di convoluzione



# Convolution e transposed convolution

- Trasponendo la matrice di convoluzione, otteniamo l'operazione opposta

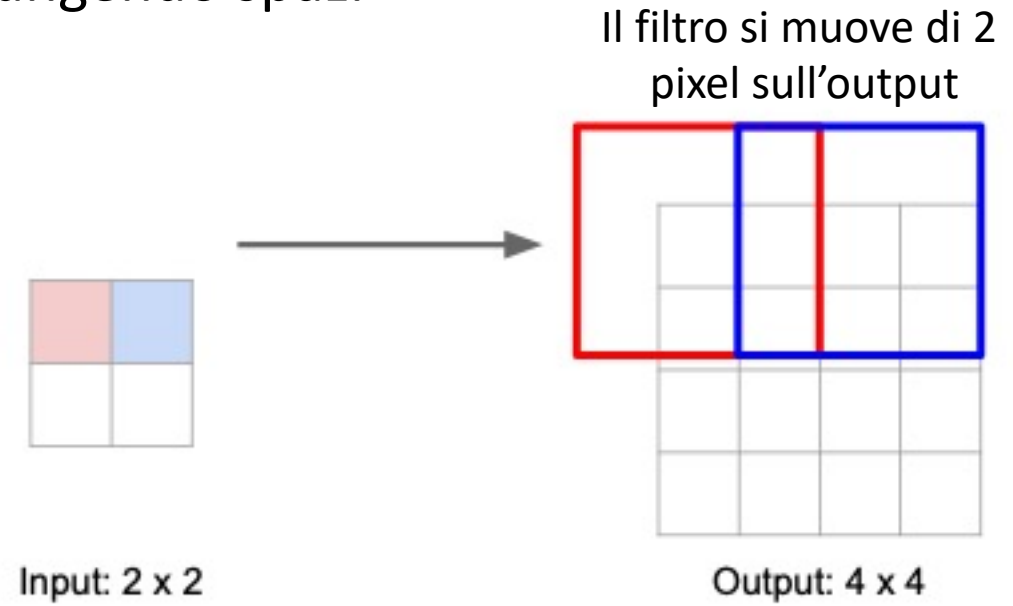




# ConvTranspose, Stride

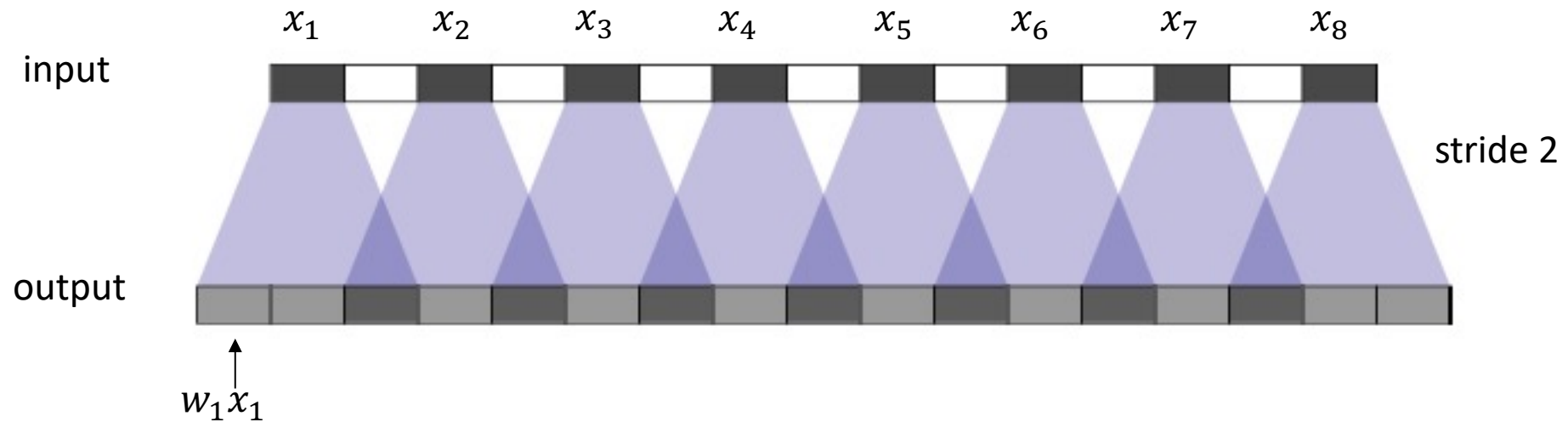
- Espande l'output
  - Di conseguenza «fraziona» l'input aggiungendo spazi

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	
1																								
2																								
3																								
4																								
5																								
6																								
7																								
8																								
9																								
10																								
11																								



# ConvTranspose, checkerboarding

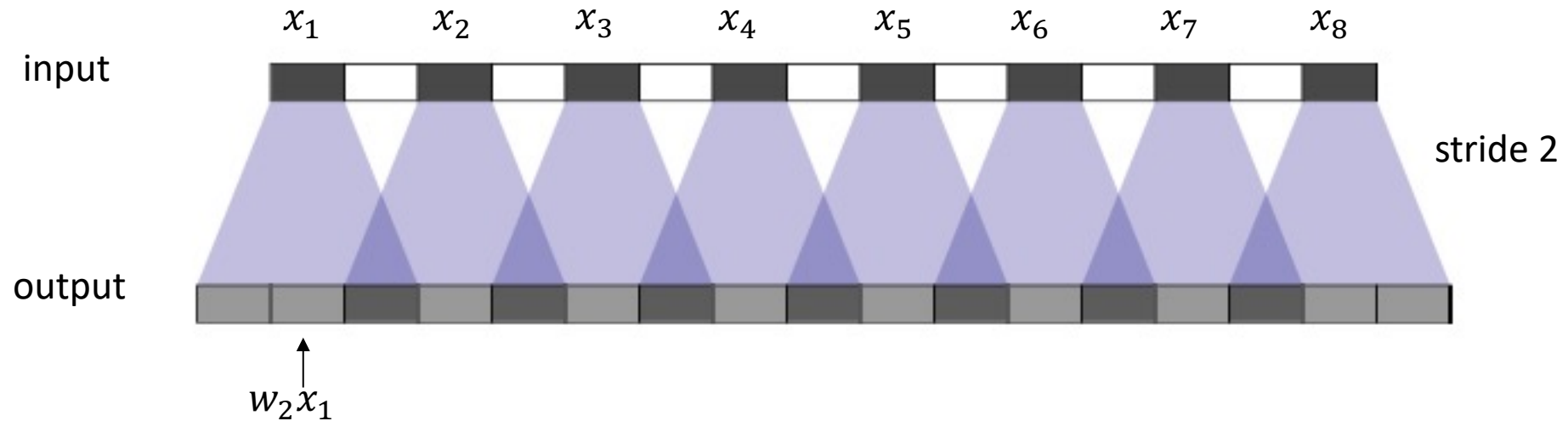
- Filter  $[w_1, w_2, w_3]$ , *output stride* = 2



Animation: <https://distill.pub/2016/deconv-checkerboard/>

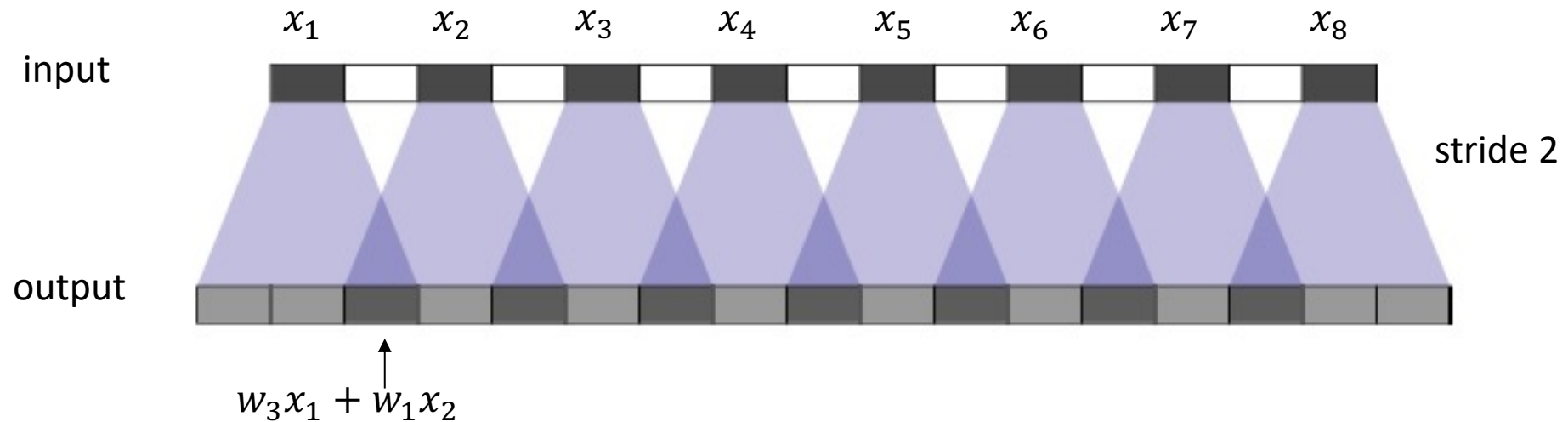
# ConvTranspose, checkerboarding

- Filter  $[w_1, w_2, w_3]$ , *output stride* = 2



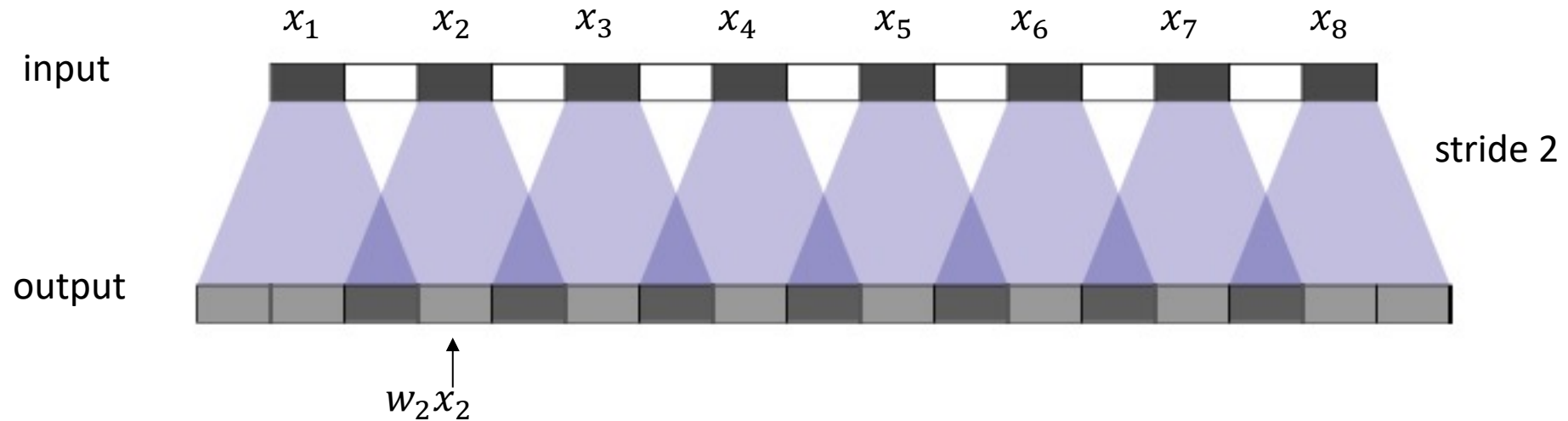
# ConvTranspose, checkerboarding

- Filter  $[w_1, w_2, w_3]$ , *output stride* = 2



# ConvTranspose, Checkerboarding

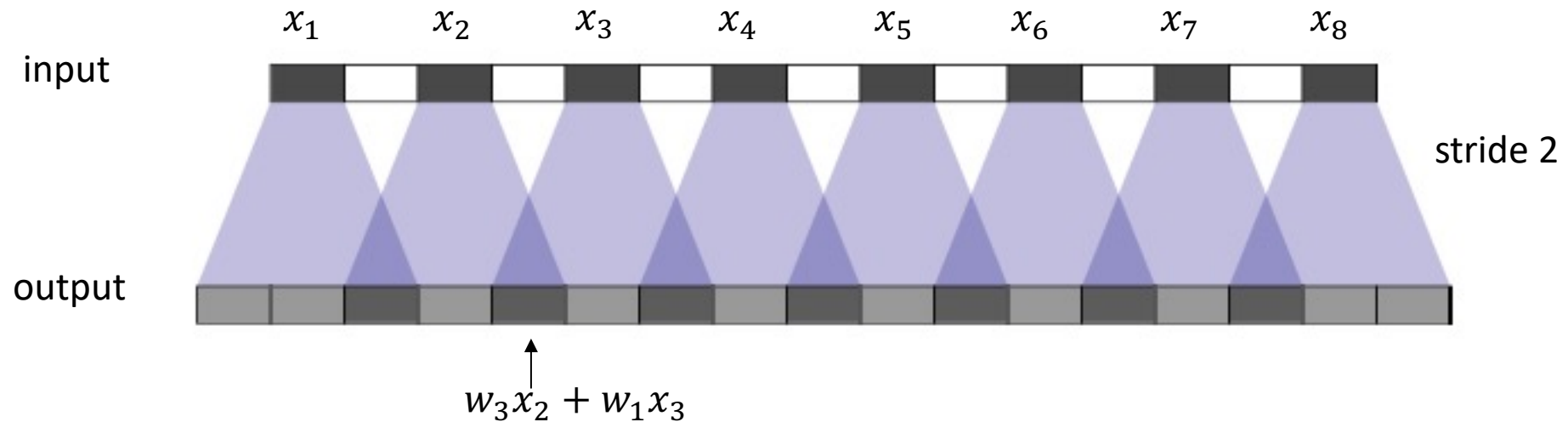
- Filter  $[w_1, w_2, w_3]$ , *output stride* = 2





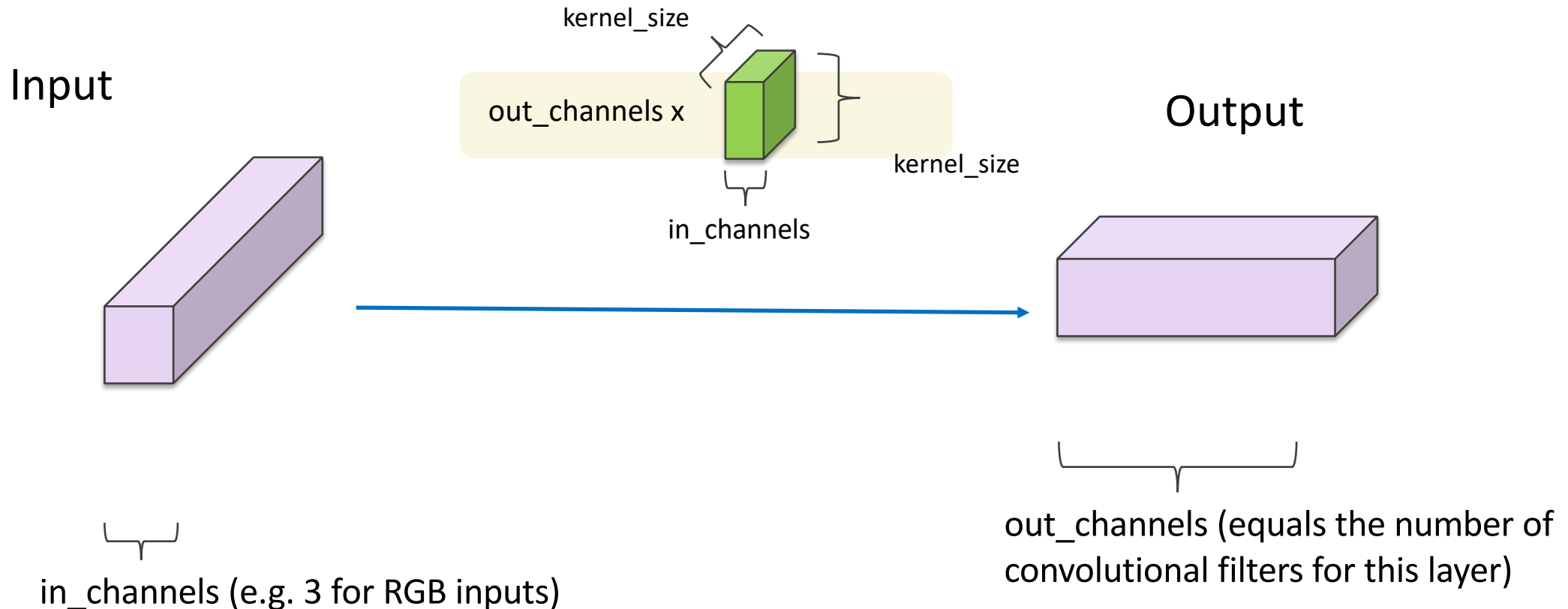
# ConvTranspose, Checkerboarding

- Filter  $[w_1, w_2, w_3]$ , *output stride* = 2



# Transposed Convolution in Pytorch

```
CLASS torch.nn.ConvTranspose2d(in_channels, out_channels, kernel_size, stride=1, padding=0, output_padding=0, groups=1, bias=True, dilation=1, padding_mode='zeros')
```



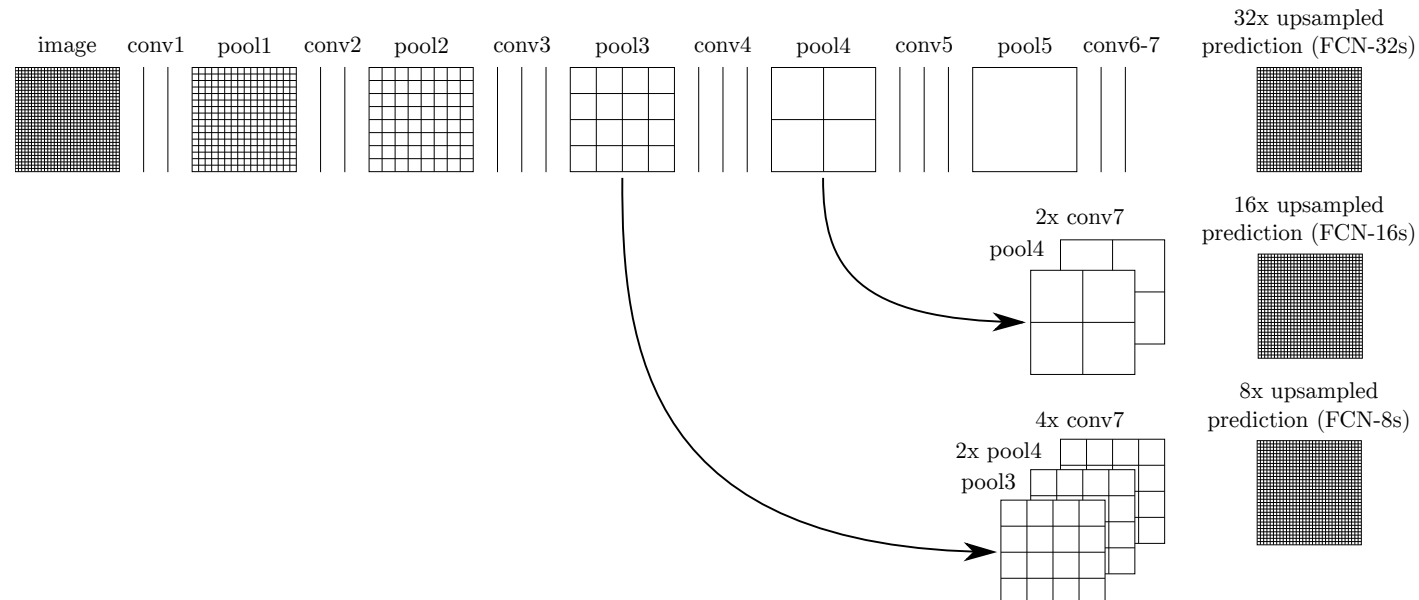
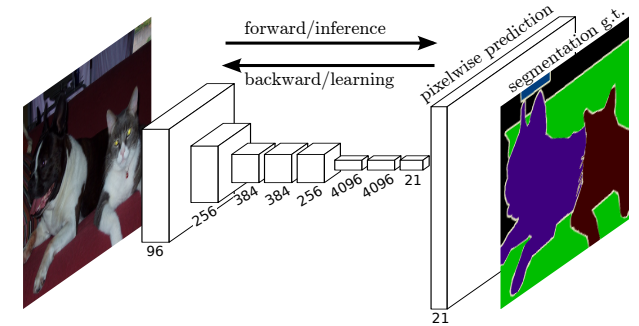
# FCN: architettura

- Principio

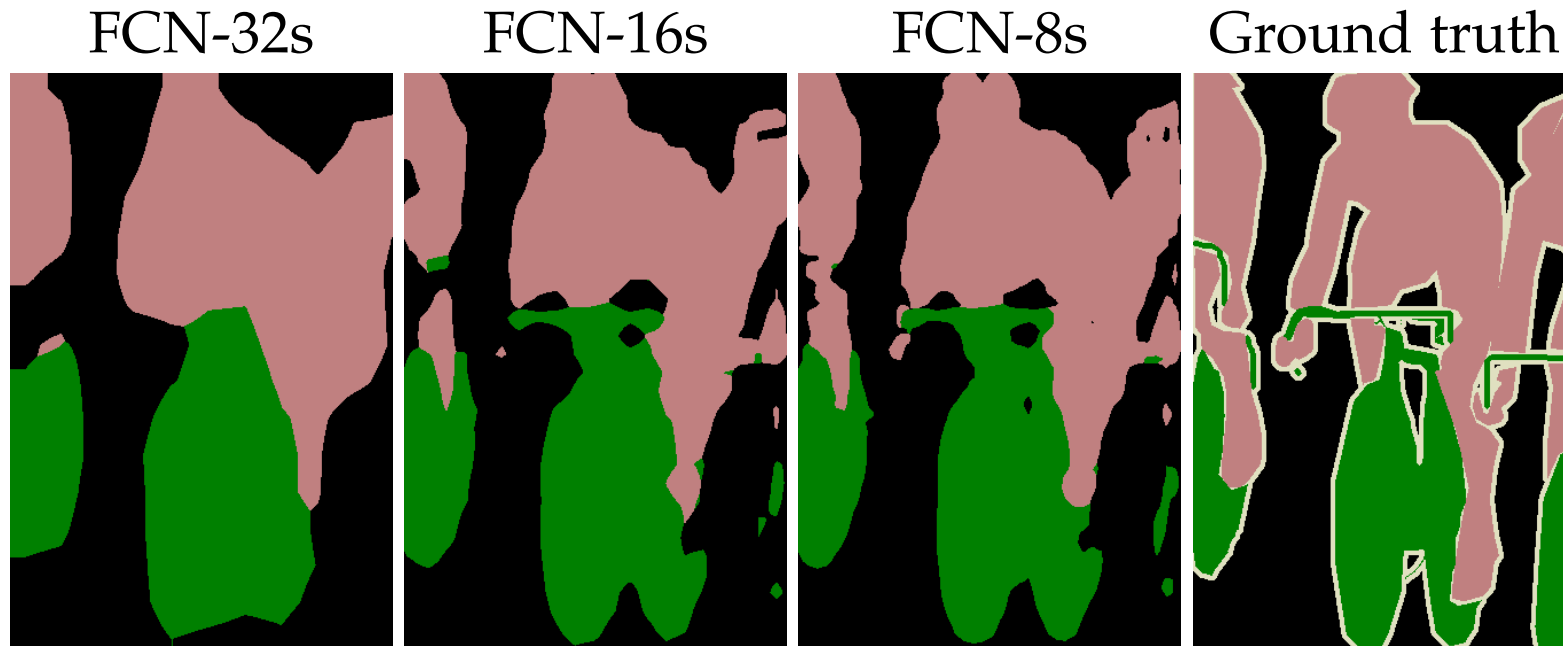
- Riduciamo la dimensione, facciamo upsampling

- Tre varianti

- Coarse upsampling
- Combined upsampling, skip connections (tramite somma)



# FCN

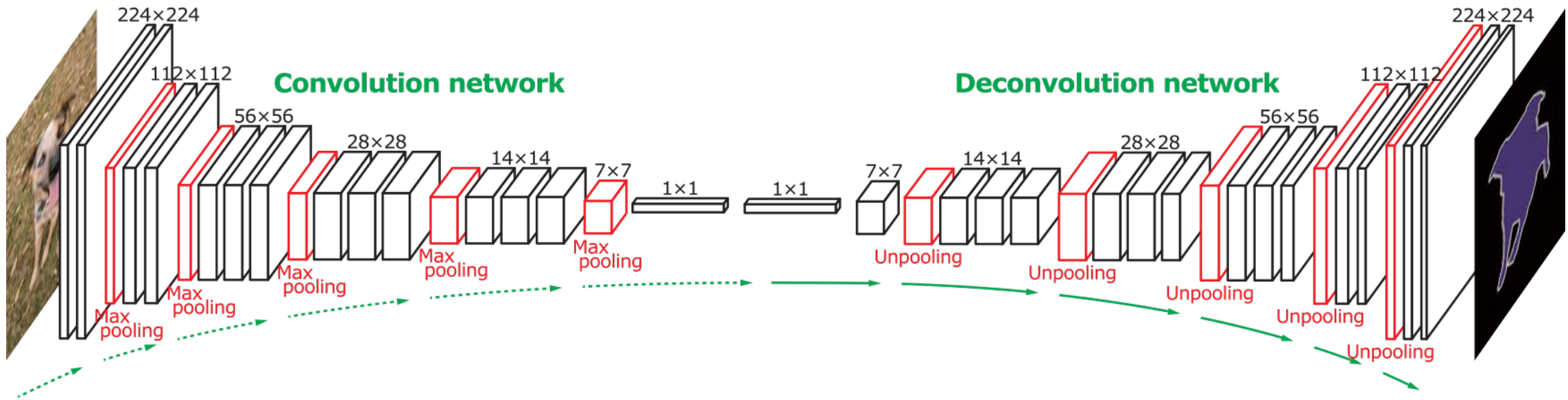


- L'utilizzo di ConvTranspose con stride di grandi dimensioni causa la presenza di artefatti
- Scarsa risoluzione ai bordi
  - L'encoding causa perdita di informazione

# DeconvNet

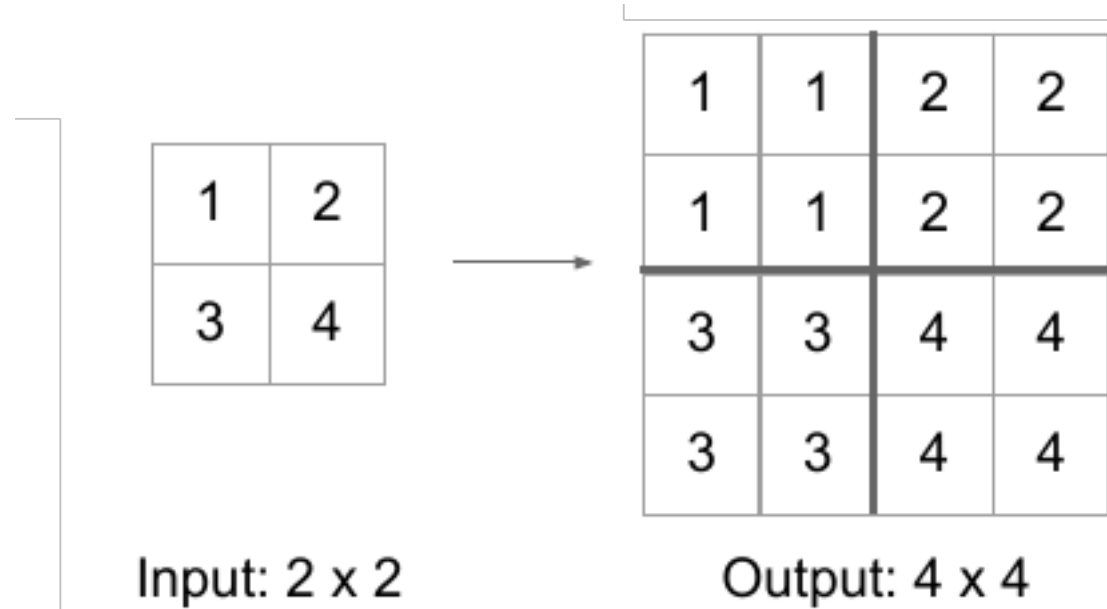
## Up-sampling Convolutions or "Deconvolutions"

- Backbone: VGG



<http://cvlab.postech.ac.kr/research/deconvnet/>

# Unpooling

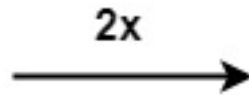


# Unpooling

- Bilinear interpolation

10	20
30	40

2x2



10	12	17	20
15	17	22	25
25	27	32	35
30	32	37	40

4x4

# Unpooling

- Bed of nails

1	2
3	4

Input: 2 x 2



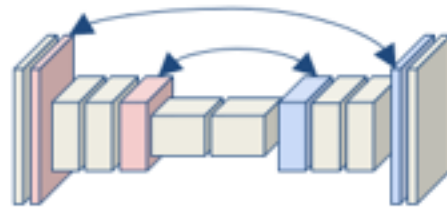
1	0	2	0
0	0	0	0
3	0	4	0
0	0	0	0

Output: 4 x 4



# Unpooling

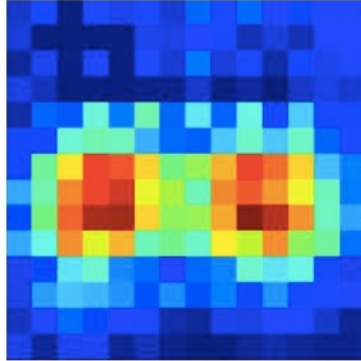
- Max unpooling



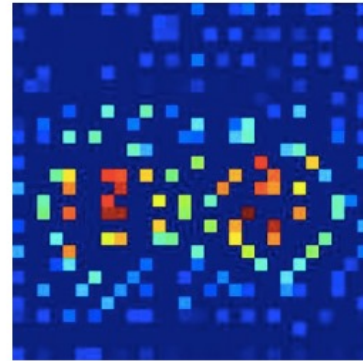
# DeconvNet



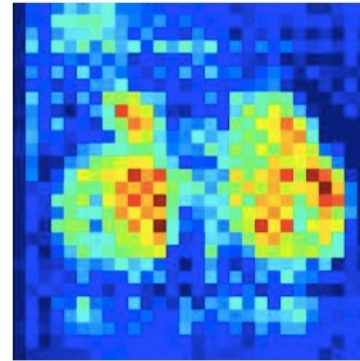
Original image



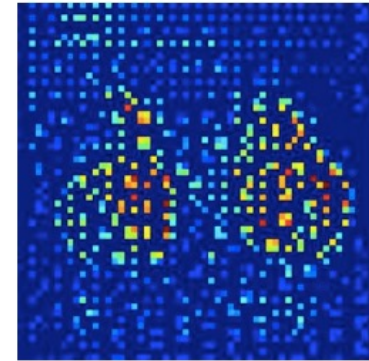
14x14 deconv



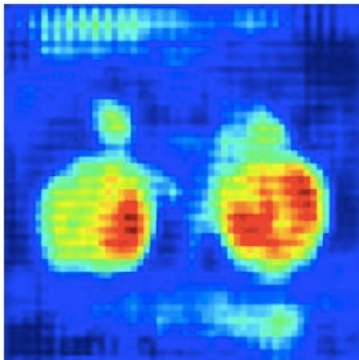
28x28 unpooling



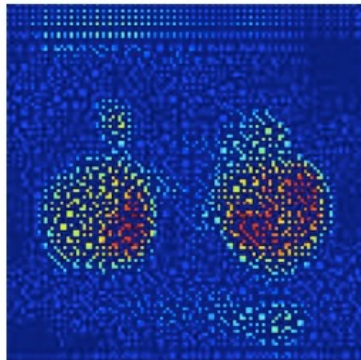
28x28 deconv



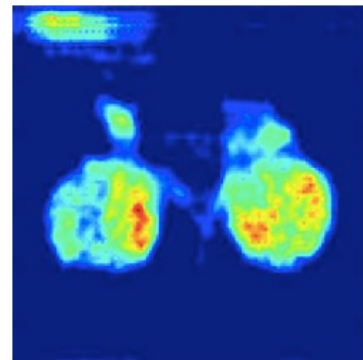
54x54 unpooling



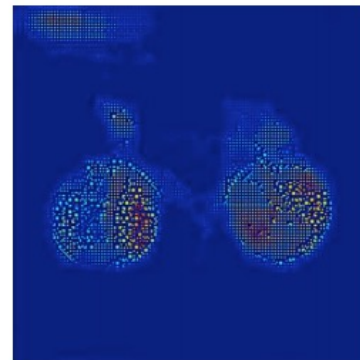
54x54 deconv



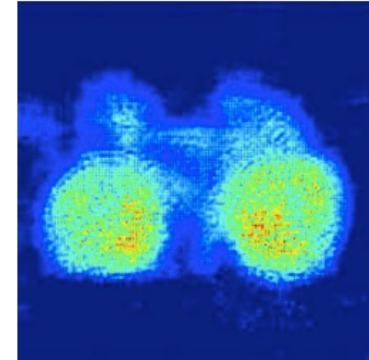
112x112 unpooling



112x112 deconv

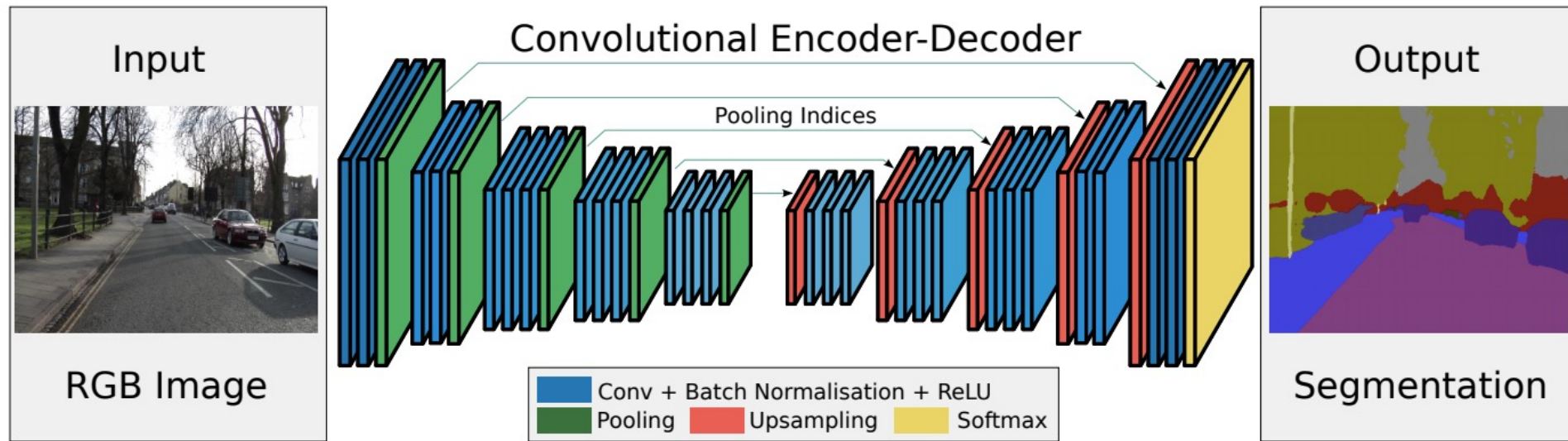


224x224 unpooling



224x224 deconv

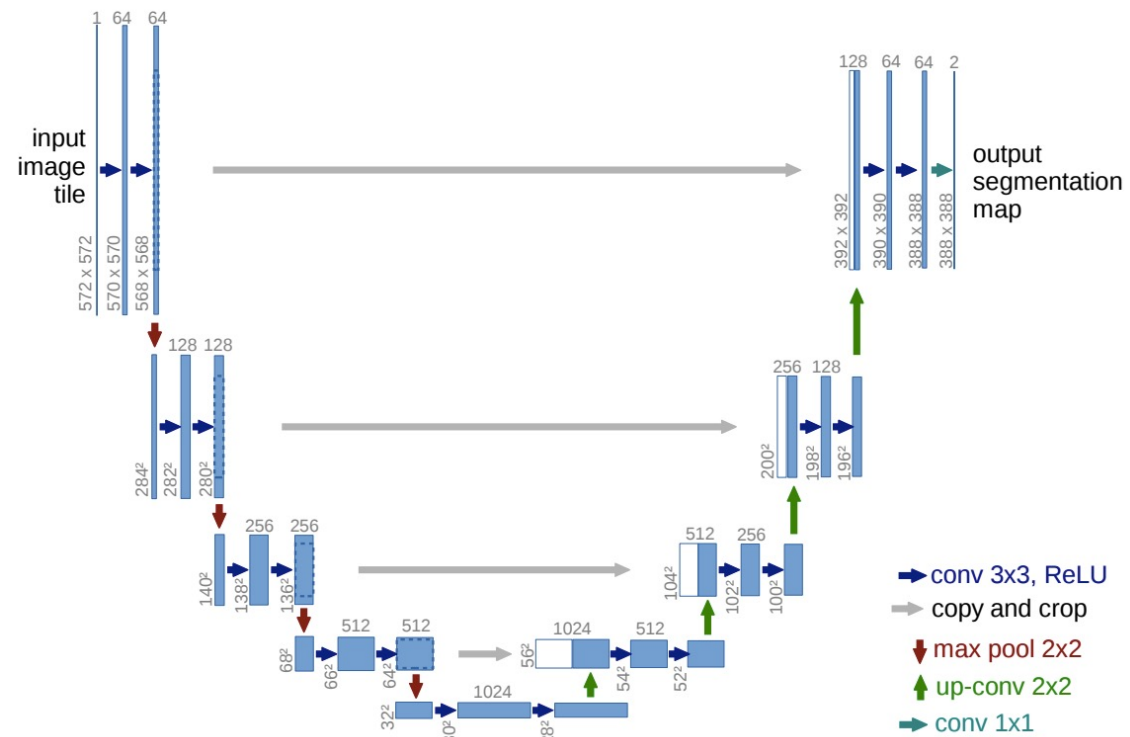
# SegNet



Eliminando i FC layer, porta a risultati migliori

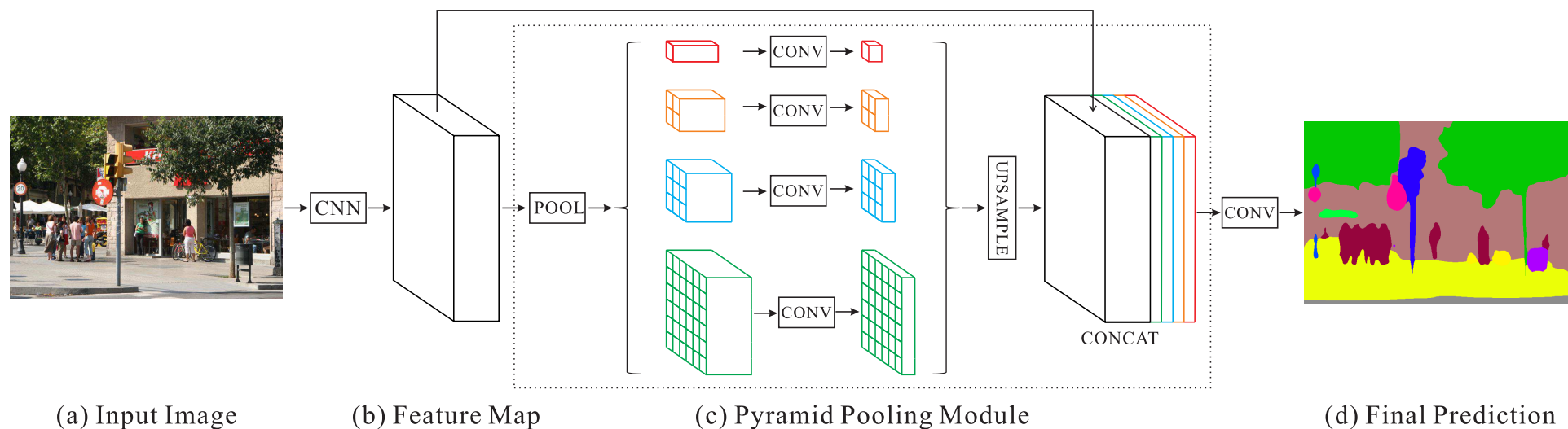
# U-Net

- Usa le skip connections per combinare le feature maps
- La combinazione viene effettuata per concatenazione



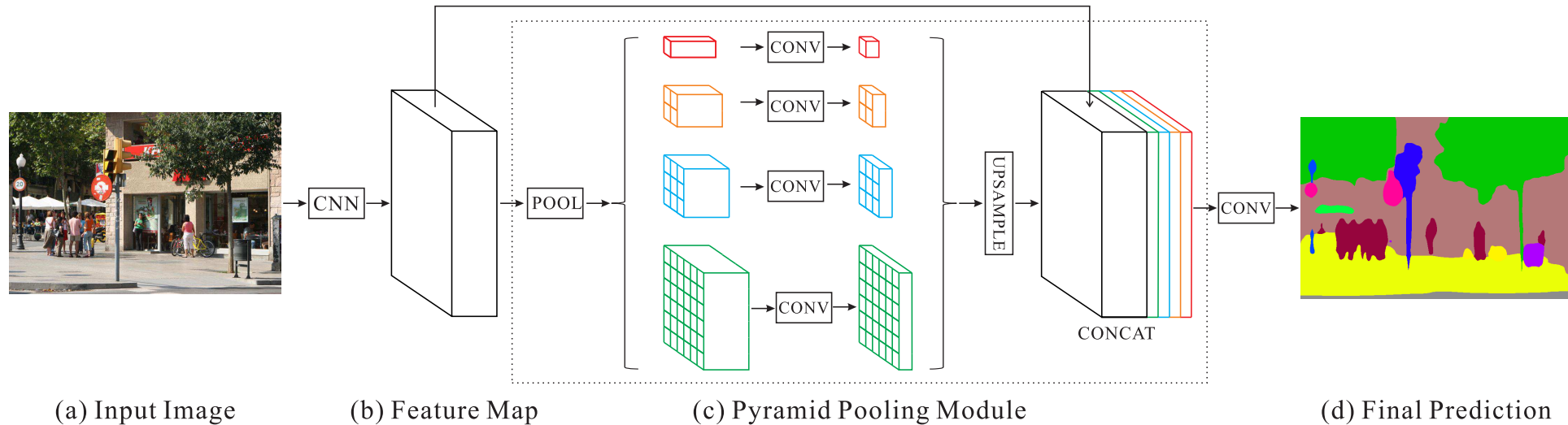
# Metodi Multi-scala

- Idea generale
  - Otteniamo una feature map utilizzando un'architettura standard (ResNet)
  - Appliciamo una serie di convoluzioni con filtri di dimensioni diverse per ottenere risoluzioni diverse
  - Encoding delle varie scale
  - Upsampling e combinazione dei risultati



# Metodi Multi-scala

- Esplosione combinatoria del numero di parametri

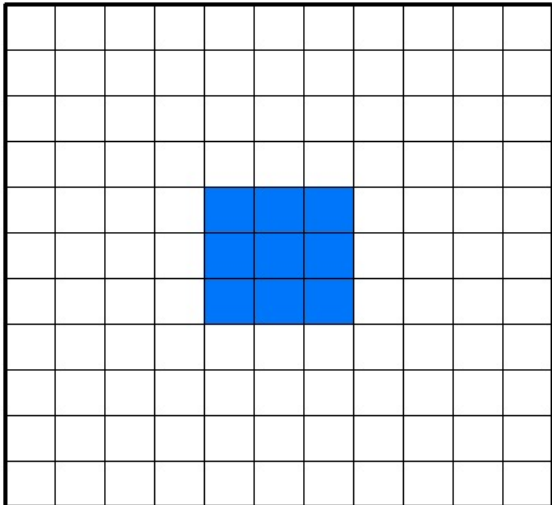


- Soluzione: Dilated convolutions

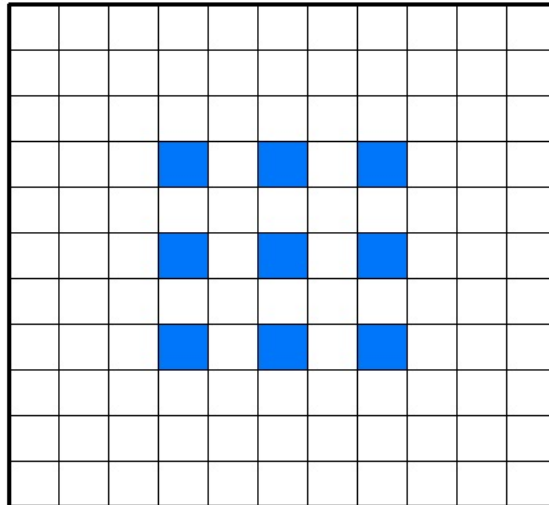
# Dilated convolutions

- Invece di ridurre la risoluzione spaziale delle feature maps, utilizziamo un filtro sparso

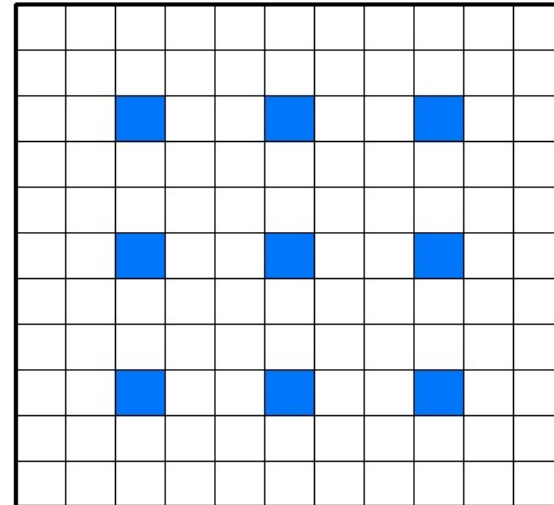
Dilation factor 1



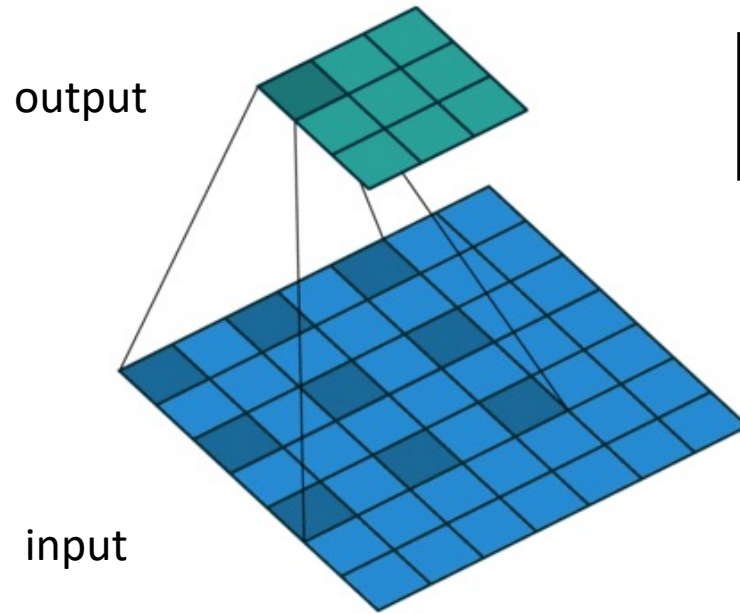
Dilation factor 2



Dilation factor 3



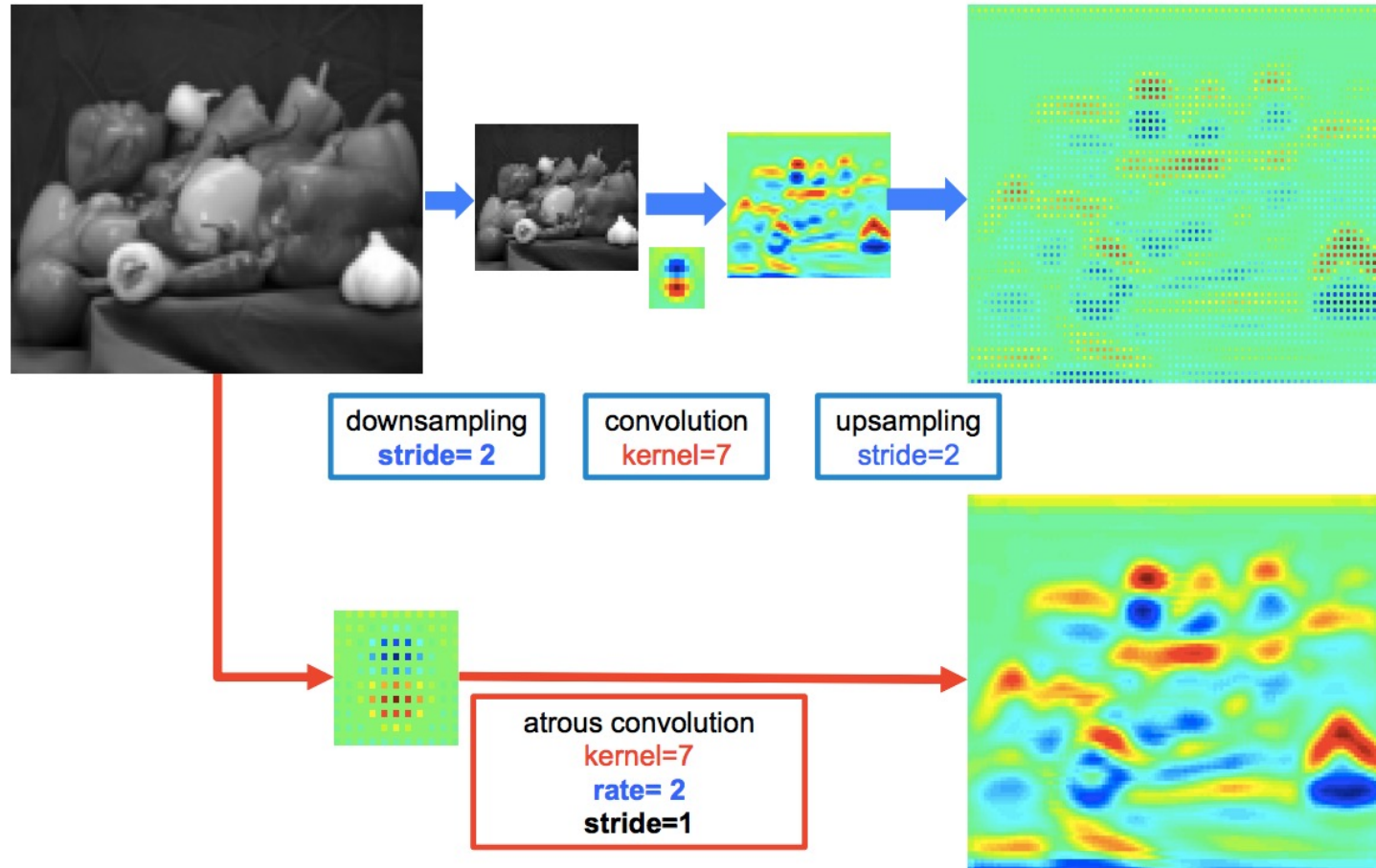
# Dilated convolutions



$$\left\lfloor \frac{I - K - (K - 1)(D - 1) + 2P}{S} \right\rfloor + 1$$



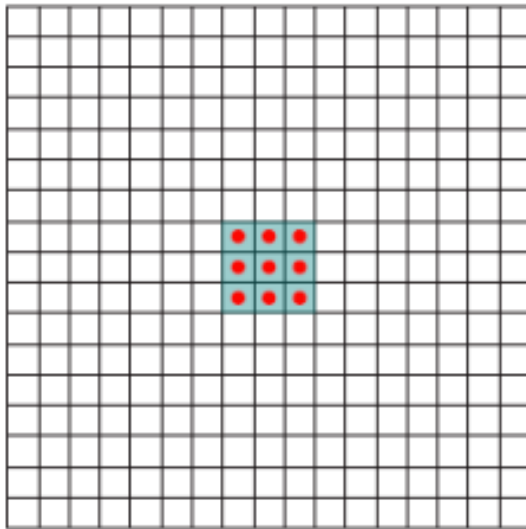
# Dilated convolutions



# Dilated convolutions

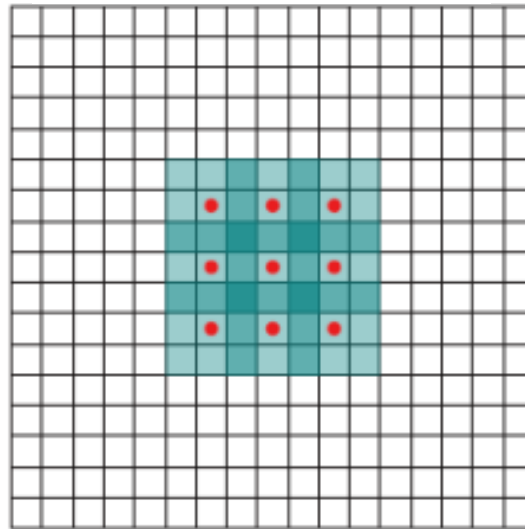
- La dimensione del receptive field cresce esponenzialmente ma il numero di parametri è lineare

F1 calcolata da F0 con  
1-dilated convolution



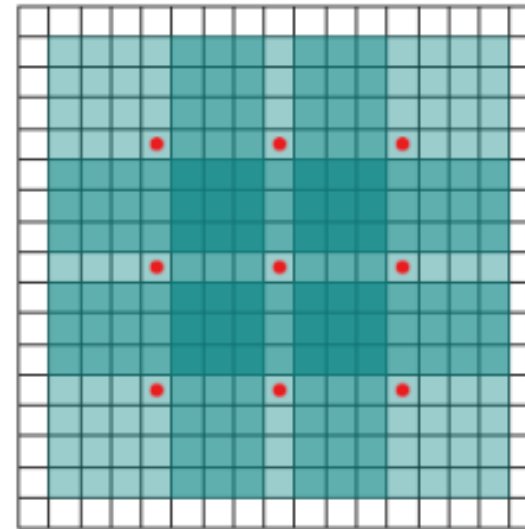
Receptive field: 3x3

F2 calcolata da F1  
con 2-dilated  
convolution



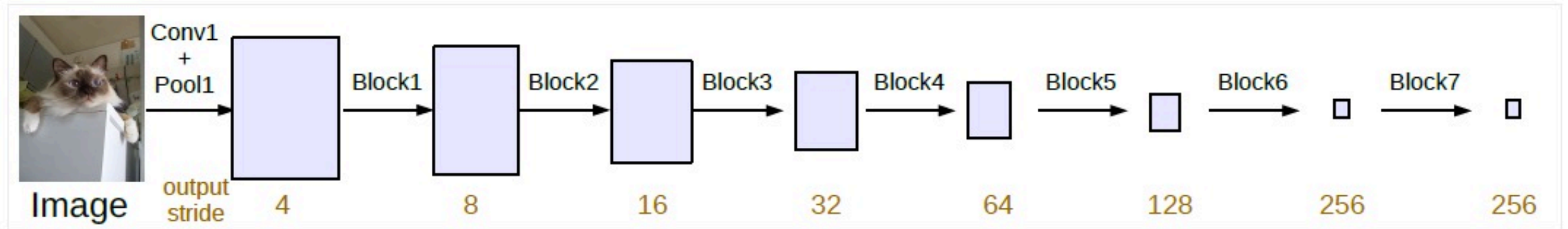
Receptive field: 7x7

F3 calcolata da F2 4-  
dilated convolution

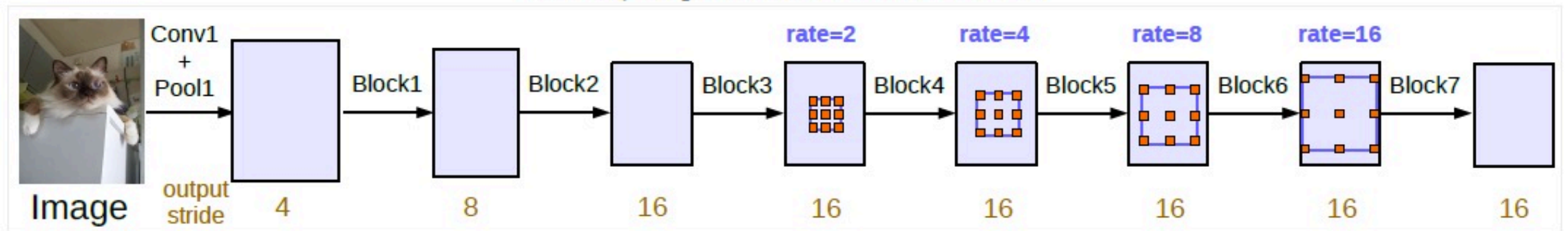


Receptive field: 15x15

# Vantaggi

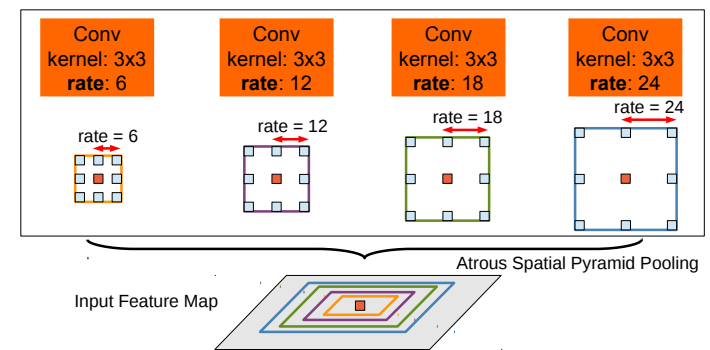
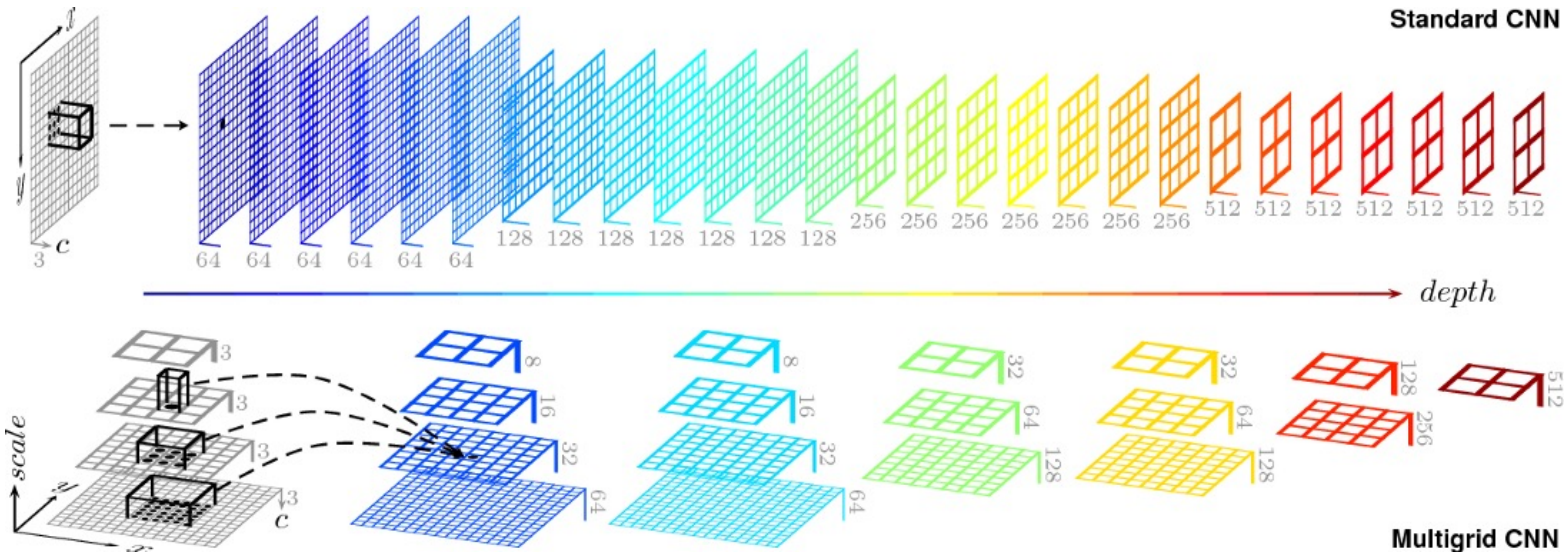


(a) Going deeper without atrous convolution.



(b) Going deeper with atrous convolution. Atrous convolution with  $rate > 1$  is applied after block3 when  $output\_stride = 16$ .

# Multigrid CNN



# Real-Time Semantic Segmentation

- ENet
- ICNet
- Fast-SCNN
- DFANet
- ...

# ENet

- Combina Encoder-Decoder con blocchi ResNet-like
  - 5 stadi
  - Sfrutta la riduzione e successivo restore dei canali per migliorare l'efficienza
- Design choices per l'efficienza
  - Early downsampling
    - I primi due blocchi riducono la size in maniera significativa
  - Ricostruzione tramite MaxUnpooling
  - Decoder size
    - Large encoder, small decoder
  - Non-linear activations
    - PReLU invece delle ReLU
  - Asymmetric convolutions, dilated convolutions
  - Regularization
    - Stochastic Depth, spatial dropout

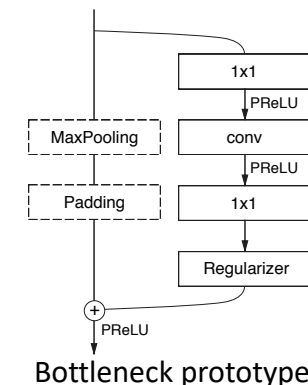
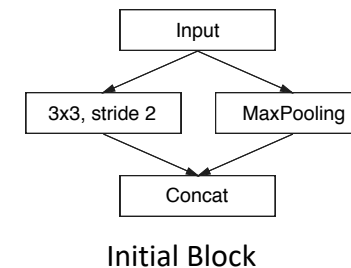


Table 1: ENet architecture. Output sizes are given for an example input of  $512 \times 512$ .

Name	Type	Output size
initial		$16 \times 256 \times 256$
bottleneck1.0	downsampling	$64 \times 128 \times 128$
4× bottleneck1.x		$64 \times 128 \times 128$
bottleneck2.0	downsampling	$128 \times 64 \times 64$
bottleneck2.1	dilated 2	$128 \times 64 \times 64$
bottleneck2.2	asymmetric 5	$128 \times 64 \times 64$
bottleneck2.3	dilated 4	$128 \times 64 \times 64$
bottleneck2.4		$128 \times 64 \times 64$
bottleneck2.5		$128 \times 64 \times 64$
bottleneck2.6	dilated 8	$128 \times 64 \times 64$
bottleneck2.7	asymmetric 5	$128 \times 64 \times 64$
bottleneck2.8	dilated 16	$128 \times 64 \times 64$
<i>Repeat section 2, without bottleneck2.0</i>		
bottleneck4.0	upsampling	$64 \times 128 \times 128$
bottleneck4.1		$64 \times 128 \times 128$
bottleneck4.2		$64 \times 128 \times 128$
bottleneck5.0	upsampling	$16 \times 256 \times 256$
bottleneck5.1		$16 \times 256 \times 256$
fullconv		$C \times 512 \times 512$

# ENet

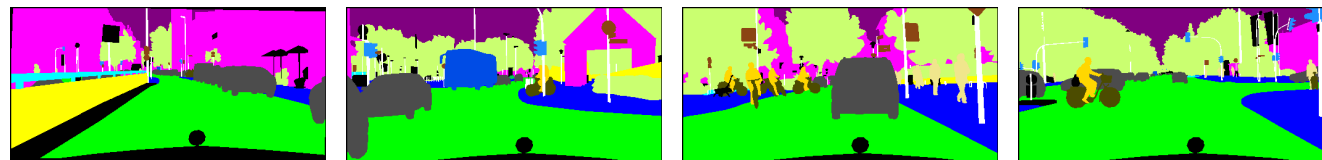
Model	NVIDIA TX1						NVIDIA Titan X					
	480×320		640×360		1280×720		640×360		1280×720		1920×1080	
	ms	fps	ms	fps	ms	fps	ms	fps	ms	fps	ms	fps
SegNet	757	1.3	1251	0.8	-	-	69	14.6	289	3.5	637	1.6
ENet	47	21.1	69	14.6	262	3.8	7	135.4	21	46.8	46	21.6

Model	Class IoU	Class iIoU	Category IoU	Category iIoU
SegNet	56.1	34.2	79.8	<b>66.4</b>
ENet	<b>58.3</b>	<b>34.4</b>	<b>80.4</b>	64.0

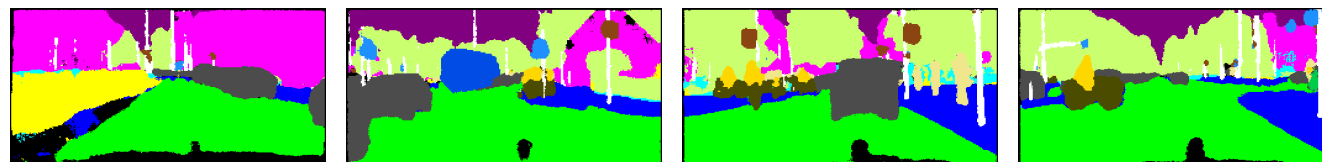
Input image



Ground truth

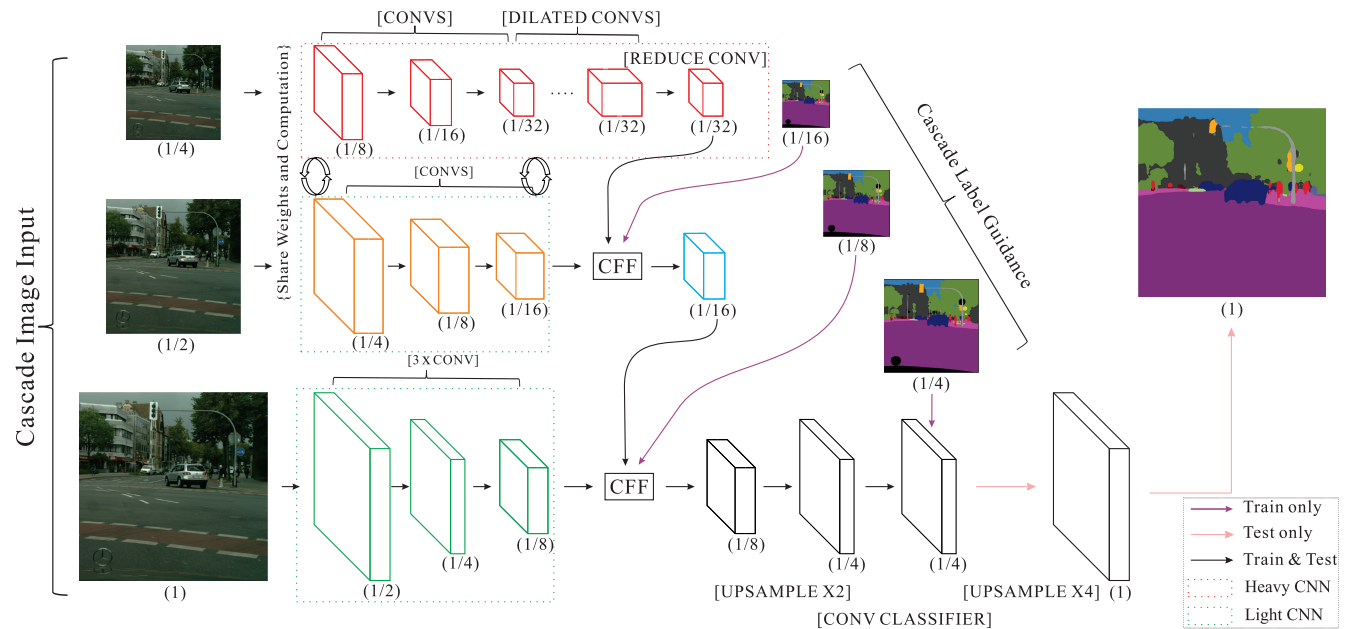
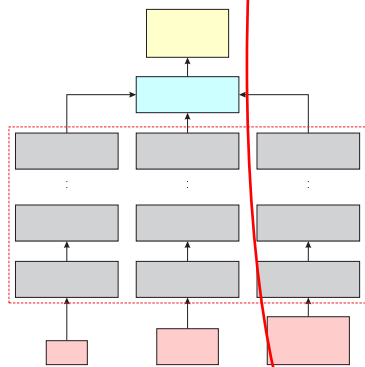
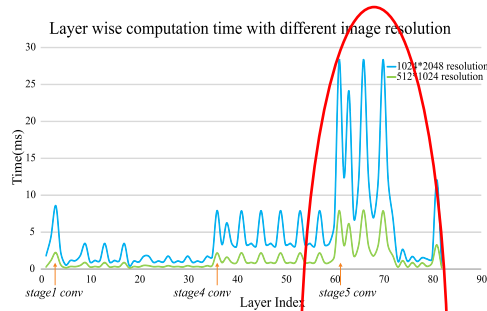


ENet output



# ICNet

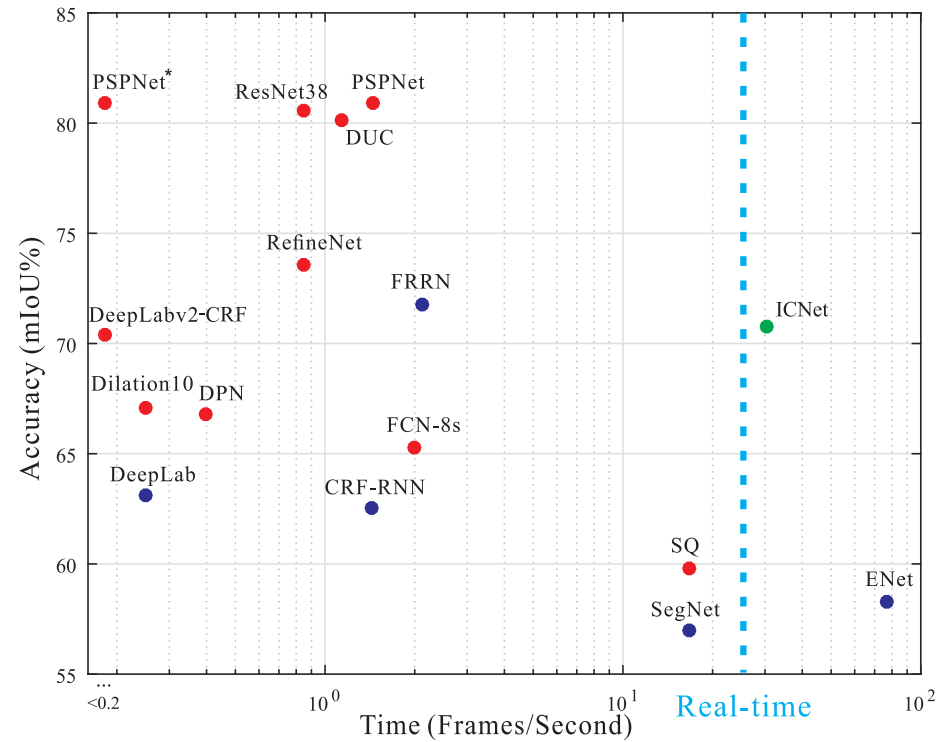
- Analisi dei bottleneck nei metodi multi-scala
  - La convoluzione su immagini grandi è costosa
  - Soluzione: combinazione «furba» per accelerare la computazione





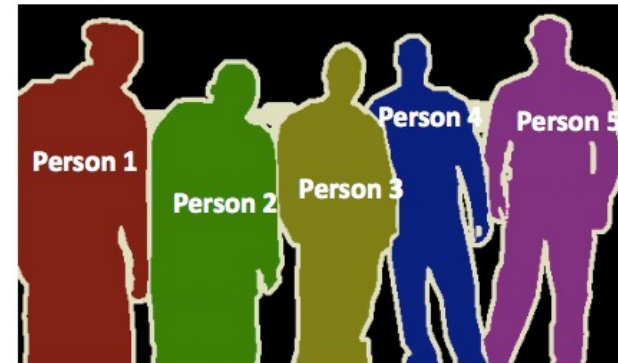
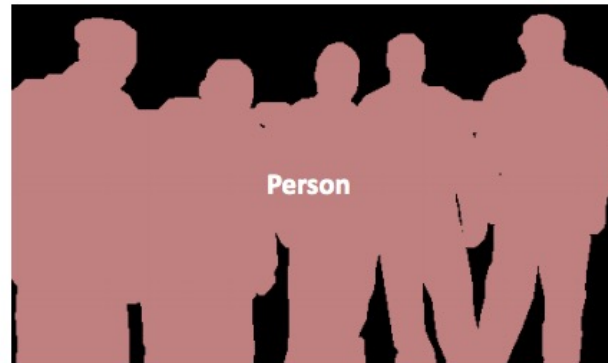
# ICNet

- Trade-off tra velocità e accuratezza



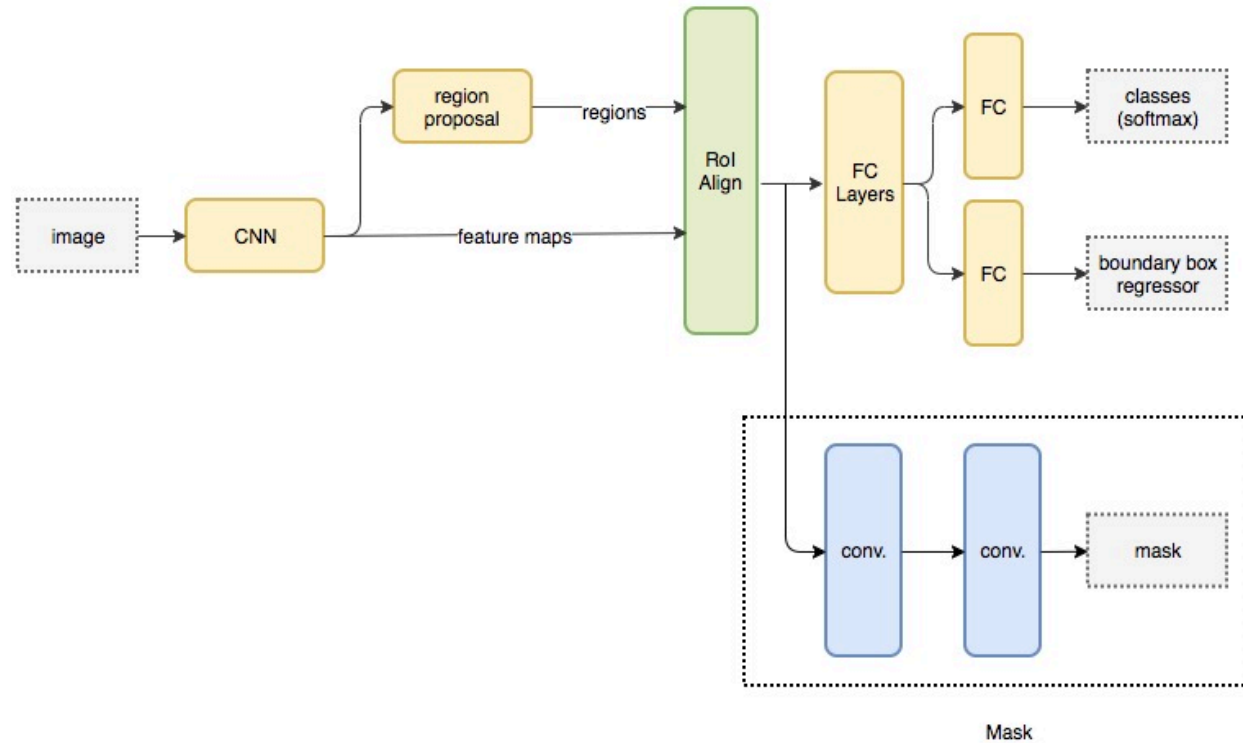
# Instance Segmentation

- Obiettivo
  - Individuare non solo la segmentazione, ma anche l'istanza



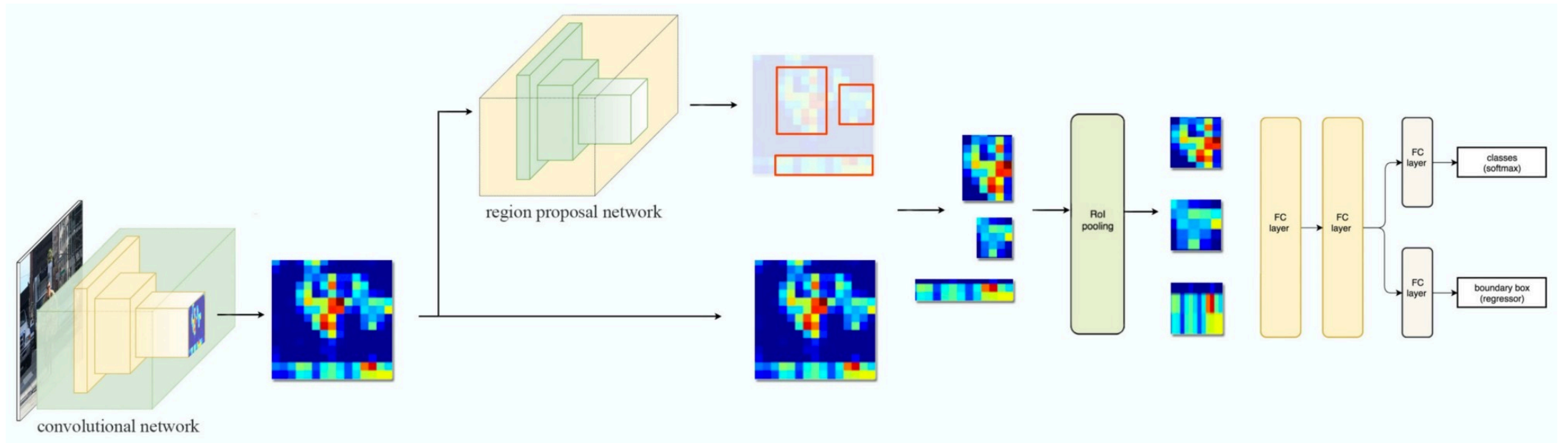
# Mask R-CNN

- Mask R-CNN = Faster R-CNN + FCN sui Rols

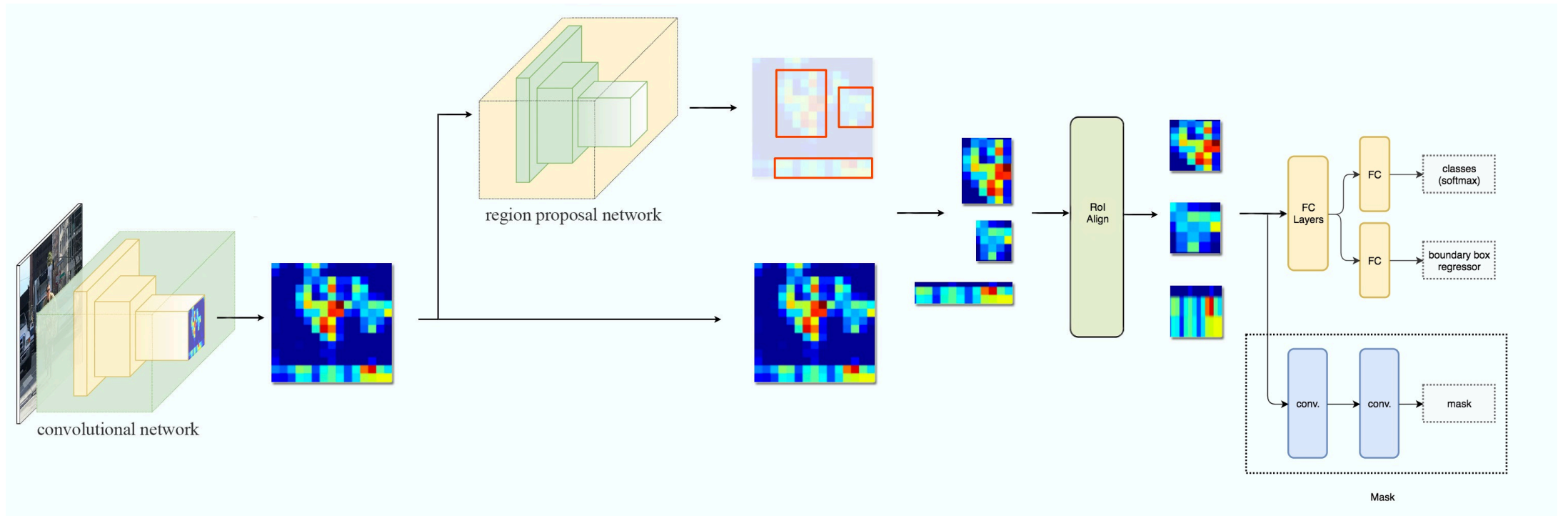


$$\text{Loss: } L_{cs} + L_{box} + L_{mask}$$

# Recap: Faster R-CNN

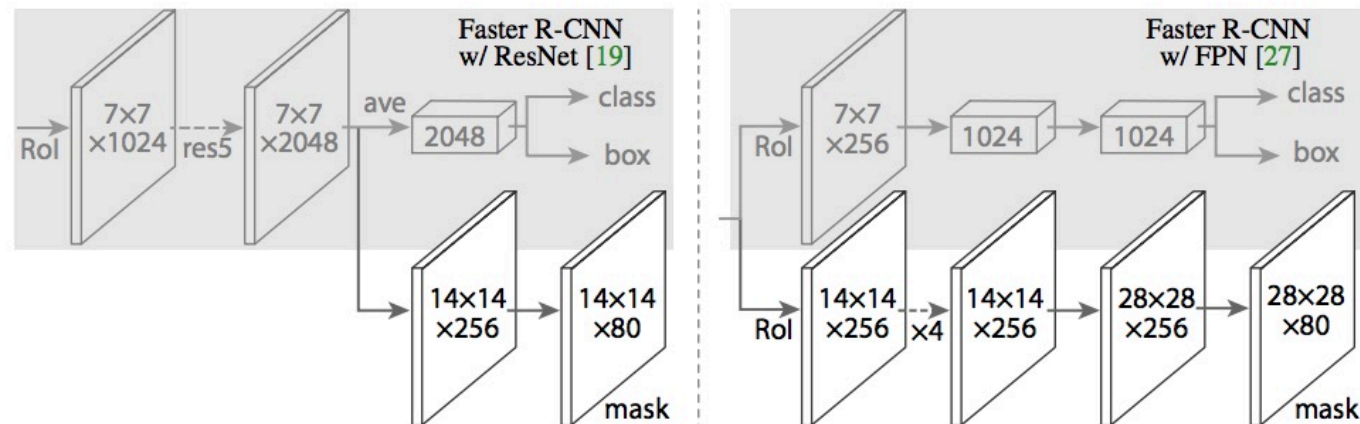


# Da Faster R-CNN a Mask R-CNN



# Mask R-CNN: Mask

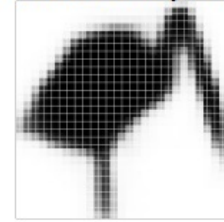
- $K \cdot m \times m$ 
  - Una maschera di dimensione  $m \times m$  per ognuna delle  $K$  classi
  - Ogni pixel è regolato da una sigmoide
  - Loss
    - Su una RoI associata alla classe  $k$ ,  $L_{mask}$  è la binary cross-entropy relativa alla maschera  $m_k$  associata
    - Le altre maschere non contribuiscono alla loss



# Mask R-CNN



28x28 soft prediction



Resized Soft prediction

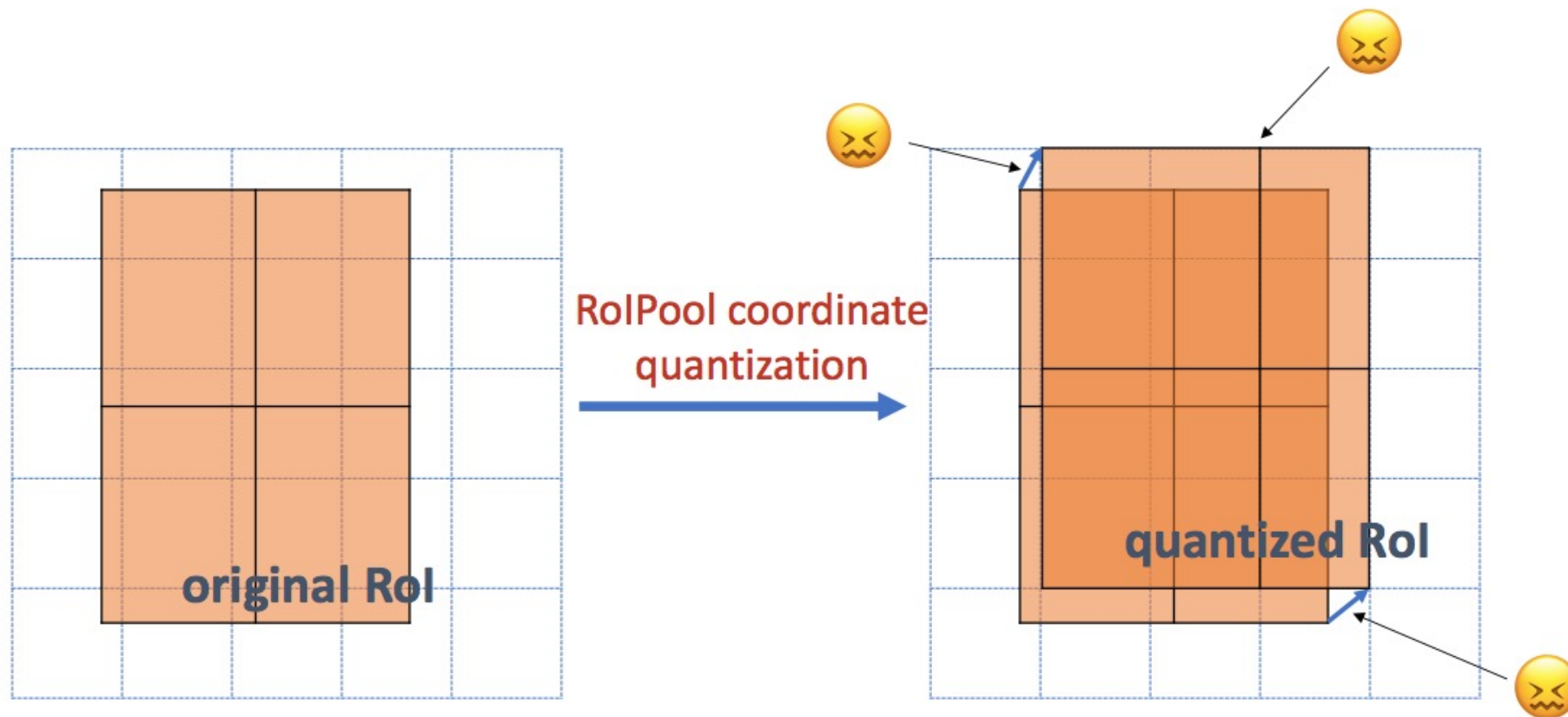


Final mask



# RoIAlign

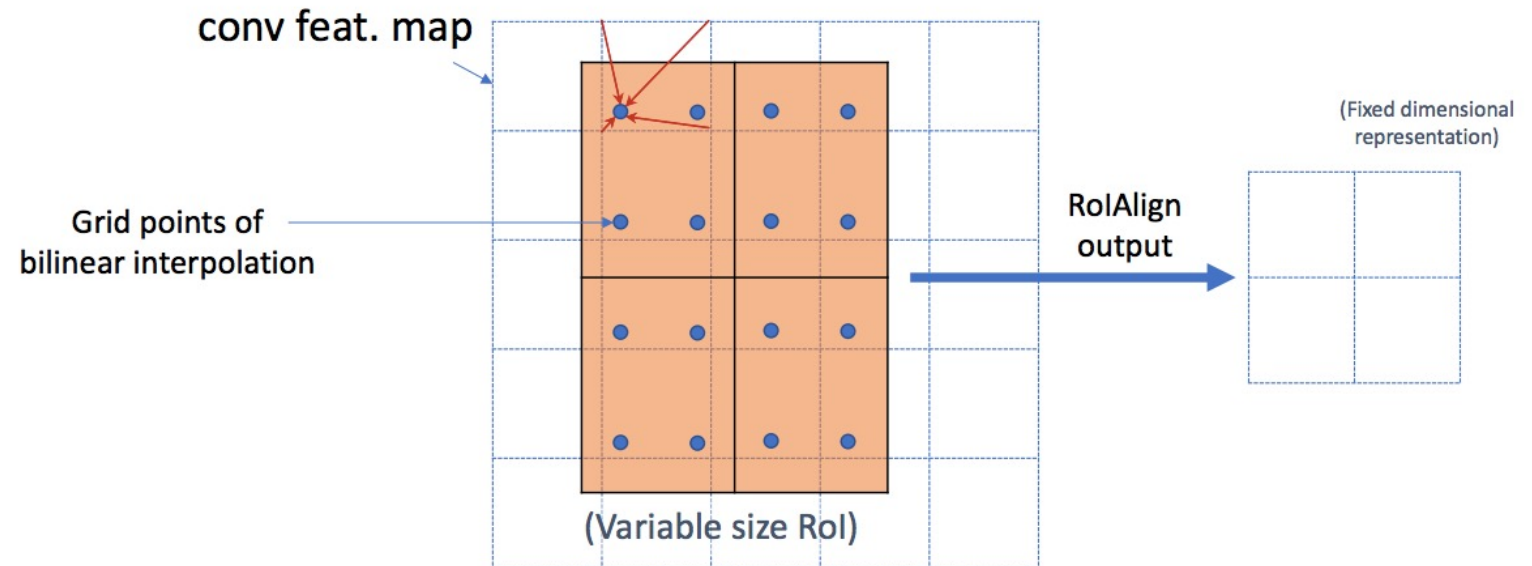
- Il mapping di una regione sulla feature map con RoIPooling causa un riallineamento





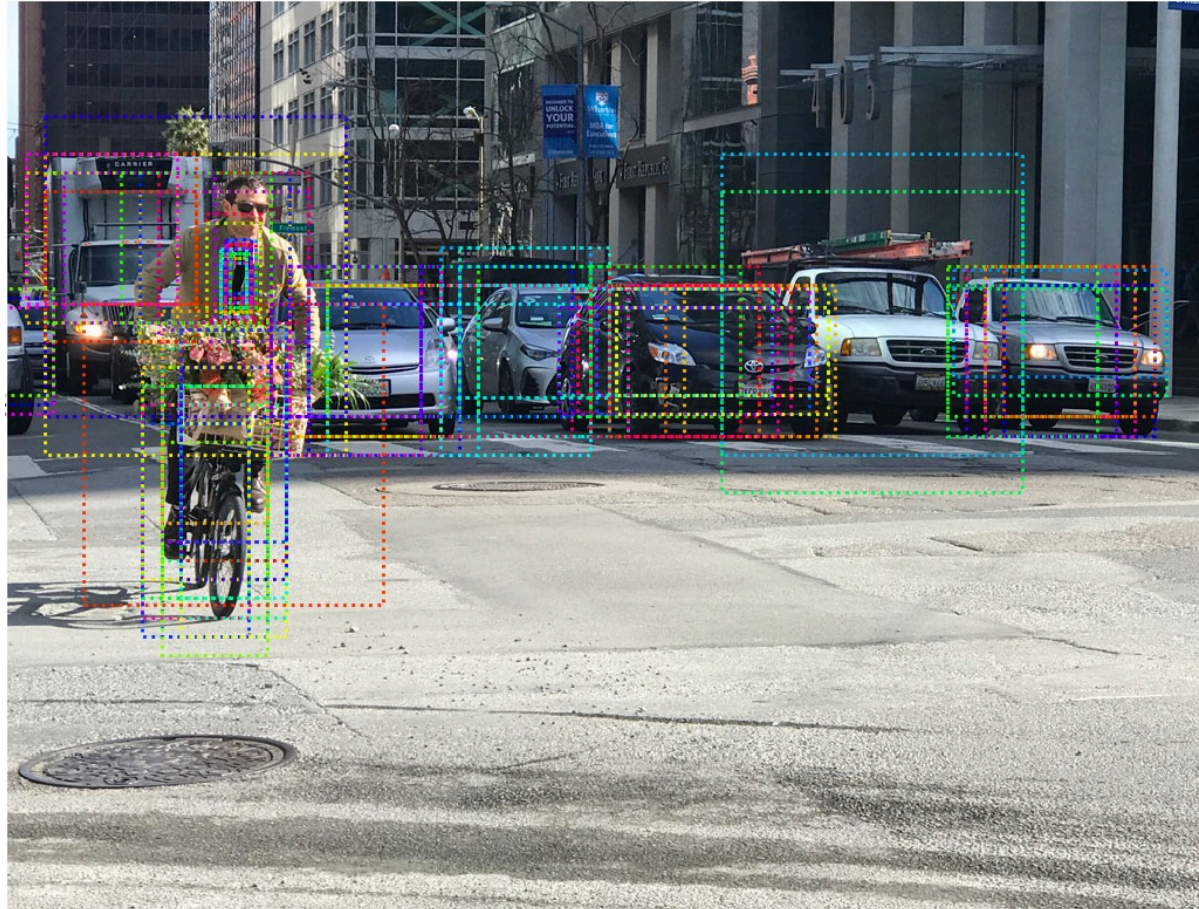
# RoIAlign

- Con RoIAlign, ogni punto viene interpolato
  - Recupera precision nella ricostruzione della maschera



# Risultati

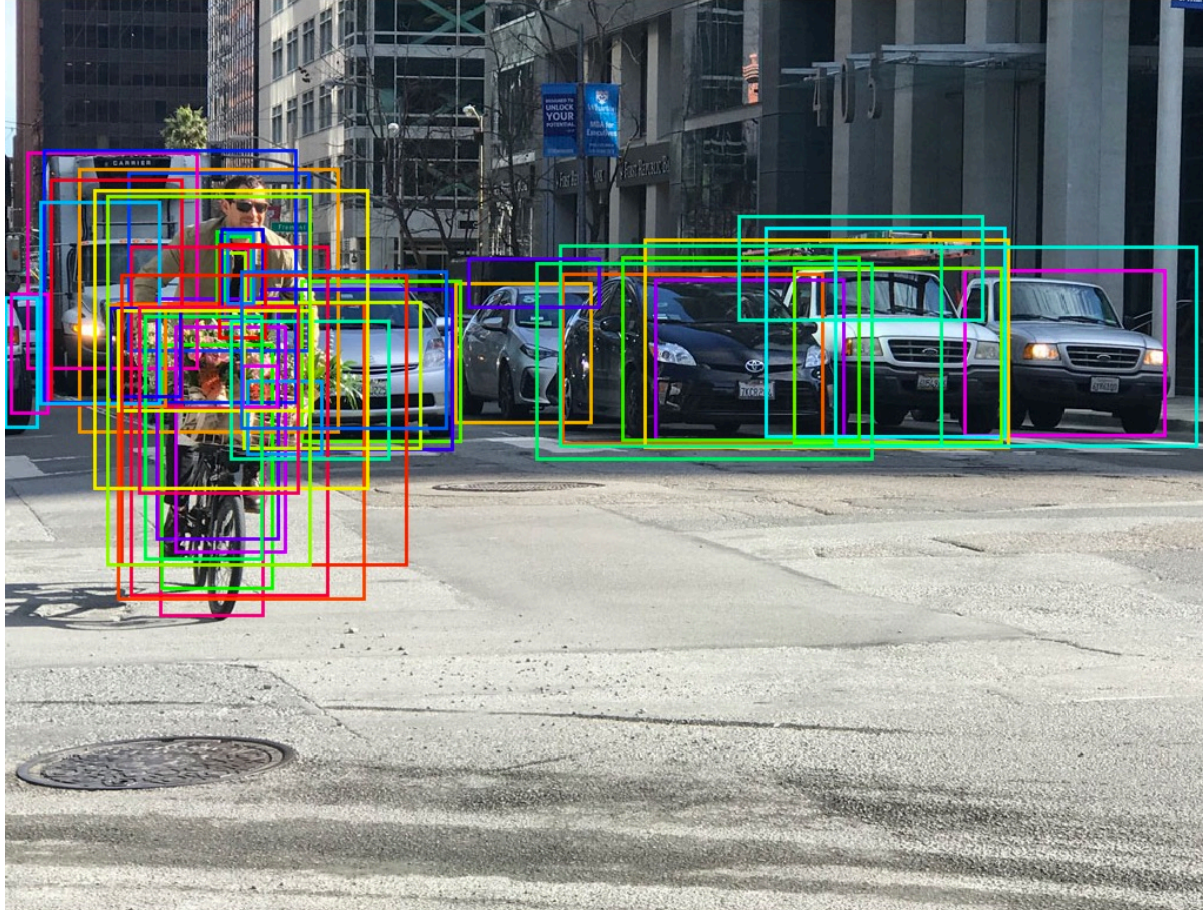
- Priors





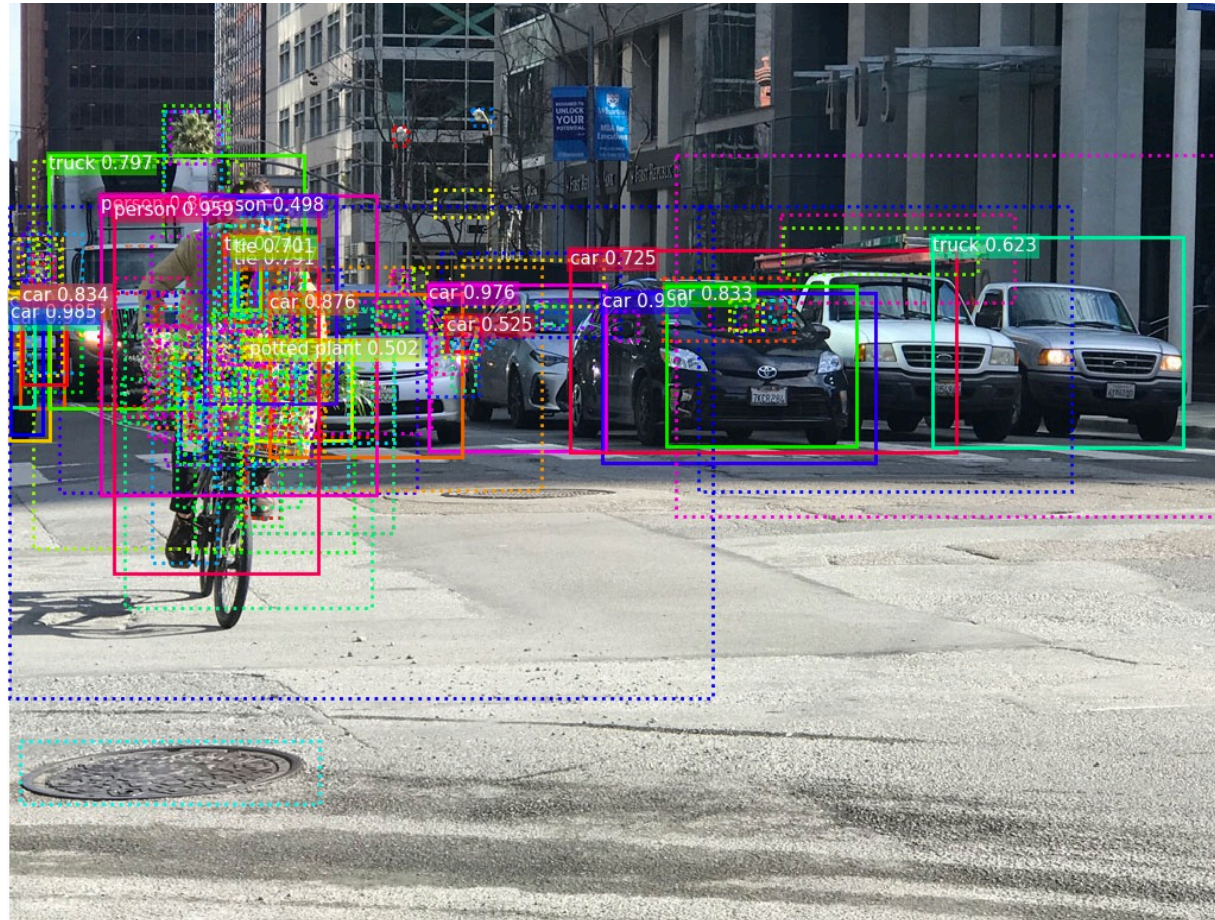
# Risultati

- Region Proposals



# Risultati

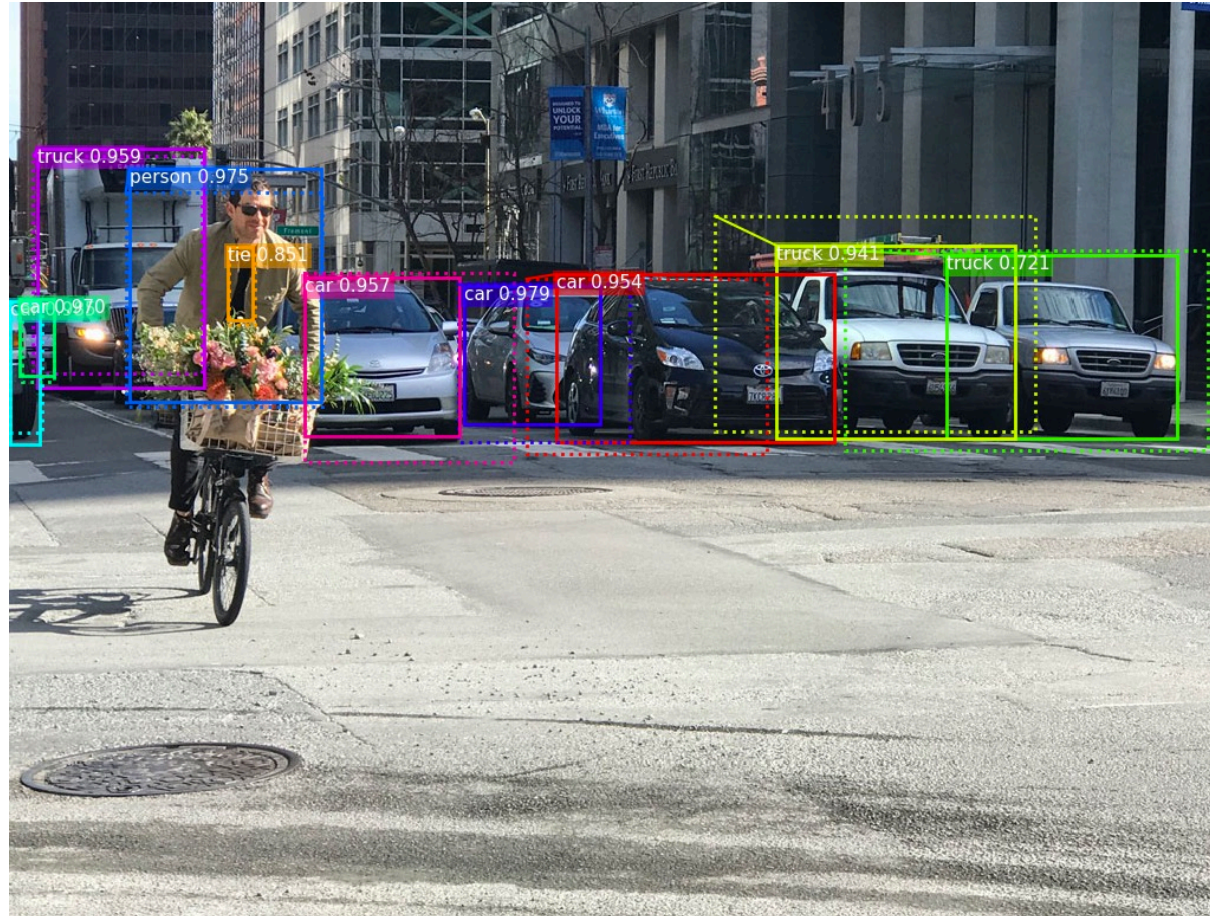
- Predizione





# Risultati

- Non-Maximum Suppression



# Risultati

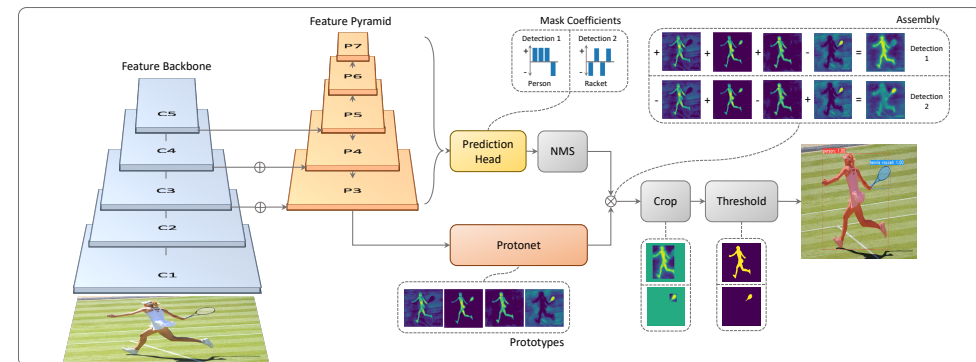
- Mask



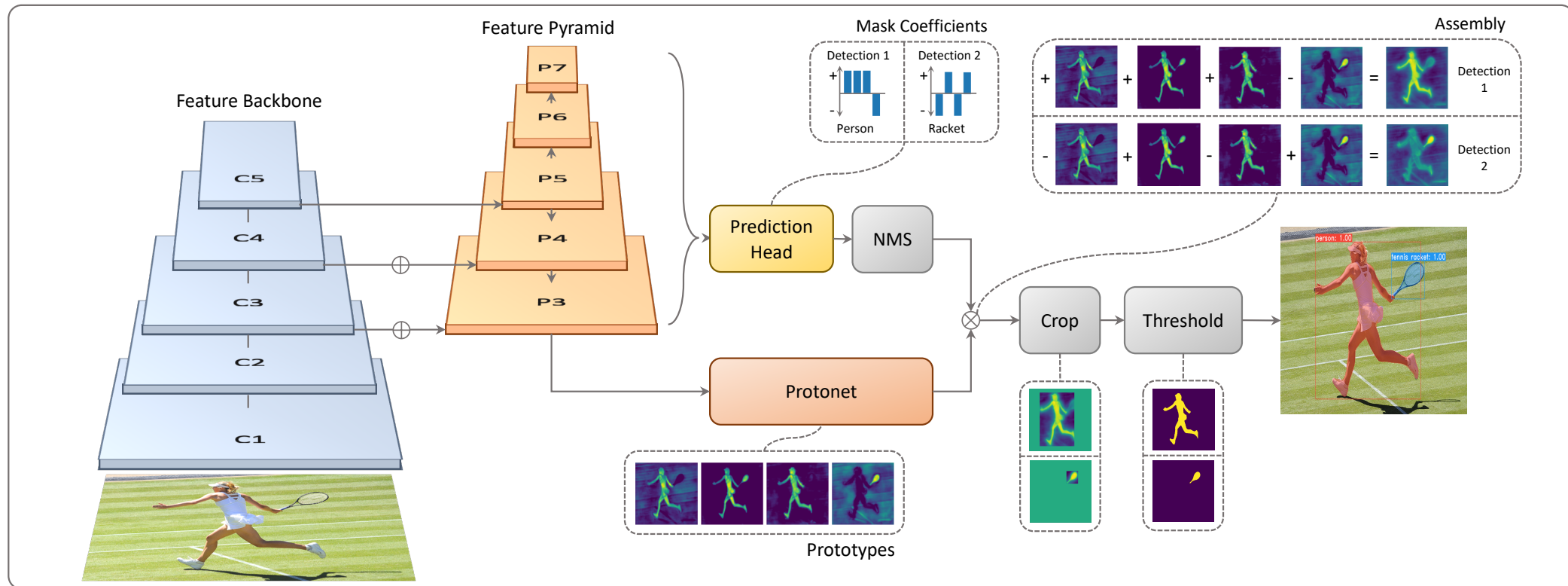


# YOLOACT: You Only Look At Coefficients

- Due task paralleli:
  - Generazione di un dizionario di non-local prototype masks sull'intera immagine
    - Basato su FCN
  - Predizione di un insieme di coefficienti di combinazione per ogni istanza
    - Aggiunge una componente all'object detection per predire un vettore di "mask coefficients"
- Per ogni istanza selezionata nel NMS viene costruita una maschera combinando i risultati dei due task.

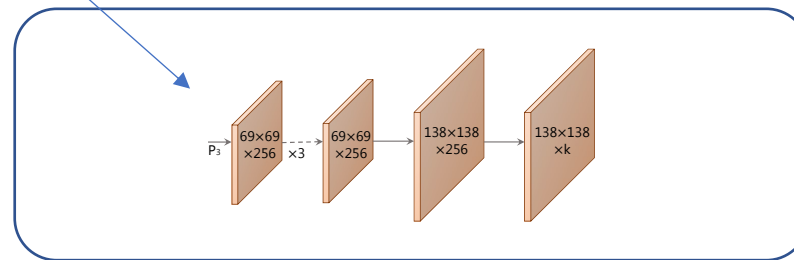
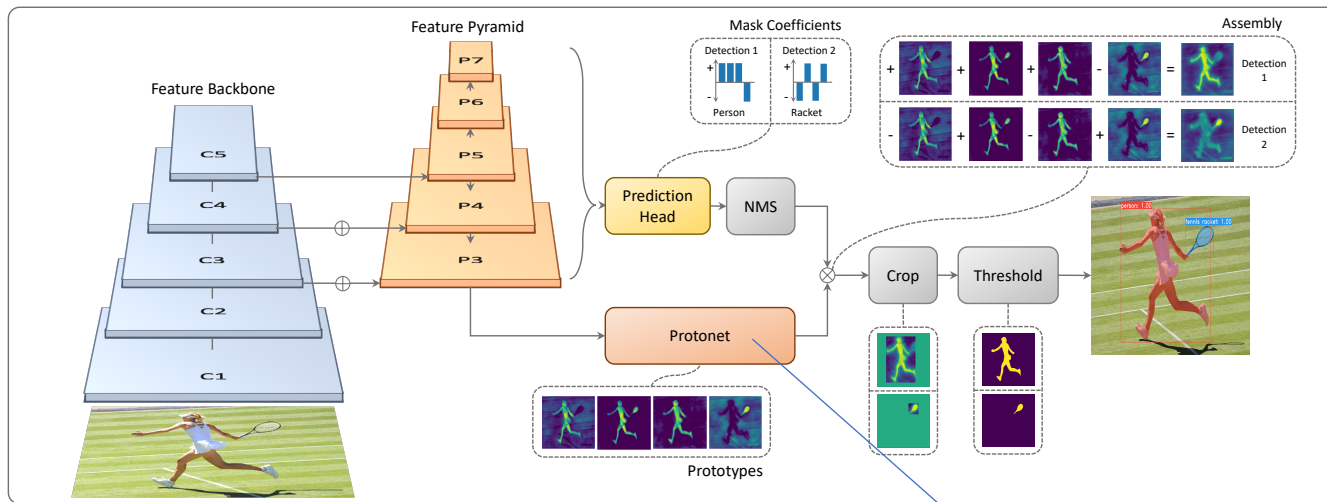


# Architettura



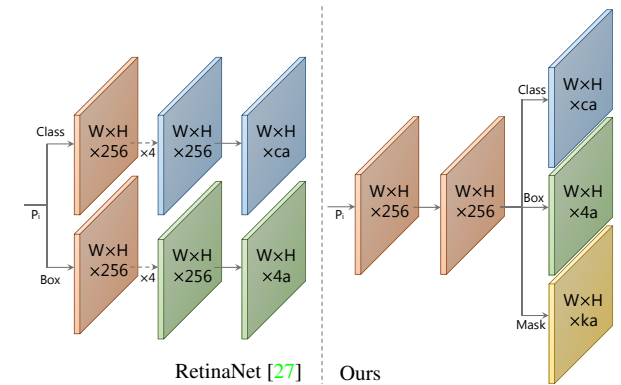
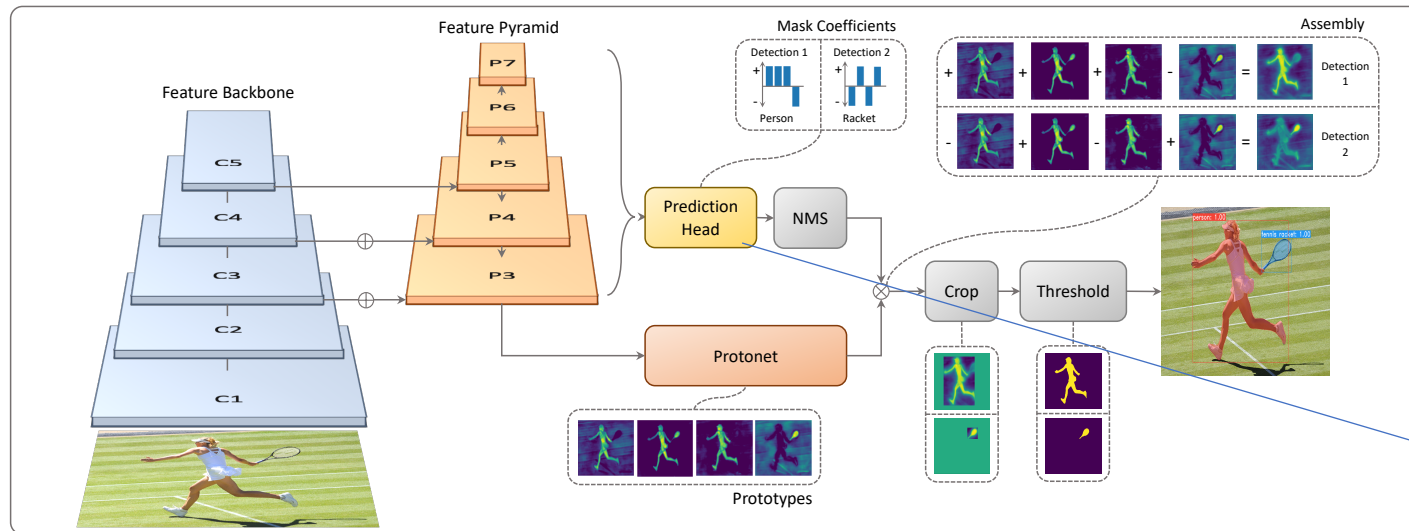


# Architettura

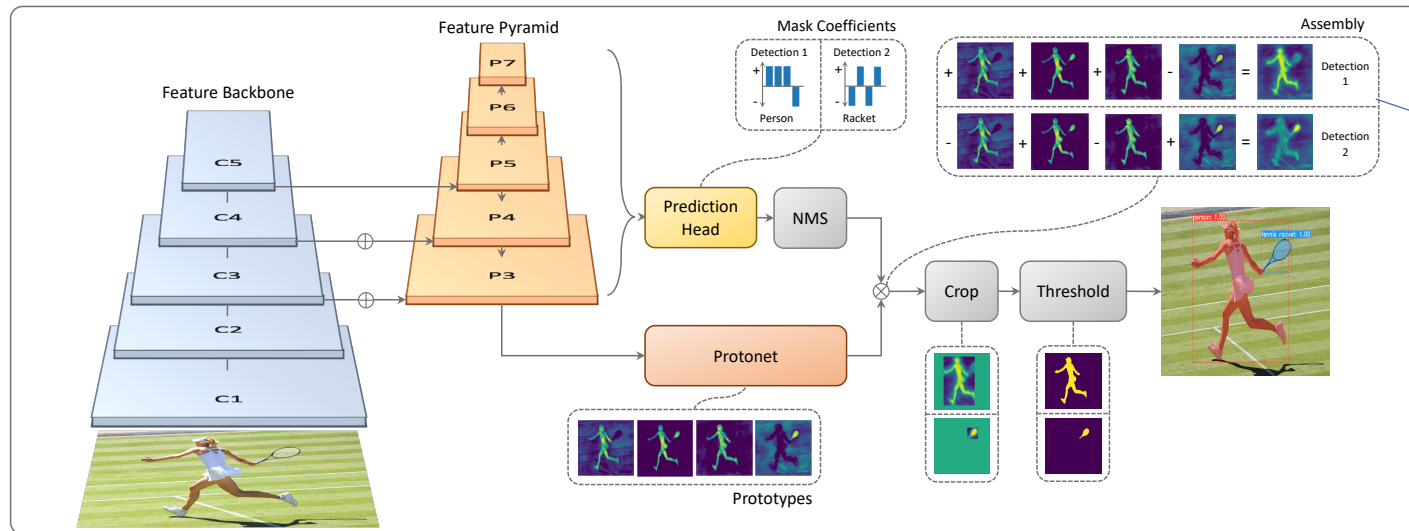


Prototype FCN

# Architettura



# Architettura

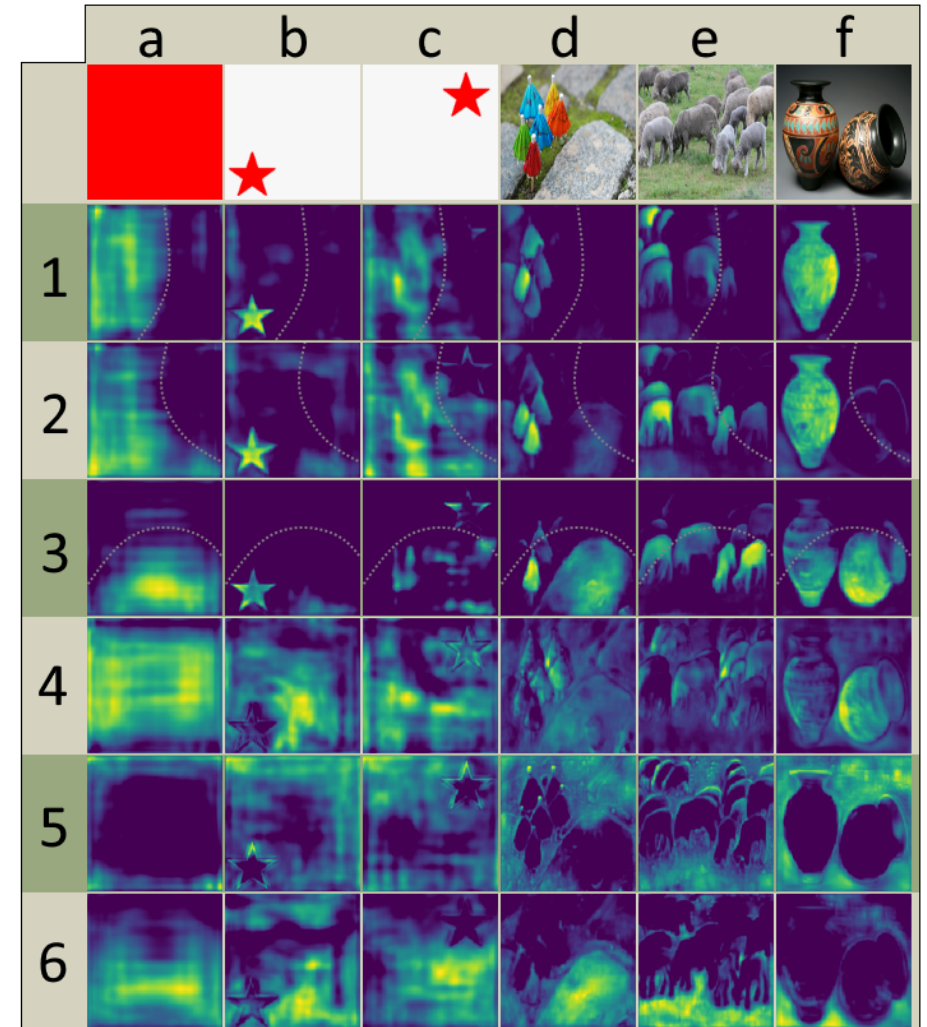


$$M = \sigma(P \cdot C^T)$$

- $P$  matrice di maschere
- $C$  vettore di coefficienti

# Caratteristiche

- Loss
  - $L_{cs} + L_{box} + L_{mask}$
  - $L_{mask} = BCE(M, M_{gt})$
- YOLACT impara a localizzare le istanze







# Riassunto

- Semantic vs. Instance segmentation
- Architetture complesse
- Base per learning task simili
  - Depth estimation
  - Surface normal estimation
  - Colorization

