Computer Vision

IA per le industrie: corso avanzato

Rosario Di Carlo

PRESENTAZIONE

Rosario Di Carlo

AI Research Fellow @LeonardoLabs, AI-BigData-HPC Lab

Cosa è la Computer Vision?

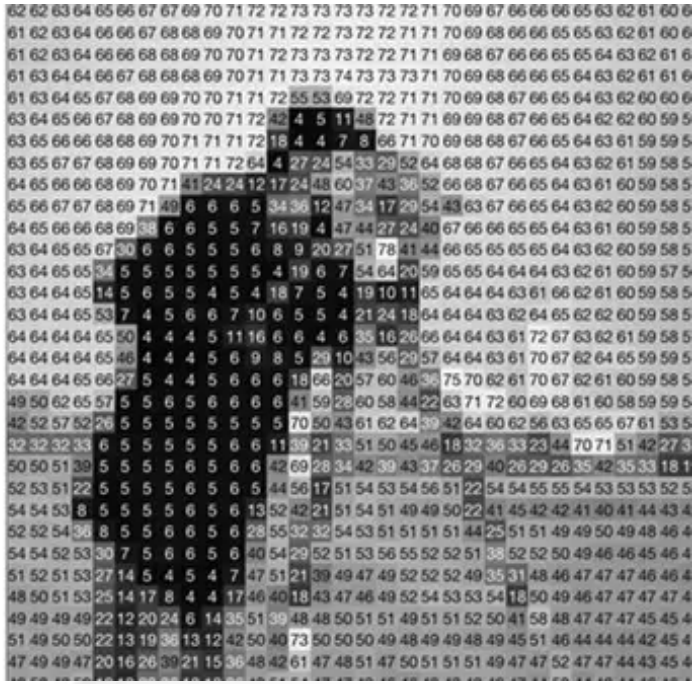
- Computer Vision è il campo di ricerca che include metodologie per l'acquisizione, il processamento, l'analisi e la comprensione delle immagini.

- Strettamente legata alla teoria per costruire sistemi artificiali che ottengono informazioni dalle immagini.

- I dati delle immagini possono assumere molte forme, come una sequenza video, immagini di profondità, viste da più telecamere o dati multidimensionali da uno scanner medico

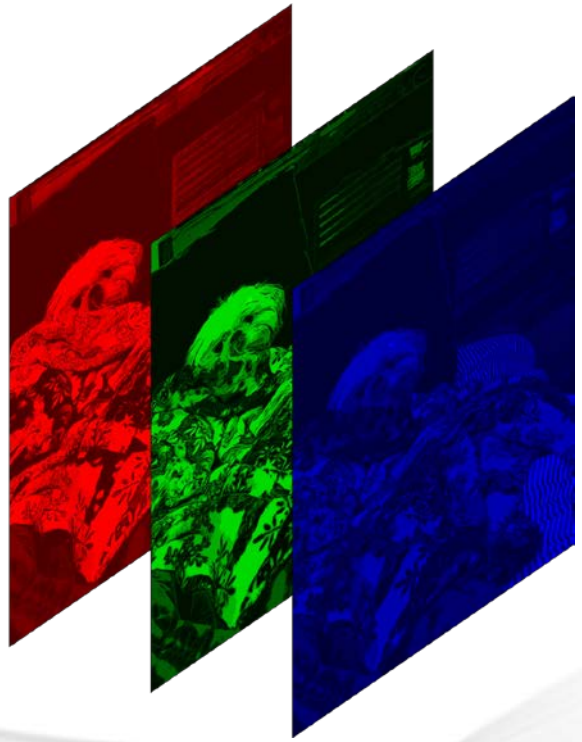
Introduzione alla Computer Vision

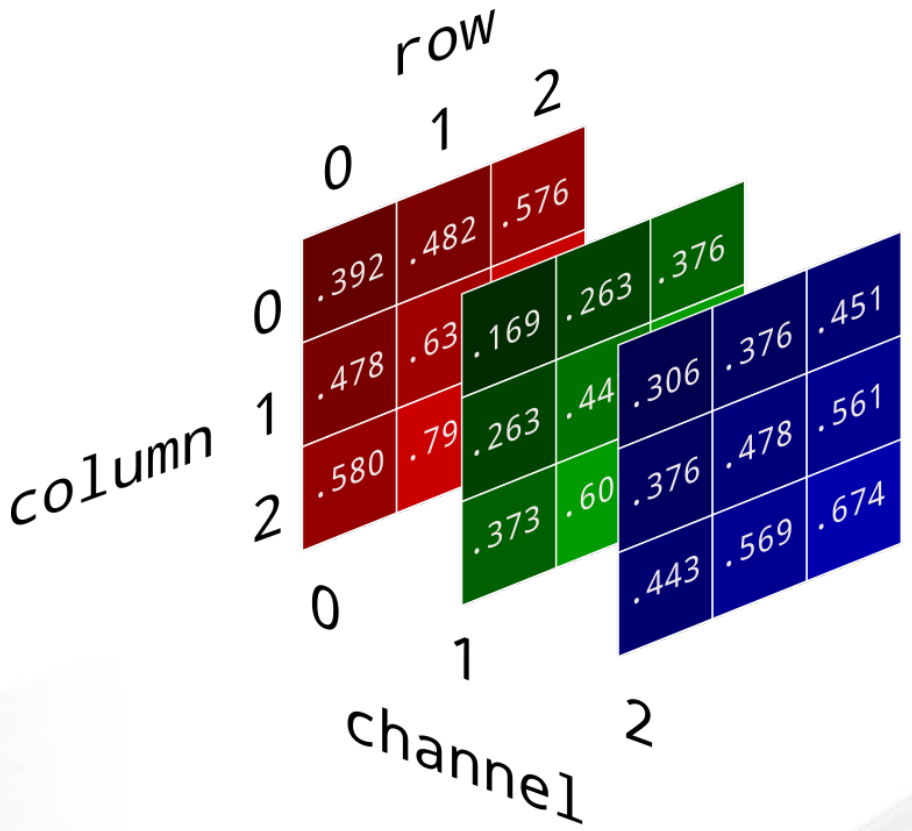
- La prima cosa importante da capire è ciò che un computer vede quando guarda un'immagine: non un contenuto visivo reale, ma un array di valori numerici di pixel.



Immagini RGB

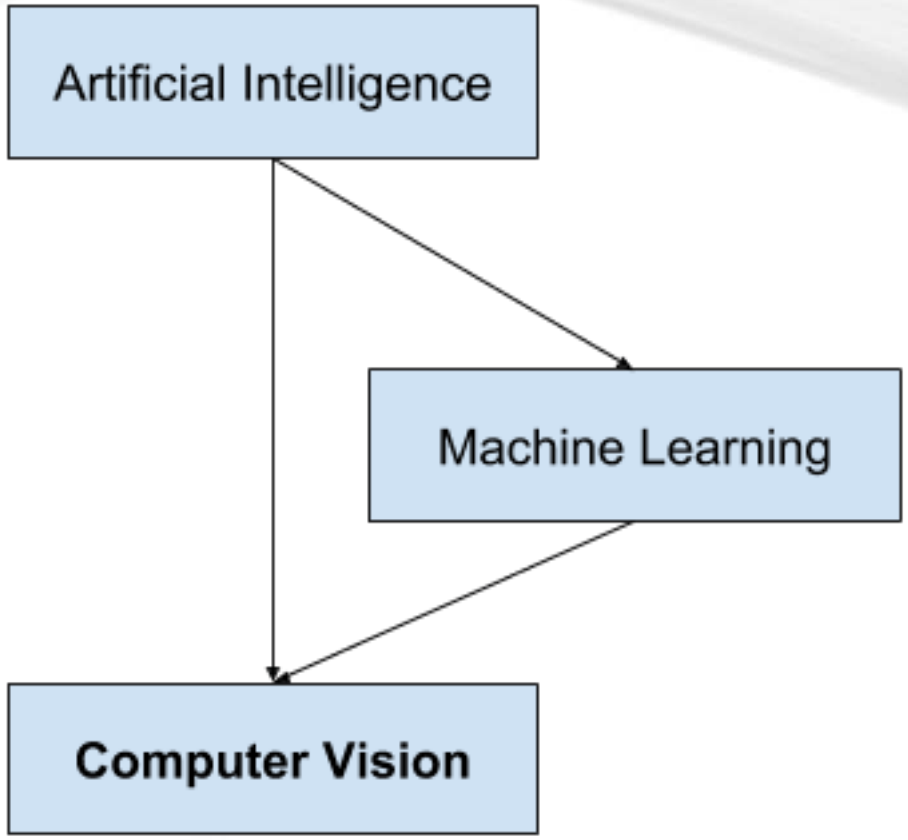
- Le immagini a colori sono rappresentate come array tridimensionali, una collezione di tre matrici, uno per ciascuno dei canali rosso, verde e blu. Ognuno di essi, ha un valore per pixel e i loro intervalli sono identici [0, 255]





Computer vision e AI

- La Computer Vision è un campo multidisciplinare che potrebbe essere definito in generale un sottocampo dell'intelligenza artificiale e del machine learning, che può comportare l'uso di metodi specifici e l'uso di algoritmi di apprendimento generali.

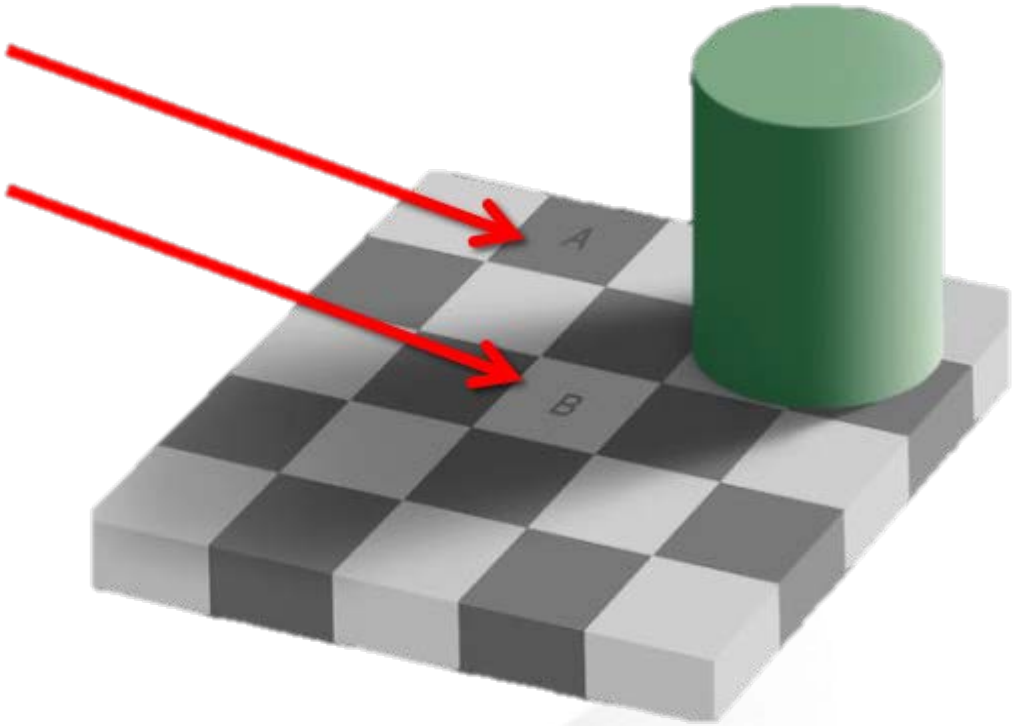


Perché è così difficile?

- La visione non è soltanto image processing

Casella scura

Casella chiara

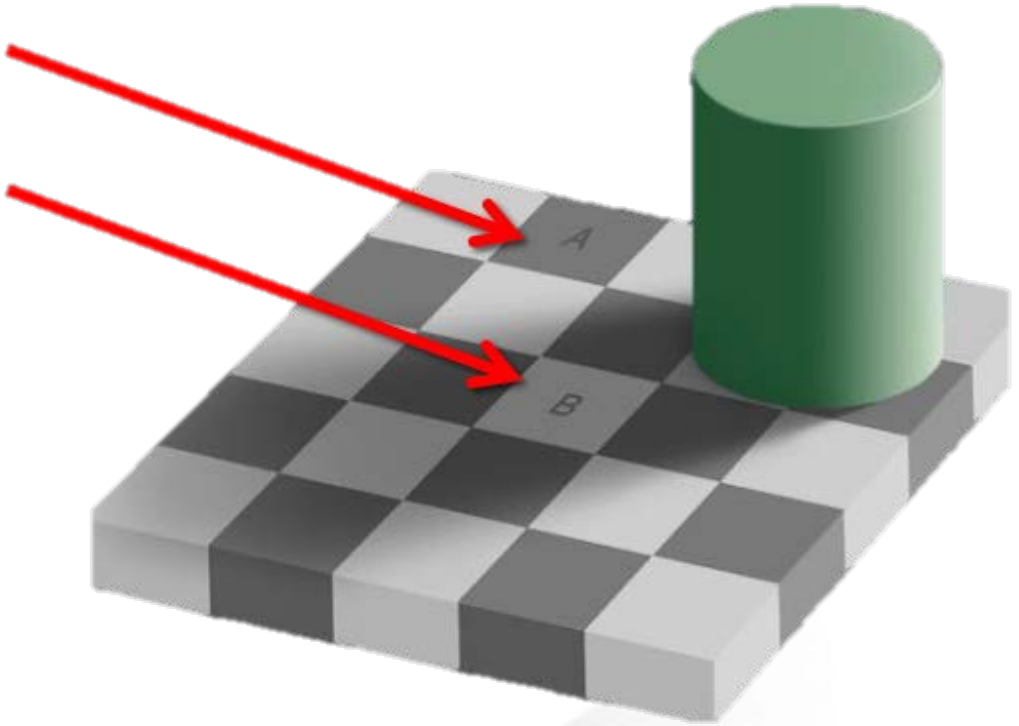


Perché è così difficile?

- La visione non è soltanto image processing

Casella scura

Casella chiara

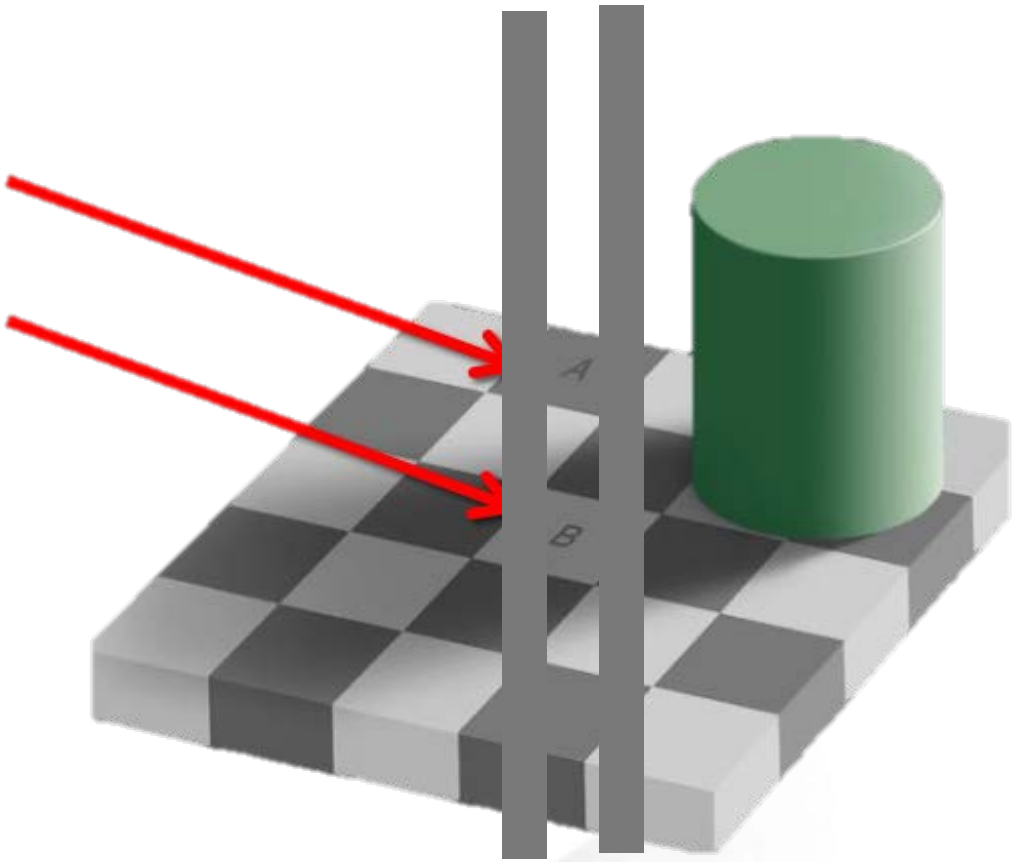


Perché è così difficile?

- La visione non è soltanto image processing

Casella scura

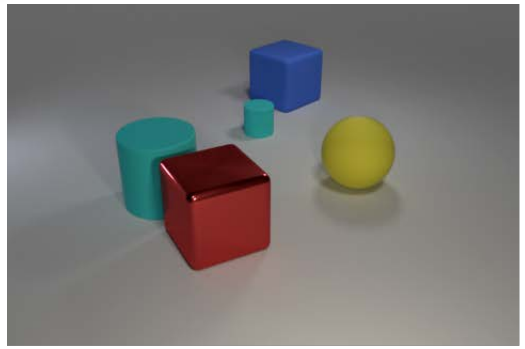
Casella chiara



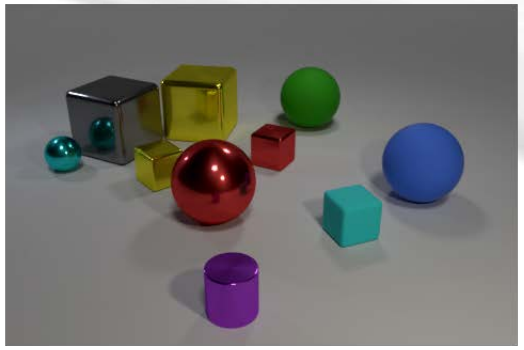
Perché è così difficile?

Nell'esempio precedente, i due quadrati hanno esattamente la stessa misura di intensità. Quindi, vedere non equivale a misurare le proprietà dell'immagine.

Piuttosto, "vedere" è costruire una percezione di ciò che è nel mondo sulla base delle misurazioni effettuate da un sensore di immagini.



Q: *What **shape** is the red thing?*
A: *Cube*



Q: *There is a large ball that is on the **right side** of the rubber sphere that is **behind** the tiny cyan object on the **left side of** the large green matte thing; what is its **color**?*
A: *Blue*

Il task di **Visual Reasoning** prova ad allenare un modello a comprendere il contenuto dell'immagine

Classificazione

- Quando noi umani vediamo e classifichiamo qualcosa, il nostro cervello dà un senso ad esso etichettando, prevedendo e riconoscendo pattern specifici. Allo stesso modo, i classificatori di immagini si basano su strutture che prendono ispirazione dalla biologia del cervello: le cosiddette **reti neurali convoluzionali** (CNN),

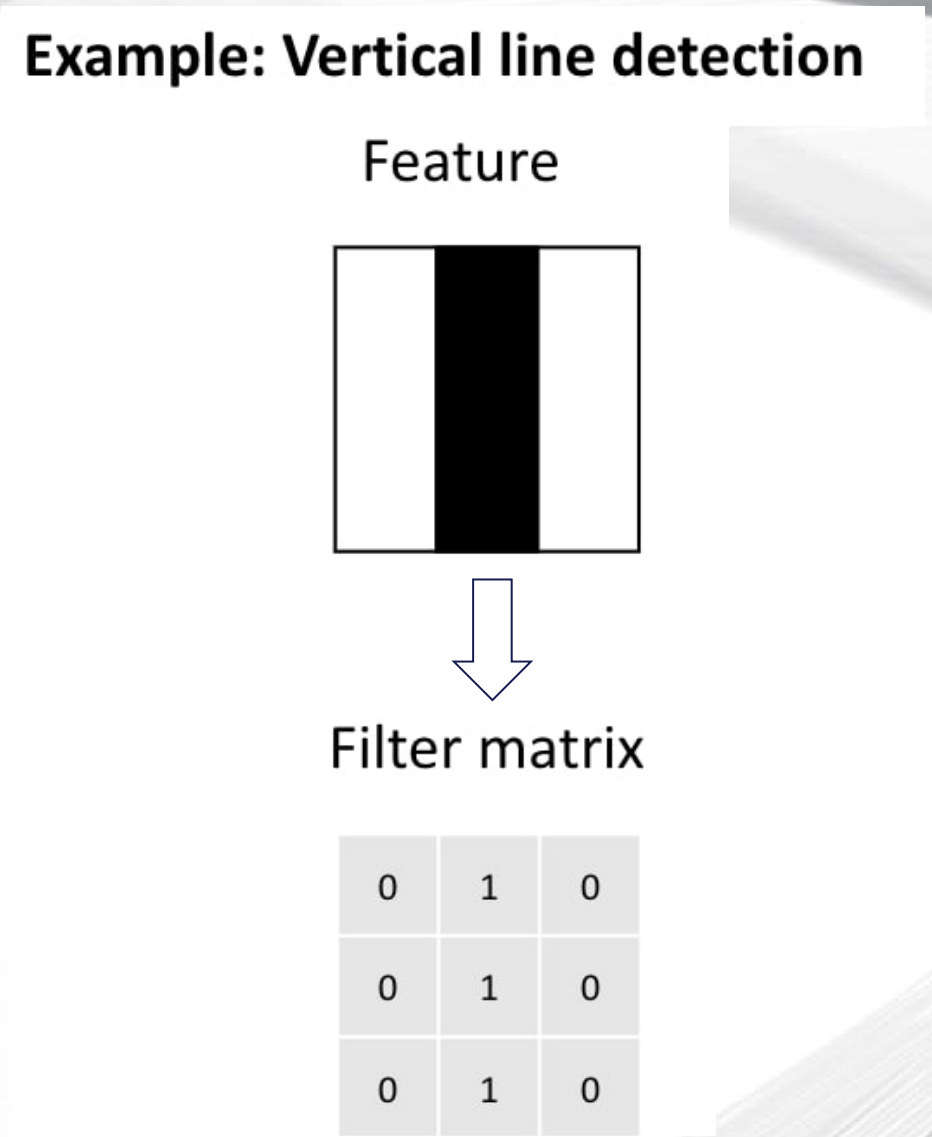
- Quello che vogliamo in un task di classificazione di immagini è prendere la matrice di numeri che costituisce un'immagine e produrre la probabilità che l'immagine sia di una certa classe (ad esempio 90% per il cane, 10% per il gatto).

Convoluzione

I layer di convoluzione sono i principali elementi costitutivi dei classificatori di immagini. Il termine matematico convoluzione si riferisce alla combinazione di due funzioni (f e g) che producono una terza funzione (z).

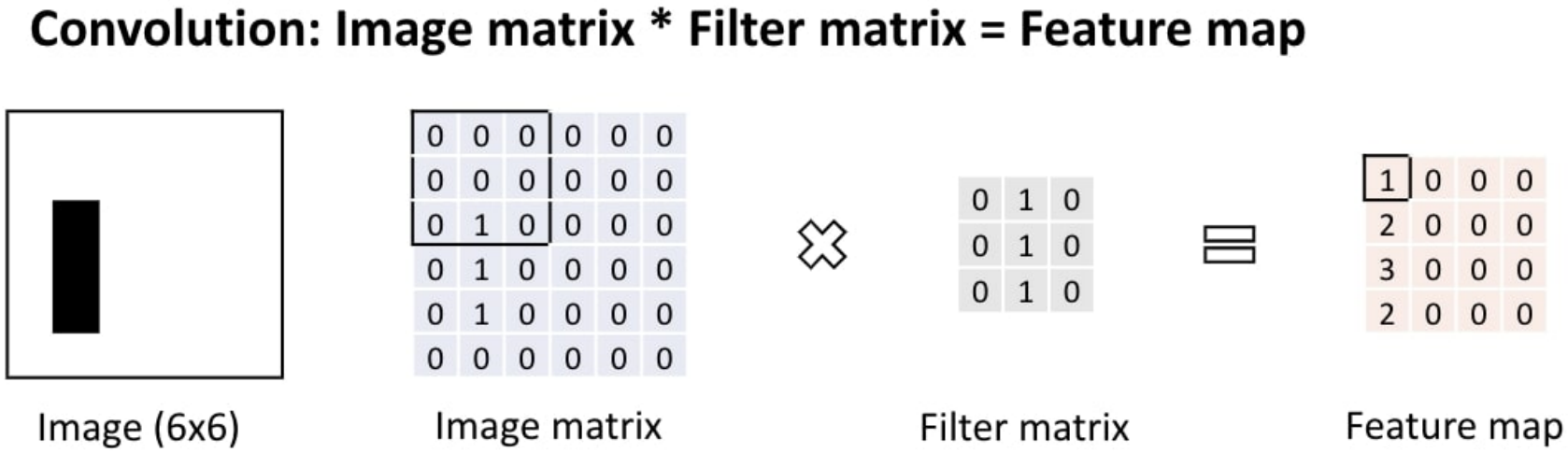
Un layer di convoluzione prende un input, applica un filtro (kernel) e produce una feature map. La feature map (z) è una combinazione di input e filtro (f e g), da cui il nome layer di convoluzione. L'obiettivo della convoluzione è di estrarre delle caratteristiche significative di un'immagine.

Una feature è una caratteristica specifica dell'immagine originale, come punti, bordi o la forma del naso di un cane. Similmente all'immagine che viene elaborata come un array di numeri, una feature si traduce in una casella di valori numerici dei pixel. Questa matrice serve come rilevatore di feature.



Convoluzione

- Per scansionare l'immagine, questo kernel si muove sull'immagine, blocco di pixel per blocco di pixel. Per ogni sottoregione, viene calcolato un valore basato su quanto è buona la corrispondenza tra il filtro e l'immagine.



Convoluzione

- In computer vision "tradizionale" i Kernel sono definiti **manualmente**, in modo tale da **estrarre determinate caratteristiche (features) di interesse** dall'immagine di input.

Input image

Convolution Kernel

$$\begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}$$

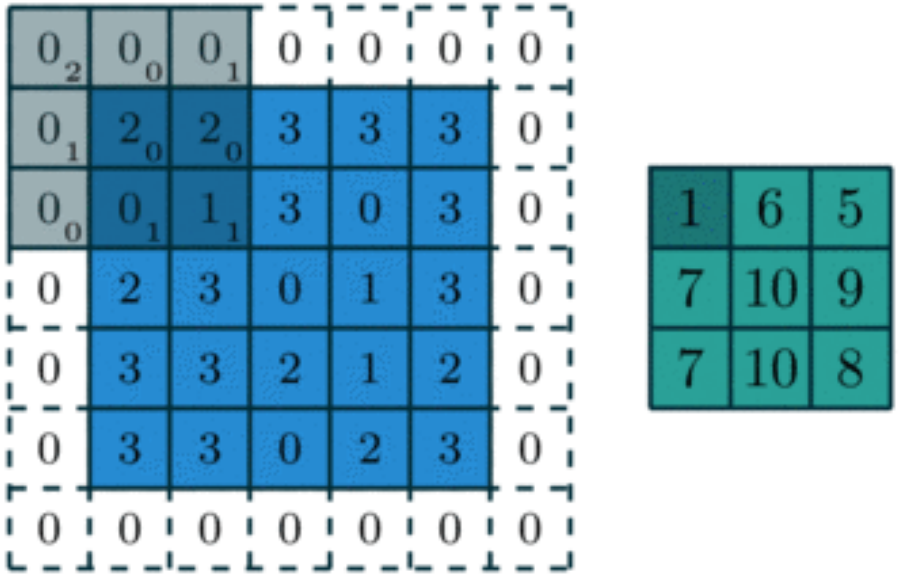
Feature map

A kernel (filtri) diversi, corrispondono diverse features estratte

Convoluzione

- Ci sono due parametri che controllano come l'operazione di convoluzione viene effettuata: **lo stride** (orizzontale e verticale) ed il **padding**.

- Questi due parametri altro non sono che il numero di pixel da "saltare" quando si fa scorrere il kernel sull'immagine di input (stride), e da aggiungere come cornice all'input (padding)

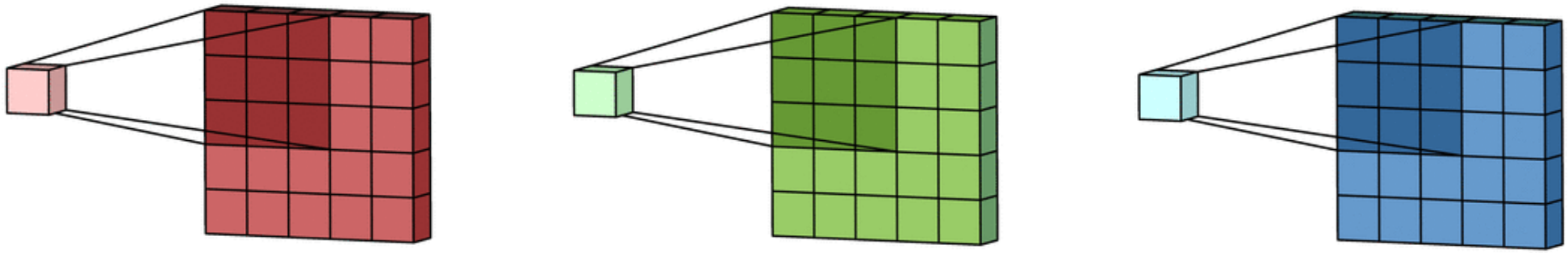


Convoluzione tra un immagine 5×5, con uno **zero padding** 1×1 ed un kernel 3×3, usando stride 2×2

Convoluzione

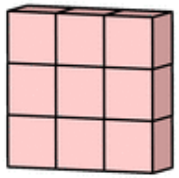
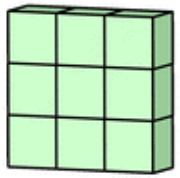
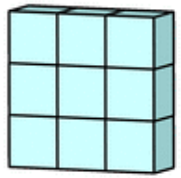
Data un immagine **RGB**, la **convoluzione** con un **kernel 3×3** viene effettuata facendo:

- La convoluzione di ogni canale per il rispettivo filtro a profondità 1



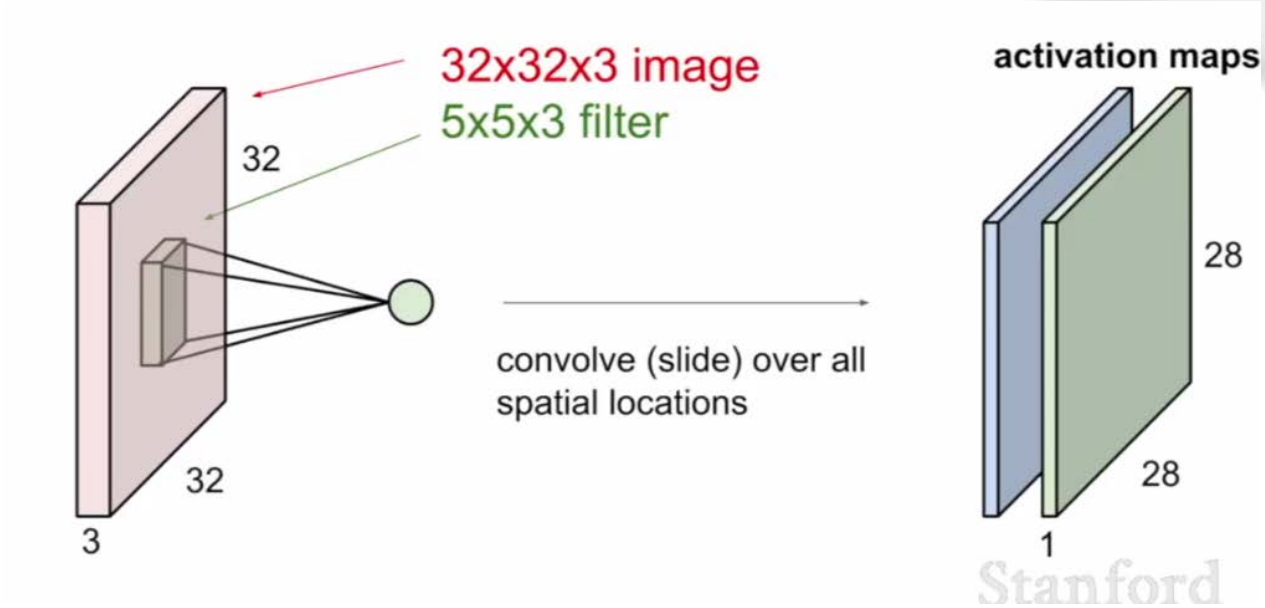
Convoluzione

- La somma delle **feature map** estratte

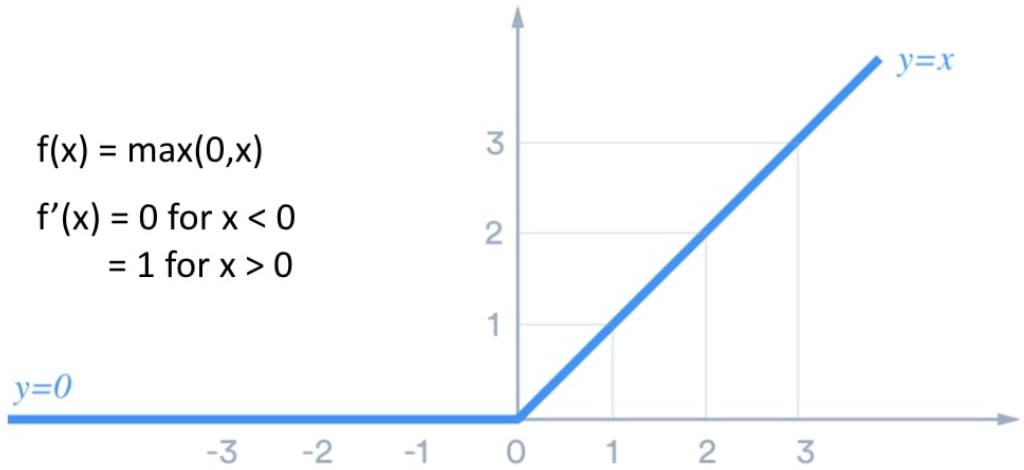
Imparare i kernel

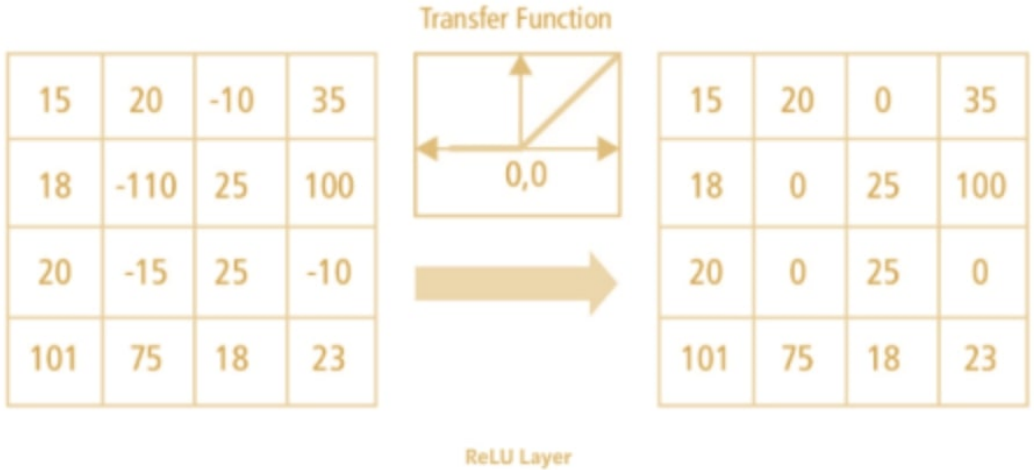
- Apprendere filtri convoluzionali consiste nel **definire un numero arbitrario N di filtri da apprendere** ed eseguire N convoluzioni tra volumi, in maniera indipendente.
- Ogni filtro convoluzionale è **un neurone** che osserva una regione locale dell'immagine (sulla quale scorre).
- Ogni convoluzione produce una feature map, che passa attraverso una funziona di attivazione e diventa una **activation map**.



ReLU

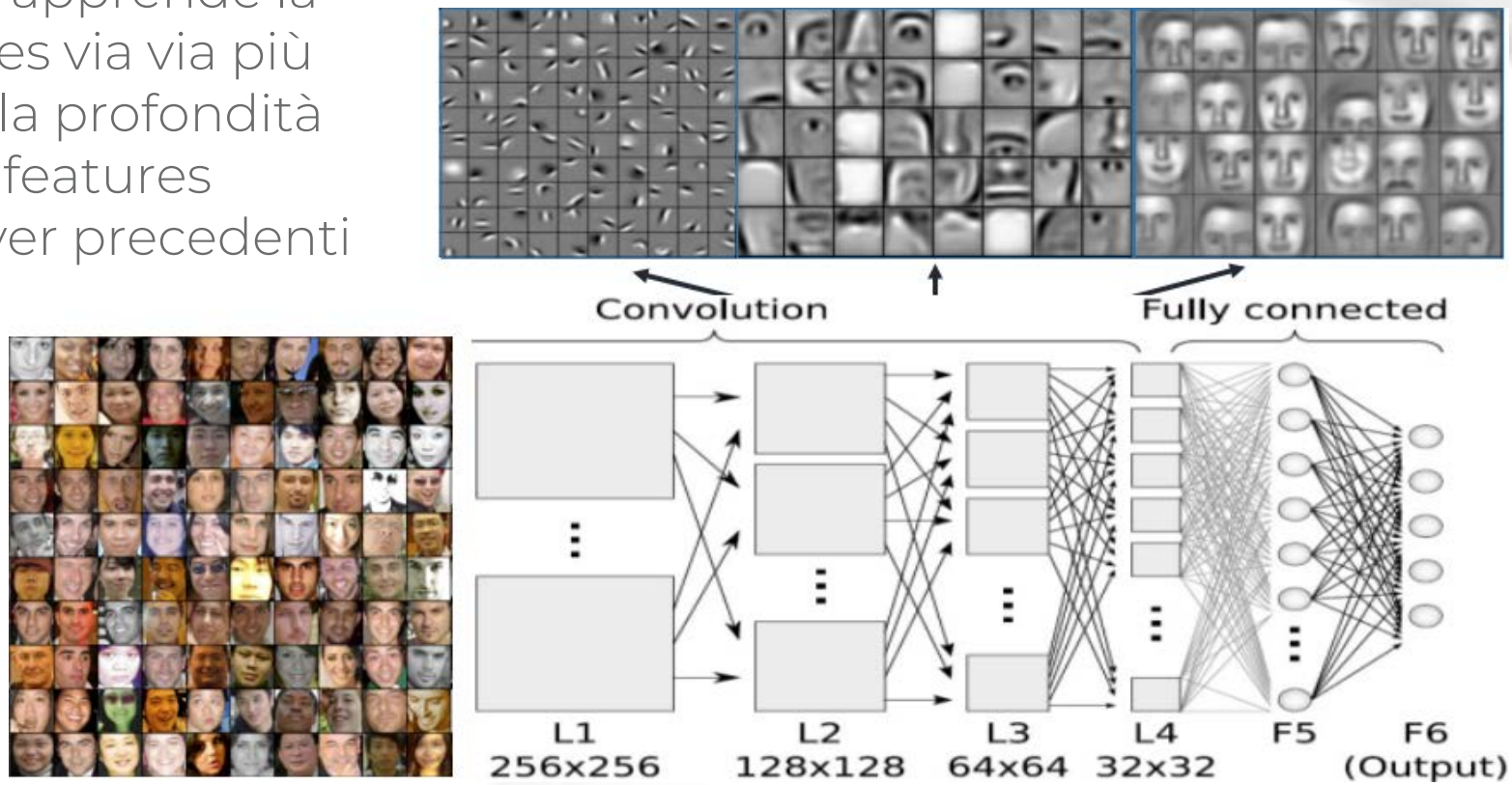
- ReLU è l'abbreviazione di unità lineare rettificata ed è una cosiddetta funzione di attivazione. L'obiettivo principale dell'uso di una funzione di attivazione è di aggiungere non linearità al calcolo. Uno strato ReLU prende una feature map e rettifica qualsiasi valore negativo a zero. I numeri positivi rimangono invariati.





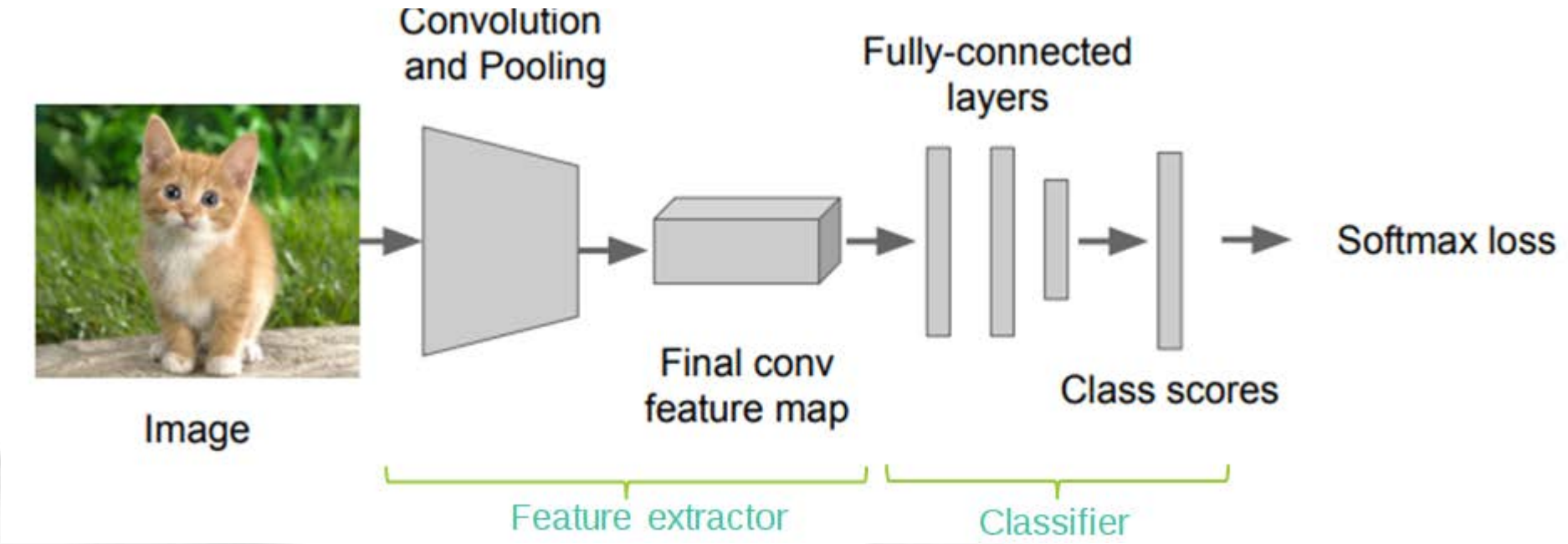
Layers di convoluzione

- Ogni layer convoluzionale apprende la capacità di estrarre features via via più astratte all'aumentare della profondità della rete, combinando le features elementari estratte dai layer precedenti



Classificazione ed estrazione di features

- Essendo a tutti gli effetti una **rete neurale**, possiamo definire l'architettura in modo tale da estrarre **un numero arbitrario di features** (a bassa dimensionalità) ed utilizzarlo come **input di una classificatore** (rete FC).



YOLO

- **YOLO** è stato introdotto per la prima volta nel 2016 ed è stato una pietra miliare nella ricerca sul rilevamento degli oggetti grazie alla sua capacità di rilevare gli oggetti in tempo reale con una migliore precisione.

- È stato proposto da **Joseph Redmon**, laureato dell'Università di Washington. L'articolo che descrive YOLO ha vinto il premio OpenCV People's Choice Award nella conferenza su Computer Vision e Pattern Recognition (**CVPR**) nel 2016.

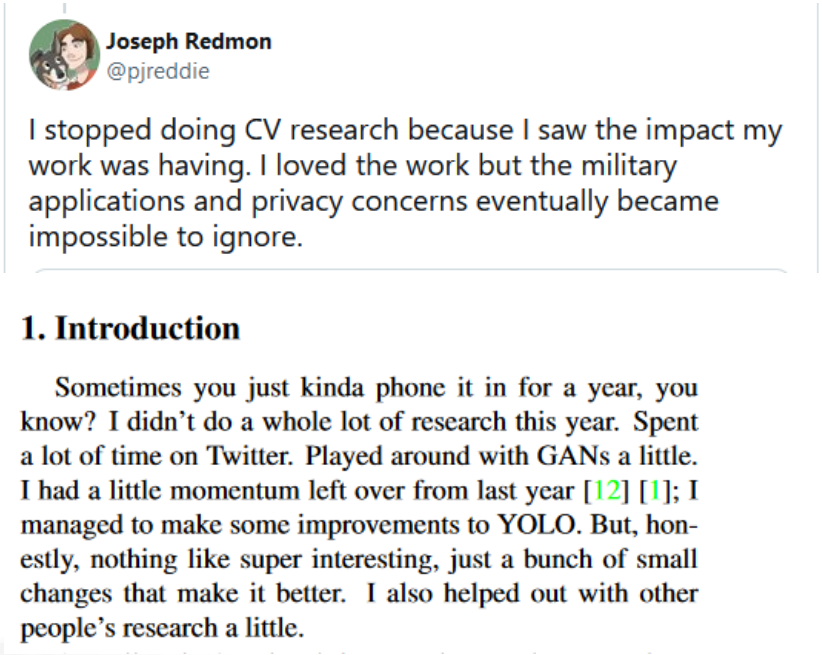
Versioni di YOLO di Joseph Redmon:

- Version 1
  'You Only Look Once: Unified, Real-Time Object Detection' (2016)
- Version 2
  'YOLO9000: Better, Faster, Stronger' (2017)
- Version 3
  'YOLOv3: An Incremental Improvement' (2018)

YOLO v3

- L'implementazione principale di YOLO di Redmon è basata su **Darknet**, che è un framework di reti neurali open source scritto in C e CUDA. Darknet imposta l'architettura di base della rete e la usa come framework per l'addestramento di YOLO. Questa implementazione è stata introdotta da Redmon stesso ed è veloce, facile da installare e supporta il calcolo su CPU e GPU.

- Tuttavia, nel febbraio 2020, Joseph Redmon, il creatore di **YOLO ha annunciato di aver interrotto la sua ricerca nel campo della computer vision!** Ha inoltre dichiarato che ciò era dovuto a diverse preoccupazioni riguardanti il potenziale impatto negativo del suo lavoro.



Yolov3 - introduzione

YOLO v4

Il ritiro di Redmon non è stata la fine di YOLO. In ricordo di molti nella comunità della computer vision, la quarta generazione di YOLO è stata rilasciata nell'aprile 2020. È stato introdotto in un documento intitolato '*YOLOv4: Optimal Speed and Accuracy of Object Detection*' by Alexey Bochkovskiy et al.

Inoltre, il lavoro di Redmon è stato continuato da Alexey nel fork del repository principale. YOLO v4 è stato considerato il modello in tempo reale più veloce e accurato per il rilevamento di oggetti.

YOLO v5

- Dopo il rilascio di YOLO v4, in soli due mesi di tempo, è stata rilasciata un'altra versione di YOLO chiamata YOLO v5! E' opera di Glenn Jocher, già noto nella comunità per aver creato la popolare implementazione PyTorch di YOLO v3.

- Il 9 giugno 2020, Jocher ha dichiarato che la sua implementazione di YOLO v5 è rilasciata pubblicamente ed è raccomandata per l'uso in nuovi progetti. L'implementazione di YOLOv5 di Jocher differisce dalle versioni precedenti in alcuni modi notevoli. In primo luogo, Jocher non ha (ancora) pubblicato un documento per accompagnare il suo rilascio. Secondo, Jocher ha implementato YOLOv5 nativamente in PyTorch, mentre tutti i modelli precedenti della famiglia YOLO sfruttano Darknet.

- Il rilascio di YOLO v5 ha attirato molta attenzione e ha causato accese discussioni nelle piattaforme della comunità Machine Learning. Questo era principalmente dovuto a diversi fatti su un articolo pubblicato dal team Roboflow riguardo a YOLO v5.

YOLO architettura

- YOLO fa uso solo di layer convoluzionali, rendendola una rete completamente convoluzionale (FCN). Ha **75 layer di convoluzione**, con **skip connection** e **layer di upsampling**. Non viene utilizzata alcuna forma di pooling, e uno layer di convoluzione con stride 2 viene utilizzato per ricampionare le feature map. Questo aiuta a prevenire la perdita di feature di basso livello.

- Essendo una FCN, YOLO è invariante alla dimensione dell'immagine di ingresso. Tuttavia, in pratica, potremmo voler attenerci a una dimensione di input costante a causa di vari problemi implementativi.

- Uno di questi problemi è che se vogliamo elaborare le nostre immagini in **batch** (le immagini in batch possono essere elaborate in parallelo dalla **GPU**, con conseguente aumento della velocità), abbiamo bisogno di avere tutte le immagini di altezza e larghezza fissa. Questo è necessario per concatenare più immagini in unico batch.

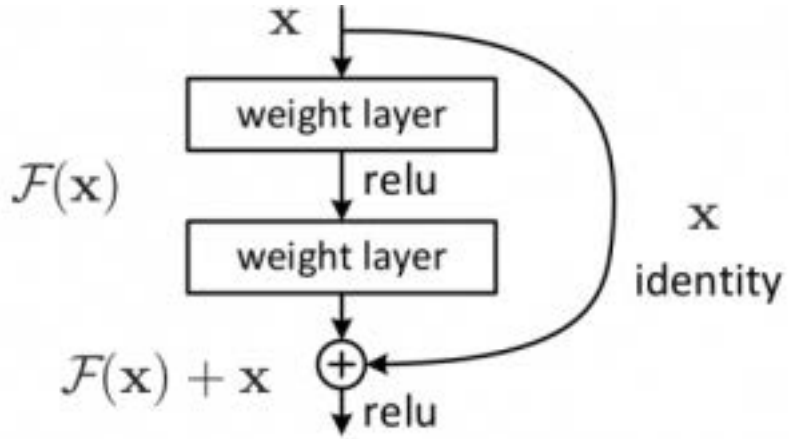
- La rete scala l'immagine di un fattore chiamato **stride della rete**. Per esempio, se lo stride della rete è **32**, allora un'immagine di input di dimensioni **416 x 416** produrrà un output di dimensioni **13 x 13**. In generale, lo stride di qualsiasi strato della rete è uguale al fattore per cui l'uscita dello strato è più piccola dell'immagine in ingresso alla rete.

YOLO – skip connection

Le reti neurali deep possono imparare funzioni complesse in modo più efficiente delle loro controparti non-deep. Tuttavia, durante l'addestramento delle reti neurali deep, le prestazioni del modello diminuiscono con l'aumento della profondità dell'architettura. Questo è noto come problema della degradazione.

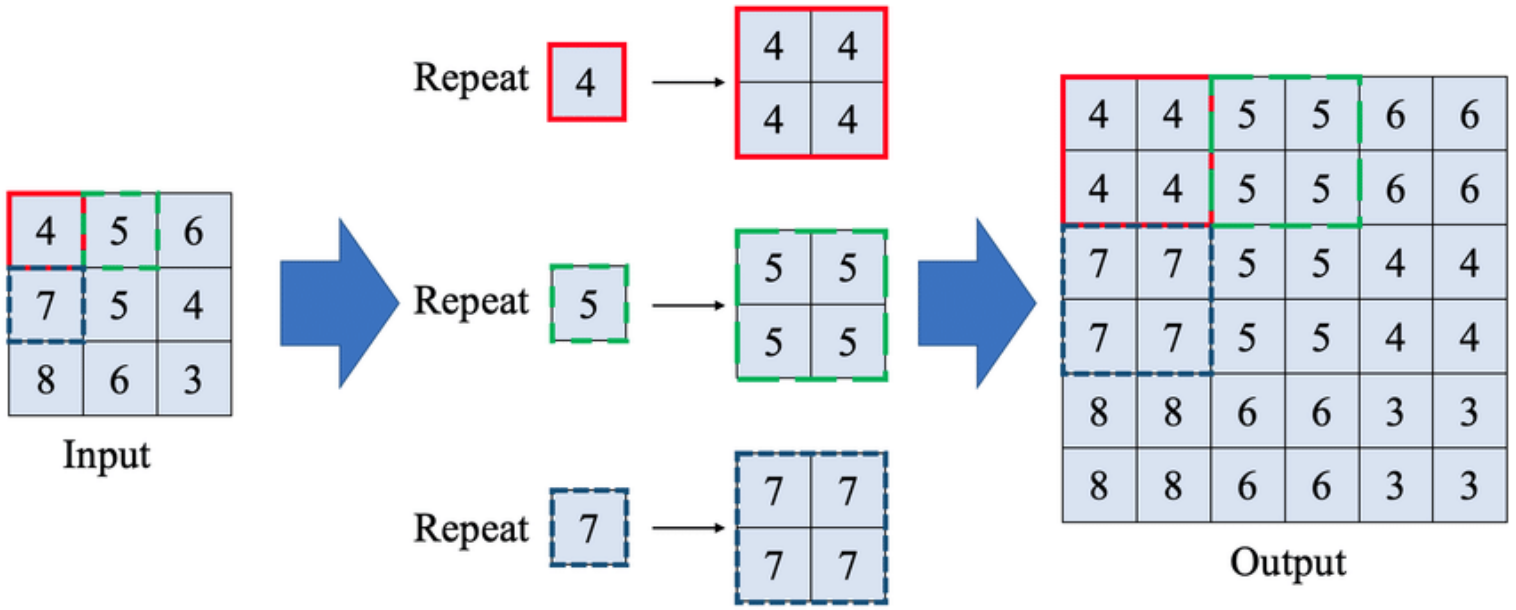
Skip Connections (o Shortcut Connections), come suggerisce il nome, salta alcuni dei layer della rete neurale e alimenta l'uscita di un layer come ingresso dei layer successivi.

Le Skip Connections sono state introdotte per risolvere diversi problemi in diverse architetture. Nel caso di ResNets, skip Connections hanno risolto il problema della degradazione.



YOLO – layer di upsampling

Un **layer di upsampling** aumenta la larghezza e l'altezza dell'array di input. Il metodo più semplice per fare upsampling è ripetere l'elemento della matrice di input in due direzioni.



YOLO vs Sliding window

Nel metodo sliding window quello che veniva fatto era una scansione lineare dell'immagine alla ricerca di oggetti conosciuti. Se all'interno di una finestra viene riconosciuto un oggetto, allora non solo potremmo dire che quell'oggetto è presente nell'immagine, ma anche indicarne la posizione.

Questa tecnica presenta però dei difetti. In particolare ogni oggetto riconoscibile tramite questo metodo deve avere una dimensione adatta per rientrare nella finestra di scansione e non deve trovarsi in una posizione tale da finire sulla linea di demarcazione tra due finestre adiacenti.

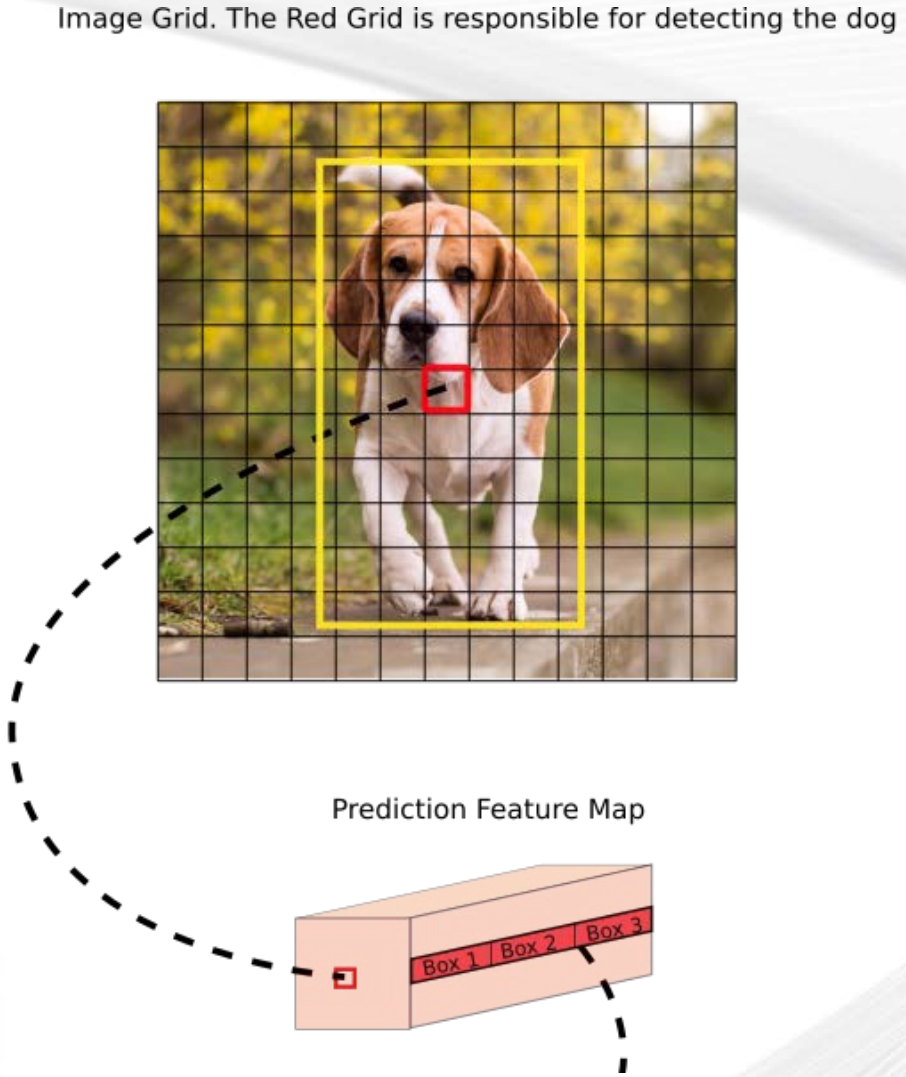
YOLO (You Only Look Once) permette di riconoscere più oggetti e, allo stesso tempo, di identificarne la posizione e lo spazio occupato effettuando una sola "lettura" dell'immagine di input.

YOLO – feature map

- Consideriamo un esempio dove l'immagine **di input è 416 x 416**, e lo **stride della rete è 32**. Come indicato in precedenza, le dimensioni **feature map saranno 13 x 13**. Dividiamo quindi l'immagine di input in 13 x 13 celle.

- La cella (sull'immagine di input) che contiene il valore di Ground Truth è scelta per essere quella responsabile della predizione dell'oggetto. Nell'immagine, è la cella segnata in rosso.

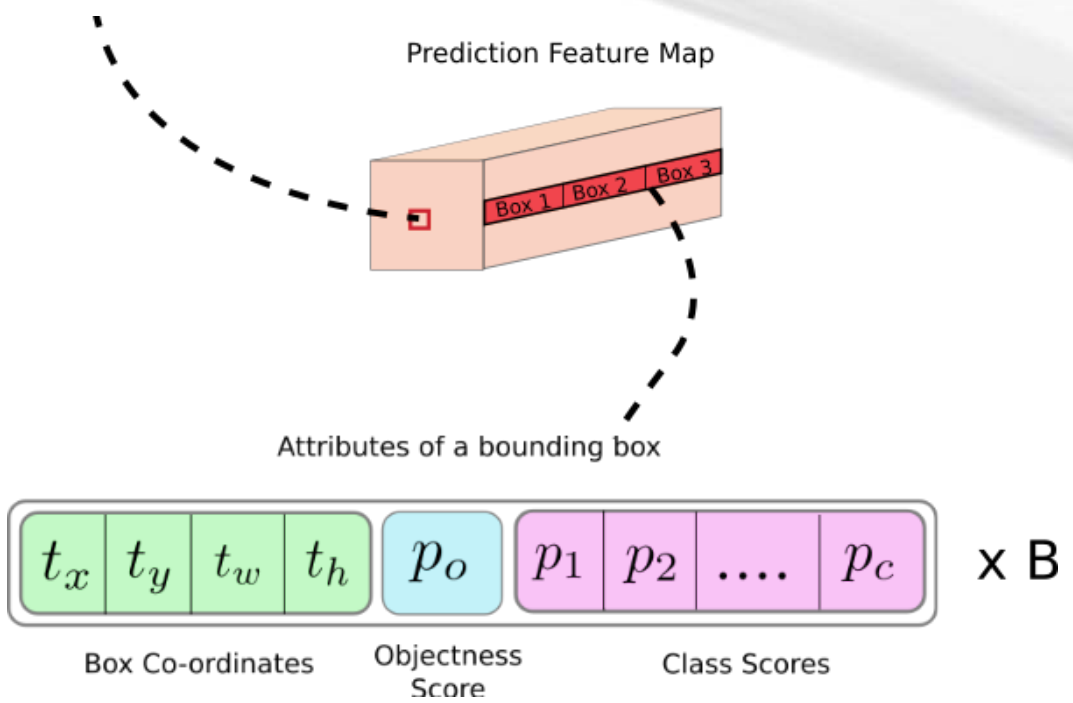
- La cella rossa è la 7a cella nella 7a riga della griglia. Ora assegniamo la 7a cella nella 7a riga sulla feature map come responsabile del rilevamento del cane.



YOLO - output

- Ognuna delle bounding box ha 5 + C attributi, dove C è il numero delle classi di oggetti che il modello può classificare.

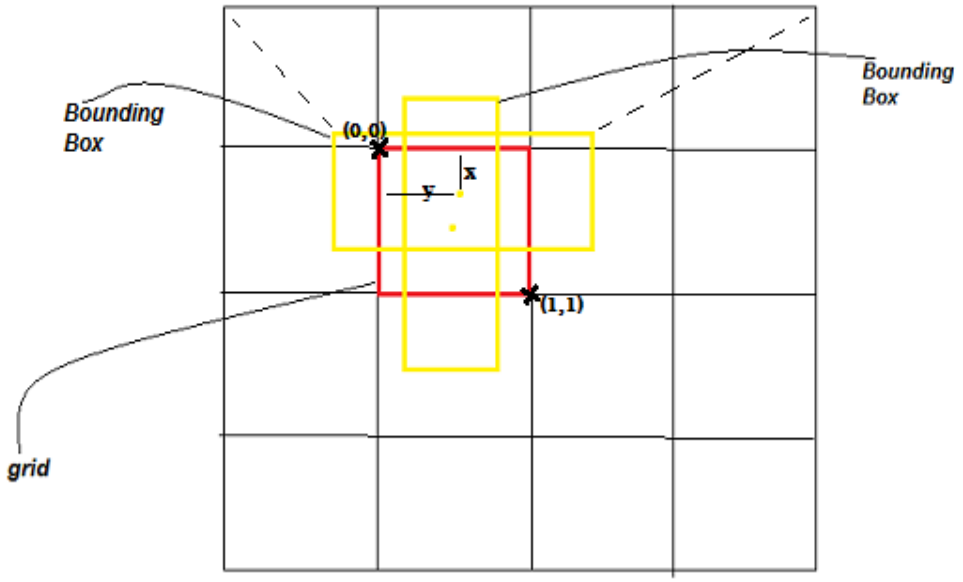
t_x : Coordinata x del centro della bounding box
t_y : Coordinata y del centro della bounding box
t_w : Larghezza della bounding box
t_h : Altezza della bounding box
p_o : Score
p_1 : Probabilità che l'oggetto appartenga alla classe 1
...
p_c : Probabilità che l'oggetto appartenga alla classe c



YOLO - output

- Inoltre in YOLO ogni cella può predire **tre** bounding box, quindi il vettore di output finale avrà dimensione:

$$3 \times (5 + N\_classi)$$



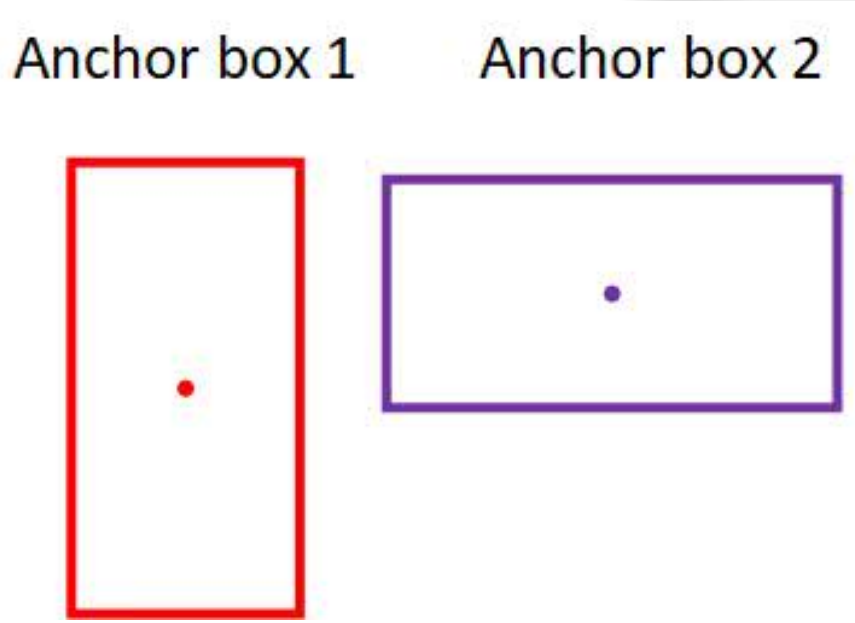
YOLO – anchor boxes

Anchor Boxes

- Potrebbe avere senso prevedere la larghezza e l'altezza della bounding box, ma in pratica, questo porta a gradienti instabili durante l'allenamento. Invece, la maggior parte dei moderni classificatori predice le trasformazioni log-space, o semplicemente gli offset delle bounding box predefinite chiamate **ancore**.

- Poi, queste trasformazioni sono applicate alle anchor boxes per ottenere la predizione. YOLO utilizza tre ancore, che si traducono nella predizione di tre bounding box per cella.

- La bounding box responsabile del rilevamento del cane sarà quello la cui ancora ha il più alto **IoU** con il riquadro del GT.



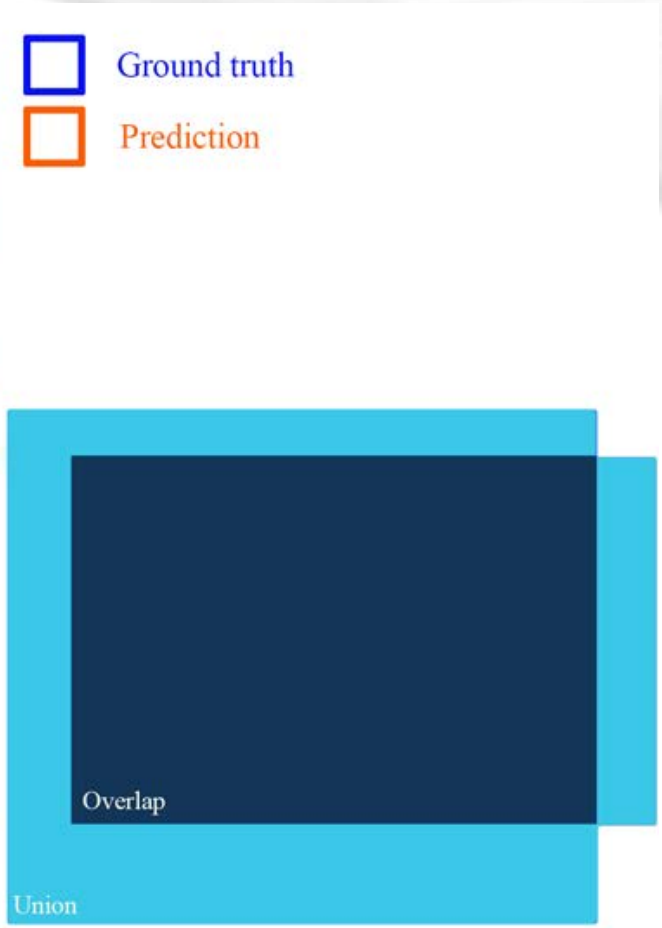
YOLO - IoU

IoU

Per decidere se una bounding box predetta è a tutti gli effetti una "detection" è possibile utilizzare l'intersection over union (delle aree):

$$IoU = \frac{A \cap B}{A \cup B}$$

Il suo valore è tra 0 (nessuna detection) ed 1 (overlap completo).

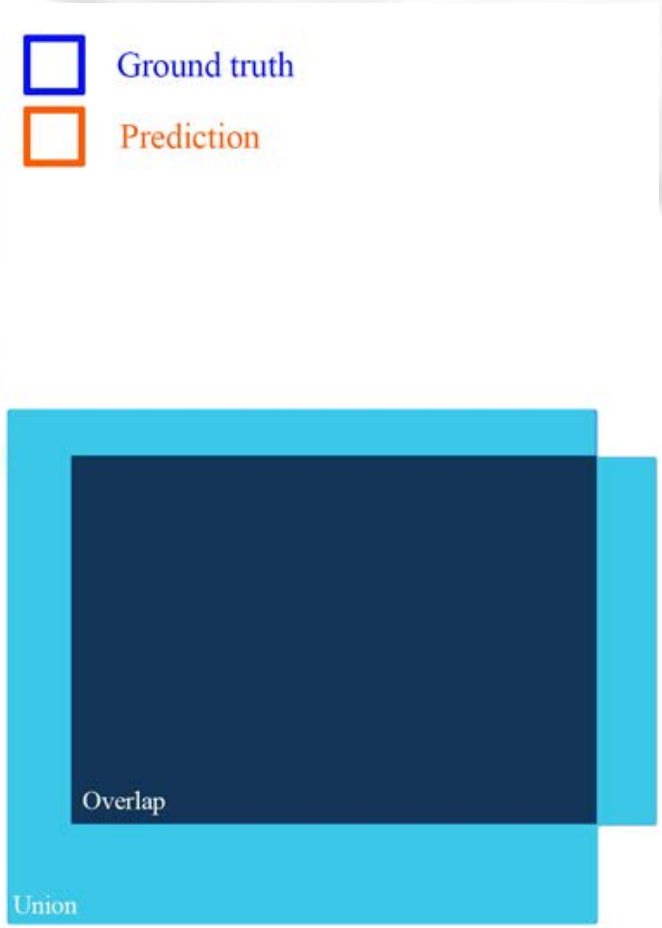


YOLO - IoU

Per *decidere* se una bounding box predetta è da considerare una **detection**, si fissa una soglia di IoU e vengono considerate come detection solo quelle predizioni di bounding box che superano la soglia.

Così facendo ci ritroviamo nel caso di una classificazione binaria, in cui possiamo avere:

- True positives: detection
- False positives: detection errate (non passano la soglia)



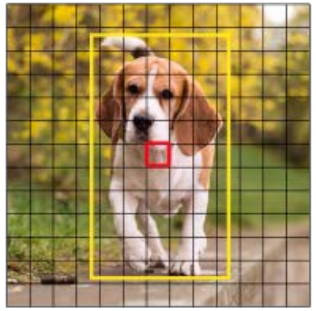
YOLO - Scales

YOLO v3 fa previsioni su **3 scale** diverse. Il layer di detection è utilizzato per predire feature map di tre dimensioni diverse, con **stride 32, 16, 8** rispettivamente. Ciò significa che, con un input di **416 x 416**, si effettuano detection su scale **13 x 13, 26 x 26 e 52 x 52**.

La rete ricampiona l'immagine di input fino al primo layer di detection, dove viene effettuato una prima detection utilizzando le feauture map di di un layer con stride 32.

Inoltre, gli strati sono sovracampionati di un fattore 2 e concatenati con le feature map di uno strato precedente con dimensioni identiche. Un'altra detection vieni quindi fatta al layer con stride 16. La stessa procedura di upsampling è ripetuta, e una detection finale è fatto al layer con stride 8.

Prediction Feature Maps at different Scales



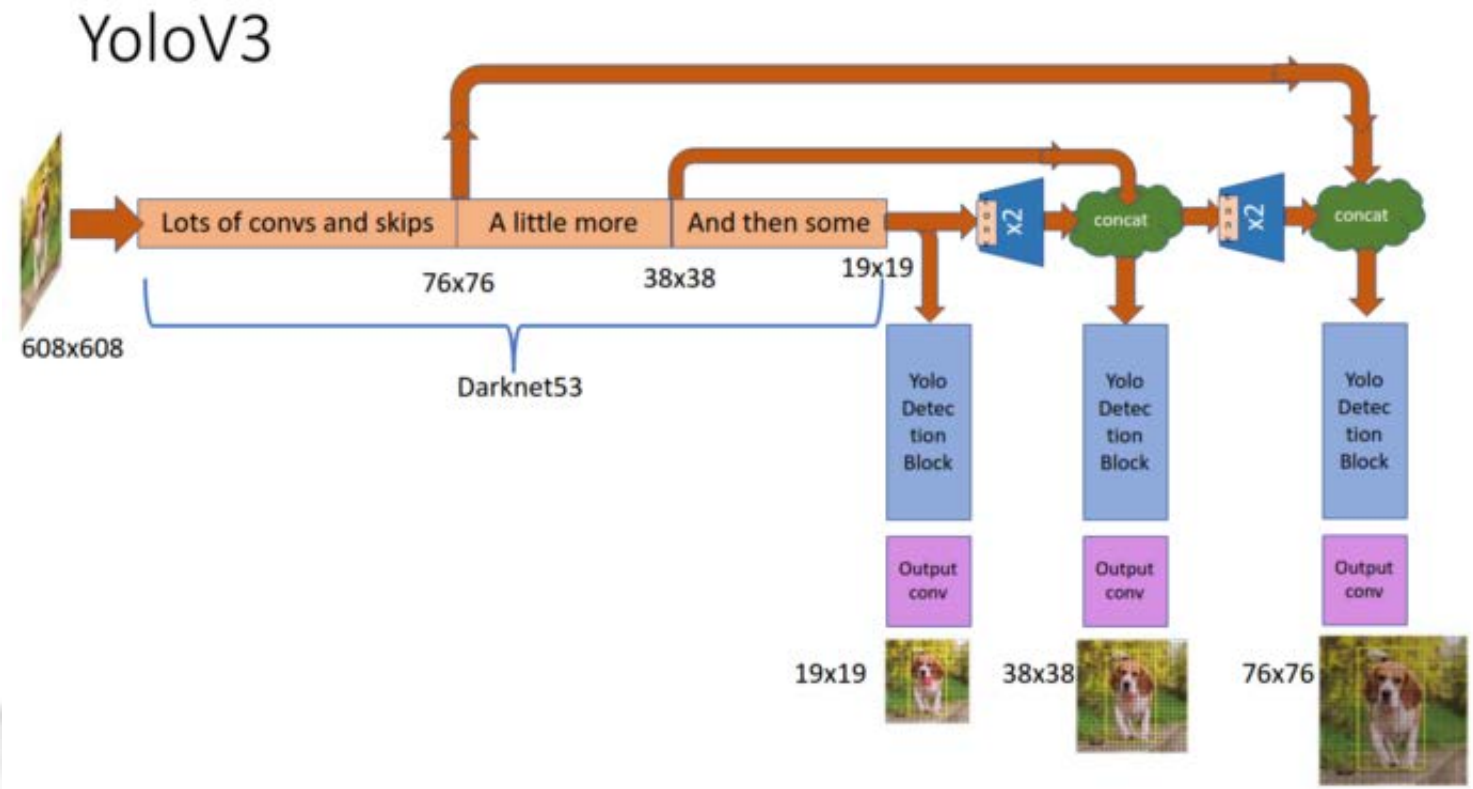
13 x 13

26 x 26

52 x 52

YOLO - Scales

Ad ogni scala, ogni cella predice 3 bounding box usando 3 ancore, rendendo il numero totale di ancore usate 9. (Le ancore sono diverse per scale diverse)



YOLO - NMS

Per un'immagine di dimensioni 416 x 416, YOLO predice ((52 x 52) + (26 x 26) + 13 x 13)) x 3 = **10647 bounding box**.

Algoritmo Non-max suppression

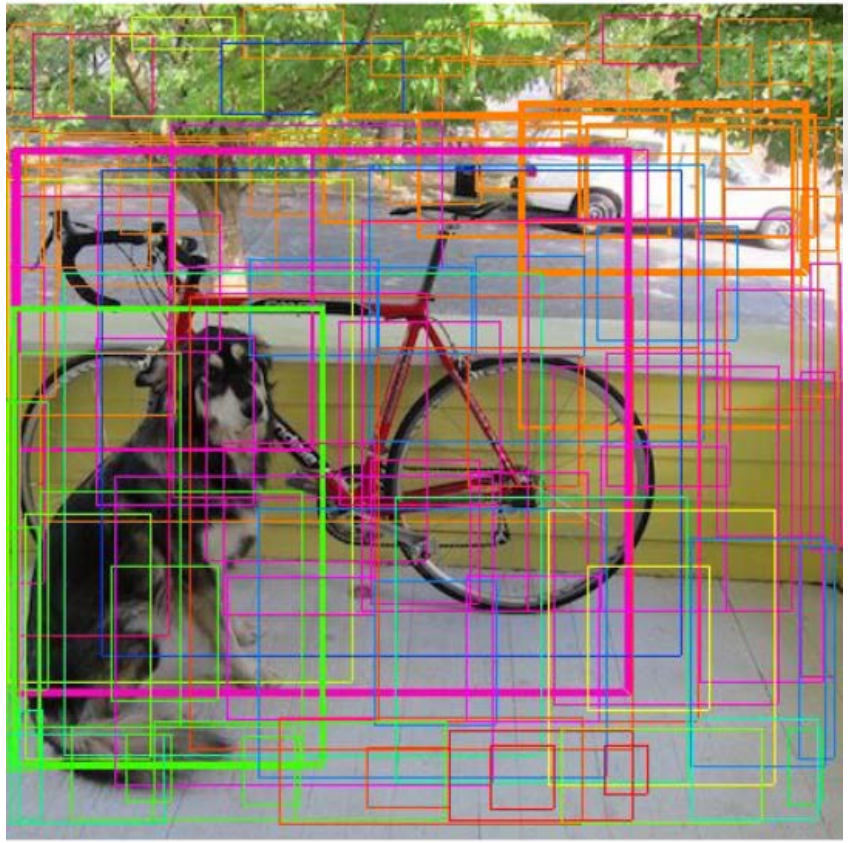
- Rimuovere tutte le bounding box con uno score p_o < 0.5

Con le rimanenti detection:

- Selezionare la bounding box con il più alto score p_o e utilizzarla come output della predizione
- Rimuovere tutte le rimanenti bounding box con

  IoU > 0.45

con la bounding box utilizzata come output allo step precedente



YOLO - NMS

Per un'immagine di dimensioni 416 x 416, YOLO predice ((52 x 52) + (26 x 26) + 13 x 13)) x 3 = 10647 bounding box.

Algoritmo Non-max suppression

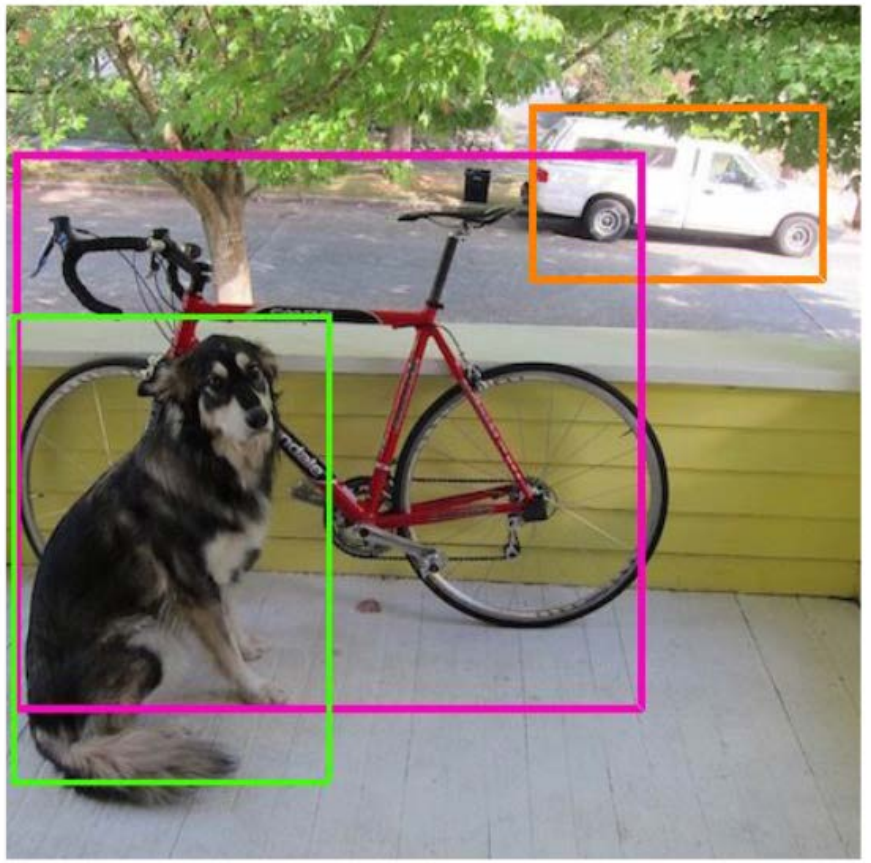
- Rimuovere tutte le bounding box con uno score $p_o < 0.5$

Con le rimanenti detection:

- Selezionare la bounding box con il più alto score p_o e utilizzarla come output della predizione
- Rimuovere tutte le rimanenti bounding box con

  IoU > 0.45

con la bounding box utilizzata come output allo step precedente



YOLO – Funzione di Loss

$$\lambda_{coord} \sum_{i=0}^{S^2} \sum_{j=0}^{B} 1_{ij}^{obj} [(x_i - \hat{x}_i)^2 + (y_i - \hat{y}_i)^2]$$

$$+ \lambda_{coord} \sum_{i=0}^{S^2} \sum_{j=0}^{B} 1_{ij}^{obj} [(\sqrt{w_i} - \sqrt{\hat{w}_i})^2 + (\sqrt{h_i} - \sqrt{\hat{h}_i})^2]$$

$$+ \sum_{i=0}^{S^2} \sum_{j=0}^{B} 1_{ij}^{obj} (C_i - \hat{C}_i)^2 + \lambda_{noobj} \sum_{i=0}^{S^2} \sum_{j=0}^{B} 1_{ij}^{noobj} (C_i - \hat{C}_i)^2$$

$$+ \sum_{i=0}^{S^2} 1_i^{obj} \sum_{c \in classes} (p_i(c) - \hat{p}_i(c))^2$$

x_i , y_i è la posizione del centroide della bounding box

w_i , h_i è la larghezza e l'altezza della bounding box

C_i è l'Objectness, cioè il punteggio di confidenza della presenza o meno di un oggetto

p_i(c) è la loss di classificazione.

YOLO - Funzione di Loss

$$\lambda_{coord} \sum_{i=0}^{S^2} \sum_{j=0}^{B} 1_{ij}^{obj} [(x_i - \hat{x}_i)^2 + (y_i - \hat{y}_i)^2]$$

$$+ \lambda_{coord} \sum_{i=0}^{S^2} \sum_{j=0}^{B} 1_{ij}^{obj} [(\sqrt{w_i} - \sqrt{\hat{w}_i})^2 + (\sqrt{h_i} - \sqrt{\hat{h}_i})^2]$$

$$+ \sum_{i=0}^{S^2} \sum_{j=0}^{B} 1_{ij}^{obj} (C_i - \hat{C}_i)^2 + \lambda_{noobj} \sum_{i=0}^{S^2} \sum_{j=0}^{B} 1_{ij}^{noobj} (C_i - \hat{C}_i)^2$$

$$+ \sum_{i=0}^{S^2} 1_i^{obj} \sum_{c \in classes} (p_i(c) - \hat{p}_i(c))^2$$

1. Penalizza la cattiva localizzazione del centro delle celle
2. Penalizza il rettangolo di selezione con altezza e larghezza imprecise.
3. Rende il punteggio di confidenza uguale all'IOU tra l'oggetto e la predizione quando c'è un oggetto
4. Rende il punteggio di confidenza vicino a 0 quando non ci sono oggetti nella cella
5. Loss di classificazione

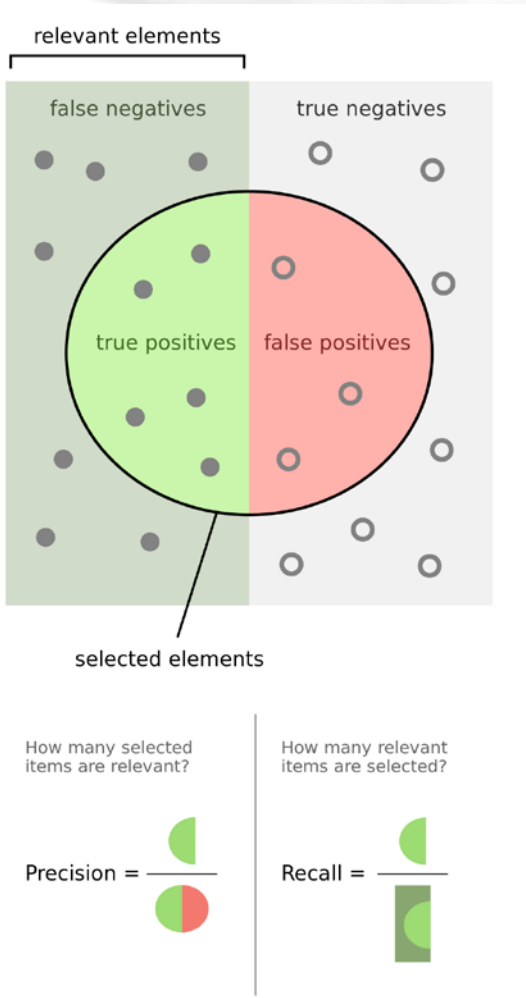
Metriche

La **precision** misura quanto sono accurate le vostre previsioni, cioè la percentuale di previsioni corrette. Misura quante delle previsioni che il modello ha fatto erano effettivamente corrette.

La **recall** misura quanto bene trovi tutti i positivi. Per esempio, possiamo trovare l'80% dei possibili casi positivi nelle nostre prime K previsioni.

$$Precision = \frac{TP}{TP + FP}$$

$$Recall = \frac{TP}{TP + FN}$$



Metriche

a) True positive, perché la sovrapposizione del GT (box giallo) e della predizione (box verde) è superiore allo 0,5

b) False positive, poiché le bounding box di predizione (verdi) non si sovrappongono a nessuna delle annotazioni di GT.

c) False negative, poiché l'annotazione di GT (giallo) non viene rilevato dal modello

(a) (b) (c)

$$class(IoU) = \begin{cases} Positive \rightarrow IoU \geq Threshold \\ Negative \rightarrow IoU < Threshold \end{cases}$$

Metriche

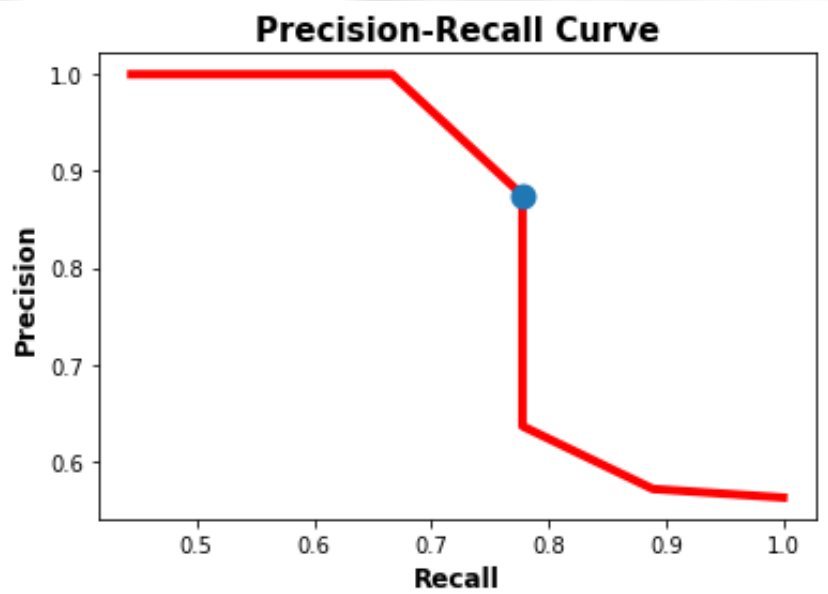
La definizione generale per l'Average Precision (AP) è trovare l'area sotto la curva precision-recall.

.

mAP (mean Average Precision) è la media dell'AP su tutte le classi di oggetti.

$$\mathrm{AP} = \int_0^1 \mathrm{P}(\mathrm{R}) d\mathrm{R}$$

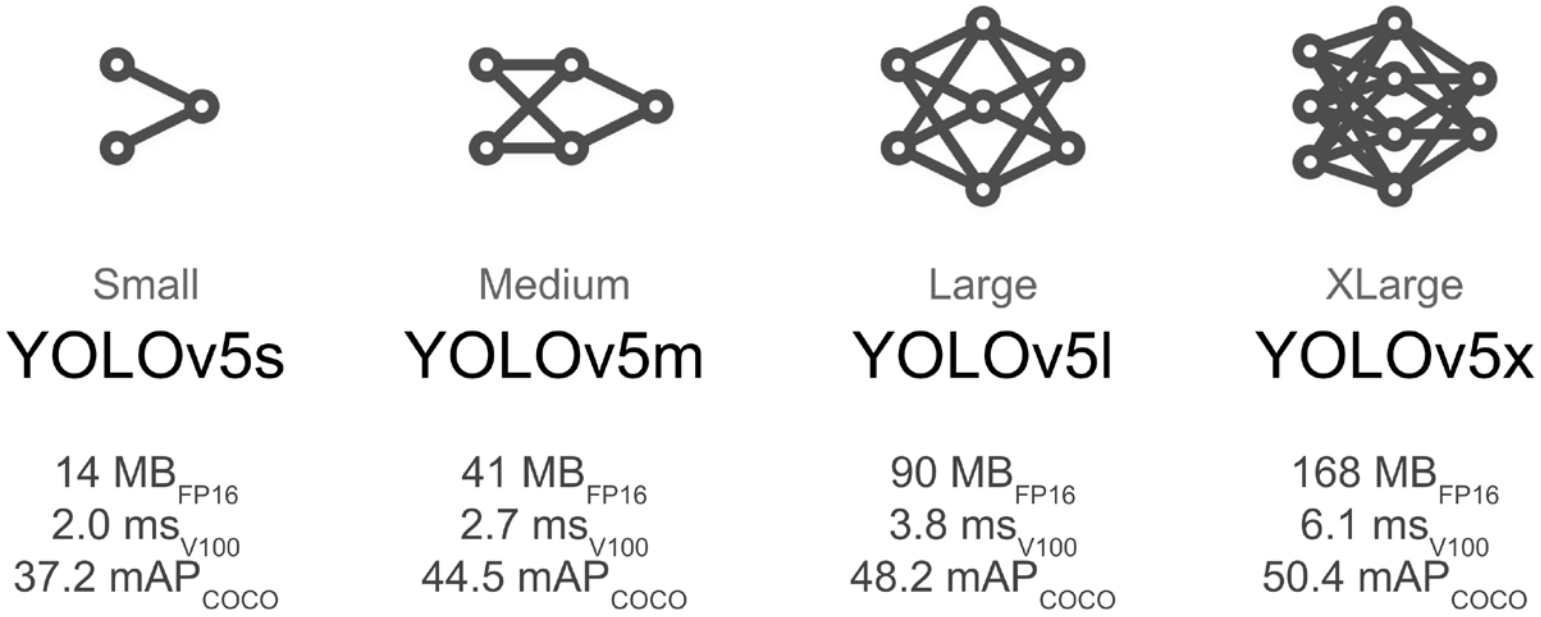
$$\mathrm{mAP} = \frac{1}{|Q_R|} \sum_{q=Q_R} \mathrm{AP}(q)$$



Curva precision (asse y) e recall (asse x). Il modello è considerato un buon modello predittivo se la precisione rimane alta mentre il richiamo aumenta

Lab

1. Exercise.py (da completare)
   Effettuare la detection di un'immagine utilizzando il modello YOLOv5s pre-allenato



Yolov5 models

Lab

2. Annotare un nuovo dataset utilizzando il tool RoboFlow, utilizzare il tool di data Augmentation per ingrandire il dataset

https://app.roboflow.com/

3. Re-allenare il modello sul nuovo dataset.

```
$ python train.py --img 640 --batch 16 --epochs 3 --data data.yaml --weights yolov5s.pt
```

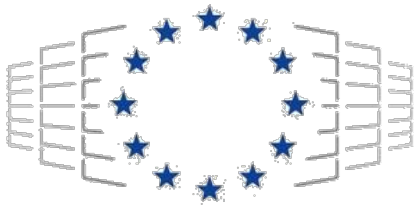
Lab

4. Visualizzare le metriche del modello allenato.
    - 4.1 Test freeze backbone
    - 4.2 Test cambiare il numero di epoche
    - 4.3 Test cambiare IoU threshold

5. Fare inference con il nuovo modello su un’immagine

6. Fare inference con il nuovo modello su un video

THANK YOU!



EuroHPC
Joint Undertaking

This project has received funding from the European High-Performance Computing Joint Undertaking (JU) under grant agreement No 951732. The JU receives support from the European Union's Horizon 2020 research and innovation programme and Germany, Bulgaria, Austria, Croatia, Cyprus, Czech Republic, Denmark, Estonia, Finland, Greece, Hungary, Ireland, Italy, Lithuania, Latvia, Poland, Portugal, Romania, Slovenia, Spain, Sweden, United Kingdom, France, Netherlands, Belgium, Luxembourg, Slovakia, Norway, Switzerland, Turkey, Republic of North Macedonia, Iceland, Montenegro

EUROCC ITALY