

Lezione 10 - Object detection -

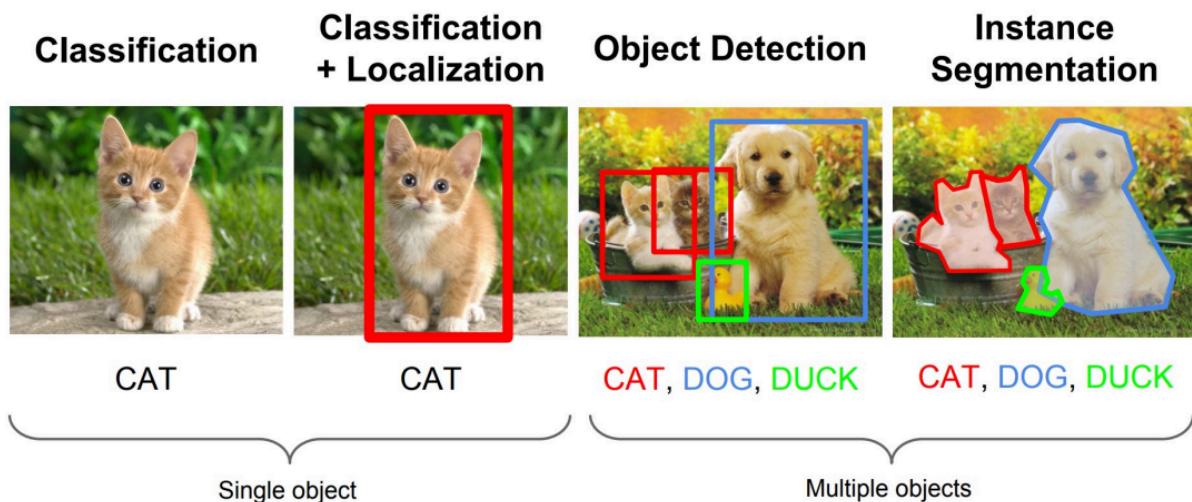
09/12/2024

Neural detectors

Questo è un approccio diverso rispetto ai keypoints, ci interessa trovare la posizione di 'ogni' gatto, non quel preciso gatto. Vogliamo trovare la posizione, la presenza o l'assenza.

Vedremo gli approcci recenti, basati su neural models.

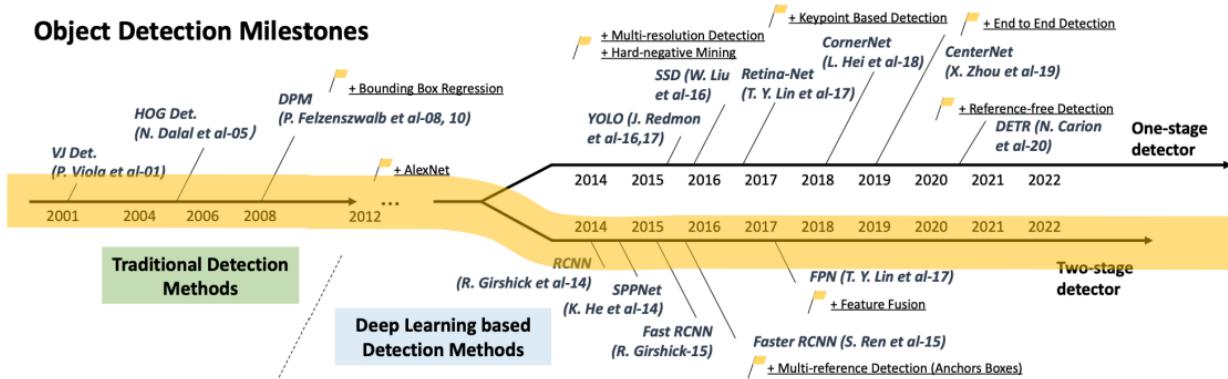
Terminologia delle task in computer vision:



A sinistra abbiamo la task 'semplice' di classificazione, dobbiamo solo sapere se l'oggetto principale dell'immagine è di quella categoria.

Man mano che andiamo su quelle a destra, la difficoltà aumenta.

La linea sopra è per gli one-stage detectors, e quella sotto per i two stage detectors.



Oggi vediamo YOLO e le prime 4 della linea sotto.

Partiamo con i two-stage detectors, e il problema di classification + localization.

Task: classification + localization

Qui abbiamo 2 problemi diversi: classificazione (sappiamo che ogni immagine contiene un oggetto principale che appartiene ad una delle classi) e localizzazione (vogliamo definire la posizione nell'immagine dove troviamo l'oggetto, quindi dobbiamo fare regression per diversi valori)

Classification: C classes

Input: Image

Output: Class label

Evaluation metric: Accuracy



Localization:

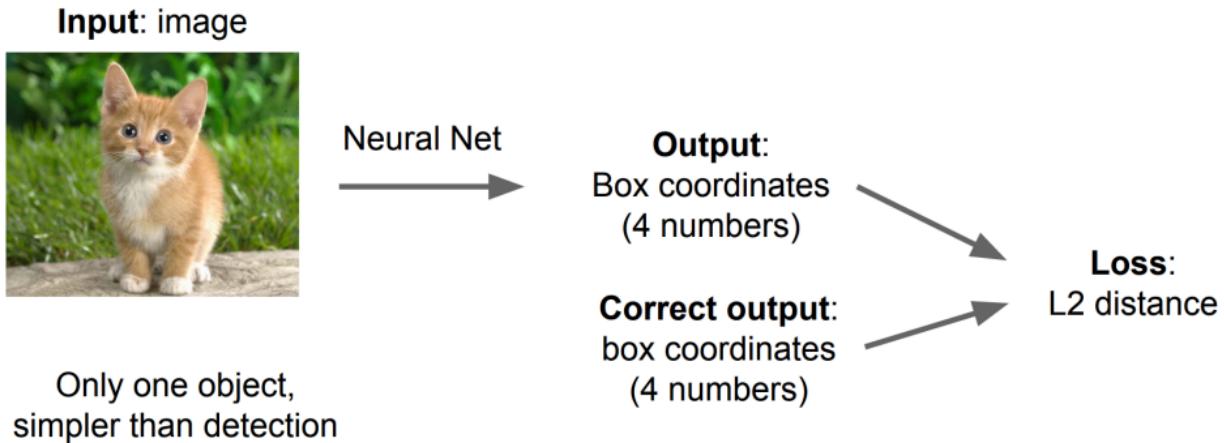
Input: Image

Output: Box in the image (x, y, w, h)

Evaluation metric: Intersection over Union



Un'altra possibilità, al posto di intersection over union, è questa_



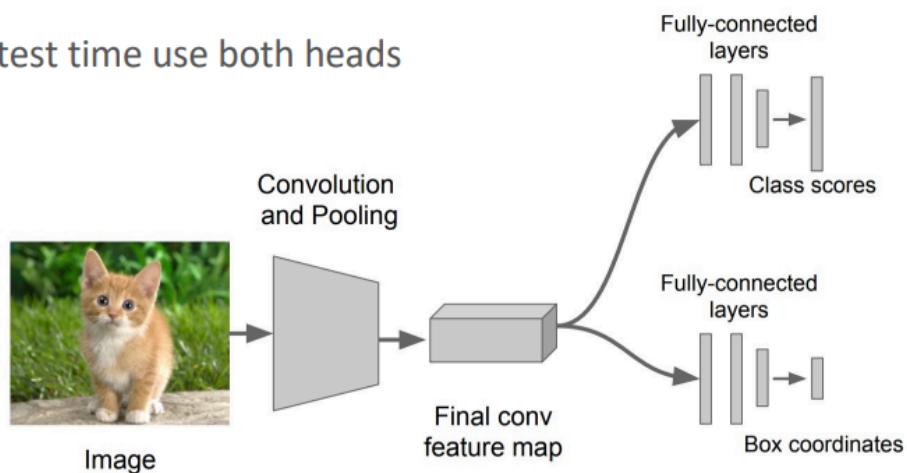
Quindi come possiamo fare a non avere 2 reti?

Innanzitutto facciamo la **localization as regression**, ...

Ad un certo punto splittiamo, freezando la prima parte dell'architettura, e alleniamo il pezzo che impara le pedizioni dei bounding boxes.

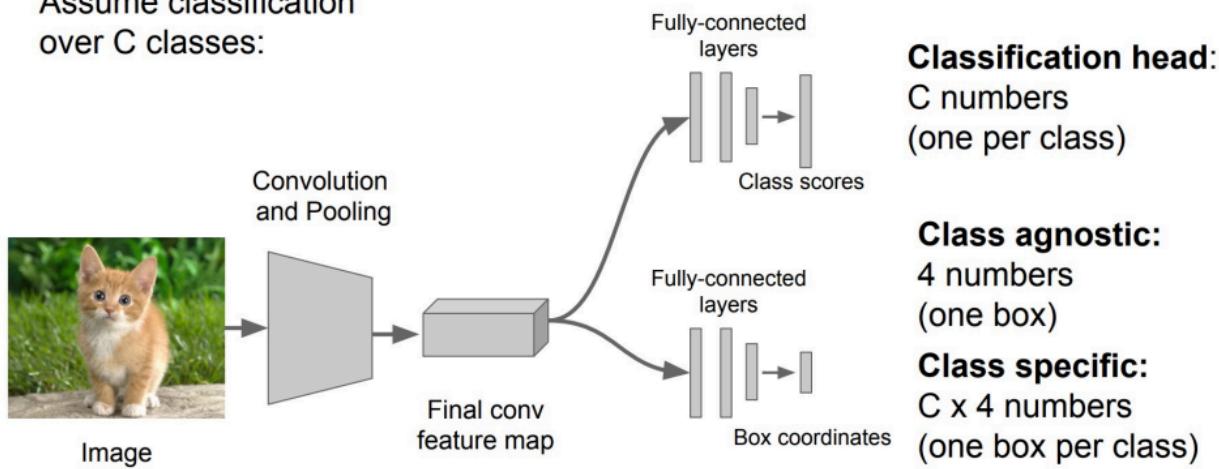
A simple recipe (a.k.a. Localization as regression):

- step 1: Train a classification model (e.g., AlexNet, VGG, GoogleNet)
- step 2: Attach a new fully-connected “regression head” to the network
- Step 3: Train the regression head only with SGD and L2 loss
- Step 4: At test time use both heads



Potremmo considerare alcune varianti:

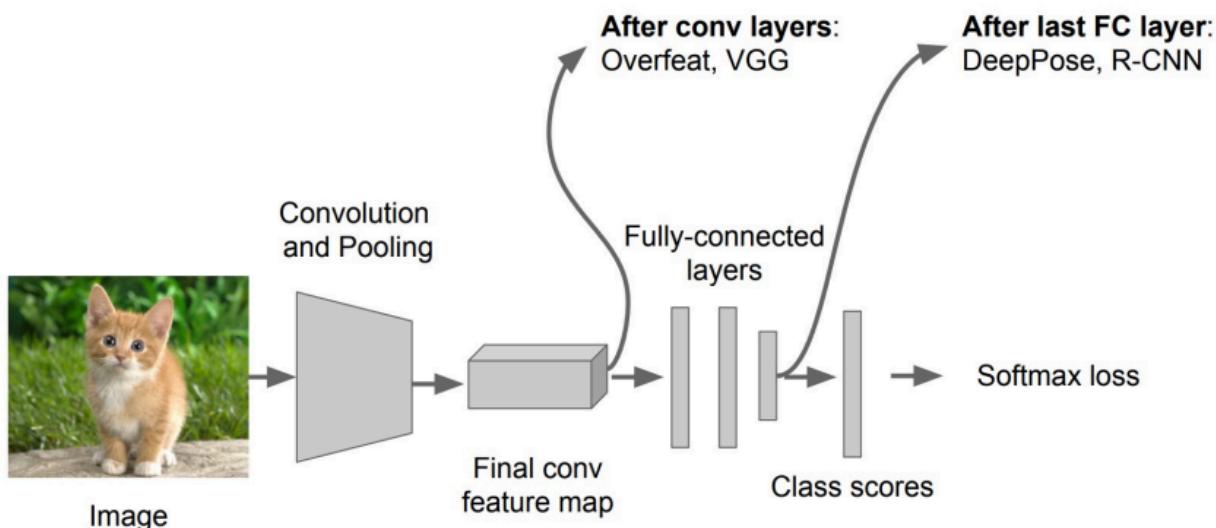
Assume classification over C classes:



Per quanto riguarda la localizzazione potremmo essere in **class agnostic**, ovvero abbiamo solo 4 numeri che rappresentano gli angoli del bounding box, oppure con **class specific** prediciamo bounding box per tutte e 4 le classi e poi prendiamo quella con la confidenza più alta.

Questo secondo caso può essere utile, per esempio se abbiamo una classe per 'bottiglie' e una per 'divani', che hanno forme ben diverse. In questo modo il modello impara che diversi oggetti hanno diversi 'aspect-ratios', e quindi rende il nostro modello migliore.

Un altro problema è che in questo modello attacchiamo i sue sub-modelli allo stesso punto, però magari potrebbero dare performance migliori, attaccandoli in punti diversi.



Un'altra idea potrebbe essere quella di una **sliding window**. Questo però richiede molti step di inferenza. Può quindi funzionare se il modello è piccolo.

- Run classification + regression network at multiple locations on a high-resolution image
- Convert fully-connected layers into convolutional layers for efficient computation
- Combine classifier and regressor predictions across all scales for final prediction

Object detection

Ora siamo nel caso in cui non sappiamo quanti oggetti ci saranno nell'immagine. Succede che per esempio in questo caso dovremmo ritornare 16 numeri (4 ciascuno).



→

DOG, (x, y, w, h)
CAT, (x, y, w, h)
CAT, (x, y, w, h)
DUCK (x, y, w, h)

= 16 numbers

Però in questa invece abbiamo 2 oggetti di interesse, e quindi dobbiamo avere 8 numeri in output. Quindi abbiamo bisogno qualcosa che genera un numero diverso di output per ciascuna immagine.



→

DOG, (x, y, w, h)
CAT, (x, y, w, h)

= 8 numbers

Potremmo avere un approccio di sliding window, per ogni posizione controlliamo le categorie.



Però qui c'è sempre lo stesso problema, bisogna controllare tante posizioni, tante grandezze... non è una soluzione fattibile.

- **Problem:** Need to test many positions and scales
- **Solution:** If your classifier is fast enough, just do it

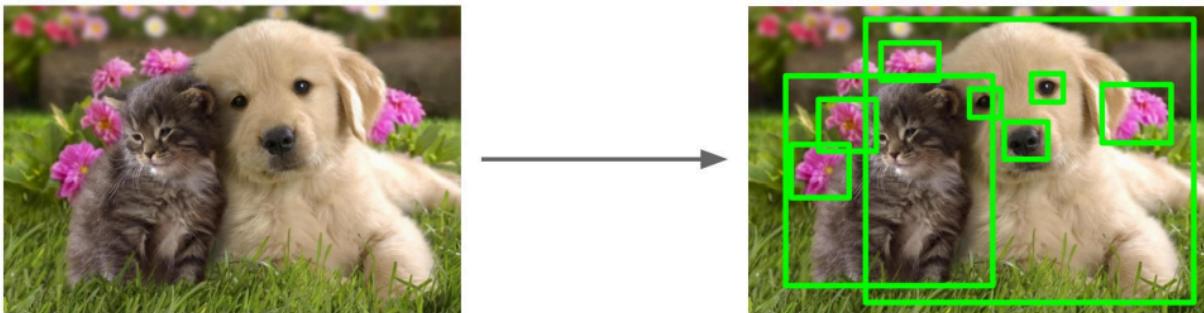
More likely:

- **Problem:** Need to test many positions and scales, and need to use a computationally demanding classifier (e.g., CNN)
- **Solution:** Only look at a tiny subset of possible positions

La soluzione ideale potrebbe essere quella di dire, non guardiamo tutte le posizioni dell'immagine, trova un modo di ridurla.

Qui si parla di approccio **region proposals**. Abbiamo l'immagine di input, troviamo delle regioni '**blobby**' che potrebbero contenere un oggetto. é come un detector class-agnostic.

Questo potrebbe essere un output, noi andremo ad analizzare solamente questo sub-set di proposte.



Region proposals (object proposals)

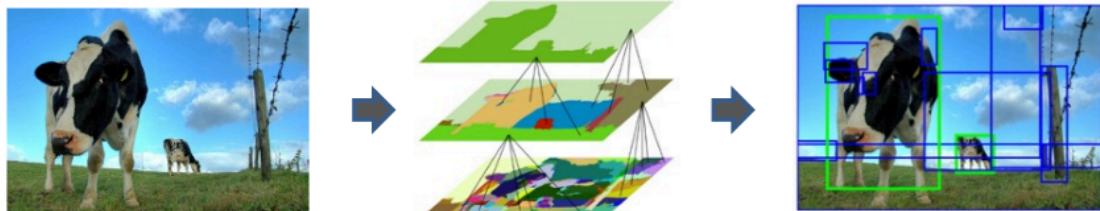
Non andremo troppo in dettaglio, vediamo l'idea.

L'idea è che partiamo da un'immagine, e partiamo con una divisione in regioni molto piccole. Queste regioni hanno la proprietà di essere molto omogenee per colore e texture.

Poi guardando ai vicini, combiniamo le regioni se sono sufficientemente simili.

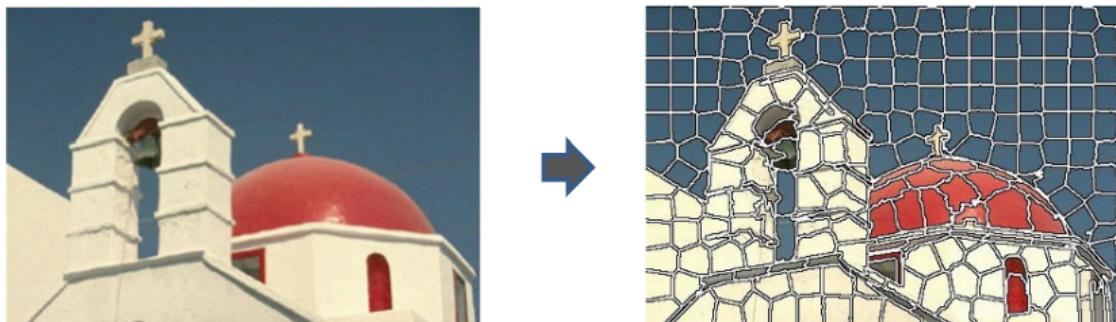
1. Initial image regions
2. Similarity computation
3. Greedy bottom-up grouping

NOT NEURAL



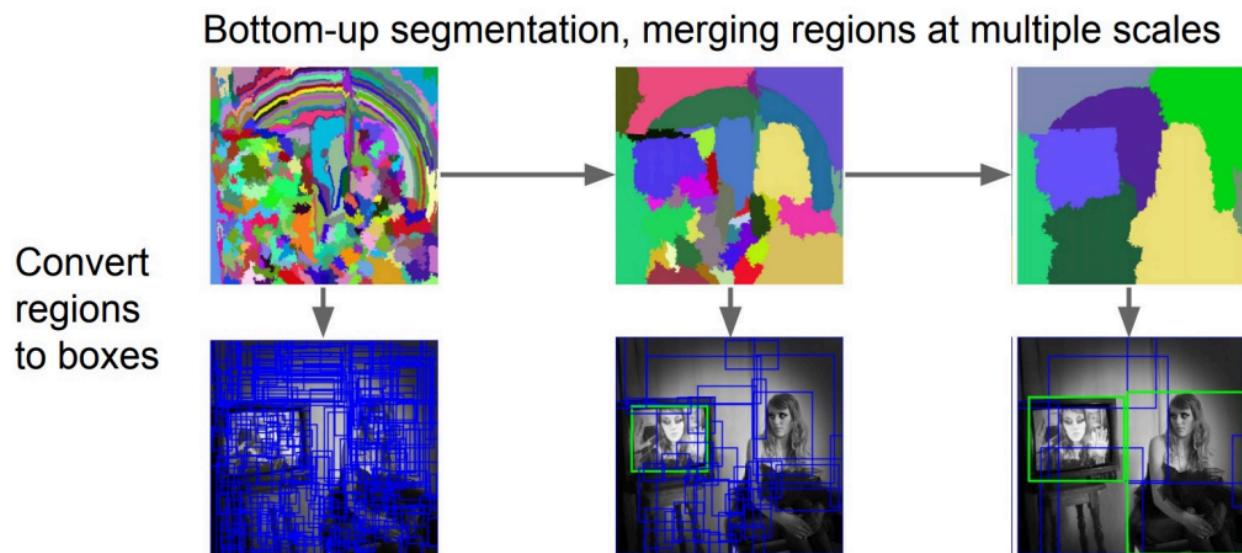
Questo tipo di approccio non è neurale.

Questo è uno degli approcci che è stato proposto, l'idea è di iniziare con una segmentazione in superpixels, che sono delle porzioni che condividono informazioni in comune.



Poi nel prossimo step, uniamo i superpixels che sono sufficientemente simili, ripetendo questo processo multiple volte.

Questa è una strategia bottom-up:



Abbiamo una scala molto piccola, una media e una grande. Ci sono altre parti di quest'algoritmo che danno uno score di confidenza per ogni regione, di modo da ridurre il numero totale di regioni che dobbiamo analizzare.

Region-based convolutional neural networks (R-CNN)

Ci sono anche varianti di R-CNN che puntano a rendere questo tipo di algoritmi sempre più veloci.

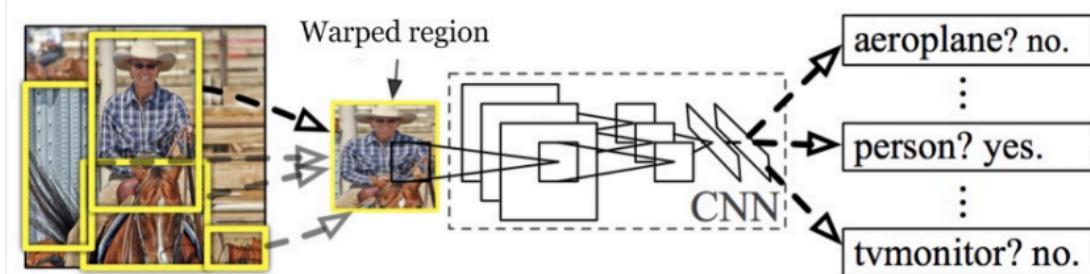
L'idea è quella di usare un approccio semi-neurale.

Quindi abbiamo un object proposal che usa l'algoritmo di selective search, e poi classifichiamo usando delle features neurali.



In particolare, le proposte diverse potrebbero avere forme diverse, ma in realtà abbiamo visto che le CNN prende input a dimensioni fisse. Quindi dobbiamo fare 'warp' di ciascuna regione.

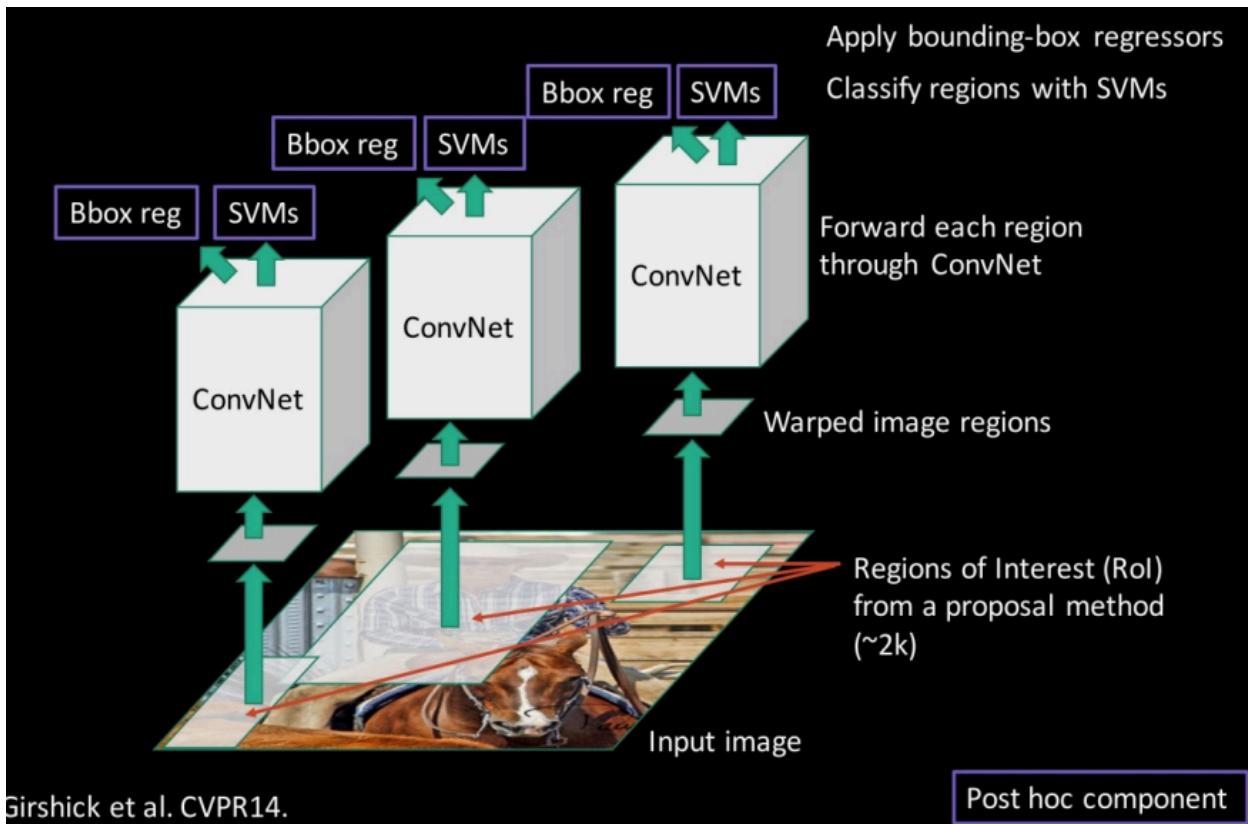
- Each candidate is warped to a common size
- Neural features are extracted from each candidate
 - Alexnet pretrained on IMAGENET (1000 classes)
 - Alexnet fine-tuned on the domain of interest
- Support Vector Machine (SVM) for final classification
 - Uses neural features as input
 - Binary classification for each possible class (+ unknown class)



è un approccio semi-neurale perché il region proposal iniziale non è neurale, e la features extraction che è neurale, e il classifier finale che non è neurale

C'è anche uno step di 'refinement'.

In generale abbiamo l'immagine di input, abbiamo dei proposals delle regioni possibili, che vengono warpage nella grandezza di input della CNN, per ciascuna abbiamo in output le features, con cui facciamo una modifica della posizione della proposal da una parte (per fissare meglio l'oggetto), e dall'altra usiamo una SVM per la classificazione.



Bottlenecks

La selective search è molto lenta, ci mette 10 secondi per immagine, non può essere usata per applicazioni real time.

Poi abbiamo anche che l'estrazione delle features è inefficiente, perchè abbiamo tanti candidati per immagine, che sono anche molto overlapped, e dobbiamo estrarre le features per ciascun candidato.

Problems

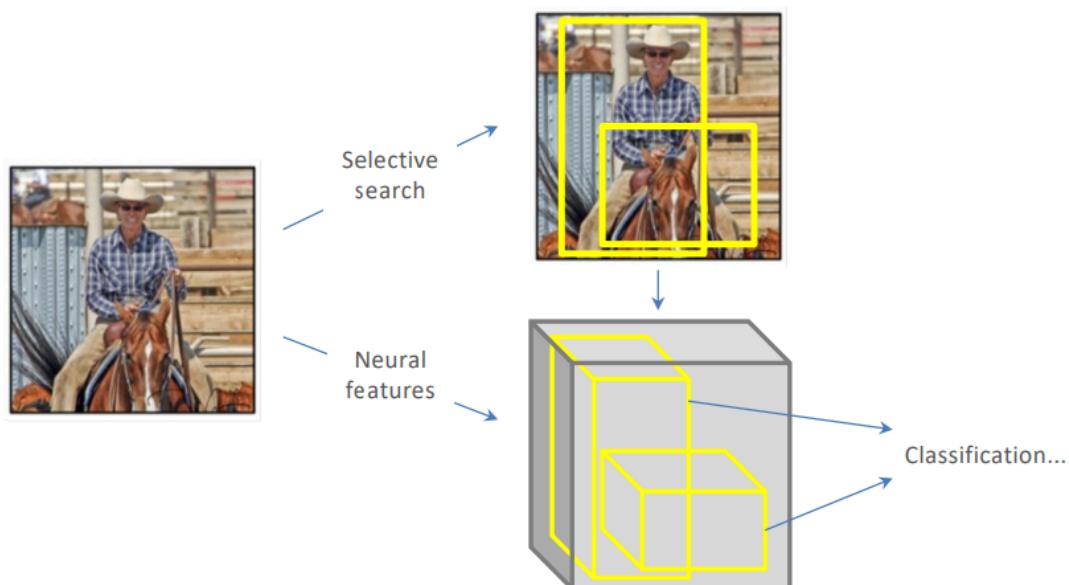
SVMs e i correctors per le bounding box sono chiamati **post hoc**, cioè sono separati, non aggiornano le features delle CNN, perchè non abbiamo dei gradienti, sono modelli separati che non si parlano.

1. SVMs and regressors are post-hoc: CNN features are not updated in response to SVMs and regressors
 2. Complex multistage training pipeline
-

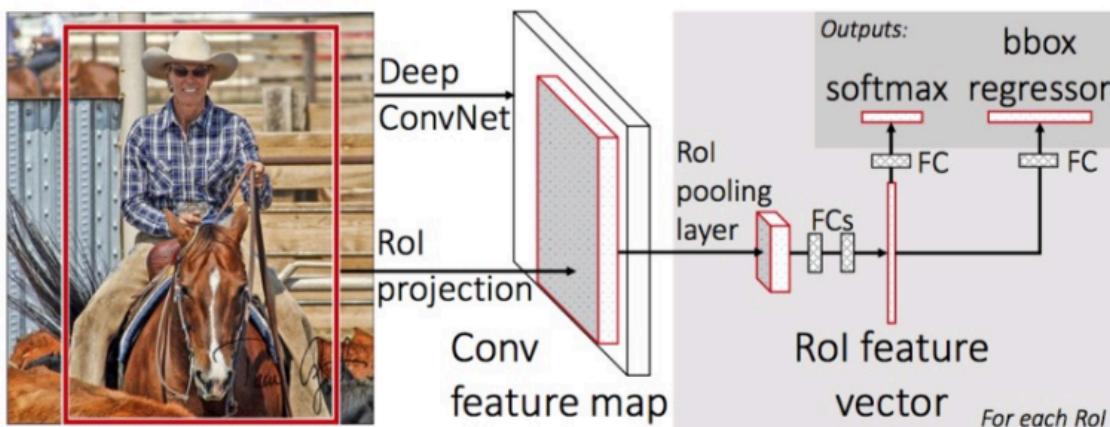
Fast R-CNN

L'idea è quella di calcolare le features dense solo una volta, e poi fare sampling in base alle proposte.

- Compute dense features once, and sample them based on proposals



Al posto di warpare i proposals, andiamo a samplare i volumi nelle posizioni corrette, quindi senza ricalcolare le features, ma recuperandole



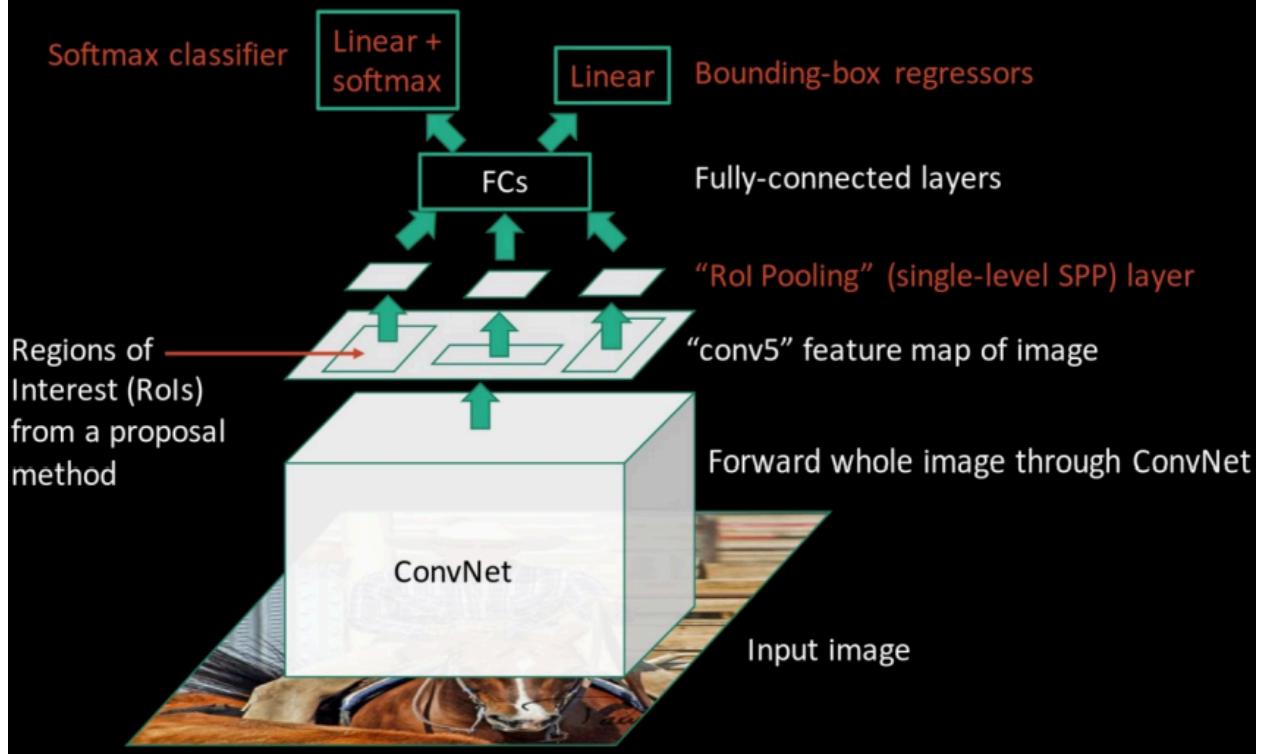
Il **ROI pooling layer** (region of interest) funziona che se abbiamo questa posizione di un candidate object, lo proiettiamo nel volume, e poi facciamo un reshape di questo volume ad una grandezza comune, di modo che possiamo collegare dei layers di NN che sono rigidi sulla grandezza dell'input.

Tutti i problemi di inefficienza per l'estrazione delle features sono in questo modo risolti.

Però abbiamo ancora il primo bottleneck, il selective search.

L'idea è che abbiamo l'immagine di input, su cui applichiamo la CNN direttamente, che ci da in output il volume di features. Poi in base ai proposals che abbiamo sull'immagine, andiamo a prendere samples dell'immagine, e poi andiamo a fare predizione con un fully connected layer le classi e le bounding boxes.

Fast R-CNN (test time)



In questo modo, è tutta neurale, quindi possiamo fare passare indietro i gradienti, e in questo modo la classification e la detection possono influenzare le features.

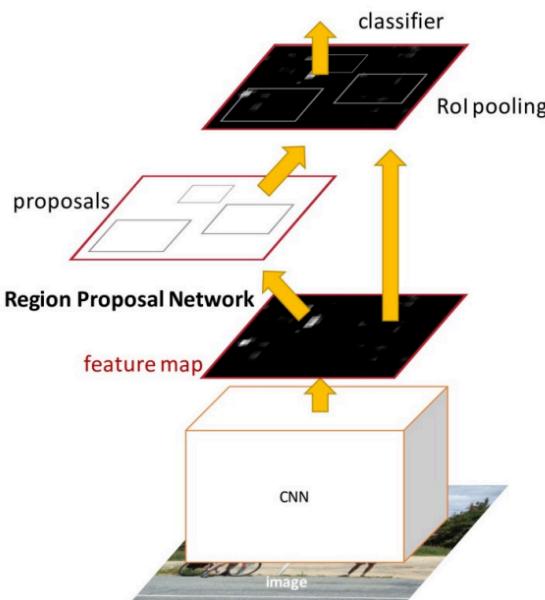
	R-CNN	Fast R-CNN
Faster!	Training Time: (Speedup)	84 hours 9.5 hours
	Test time per image (Speedup)	47 seconds 0.32 seconds
FASTER!	mAP (VOC 2007)	146x
	66.0	66.9

Le FAST R-CNN migliorano in tutti gli aspetti. Siamo ancora lontani da applicazioni real-time però.

Innanzitutto, possiamo far fare al NN anche le region proposal.

	R-CNN	Fast R-CNN
Test time per image	47 seconds	0.32 seconds
(Speedup)	1x	146x
Test time per image with Selective Search	50 seconds	2 seconds
(Speedup)	1x	25x

Test-time speeds don't include region proposals
 → Just make the CNN do region proposals too!



Insert a **Region Proposal Network (RPN)** after the last convolutional layer

RPN trained to produce region proposals directly; no need for external region proposals!

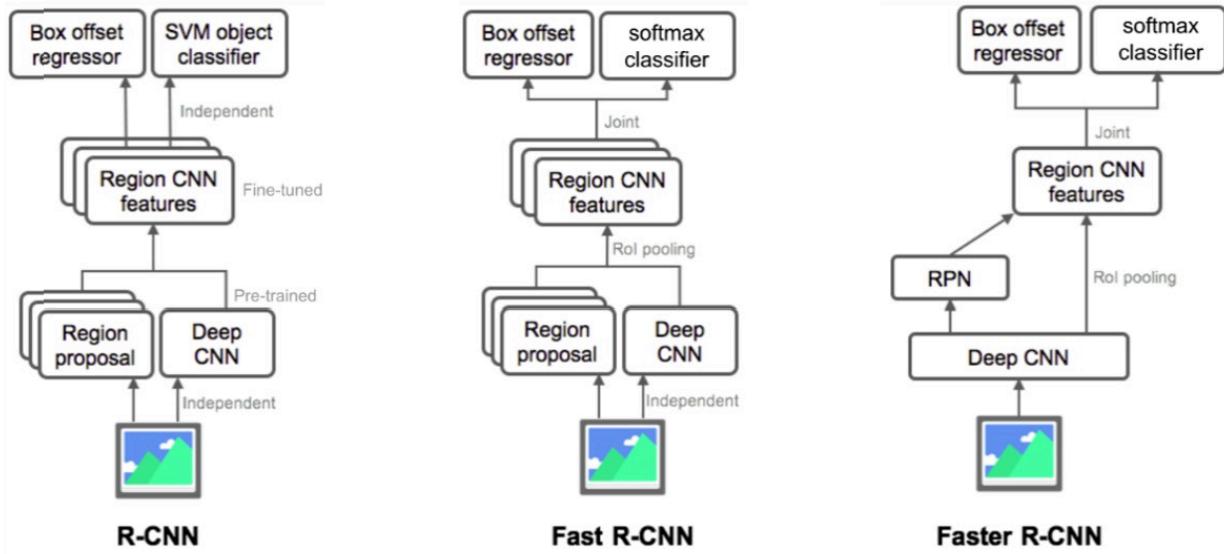
After RPN, use RoI Pooling and an upstream classifier and bbox regressor just like Fast R-CNN

L'idea è quella di avere la nostra input image, la CNN che estrae le features, e poi prendiamo una feature map da cui fare le region proposals, che poi sono usate dal region of interest pooling layer che va al classificatore. In questo caso quindi direttamente dalla features della CNN prendiamo i region proposals.

I gradienti possono backpropagare anche all'interno della CNN.

Quindi non ci affidiamo ad un object detector generico (selective search) ma alleniamo per fare proposals che hanno senso per il nostro dominio, la nostra applicazione.

Summary of models in the R-CNN family



Nel primo caso tutto è separato.

Nelle fast R-CNN abbiamo il region proposal che è ancora esterno e indipendente alla CNN, però le features delle regioni sono collegate direttamente al regressor e al classifier, di modo che il gradiente può backpropagare fino al CNN (passa per tutto tranne il region proposal, perchè è un modulo esterno, fisso).

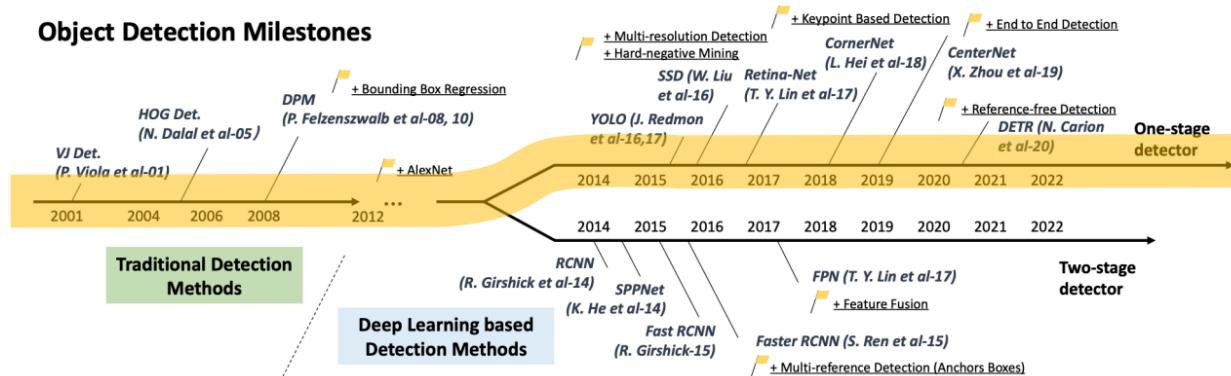
Nel terzo caso l'immagine va nel deep CNN, che fa delle proposte delle regioni, e poi estraiamo le features usando queste proposte. Questo poi è fully connected con il regressor e il classifier. Il vantaggio è che non dobbiamo usare un region proposal module esterno lento, ma invece i gradienti possono backpropagare in entrambe le direzioni (dx e sx), e quindi visto che passano per il RPN, le proposte che facciamo sono fatte ad hoc per il nostro dominio.

Quindi gli altri modelli avranno proposals per tante categorie diverse, mentre il terzo le avrà solo per quelle che ci interessano.

Con questo terzo metodo, a livello di accuracy siamo vicini, però miglioriamo molto il tempo totale richiesto.

	R-CNN	Fast R-CNN	Faster R-CNN
Test time per image (with proposals)	50 seconds	2 seconds	0.2 seconds
(Speedup)	1x	25x	250x
mAP (VOC 2007)	66.0	66.9	66.9

Quindi, questi sono gli two-stage detectors. Dopo il 2018 non abbiamo delle proposte interessanti, mentre invece ce li abbiamo per gli one-stage. Andiamo a vedere questi.



Two-stage vs one-stage

I metodi visti fin ora sono chiamati **two-shot object detection** perché prima dobbiamo trovare le regioni, e poi le dobbiamo classificare.

I **single-shot object detection** invece in un singolo passo fanno già le predizioni delle posizioni e delle classi. Però sono meno accurati. In particolari, funzionano meglio nel trovare oggetti piccoli.

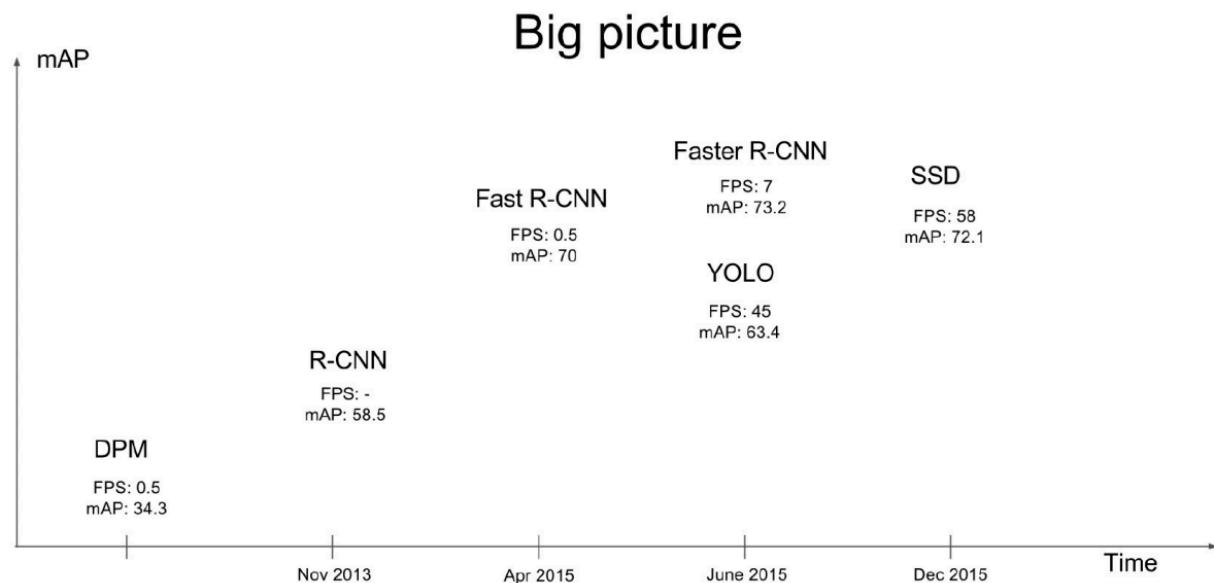
Quindi questi algoritmi possono funzionare bene per applicazioni real time, in ambienti con risorse limitate (come le self driving cars, ci interessano gli oggetti grandi, vicini).

Invece, per esempio, se abbiamo delle immagini x-ray da cui vogliamo trovare il cancro, che può essere molto piccolo, allora siamo interessati anche a trovare delle zone molto piccole. Per questo, i single-shot non sono ideali.

YOLO (You Only Look Once)

In una sola passata di inferenza, facciamo l'object detection.

La performance di YOLO, rispetto a quello che abbiamo visto, sull'asse y abbiamo la mean average precision. C'è anche scritta la velocità del metodo in termini di frames per second.



Possiamo vedere che per real time application, YOLO funziona bene, mentre le R-CNN sono troppo lente.

Nell'immagine c'è la prima versione di YOLO, ora c'è la versione 11. Per ogni versione abbiamo versioni più o meno grandi, in base alle risorse che abbiamo disponibili.

Main concepts

- Object detection: il problema è castato in un problema di regressione (come le R-CNN)
- è basato su un feedforward processing unico

- C'erano due versioni: una che va a 45FPS e una più veloce (fast YOLO) che andava a 155FPS.
- Il modello riesce ad imparare una rappresentazione generale che è robusta su molti domini

Single regression problem, partendo da un'immagine abbiamo sia coordinate delle bounding box che le probabilità delle classi.

- YOLO: single regression problem
- Image → bounding box coordinates and class probability
- Extremely fast
- Global reasoning
- Generalizable representation

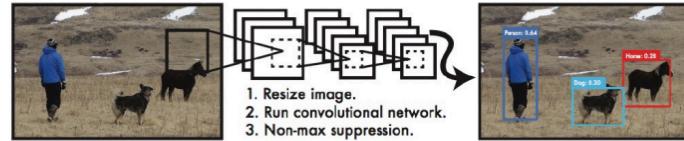


Figure 1: The YOLO Detection System. Processing images with YOLO is simple and straightforward. Our system (1) resizes the input image to 448×448 , (2) runs a single convolutional network on the image, and (3) thresholds the resulting detections by the model's confidence.

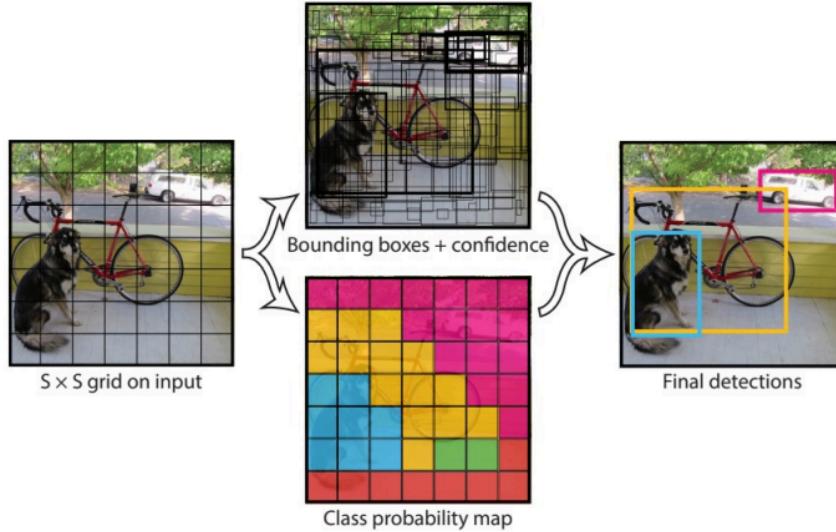
Come funziona

Innanzitutto divide l'immagine in un grid regolare $s * s$.

Per ciascuna cella, ritorna il displacement del bounding box e il confidence score. Ritorna 'B' bounding boxes, non 1. Per ciascuna grid cell, predico per esempio 2 bounding boxes.

Per ciascuna, diamo in output anche una cluster probability map.

Poi dopo aver combinato le due informazioni, e facendo thresholding, abbiamo l'output.



Se siamo sul dataset VOC (20 classe) abbiamo:

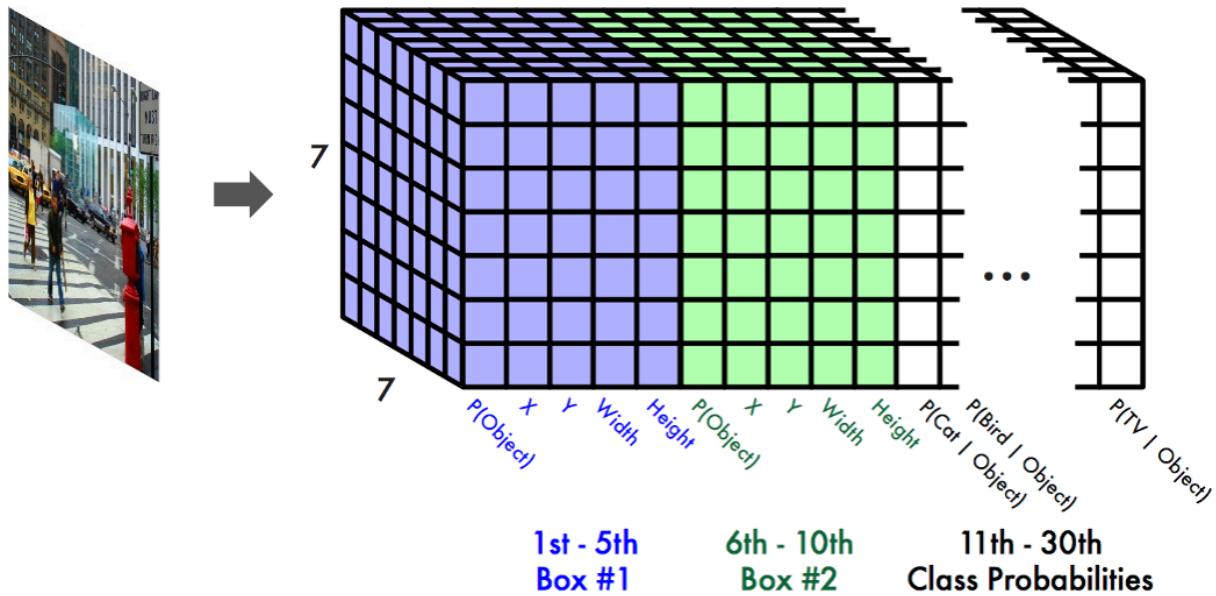
- On Pascal VOC (20 classes):

- $S=7$
- $B=2$
- $C=20$
- prediction is a $7 \times 7 \times 30$ tensor

Figure 2: The Model. Our system models detection as a regression problem. It divides the image into an $S \times S$ grid and for each grid cell predicts B bounding boxes, confidence for those boxes, and C class probabilities. These predictions are encoded as an $S \times S \times (B * 5 + C)$ tensor.

Quindi in totale abbiamo un tensore $7 \times 7 \times 30$ (7×7 posizioni spaziali, e per ciascuno facciamo un encoding di $5 * B + C$ (C è la class probability))

Questo è come il nostro volume di output è organizzato: è un 7×7 , poi nel primo blocco abbiamo la probabilità che contenga un'oggetto, la posizione e la grandezza del bounding box. Poi abbiamo la probabilità del secondo bounding box, centro del bounding box, altezza e grandezza. E poi abbiamo tutte le probabilità che sia un oggetto di una classe, per tutte le classi.



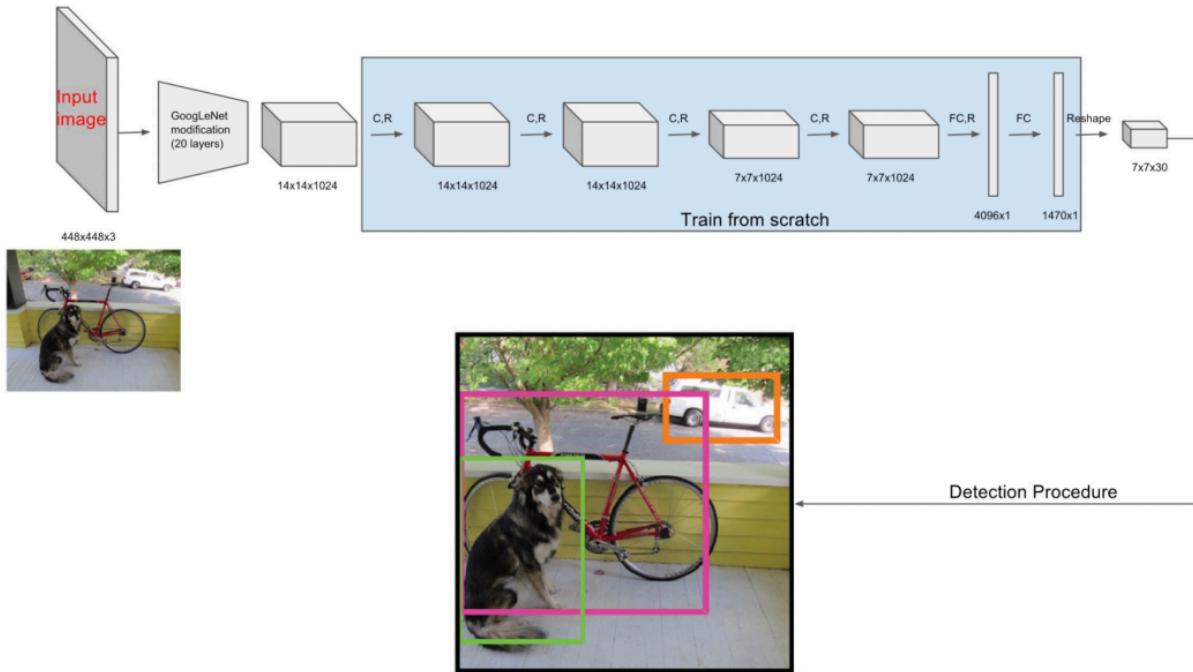
Abbiamo detto che YOLO divide l'immagine in un grid sxs.

Se il centro di un oggetto è all'interno della grid cell, allora la grid cell è responsabile di trovare l'oggetto.

Ogni grid cell predice B bounding boxes e i punteggi di confidence (quanto il modello pensa che quel bounding box contenga un oggetto, e quanto pensa che la classificazione sia corretta).

Che tipo di NN è usato?

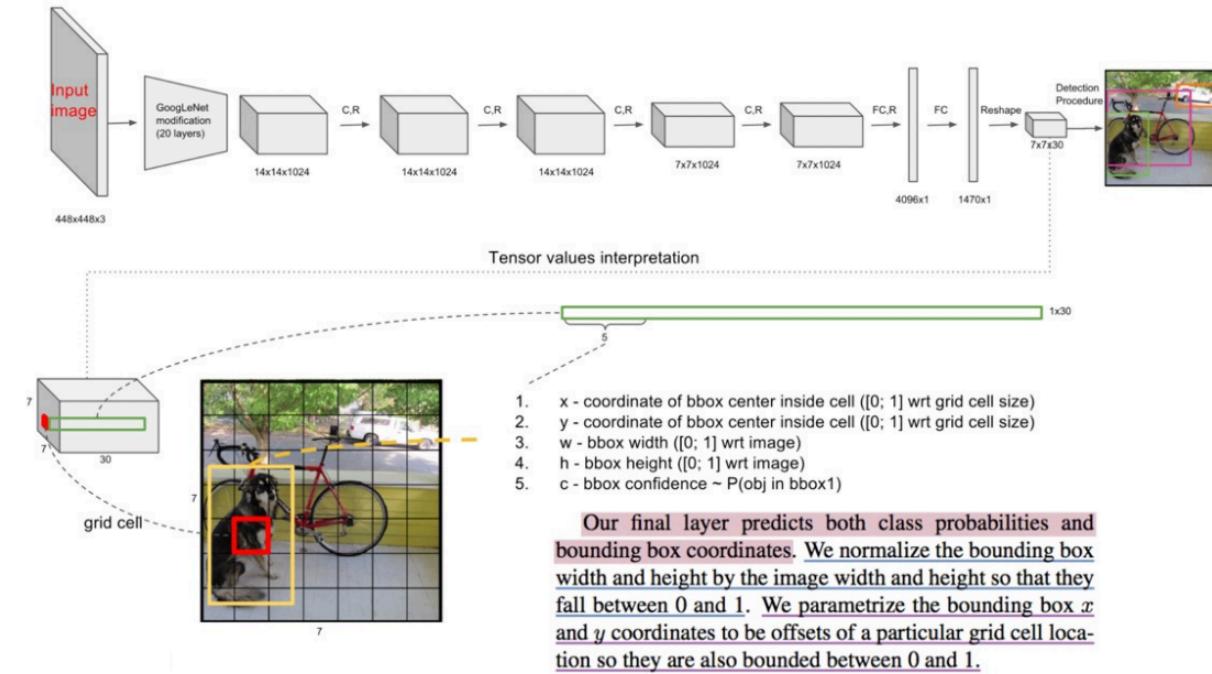
è una versione modificata di GoogLeNet. (nella prima versione di yolo)



Dall'input, si taglia la rete di googlenet al punto di 14×14 , e hanno aggiunto altri 4 layer convoluzionali, dove la depth è mantenuta la stessa, (in ciascuno step c'è convoluzione e non linearità), da un 14×14 arriviamo ad un 7×7 .

Poi andiamo in dei layer fully connected. Infine l'informazione è ridimensionata in $7 \times 7 \times 30$ da cui si trovano i risultati.

Quell'output $7 \times 7 \times 30$ è letto in questo modo:



Alla fine teniamo solo la predizione con la confidenza più alta.

Real-Time Detectors	Train	mAP	FPS
100Hz DPM [31]	2007	16.0	100
30Hz DPM [31]	2007	26.1	30
Fast YOLO	2007+2012	52.7	155
YOLO	2007+2012	63.4	45
<hr/>			
Less Than Real-Time			
Fastest DPM [38]	2007	30.4	15
R-CNN Minus R [20]	2007	53.5	6
Fast R-CNN [14]	2007+2012	70.0	0.5
Faster R-CNN VGG-16[28]	2007+2012	73.2	7
Faster R-CNN ZF [28]	2007+2012	62.1	18
YOLO VGG-16	2007+2012	66.4	21

Nella parte sopra, i DPM hanno tanti fps, ma hanno una performance molto bassa.

YOLO invece ha una performance che era una via di mezzo, rispetto ai metodi lenti, ma con una velocità molto più alta.

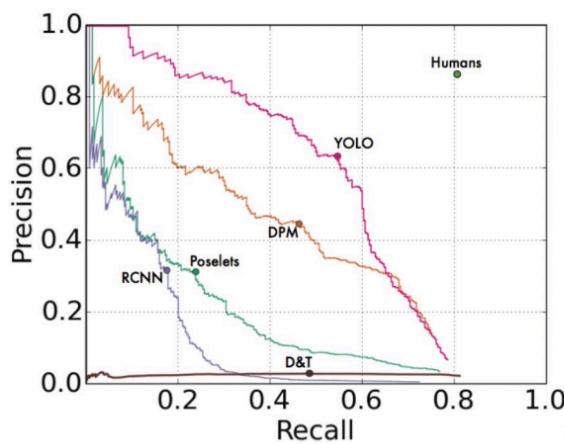
YOLO generalizability

YOLO ha un'alta generalizzabilità.

Hanno provato a trainare dei detectors per trovare delle persone su immagini fatte al telefono, e poi hanno trainato dei detectors su dei dipinti.

Possiamo vedere che la performance delle RCNN sono scarse, molto lontane da quello che possono fare gli esseri umani. Con YOLO invece siamo sopra alla diagonale, e siamo più vicini alla capacità umana, anche facendo training su un dominio e facendo inferenza su un'altro.

Person detection in artwork



(a) Picasso Dataset precision-recall curves.

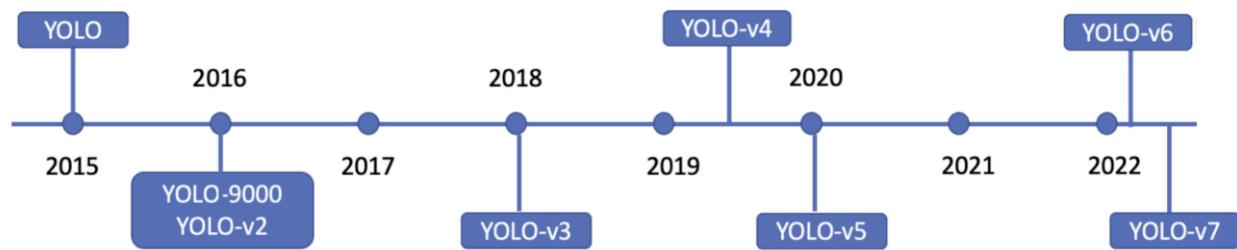
	VOC 2007 AP	Picasso		People-Art AP
		AP	Best F_1	
YOLO	59.2	53.3	0.590	45
R-CNN	54.2	10.4	0.226	26
DPM	43.2	37.8	0.458	32
Poselets [2]	36.5	17.8	0.271	
D&T [4]	-	1.9	0.051	

(b) Quantitative results on the VOC 2007, Picasso, and People-Art Datasets.
The Picasso Dataset evaluates on both AP and best F_1 score.

Figure 5: Generalization results on Picasso and People-Art datasets.



YOLO timeline



Questa è una timeline incompleta. Ci sono alcune varianti ufficiali e altre non ufficiali.

Lo standard è YOLO-v5, oggi si usano v7-v8. Ci sono librerie che permettano di trainare diverse varianti con diverse grandezze.

YOLO-v5 utilizza delle bounding boxes di dimensioni preseggiate.