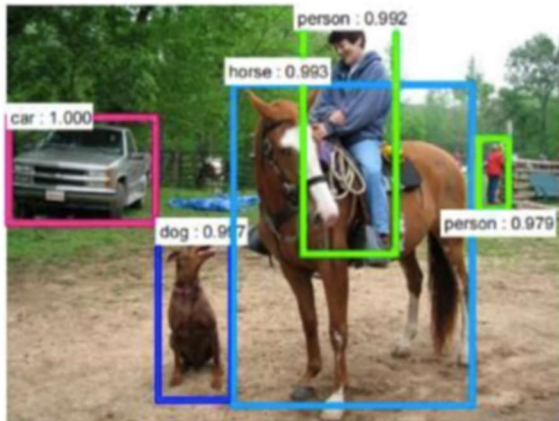


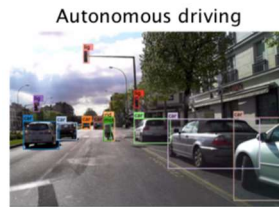
11 - BEYOND IMAGE CLASSIFICATION: OBJECT DETECTION

From classification to detection

Capire cosa c'è in una certa immagine con una certa percentuale di accuratezza



Detection: localizzare oggetti, in una stessa immagine possiamo avere più oggetti dello stesso tipo e più oggetti diversi.



Pedestrian detection



Image captioning



Industry 4.0 : esempio -> detection dei difetti

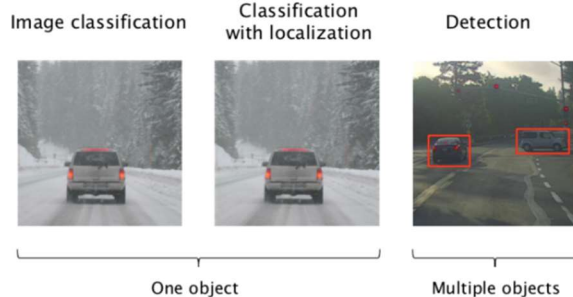


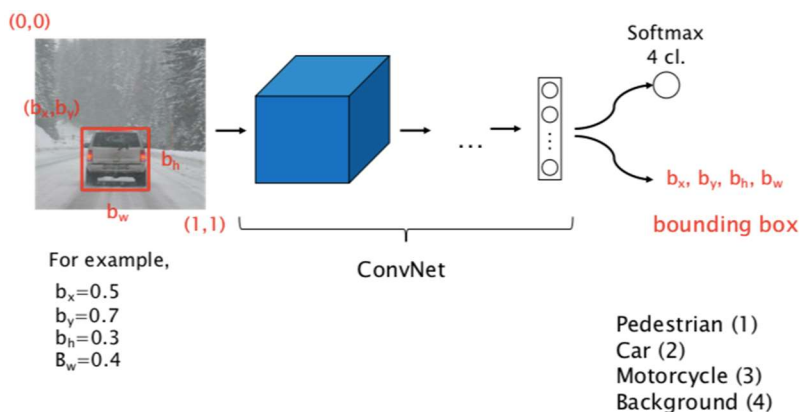
Image classification: un unico oggetto

Classificazione con localizzazione: oltre a sapere se c'è o meno la macchina, devo anche identificare dove questa si trovi nell'immagine

Detection: possiamo avere più oggetti dello stesso tipo all'interno della scena

Classification with localization

Rispetto alla classificazione che si ottiene come output della softmax, mi interessa trovare il bounding box. Si può immaginare di portare in output anche le informazioni della **posizione e grandezza del bounding box**.



Adatterei l'allenamento supervisionato in modo tale da apprendere non solo le categorie ma anche le bounding box.

Il vettore di uscita sarà adattato nella seguente maniera

Nota: p_c è una probabilità discretizzata (0 o 1)

$$y = \begin{bmatrix} p_c \\ b_x \\ b_y \\ b_h \\ b_w \\ c_1 \\ c_2 \\ c_3 \end{bmatrix}$$

Probability object (any class of interest) present

Bounding box coordinates

1 if object of class 1 is present, 0 otherwise
 1 if object of class 2 is present, 0 otherwise
 1 if object of class 3 is present, 0 otherwise

Esempio:

$$y = \begin{bmatrix} p_c \\ b_x \\ b_y \\ b_h \\ b_w \\ c_1 \\ c_2 \\ c_3 \end{bmatrix}$$

$$y = \begin{bmatrix} 1 \\ b_x \\ b_y \\ b_h \\ b_w \\ 0 \\ 1 \\ 0 \end{bmatrix} \text{ Present}$$

$$y = \begin{bmatrix} 0 \\ ? \\ ? \\ ? \\ ? \\ ? \\ ? \\ ? \end{bmatrix} \text{ Not pres.}$$

"don't care"

inserisco don't care in quanto non mi interessa

Pedestrian (1)
Car (2)
Motorcycle (3)
Background (4)



Ora senza pensare al tipo di uscite, categoriche e continue, enumero tutte le uscite allo stesso modo y_1, \dots, y_8

$$y = \begin{bmatrix} p_c \\ b_x \\ b_y \\ b_h \\ b_w \\ c_1 \\ c_2 \\ c_3 \end{bmatrix} \begin{matrix} y_1 \\ y_2 \\ y_3 \\ y_4 \\ y_5 \\ y_6 \\ y_7 \\ y_8 \end{matrix}$$

$$y = \begin{bmatrix} 1 \\ b_x \\ b_y \\ b_h \\ b_w \\ 0 \\ 1 \\ 0 \end{bmatrix}$$

$$y = \begin{bmatrix} 0 \\ ? \\ ? \\ ? \\ ? \\ ? \\ ? \\ ? \end{bmatrix}$$

"don't care"

$$\text{Loss}(\hat{y}, y) = \begin{cases} (\hat{y}_1 - y_1)^2 + (\hat{y}_2 - y_2)^2 + \dots + (\hat{y}_8 - y_8)^2 & \text{if } y = 1 \\ (\hat{y}_1 - y_1)^2 & \text{if } y = 0 \end{cases}$$

Posso scrivere la **loss function** che dice: per il singolo example se l'oggetto è presente -> i quadrati delle differenze, se l'oggetto non c'è mi interessa solo avere il primo termine.

Questo fa intuire la natura binaria del primo termine e la natura continua degli altri.

Però questa Loss Function è troppo semplice perché non tiene conto della diversa natura dei dati di uscita;

posso **distinguere i tre tipi di uscite** trattandoli come problemi diversi:

- y_1 è una logistic regression (0,1) -> probabilità discretizzata che ci sia
- $y_2 \dots y_5$ è una linear regression -> posizione
- $y_6 \dots y_8$ è un softmax

Posso aggregare la prima e l'ultima classe in un'unica softmax con 4 classi.

Landmark detection



b_x, b_y, b_h, b_w



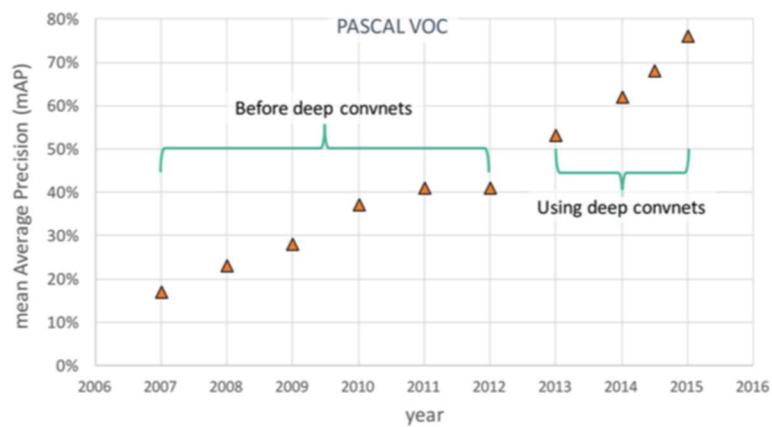
l_{1x}, l_{1y}
 l_{2x}, l_{2y}
 l_{3x}, l_{3y}
 l_{4x}, l_{4y}
...
 l_{64x}, l_{64y}



l_{1x}, l_{1y}
 l_{2x}, l_{2y}
 l_{3x}, l_{3y}
...

Landmark: punti salienti che possono essere usati per face-recognition, action-recognition, etc

Object detection








usando le reti profonde si è aumentata la precisione

Sliding window detection

Nel training set abbiamo dei crop molto piccoli. Abbiamo una convnet a cui passando i crop mi dice car e not car.

Per il momento ignoro i bounding box perché non sapendo quante volte può esserci una car aggiungere tale informazione può essere un problema.

Training set:

x	y
	1
	1
	1
	0
	0



→ ConvNet → y

-> do in pasto alla rete la finestra piccola fino a quando non mi dirà che c'è l'immagine: mi dirà 50% quando avrò metà auto, continuo a spostare e mi darà 100%. Dunque, facendo una sliding window riesco a farlo. Molto importante la **dimensione della finestra** e il **passo**.

Parto prendendo una porzione dell'immagine da dare in pasto e continuo spostando questa finestra, scegliendo un passo, fino a dare in pasto tutta l'immagine al classificatore convnet.



- Non sto sfruttando la similarità tra il crop è quello successivo

Window 

Bisogna anche decidere la dimensione della window.



Window 



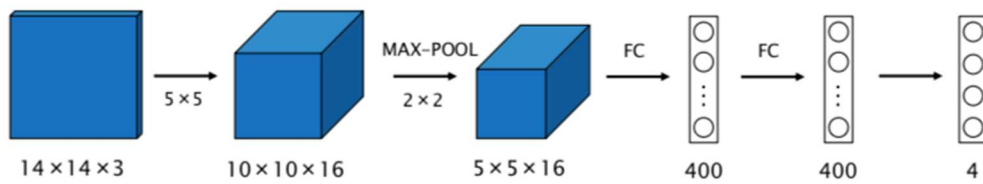
La dimensione della finestra e il passo si possono scegliere a mano oppure brute force aumentando il carico computazionale.

È un approccio forza bruta e costa tantissimo -> cerchiamo altri approcci

Trasformare una rete FullyConnected in un layer Convolutivo

L'idea è di implementare l'approccio sliding window in **maniera convolutiva**.

Prendiamo una rete che ad un certo punto ha dei livelli fully connected e infine viene fatto l'unrolling ottenendo il softmax.

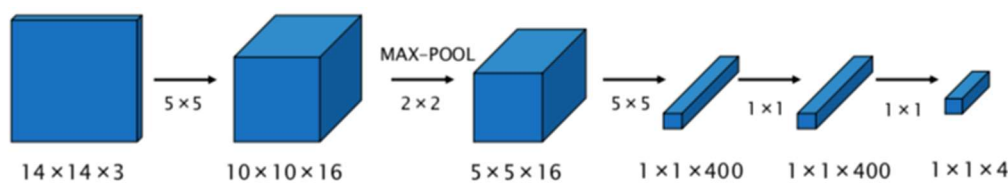


Proviamo sostituire i livelli fully connected con dei livelli convoluzionali:

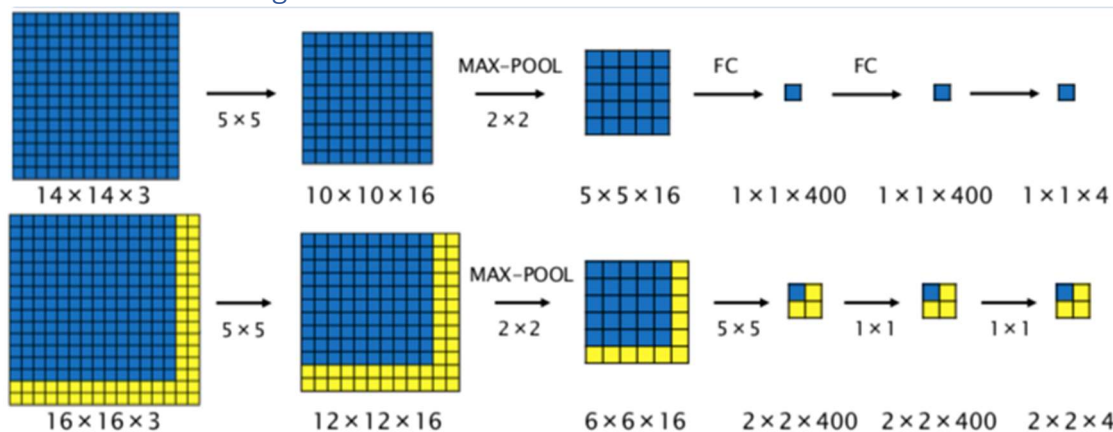
- 400 filtri 5×5 -> mi permettono di passare dal $5 \times 5 \times 16$ a $1 \times 1 \times 400$
- Convoluzione $1 \times 1 \times 400$
- Convoluzione $1 \times 1 \times 4$ filtri

Non c'è la stessa informazione all'interno in quanto ci sono operazioni differenti

- Sopra: Fully connected
- Sotto: copertura dei filtri



Convolutional sliding window



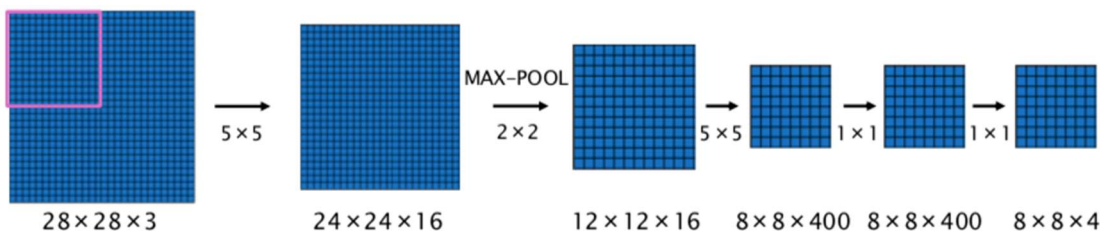
Immaginando
l'immagine fosse
 16×16

Nel caso 16×16 si può pensare di passare tutta l'immagine alla rete sfruttando la capacità convolutiva della rete perché applicando la sliding window, ad esempio 14×14 , si avrebbe ridondanza inutile di computazione.

In questo caso l'output che non è più 1×1 ma 2×2 ci dice che nel primo parallelepipedo blu ci sono le informazioni della parte blu dell'immagine 16×16 . Il blocco in basso a dx corrisponde alla $13 \times 14 \times 3$ partendo da in basso a dx.

In pratica si fanno tutte le predizioni in un colpo solo evitando la sequenzialità.

Questo è scalabile ad un'immagine di dimensioni più grandi come ad esempio 28×28 .



Ci sono 3 problemi:

- Per immagini molto grandi ottengo degli output molto grandi e anche molta complessità computazionale
- La sliding window ha un fattore di forma limitato che potrebbe presentare problemi con oggetti ad esempio allungati etc -> aspect ratio
- Accuratezza dovuta allo stride

Region proposal: R-CNN

Può succedere che il nostro target non abbia dimensioni compatibili con la finestra o che non lo centro.

- Difficile che l'intera persona sia in un quadrato precisamente

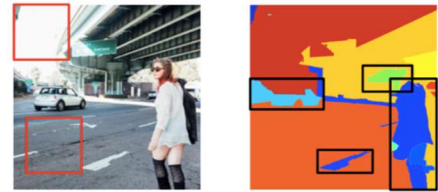


L'algoritmo propone delle regioni in cui andare a cercare l'oggetto.

TWO STAGE:

- Dove andare a cercare: genero delle regioni proposte
- Poi vado a cercare: classifico ogni regione proposta

Two-stage methods, decompose the problem in two stages:

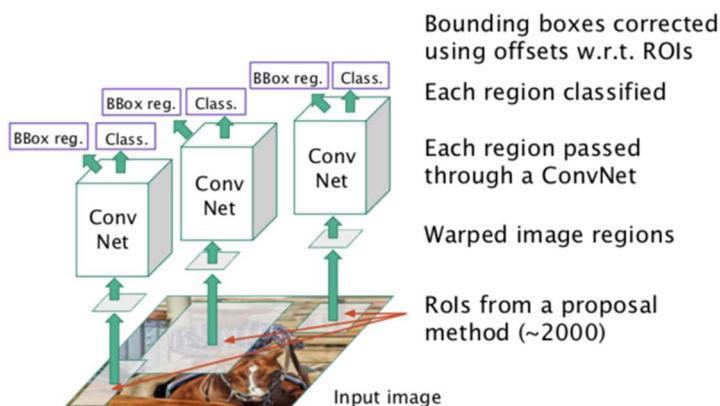


First, generate sensible region proposals (Regions of Interest, ROIs)
Then, classify each proposed region

L'algoritmo prende il nome di R-CNN:

- Sull'immagine di input viene fatto girare un algoritmo di **selective search** spesso non convolutiva che tira fuori delle ROIs (Region Of Interest) -> regione dove è probabile trovare l'oggetto
- Sulle ROIs viene fatto un **warping**, ovvero vengono adattate le dimensioni per andare in pasto alla convnet
- Le ROIs warpate vengono passate alla convnet che fanno la classificazione ottenendo se ci sono gli oggetti cercati e il bounding box in relazione alla ROI. (si può fare in parallelo)

- ♦ Propose regions
- ♦ Classify proposed regions one at a time
- ♦ Output label + bounding box
- ♦ Training is slow (84h), and takes a lot of disk space
- ♦ Making predictions (detections) is slow too (47s/image with VGG16)



- Rete convolutiva con 2 teste (classificazione + parte regressiva per bounding boc)

Il problema è che l'addestramento è molto lento (84h) e richiede molta archiviazione.

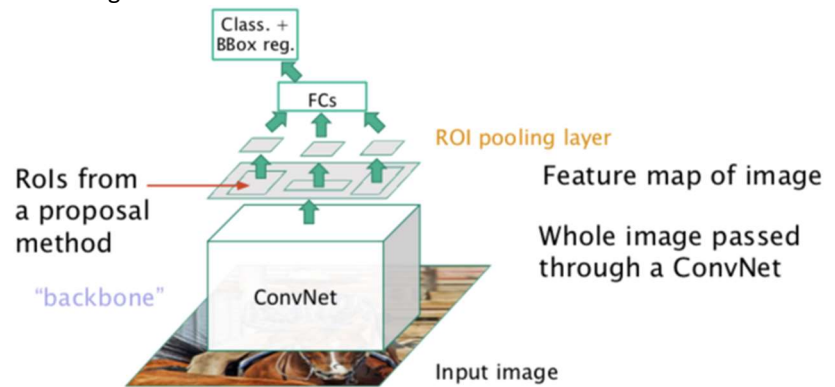
È lento anche nel fare le predizioni (47 s/image con VGG 16). Influisce anche la scelta della rete convolutiva da utilizzare con questo meccanismo.

Region proposal: Fast R-CNN

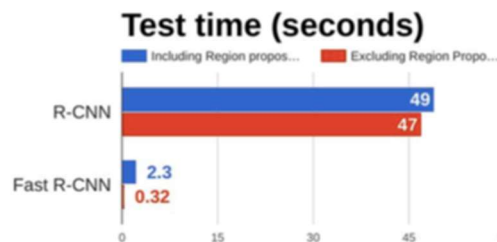
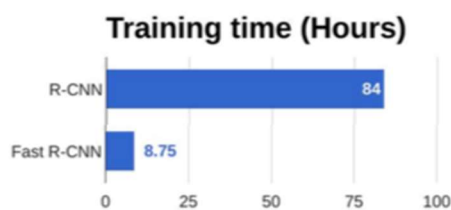
Nel 2015 è stata proposta una versione evoluta chiamata Fast R-CNN che usa lo stesso principio ma utilizza un'implementazione **convolutiva** delle sliding windows per classificare tutte le regioni in una sola volta.

Per prima cosa fa la convoluzione che:

- Traduce il dominio dell'immagine nello spazio delle feature
- Cerca delle ROI sulla feature map dell'immagine
- Prende queste regioni (della feature map) e le ridimensiona (pooling)
- Esegue classificazione su queste regioni con dei livelli fully-connected



Confrontiamo i tempi delle due reti:

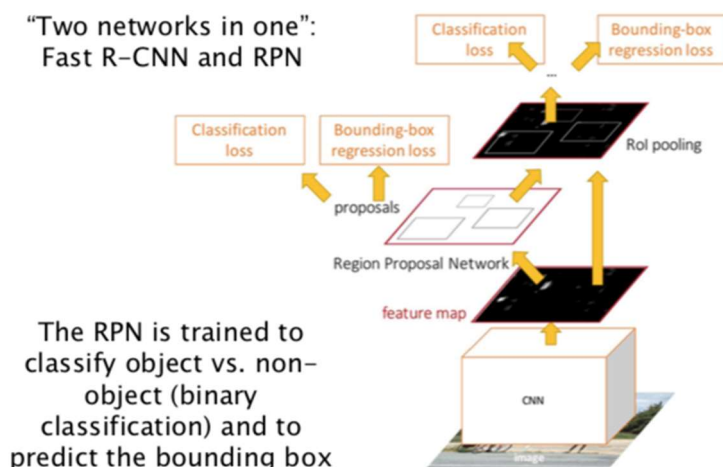


Nella Fast R-CNN il collo di bottiglia è dato dalla generazione delle ROI (2,3s).

Per andare ancora più veloci Faster R-CNN -> trasformare in convolutivo anche il region-proposal.

Region proposal: Faster R-CNN

"Two networks in one":
Fast R-CNN and RPN



- Rete convolutiva trasforma immagine in dominio delle feature -> feature map
- Rete RPN (Region Proposal Network) che tira fuori dalla feature map, delle ROI dello spazio delle feature
- Viene fatto il pooling sulle ROI -> trasformate in regioni quadrate
- Si fa la classificazione

Continuiamo ad avere una CNN per estrarre la feature map dell'immagine; su questa non uso un algoritmo tipo selective search ma vado ad utilizzare un'altra rete per tirare fuori le ROIs. Quindi la feature map viene passato ad un pooling di ROI che le porta tutte alla stessa dimensione e a questo punto si vanno a tirare fuori le informazioni volute.

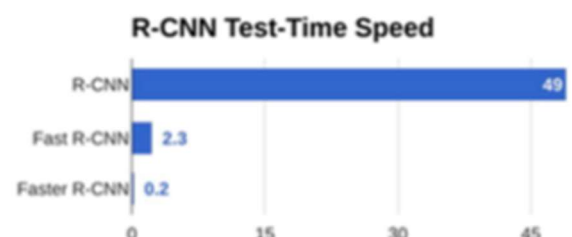
In questo caso è come se si avesse un algoritmo con **due reti l'una nell'altra**: una Fast R-CNN e una RPN (Region Proposal Network).

L'idea è quella di avere loss sia per la parte di classificazione e bounding box e sia per la proposal delle ROIs.

Vedendo i tempi di tutte e tre le reti:

Tempo di inferenza si è abbassato da 49 a 0.2s.

- Convoluzioni più possibile per risparmiare calcoli
- Convoluzioni il prima possibile perché andando nello spazio delle feature si riduce la rappresentazione
- Andiamo a prendere in partid iverse della rete quello che ci serve e trattiamo in maniera convolutiva

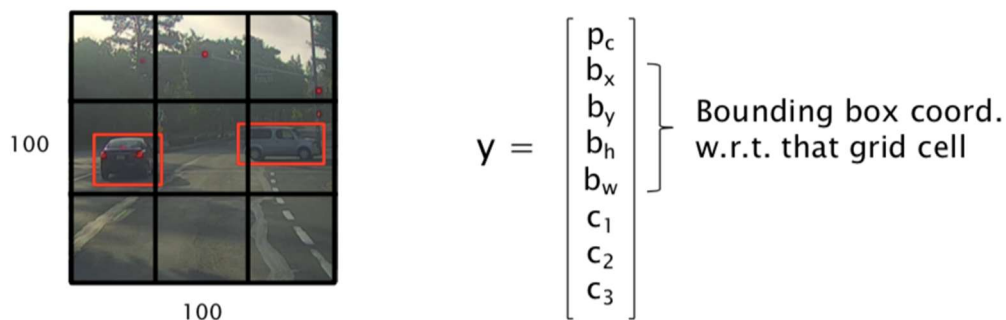


Successivamente si è tentato di fare tutto in un colpo solo -> Single Shot

YOLO

BB predicitons (Basis of YOLO)

Prima implementazione si basa sulla SSD (Single-Shot Detection) che non usa la region proposal. Si cerca di fare localizzazione e classificazione contemporaneamente sfruttando la divisione in griglia per diminuire la complessità. (Divide and Impera)

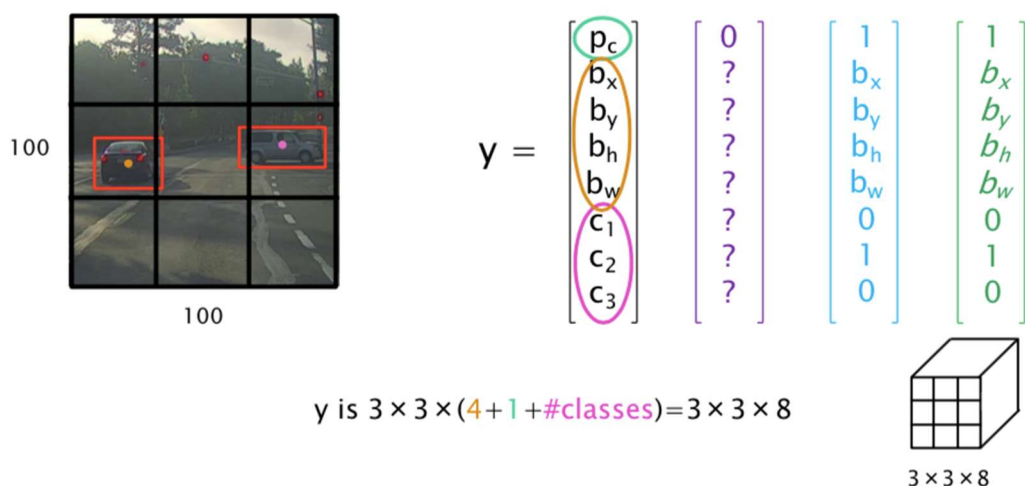


Si crea **per ogni unità della griglia un vettore y**.

Per descrivere l'appartenenza di un oggetto si considerano i midpoint, ovvero il centro di dove immaginiamo verrà posizionato il bounding box e in base alla cella a cui appartiene questo oggetto usiamo il midpoint.

La dimensione di uscita è 3×3 (dimensione della griglia) $\times 8$ (4 di bounding box + 1 sì/no + 3 classi).

Il limite è che posso avere al più un oggetto per unità di griglia.

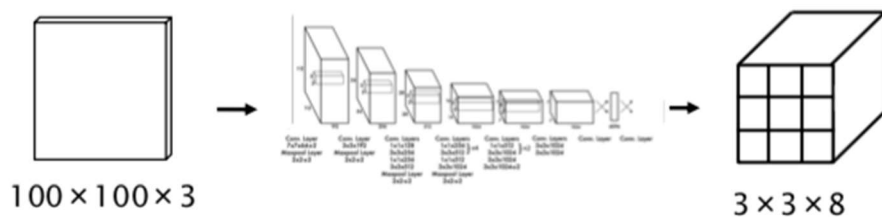


In realtà **YOLO** (*You Only Look Once*) fa qualcosa di più sofisticato:

- Prende l'immagine $100 \times 100 \times 3$
- Fa una serie di operazioni convoluzionali
- Ottengo l'uscita $3 \times 3 \times 8$ dove posso avere al più un oggetto per vettore

Nota:

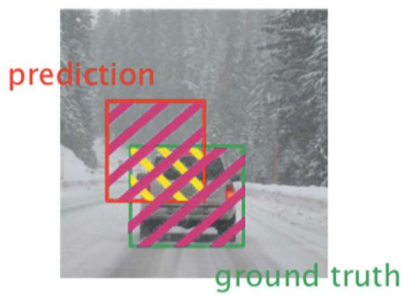
- Non usa sliding window
- Divide problema in sotto-problemi
- Funziona con più classi diverse
- Funziona con più oggetti diversi o anche di stessa classe in celle diverse
- Non funziona se oggetti in stessa cella



Nella realtà viene utilizzato con griglie con celle più piccole per far scendere la probabilità e la criticità di avere più midpoint nella stessa griglia

Evaluating object localization

Per dire se la predizione di un algoritmo è buona ci sono diversi approcci ma quello base è la "Intersection over Union": il rapporto tra l'area sovrapposta (gialla) e l'unione della predizione con il ground truth (fucsia).



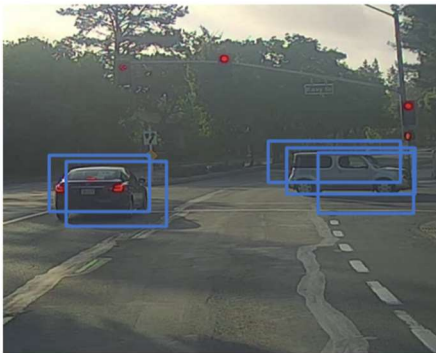
Intersection over Union (IoU)

$$\text{IoU} = \frac{\text{size of } \text{yellow box}}{\text{size of } \text{pink box}}$$

Si imposta quindi una soglia (spesso di 0.5) per dire se la predizione è corretta. (1 solo se coincidono perfettamente)
C'è il problema che questa dimensione cresce sia se sottostimiamo sia se sovrastimiamo.

Non-max suppression

Siccome si suddivide l'immagine in una griglia, ci possono essere per ogni oggetto diverse celle della griglia in cui si trova e quindi l'output dell'algoritmo restituirà diverse bounding box.



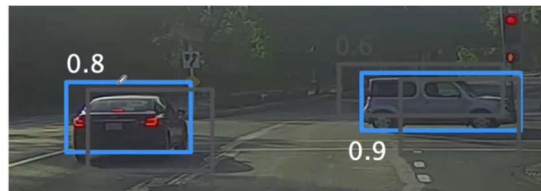
Per ognuno di essi la rete dà in output le probabilità ovvero i livelli di confidenza.



L'algoritmo di non-max suppression prende il bounding box con la probabilità più alta, e calcola la IoU con tutti gli altri presenti nella scena. Tutti quelli che si sovrappongono oltre una certa soglia, li butta via. A questo punto prende gli altri bounding box, non

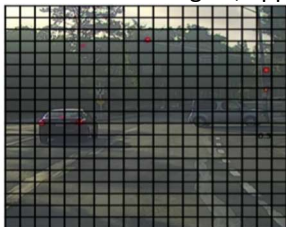
sovrapposti, prende il massimo e ripete la stessa cosa.

Otteniamo:



Non-max suppression algorithm

Prendiamo l'immagine, applico una griglia grande per evitare che ci siano due midpoint nella stessa cella.



19x19

$$\text{Each prediction } y = \begin{bmatrix} p_c \\ b_x \\ b_y \\ b_h \\ b_w \end{bmatrix}$$

*Assumption: only one class

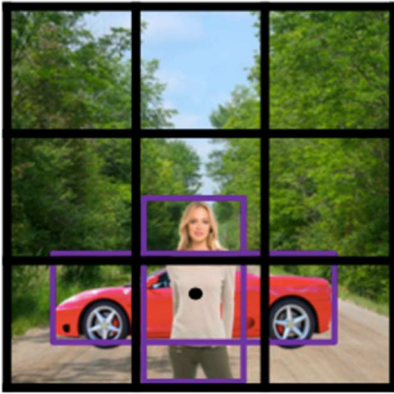
Ottengo dei vettori di output, **scarto tutti i box che hanno $p_c \leq 0.6$ e scarto qualsiasi box rimanente con $\text{IoU} \leq 0.5$.**

Se ci sono più classi si ripete il procedimento per ogni classe.

Nota: IoU alta -> maggior precisione nell'identificare le region. Soglia più bassa -> recall più alto, esser sicuri di non mancare nessuna automobile.

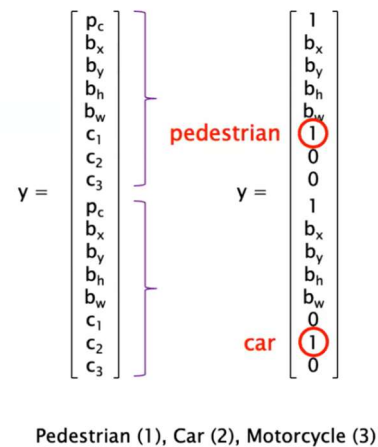
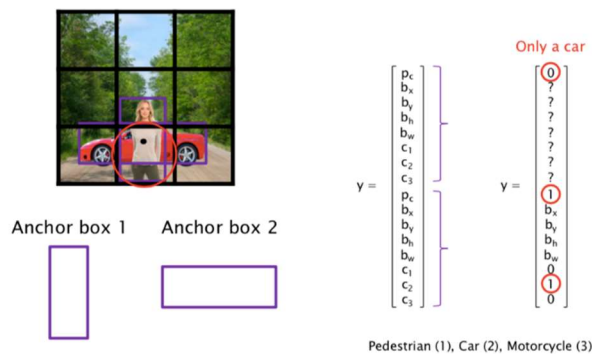
Overlapping objects

Cosa succede se ci sono più oggetti in una cella della griglia? Dobbiamo essere in grado di fare più predizioni per cella



L'idea alla base è quella di avere **anchor boxes**, ovvero si definiscono a priori delle ancore di una dimensione prefissata con dei propri fattori di forma.

Prima di usare questa idea avremmo avuto un output di otto valori per cella; invece, quello che si è fatto è stato avere nello stesso output l'unione delle predizioni per le singole ancore.



Previously:

Each object in training image is assigned to grid cell that contains that object's midpoint

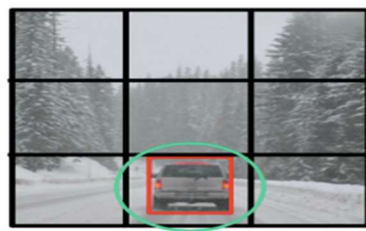
With two anchor boxes:

Each object in training image is assigned to grid cell that contains object's midpoint and anchor box for the grid cell with highest IoU

Anchor boxes are human-encoded priors on the size and aspect ratio (shape) of objects: designing anchors is deciding how many and which shapes to use

➔ Hand-engineering

YOLO algorithm



Anchor box 1



Anchor box 2



$y =$

$$y = \begin{bmatrix} p_c \\ b_x \\ b_y \\ b_h \\ b_w \\ c_1 \\ c_2 \\ c_3 \end{bmatrix} \begin{bmatrix} 0 \\ ? \\ ? \\ ? \\ ? \\ ? \\ ? \\ ? \end{bmatrix} \begin{bmatrix} 0 \\ ? \\ ? \\ ? \\ ? \\ ? \\ ? \\ ? \end{bmatrix} \dots \begin{bmatrix} 0 \\ ? \\ ? \\ ? \\ ? \\ ? \\ ? \\ ? \end{bmatrix} \begin{bmatrix} 0 \\ ? \\ ? \\ ? \\ ? \\ ? \\ ? \\ ? \end{bmatrix} \begin{bmatrix} 0 \\ ? \\ ? \\ ? \\ ? \\ ? \\ ? \\ ? \end{bmatrix}$$

3×3

Pedestrian (1), Car (2), Motorcycle (3)

y is $3 \times 3 \times \# \text{anch.} \times (4 + 1 + \# \text{classes}) = 3 \times 3 \times 16$

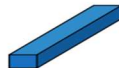
Training YOLO

Volume $3 \times 3 \times 16$ con 2 ancore -> ci devo mettere una rete e lo addestro -> mi dira che non c'è nulla per l'anchor box verticale mentre per l'anchor box orizzontale c'è l'automobile



$100 \times 100 \times 3$

→ ConvNet →



$3 \times 3 \times 16$

$$y = \begin{bmatrix} p_c \\ b_x \\ b_y \\ b_h \\ b_w \\ c_1 \\ c_2 \\ c_3 \end{bmatrix}$$

Non-max suppression with YOLO



For each grid cell, get 2 predicted bounding boxes

Get rid of low probability predictions

For each class (pedestrian, car, motorcycle) use non-max suppression to generate final predictions

Datasets for object detection

- PASCAL VOC: training set da 11k immagini con in media 2.7 oggetti ciascuna
- IMAGENET large scale: tantissime immagini, circa il 60% con un solo oggetto all'interno
- MS COCO: solo il 10% dell'immagini contiene una singola categoria di oggetti.

2007 PASCAL VOC

20 classes
11K training images
27K training objects

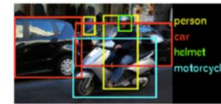
Was de-fact standard, currently used as quick benchmark to evaluate new detection algorithms



2013 ImageNet ILSVRC

200 classes
476K training images
534K training objects

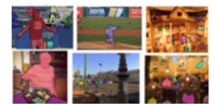
Essentially scaled up version of PASCAL VOC, similar object statistics



2015 MS COCO

80 classes
200K training images
1.5M training objects

More categories and more object instances in every image; only 10% of images contain a single object category (60% in PASCAL)



Evaluating detection performance

La valutazione viene fatta innanzi tutto sul classificatore usando la precision e la recall.

		Actual value	
		1	0
Predicted value	1	True positive	False positive
	0	False negative	True negative

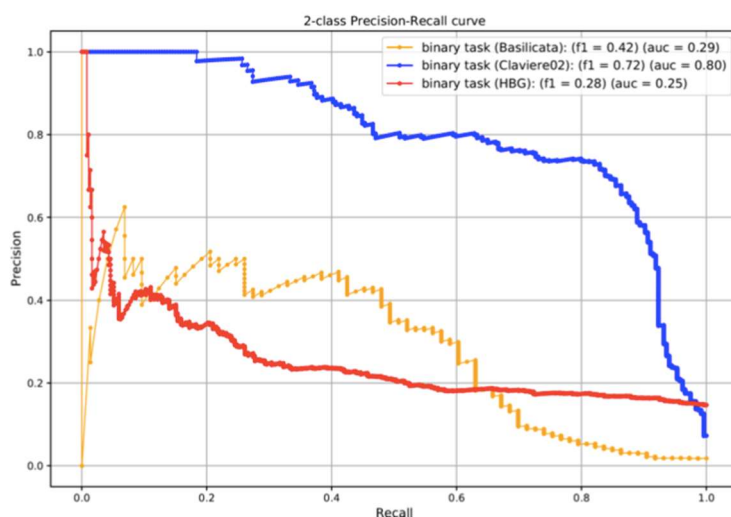
$$\text{Precision} = \frac{\text{True pos}}{\# \text{pred. positive}} = \frac{\text{True pos}}{\text{True pos} + \text{False pos}}$$

$$\text{Recall} = \frac{\text{True pos}}{\# \text{actual positive}} = \frac{\text{True pos}}{\text{True pos} + \text{False neg}}$$

Si combinano queste metriche con la IoU

- True positive
 - ♦ Correct class prediction and, e.g., IoU ≥ 0.5
- False positive
 - ♦ Wrong class prediction or, e.g., IOU < 0.5
- False negative
 - ♦ Missed (not detected) objects
- Only one detection can be matched to an object
- Distinctive value of precision and recall defined for each class

Si disegnano delle **curve chiamate precision-recall** e guardandole si può scegliere dove posizionarsi se andare più sulla precision o sulla recall.



Il problema è che l'andamento delle curve non è necessariamente monotono perché, se abbassiamo la soglia facciamo crescere sia i falsi positivi che i falsi negativi e la precision può andare un po' dove vuole.

La curva dipende anche dalla proporzione tra true positive e true negative (data imbalance).

Una buona misura di bontà di un algoritmo è data dall'area sotto la curva -> F1-score.

mAP (mean Average Precision)

Si prende l'area sotto tutte le curve (una curva per classe) e si calcola la media.

- Object detectors are usually ranked using the mean AP or mAP, which is the average AP over all classes

$$mAP = \frac{\sum_{c \in C} AP_c}{|C|}$$

- Sometimes, the mAP is denoted as mAP^{IoU} , where IoU denotes the threshold used to identify True positives and False positives
 - The higher the threshold, the better the detector must be at locating objects
 - In some benchmarks, the mAP is calculated at different IoU thresholds and then averaged

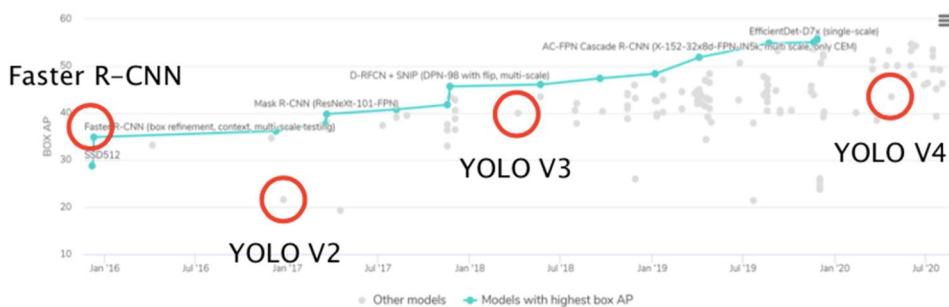
Altre metriche

- Other metrics have been proposed and used in order to focus on
 - Objects of different scales
 - Performance given a certain "budget", i.e., maximum detection per image

Average Precision (AP):	
AP	% AP at IoU=0.50:05:95 (primary challenge metric)
AP ^{IoU=50}	% AP at IoU=0.50 (PASCAL VOC metric)
AP ^{IoU=75}	% AP at IoU=0.75 (strict metric)
AP Across Scales:	
AP ^{small}	% AP for small objects: area < 32 ²
AP ^{medium}	% AP for medium objects: 32 ² < area < 96 ²
AP ^{large}	% AP for large objects: area > 96 ²
Average Recall (AR):	
AR ^{max=1}	% AR given 1 detection per image
AR ^{max=10}	% AR given 10 detections per image
AR ^{max=100}	% AR given 100 detections per image
AR Across Scales:	
AR ^{small}	% AR for small objects: area < 32 ²
AR ^{medium}	% AR for medium objects: 32 ² < area < 96 ²
AR ^{large}	% AR for large objects: area > 96 ²

COCO leaderboard

Si vede quali sono gli algoritmi che hanno performato meglio col tempo



Il grafico non tiene conto dei tempi di esecuzione.

Considerazioni

Object detectors non sono architetture end-to-end, cioè dove do l'immagine in input e ottengo l'output ma in mezzo ci sono una serie di meccanismi. -> Serve lavoro ulteriore, hand-crafted post-processing.

Tutti utilizzano un'architettura di base che poi viene variata e quindi queste si possono considerare come hyper-parametri o delle procedure da seguire.

La rappresentazione di bounding box non è ottimale in tutti i casi (shapes non regolari, classi sovrapposte, oggetti in posizioni non canoniche), così come le anchors (anche se convenienti se ci sono molte informazioni a priori).

La loss standard della regressione non necessariamente garantisce un buon IoU.