RELAZIONE PROGETTO DI METODI QUANTITATIVI PER L'INFORMATICA

Implementazione di YOLOv2 su un sottoinsieme del dataset COCO

Marco Collepiccolo - 1795881

Data 16/08/2020

ABSTRACT

La rete YOLOv2 è stata addestrata per l'object detection di tre classi di veicoli attraverso un dataset sottoinsieme di COCO. Viene implementato un algoritmo per il calcolo della mean Average Precision e si ottengono risultati in linea con quelli presentati nell'articolo scientifico di YOLOv2. Si effettuano confronti poi tra vari settaggi di iperparametri.

PRELIMINARIES

Per comprendere il lavoro svolto è necessario introdurre le metriche usate e il funzionamento della rete YOLOv2.

Le metriche

Le metriche disponibili sono diverse e forniscono metodi per misurare la performance del modello. Scegliere la giusta metrica è cruciale poiché non tutte sono adatte al task e al dataset che abbiamo. Nel nostro caso avremo infatti un dataset con classi non uniformemente distribuite.

IoU (Intersection over Union)

È una metrica usata nell'object detection e consiste nel rapporto tra area di intersezione e area di unione di due bounding box. Sarà utile nel processo di non-max suppression e nel calcolo di predizioni vere positive (true positive TP) o false positive (FP).

$$IoU = \frac{Area\ di\ intersezione}{Area\ di\ unione}$$

Precision

La precision consiste nel rapporto tra le predizioni corrette positive e tutte le predizioni positive effettuate.

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

Recall

La recall misura invece il rapporto tra i positivi correttamente predetti dal modello e tutti i positivi presenti, ovvero la somma tra i veri positivi correttamente identificati e i falsi negativi (FN).

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

Accuracy

Una metrica molto utilizzata ma che non useremo è l'accuracy. In generale è definita come il numero di elementi classificati correttamente diviso il totale, nel caso binario è:

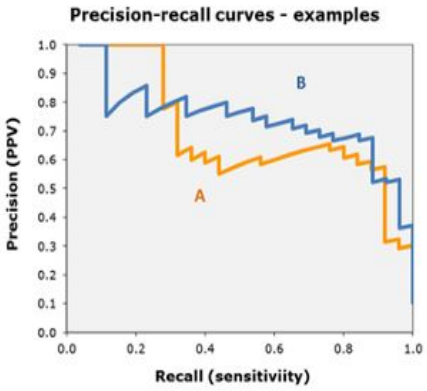
$$A = \frac{TP+TN}{TP+FP+TN+FN}$$

Questa metrica sembrerebbe quella più logica da usare per controllare la performance del modello, tuttavia nel caso di dataset con classi non uniformemente distribuite (come il nostro, si veda la sezione Selected Dataset) introduce bias rispetto alle classi più numerose, perciò è meglio usare altre metriche come la mean average precision (mAP), calcolata attraverso la precision-recall curve.

Precision-recall curve

La precision-recall curve viene ricavata graficando i valori di precision e recall (precision su ordinate, recall su ascisse) in funzione di un confidence threshold che decresce. Quando il confidence score decresce la recall può solo aumentare mentre la precision tende a diminuire. Per questa ragione la precision-recall curve ha generalmente un andamento decrescente, come mostrato nell'immagine che mostra due precision-recall curve.

ID#	Parameter, concentration	Disease Yes/No	Y (sum)	N (sum)	Precision (PPV)	Recall (sensitivity)
1	33.63	Y	1	0	1.00	0.013
2	10.63	Y	2	0	1.00	0.025
3	9.90	N	2	1	0.67	0.025
4	6.87	Y	3	1	0.75	0.038
5	6.15	Y	4	1	0.80	0.050
6	6.15	Y	5	1	0.83	0.063
7	5.53	Y	6	1	0.86	0.075
8	5.08	Y	7	1	0.88	0.088
....	....	....	....	....	....	....



Nell'object detection tuttavia spesso non sarà possibile raggiungere una recall di 1, dato che questo significherebbe aver riconosciuto tutte le bounding box di una classe.

Average Precision (AP)

È definita come l'area sotto la precision-recall curve. Generalmente prima di calcolare

l'average precision viene effettuato uno "smoothing" della P-R curve, ovvero si eliminano i piccoli "zig-zag" per rendere la curva monotona decrescente. Per il calcolo si può optare per un'interpolazione e somma oppure direttamente il calcolo dell'area sotto la curva (area under curve AUC) attraverso rettangoli.

$$AP = \int_0^1 P(r)\, dr$$ dove P(r) è la precision-recall curve

mean Average Precision (mAP)

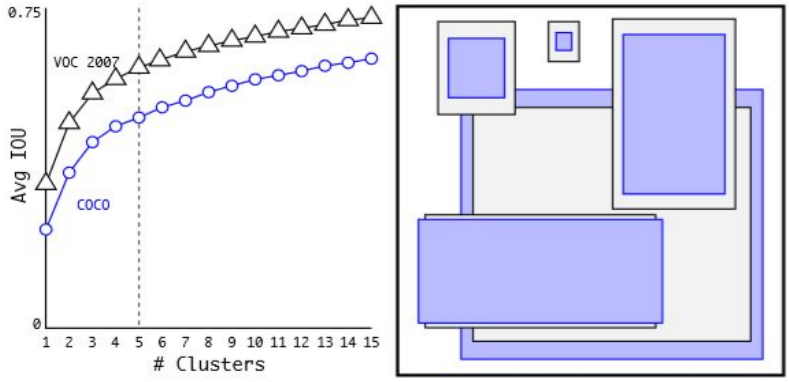
Nell'object detection la precision-recall curve e l'average precision vengono calcolate per ogni classe, perciò la media delle average precision sarà la mean Average Precision (mAP). La mAP si può calcolare anche facendo una media tra diversi threshold di IoU (Intersection over Union) usati per distinguere tra veri positivi e falsi positivi, come si fa per la mAP calcolata per la COCO challenge.

YOLOv2

L'algoritmo YOLOv2 è un'evoluzione dell'algoritmo YOLO (you only look once). Si tratta di un sistema di object detection che in una sola volta predice sia le bounding box che le classi associate a queste. Questo viene realizzato dividendo l'immagine in una griglia e risolvendo un problema di localization e classification per ogni elemento della griglia.

La griglia

Alla base del funzionamento di YOLOv2 c'è la suddivisione dell'immagine di input in una griglia SxS (nel nostro caso S=16). Ad ogni elemento della griglia vengono associate B anchor boxes. Le anchor boxes sono bounding box iniziali che hanno delle grandezze e forme standard. Da queste si predicono degli offset che formano la bounding box finale. Le anchor boxes in YOLOv2 vengono impostate a B = 5 come un buon compromesso tra la complessità del modello e una buona recall. Le forme delle bounding box sono quelle mostrate nell'immagine e vengono ricavate effettuando k-means clustering sulle bounding box di ground truth del dataset COCO o VOC 2007.



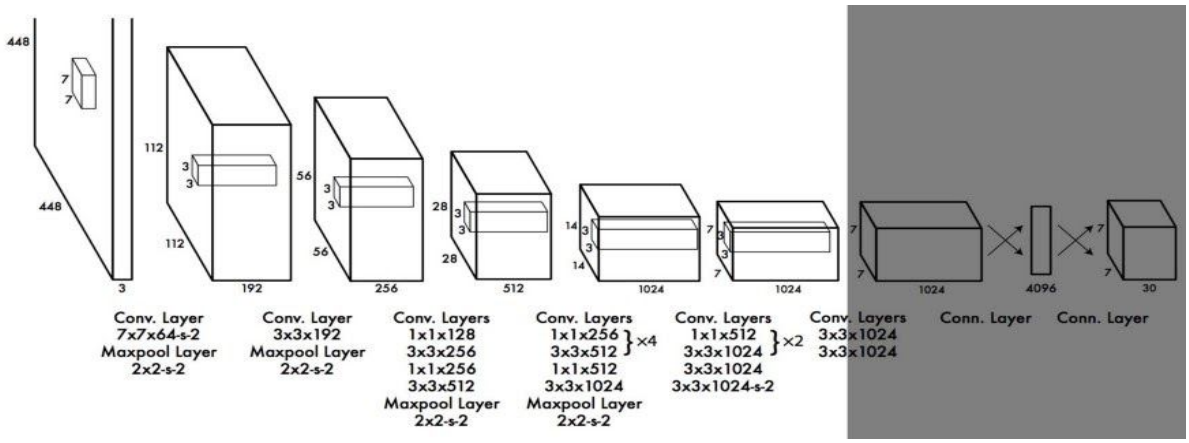
A sua volta per ogni anchor box vengono definiti 4 valori di dimensione della bounding box: (x, y, w, h), un valore di “objectness” che indica la probabilità che nella bounding box sia presente un oggetto e infine C valori di class probability, ovvero valori che indicano la probabilità della classe per ogni bounding box (nel nostro caso C = 3).

Dai valori di objectness e class probability si può anche ottenere un valore di class confidence score, ottenuto moltiplicando i due precedenti. Questo valore indicherà sia la confidenza della localization che della classification.

La rete

La rete associata a YOLOv2 perciò prenderà in input un’immagine e restituirà come output un tensore di dimensione SxSxBx(4+1+C) (nel nostro caso 16x16x5x(4+1+3)).

L’interno della rete è molto simile alla rete della prima versione di YOLO; questa è composta da 24 livelli convoluzionale e 2 livelli fully connected. Uno dei cambiamenti effettuati a questa architettura per passare a YOLOv2 è rimuovere i livelli finali fully connected (mostrati nell’immagine come oscurati) e rimpiazzare l’ultimo livello convoluzionale con uno che abbia come output la dimensione finale della predizione.



Il training

I pesi della rete vengono prima inizializzati con dei valori pretrained che sono stati addestrati per la classificazione sul dataset ImageNet con 1000 classi. Successivamente il training viene effettuato con questa funzione di loss:

$$
\begin{aligned}
&\lambda_{\text{coord}} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{obj}} \left[(x_i - \hat{x}_i)^2 + (y_i - \hat{y}_i)^2 \right] \\
&+ \lambda_{\text{coord}} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{obj}} \left[\left(\sqrt{w_i} - \sqrt{\hat{w}_i} \right)^2 + \left(\sqrt{h_i} - \sqrt{\hat{h}_i} \right)^2 \right] \\
&+ \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{obj}} \left(C_i - \hat{C}_i \right)^2 \\
&+ \lambda_{\text{noobj}} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{noobj}} \left(C_i - \hat{C}_i \right)^2 \\
&+ \sum_{i=0}^{S^2} \mathbb{1}_{i}^{\text{obj}} \sum_{c \in \text{classes}} \left(p_i(c) - \hat{p}_i(c) \right)^2
\end{aligned}
$$

La funzione di loss tiene conto di tutte le caratteristiche necessarie ed è quindi composta da vari termini che calcolano i diversi errori.

I primi due termini costituiscono la localization loss, ovvero l'errore tra le coordinate e la dimensione della bounding box reale e quella predetta. Questi due termini vengono considerati solo quando nella bounding box è presente un oggetto, per questa ragione c'è una funzione indicatrice che vale 1 se nella posizione i della griglia, per la bounding box j, c'è un oggetto. Si usa la radice quadrata per larghezze e altezze poiché piccole

deviazioni in bounding box grandi importano meno.

Il terzo e il quarto termine rappresentano la confidence loss, ovvero l'errore tra i confidence score ("objectness") delle varie bounding box. Il terzo termine vale per quelle bounding box in cui è presente un oggetto, mentre per quelle in cui non è presente l'oggetto si usa un parametro noobj che è 0.5 e serve per evitare di far divergere il modello data la grande quantità di bounding box che non presentano oggetti.

L'ultimo termine invece costituisce la classification loss, ovvero la loss relativa alla classe predetta per ogni bounding box (nell'immagine la loss function è per YOLOv1, basterà aggiungere un altro simbolo di sommatoria per sommare anche per ogni anchor box).

Non max suppression

La rete di YOLOv2 restituisce quindi una serie di valori relativi a bounding box e classi predette. Per filtrare questi risultati si effettua non max suppression:

1. Ordina i risultati per confidence score (objectness o class confidence score)
2. Prendi il primo ed eliminalo dalla lista
3. Elimina tutti gli altri risultati che hanno un IoU maggiore di un threshold (IOU_THRESHOLD) con il risultato scelto
4. Ritorna al passo 2 se ci sono ancora risultati

Questo significa che se l'IOU_THRESHOLD è basso molti risultati verranno filtrati ed è meno probabile che uno stesso oggetto venga identificato due volte, mentre se è alto è possibile che oggetti vengano riconosciuti più volte da bounding box leggermente scostate tra loro.

Nel codice del progetto il confidence score usato è il "class confidence score", che indica la confidenza sia della localization che della classification. In ogni caso la differenza tra questi due score è generalmente piccola, poiché se abbiamo class confidence score elevato necessariamente c'è objectness elevata, allo stesso modo se c'è objectness elevata significa che è presente un oggetto da classificare e il classificatore sarà poi in grado di riconoscerlo.

SELECTED DATASET

Il dataset scelto è stato ricavato da un sottoinsieme del dataset COCO.

E’ composto da tre classi: automobili, veicoli di grandi dimensioni e veicoli a due ruote, identificate dalle label “car”, “truck” e “bicycle”. La classe car è la stessa del dataset COCO, la classe truck è stata ricavata con elementi delle classi COCO “truck” e “bus”, infine la classe bicycle è stata ricavata da elementi delle classi COCO “motorcycle” e “bicycle”.

Le immagini del dataset in questo modo avranno solo annotazioni relative ai veicoli (tutte le altre annotazioni saranno rimosse) e la rete dovrà effettuare object detection per automobili, veicoli di grande misura (“truck”) o veicoli a due ruote (“bicycle”).

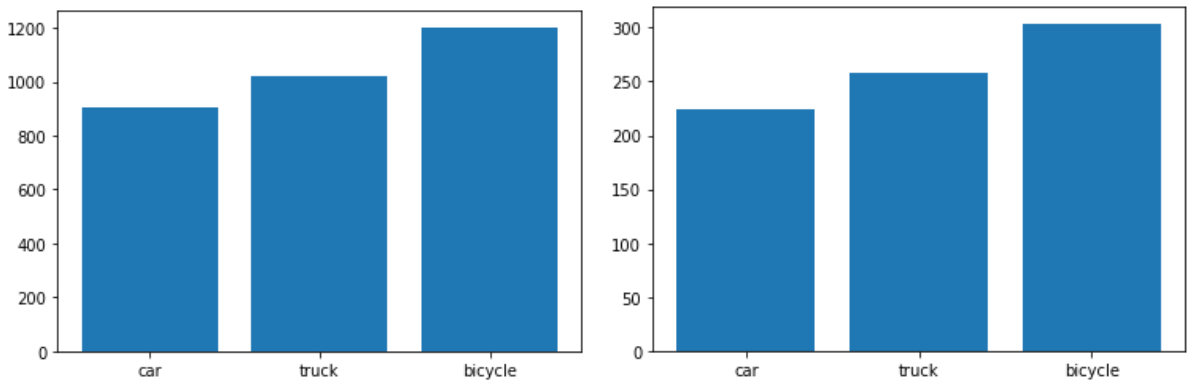
Si tratta di 1398 immagini di training e 340 immagini di validation. Ci sono almeno 400 immagini di training e 100 immagini di validation per ogni classe (ovvero almeno 400 bounding box di training per classe e 100 bounding box di validation per classe).

Nel dataset COCO ogni immagine può contenere numerosi oggetti, scegliendo 400 immagini per classe molte immagini hanno sia la classe voluta che altre classi (un’immagine con una bicicletta può anche contenere un’automobile o un autobus). Questo implica che la distribuzione delle classi non è uniforme (ovvero non ci sono precisamente 400 bounding box per classe).

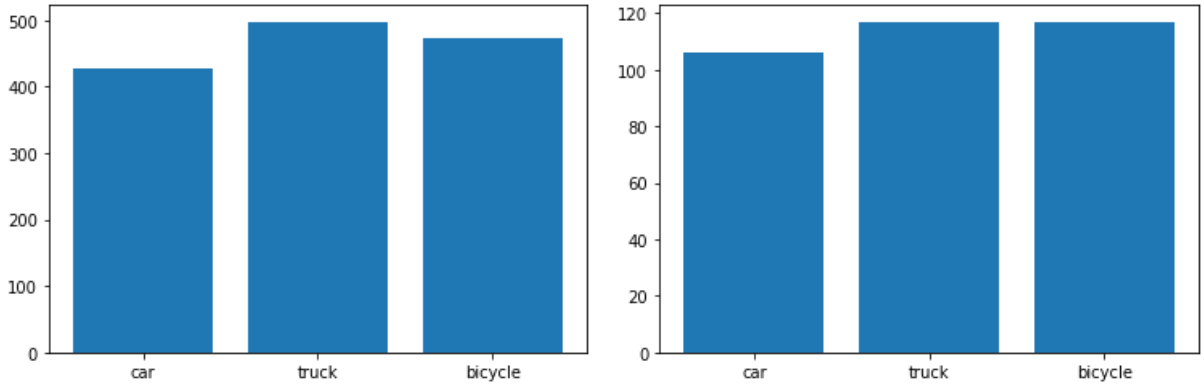
La distribuzione leggermente non uniforme delle classi replica la distribuzione più generale delle classi nel dataset COCO o in qualsiasi altro dataset (in cui quasi sempre alcune classi appaiono più frequentemente di altre). Per questa ragione usare l’accuracy come metrica è inaccurato e introdurrebbe un bias, non tenendo conto della distribuzione delle classi.

Class Name	**Training Set**	**Validation Set**
car	904	225
truck	1020	258
bicycle	1202	304

Gli istogrammi riportano questi dati per il training set (a sinistra) e il validation set (a destra)



Otteniamo invece un grafico con una distribuzione più uniforme se raggruppiamo le immagini in base alla classe “dominante” in ogni immagine (in questo caso consideriamo la classe più numerosa per ogni immagine).



Una delle maggiori sfide del dataset è trainare il modello affinché sia in grado di generalizzare. Dal dataset infatti non è possibile rimuovere le immagini troncate attraverso le API di COCO, anche perché non è presente nessuna annotazione che le identifichi. Inoltre molte immagini sono peculiari e distanti da quello che comunemente si può identificare in una delle tre classi. Ad esempio l’immagine a sinistra ha una bounding box in basso “car”, che corrisponde al tetto della macchina, mentre quella a destra ha una bounding box “motorcycle” che corrisponde allo specchietto.

METHOD

Il task scelto è quello di misurare la performance del sistema di object detection YOLOv2 sul dataset scelto, utilizzando diverse metriche. In particolare verranno calcolate precision e recall, ma anche la mean average precision, implementando il relativo algoritmo per il calcolo. Si procederà poi a confrontare queste metriche cambiando alcuni iperparametri. Infine si confronteranno i risultati ottenuti con quelli presentati nella pubblicazione scientifica di YOLOv2, tenendo tuttavia presente che i dataset sono diversi.

Il calcolo di precision e recall

Il calcolo di precision e recall viene effettuato per ogni classe e su tutto il validation dataset. Per ottenere un solo valore di precision poi si fa la media dei valori di precision e recall di ogni classe.

Il calcolo richiede che si trovino i veri positivi, i falsi positivi e i falsi negativi. Per identificare una predizione come vera positiva si seguono queste condizioni:

1. La bounding box predetta deve superare un threshold di IoU (che chiameremo IOU_THRESHOLD_AP per distinguerlo da quello usato per non max suppression) con una bounding box di ground truth
2. La classe predetta e quella della bounding box ground truth devono essere medesime (questo è banale nel caso in cui si effettua il calcolo di volta in volta per ogni classe, filtrando preventivamente bound box predette e ground truth di classi diverse)
3. La bounding box di ground truth non deve essere già stata riconosciuta precedentemente

Se la predizione non soddisfa queste condizioni allora viene classificata come falsa

positiva.

Iterando questo processo per ogni immagine del validation dataset si possono calcolare i veri positivi e i falsi positivi totali.

I falsi negativi vengono invece calcolati come il numero totale di bounding box ground truth appartenenti ad una classe nel validation dataset meno i veri positivi di quella stessa classe.

Si ottengono quindi tre valori di precision e recall, ognuno relativo ad una classe del dataset. Per trovare la precision e recall finale si effettua una media dei tre.

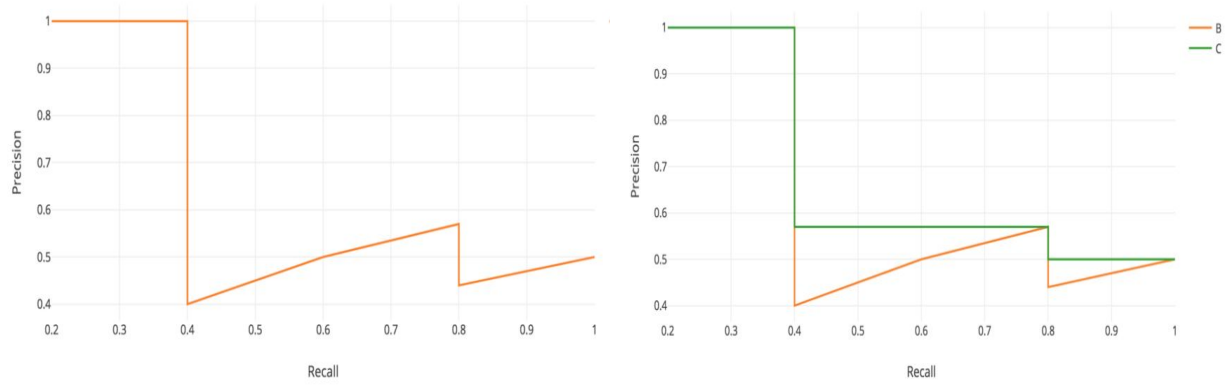
La precision-recall curve e la mean average precision

La precision-recall curve, così come precision e recall viste precedentemente, si calcola per ogni classe attraverso un algoritmo:

1. Si usa il modello per effettuare predizioni (non max suppression compresa) su tutto il validation dataset, ad ogni predizione è associato un confidence level (ovvero l'objectness che è stata vista nella sezione La Griglia)
2. Si ordinano tutte le predizioni in maniera decrescente in base a questo confidence level
3. Per ogni predizione si calcola se è vera positiva o falsa positiva utilizzando il metodo visto nella sezione precedente
4. Ad ogni confidence level si calcola quindi precision e recall considerando uno confidence level threshold che che decresce

Iterando questo algoritmo per ogni classe si ottengono tre precision-recall curve che possono essere usate per il calcolo dell'average precision di ogni classe.

Tuttavia, prima di calcolare l'average precision è necessario effettuare un'operazione di smoothing della precision-recall curve ottenuta, ovvero bisogna passare da una curva simile alla curva arancione nell'immagine a sinistra a una curva simile alla curva verde nell'immagine a destra.



Nell'esempio infatti se per un valore di recall pari a 0.6 abbiamo una precision di 0.5, possiamo aumentare sia precision che recall contemporaneamente ottenendo una recall di 0.8 per una precision di circa 0.6. Eliminiamo quindi tutti gli "zig-zag" per ottenere una curva che per ogni valore di recall $\mathbf{r}$ ci da il massimo della precision per ogni valore di recall $r \geq \mathbf{r}$, ovvero:

$$p_{interp}(r) = \max_{\tilde{r} \geq r} p(\tilde{r})$$

Per il calcolo dell'average precision si è optato per il calcolo dell'area sotto la curva. Questo calcolo è molto semplificato dopo lo smoothing della curva e consiste in una somma di rettangoli per ogni valore di recall. La formula usata è quindi:

$$\mathrm{AP} = \sum \left(r_{n+1} - r_n \right) p_{interp}(r_{n+1})$$

Infine la mean average precision viene calcolata come la media delle tre average precision calcolate per ogni classe.

Studio degli iperparametri

Ci sono diversi iperparametri presenti nel codice e a seconda di come vengono impostati determinano risultati diversi. Lo score threshold e l'iou threshold sono parametri usati per non max suppression, mentre iou threshold ap è un parametro usato per distinguere i veri positivi, infine le epoche determinano la durata del training.

- Lo score threshold è un threshold usato per effettuare non max suppression. Nel codice viene implementato come il minimo valore di class confidence score che deve avere una bounding box affinché non venga filtrata. Il class confidence score è un confidence score ottenuto moltiplicando l'objectness con la class probability.

- L'iou threshold viene usato per non max suppression ed elimina le bounding box sovrapposte tra loro oltre tale threshold.
- L'iou threshold ap (average precision) è un threshold usato per il calcolo dei veri positivi. Vengono identificati come veri positivi quelle bounding box che superano questo threshold.
- Il numero di epoche determina la durata del training ed è generalmente definito come il numero di elementi del dataset diviso il batch size, perciò è il numero di passi necessari per far passare tutto il dataset nella rete.

Modificando questi iperparametri il modello restituisce risultati differenti e i valori di precision, recall e mean average precision cambiano. Comparando questi risultati e alcuni esempi di predizione è possibile scegliere iperparametri adatti al task che ci si pone. Nel progetto ci limiteremo ad evidenziare le differenze e i trade off, senza scegliere definitivamente alcun iperparametro.

IMPLEMENTATION

Il dataset e preprocessing

Le operazioni sul dataset vengono tutte effettuate dallo script python create_dataset.py [9].

Il dataset è stato dapprima scaricato attraverso le API di COCO che permettono di scaricare un numero arbitrario di immagini con bounding box di classi scelte.

Successivamente è necessario svolgere delle operazioni di preprocessing sulle immagini per darle come input al codice della rete [7]. La rete infatti accetta immagini 512x512, perciò è necessario ridimensionare le immagini tuttavia mantenendo l'aspect ratio che costituisce un'informazione importante per il training. Vengono quindi aggiunte bande laterali scure dove necessario e ricalcolate le dimensioni e coordinate delle bounding box. Il ridimensionamento delle immagini ha portato alcune bounding box già piccole ad avere dimensioni ancora più piccole (pochi pixel), queste sono state rimosse.

Un'altra operazione da svolgere è modificare le annotazioni dal formato COCO al formato PASCAL VOC. Bisogna quindi fare il parsing del file json fornito dal dataset COCO, estrarre le informazioni rilevanti e costruire un file xml compatibile con la rete, effettuando anche l'opportuna traslazione delle coordinate e dimensioni delle bounding box nel nuovo formato.

L'ultima operazione che viene svolta sul dataset si trova nel codice della rete e consiste nel dataset augmentation.

Gli algoritmi

Gli algoritmi descritti nella sezione Method vengono implementati nel codice della rete attraverso le funzioni tp_fp_batch e average_precision.

La funzione `tp_fp_batch(y_pred, true_boxes, score_threshold, iou_threshold, iou_threshold_ap)` viene usata durante il training ed è usata nel calcolo di precision e recall per ogni epoca. Prende in input una predizione e le bounding box di ground truth, assieme a tre threshold descritti precedentemente. Restituisce poi il numero di veri positivi e falsi positivi presenti seguendo l'algoritmo descritto in Method.

L'implementazione è realizzata attraverso un post processing sulla predizione y_pred, un'iterazione per ogni classe e a sua volta un'iterazione per ogni immagine del batch. Di ogni immagine si esaminano le bounding box predette e si verificano le tre condizioni affinché sia vera positiva. Alla fine si aggiorna un array di contatori che tiene il numero di veri positivi e falsi positivi per ogni classe.

Una volta che questa funzione ritorna il numero di veri positivi e falsi positivi per ogni classe si calcola la precision e recall di ogni classe e infine la precision e recall finale è data dalla media di queste tre. Questo valore verrà salvato.

La funzione `average_precision(model, val_dataset, n_val_batches, APclass, totClass, score_threshold, iou_threshold, iou_threshold_ap)` viene invece usata con il modello trainato per calcolare una serie di metriche che verranno utilizzate nei results. La funzione ha come input il modello, il validation dataset e il suo numero totale di batch, la classe su cui vogliamo calcolare l'average precision e il numero totale di elementi di quella classe e infine i tre threshold. Restituisce invece l'average precision, i valori della precision-recall curve e della precision-recall curve dopo lo smoothing e infine la curva dei confidence score (ovvero i valori di objectness associati a ogni predizione).

L'implementazione è effettuato dando in input al modello tutti i batch del validation dataset di volta in volta, verrà poi effettuata l'operazione di post processing sulla predizione e si deciderà di volta in volta se le predizioni effettuate sono vere positive o false positive, come per la funzione tp_fp_batch. Questa volta tuttavia non verrà incrementato un contatore, poiché non siamo interessati al numero totale di elementi veri positivi e falsi positivi. Verrà invece creata una tabella in cui ad ogni predizione si associa il corrispettivo valore di objectness e un valore che indica se la predizione è vera positiva o falsa positiva. Questo è necessario per implementare l'algoritmo descritto

precedentemente. Dobbiamo infatti ordinare questa tabella in ordine decrescente di objectness, e poi calcolare precision e recall per ogni livello di objectness. Ad ogni livello infatti vengono aggiornati dei contatori che hanno il numero totale di veri positivi e falsi positivi. Se la predizione è vera positiva si aggiunge 1, altrimenti si aggiunge 1 ai falsi positivi. In questo modo ad ogni livello di objectness è associato un livello di veri positivi e falsi positivi cumulativi con cui si può calcolare precision e recall. L'operazione svolta equivale a calcolare precision e recall del modello diminuendo di volta in volta lo score threshold che filtra in prima istanza le predizioni. Perciò l'algoritmo restituirà una tabella con tutti i valori di objectness ordinati in modo decrescente e ad ognuno di questi è associato un valore di precision e recall, da questi valori si può rappresentare il grafico della precision-recall curve.

L'altra operazione che viene effettuata è lo smoothing della precision recall curve, implementando la formula vista precedentemente. Una volta effettuata questa operazione si possono sommare le varie aree dei rettangoli sotto la precision recall curve e ottenere l'average precision.

Il codice viene omesso per brevità ma è disponibile in [9] nella sezione "Evaluation metrics functions" del notebook jupyter.

Per permettere di calcolare la IoU inoltre è stata implementata una nuova funzione iou_scalar. La funzione iou già presente infatti lavora coerentemente con le richieste della rete e quindi se usata per precision e recall potrebbe dare valori di IoU non consentiti (ad esempio negativi), cosa che non accade invece se usata per il calcolo della loss. Inoltre la funzione iou presente lavora con vettori numpy. Per questa ragione è stata implementata una nuova funzione adatta all'uso per il calcolo delle metriche.

Modifiche alla rete

È stato poi modificato il numero di passi per epoca. Sebbene il numero di passi per epoca sia generalmente uguale al numero di passi necessari per far trainare la rete una volta su tutto il dataset, nel codice è stato inizialmente impostato a 10 per il training e a 2 per il validation. Per poter calcolare precision, recall e loss per tutto il validation dataset tuttavia il numero di passi per epoca è stato impostato al numero totale di batch del validation dataset. Sul validation dataset infatti non è stata effettuata nessuna operazione di dataset augmentation e inoltre è stata anche rimosso lo shuffling per permettere di passare il validation dataset sempre nello stesso ordine. Allo stesso modo sono stati aumentati i passi per il training di circa dieci volte.

Tensorboard

Infine un dettaglio implementativo è l'uso di tensorboard per salvare i dati rilevanti e creare un report finale delle metriche di loss, validation loss, precision e recall. Al termine di ogni epoca si registrano i nuovi valori.

EXPERIMENTS

Gli esperimenti che vengono svolti hanno come risultato, oltre al grafico decrescente della loss, quattro metriche: precision, recall e mean average precision assieme agli elementi relativi (precision-recall curve, average precision per classe, etc...).

Il numero di epoche

I primi esperimenti svolti sono serviti a determinare un numero di epoche che non mandasse il modello in overfitting. Sono stati confrontati due iperparametri: numero di epoche pari a 10 e numero di epoche pari a 30. Nel secondo caso, dalle curve di loss è possibile evincere che il modello era in overfitting: la validation loss non diminuiva mentre la training loss continuava a diminuire. Con 10 epoche invece la validation loss si stabilizza e la training loss non si discosta molto dalla validation loss. Perciò per trainare il modello sono state scelte 10 epoche.

Gli esperimenti

La precision e la recall sono state calcolate per ogni epoca, mentre la mean average precision e la precision-recall curve è stata calcolata con il modello trainato.

Gli esperimenti (sia durante il training che a modello trainato) sono stati effettuati con tre combinazioni di iperparametri

1. score_threshold = 0.50 e iou_threshold = 0.45
2. score_threshold = 0.55 e iou_threshold = 0.35
3. score_threshold = 0.60 e iou_threshold = 0.25

Questi threshold diventano sempre più stringenti e quindi rimangono progressivamente solo le predizioni con confidence score elevato e che non si sovrappongono con altre predizioni durante il non max suppression. Questi esperimenti forniscono un'indicazione sull'importanza della scelta dei parametri in base all'uso che si deve fare del modello. Avere threshold alti infatti porterà ad avere precision più elevate, a scapito tuttavia della recall. Cambieranno anche i valori della mean average precision.

Il threshold di IoU per discriminare i veri positivi (iou_threshold_ap) usato per questi

esperimenti è sempre 0.50. Di conseguenza le average precision e la mean average precision si indicano con mAP@0.5.

È stato poi effettuato un esperimento simile alla coco challenge, ovvero è stata calcolata la mean average precision per diversi iou_threshold_ap.

I threshold utilizzati, come nella coco challenge, sono 10, partendo da 0.5 si aggiunge 0.05 fino a giungere a 0.95. Ogni threshold restituirà una mean average precision differente, la media di queste costituisce la mean average precision utilizzata per la coco challenge.

Questi threshold sono via via sempre più stringenti ed aumentando causano logicamente un decremento della mean average precision.

Dettagli tecnici

Il batch size con cui è stato effettuato il training è pari a 10 sia per il validation set che il training set. Non è stato cambiato rispetto a quello fornito originariamente dal codice [7]. Il numero di passi per epoca è stato impostato a 140 per il training set e a 34 per il validation set. Questo ovviamente ha portato a diminuire il numero di epoche dalle centinaia alle decine.

Il setting fisico è costituito dalle risorse GPU cloud messe a disposizione dalla piattaforma google colab. Questo hardware ha permesso di avere un tempo di addestramento per epoca è di circa 6 minuti.

RESULTS

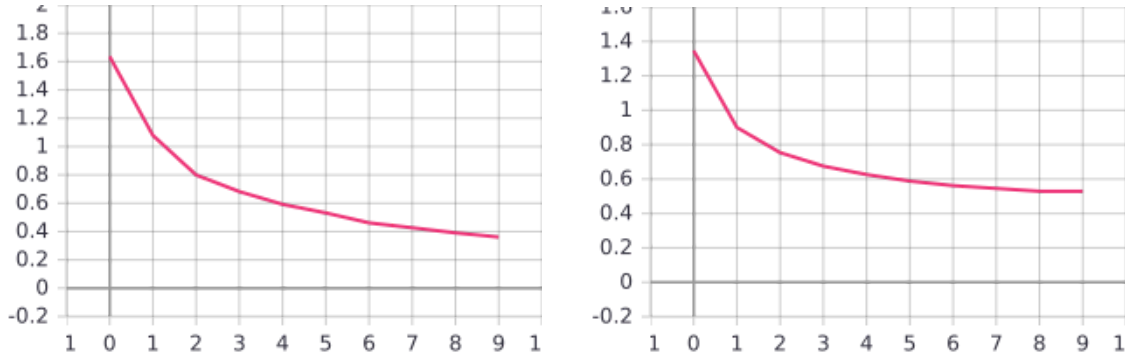
I risultati presentati sono divisi in quattro parti:

1. Nella prima ci sono i risultati registrati durante il training (loss, precision e recall) messi in relazione al numero di epoche
2. La seconda ha i valori di average precision, mean average precision e precision-recall curve del modello trainato
3. Nella terza vengono presentati diversi valori di mean average precision a diversi threshold di IoU, in maniera simile a quanto si fa per la coco challenge
4. Infine vengono presentati esempi di predizioni positive e negative, in cui, se presenti, vengono sottolineati gli errori di classificazione e localizzazione

Loss, precision e recall

Come spiegato precedentemente, si è optato per 10 epoche di training. Come si vede dal

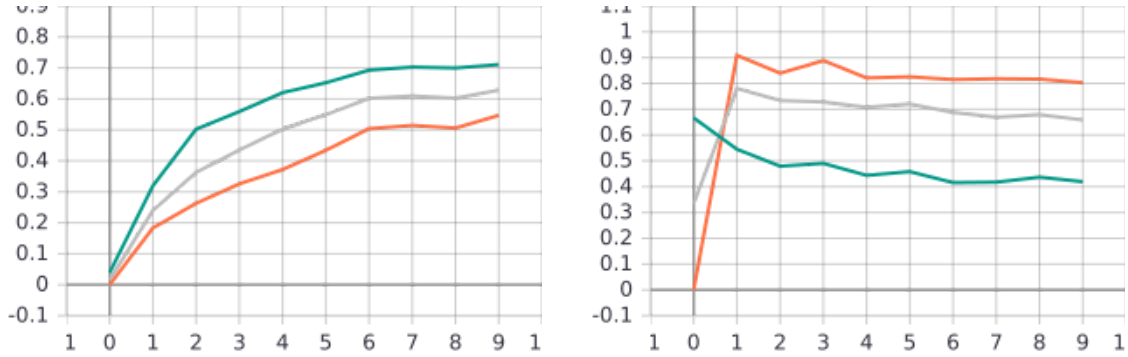
grafico, a sinistra abbiamo la training loss mentre a destra la validation loss. Le due curve hanno una differenza finale pari a circa 0.1, la validation loss si stabilizza attorno al valore 0.5 e continuando il training oltre le 10 epoche rimane su quel valore, tuttavia continuando a trainare sul training set il modello va in overfitting.



I valori di precision e recall vengono invece calcolati secondo i tre differenti threshold illustrati in "Gli esperimenti":

4. in verde score_threshold = 0.50 e iou_threshold = 0.45
5. in grigio score_threshold = 0.55 e iou_threshold = 0.35
6. in arancione score_threshold = 0.60 e iou_threshold = 0.25

Come previsto vediamo che a sinistra la recall è maggiore per i threshold meno esigenti mentre più bassa per quelli più esigenti. La precision a destra invece segue un andamento opposto, in cui i threshold meno esigenti corrispondono a una precision più bassa, mentre i threshold più esigenti danno una precision più alta.



mean Average Precision

Allo stesso modo usiamo i tre threshold per il calcolo della mean average precision. In questo caso l'unico threshold che rimarrà fisso è l'iou threshold ap, settato a 0.5. Notiamo dai valori ricavati una peculiarità, da mettere in relazione con i risultati precedenti: la

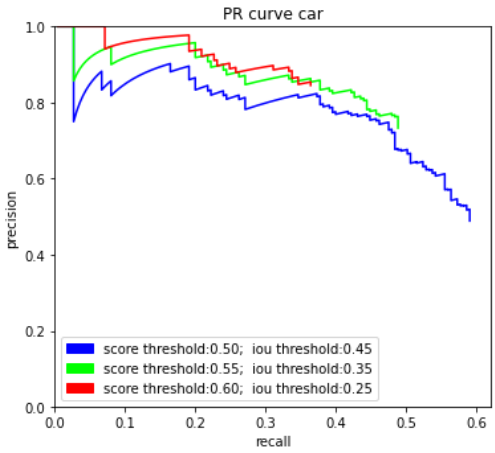
mean average precision tende a diminuire all'aumentare dei threshold. Questo perché (come si vedrà poi nelle precision-recall curve) la perdita che si ha per la diminuzione della recall è più alta rispetto al guadagno relativo alla precision.

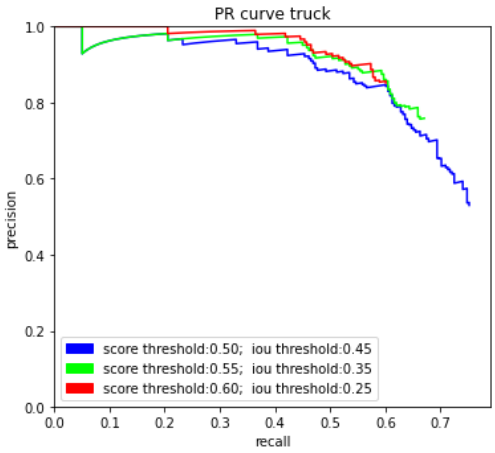
score threshold/ iou threshold	mAP@0.5	AP@0.5 car	AP@0.5 truck	AP@0.5 bicycle
0.50/0.45	0.594	0.523	0.670	0.590
0.55/0.35	0.563	0.497	0.641	0.551
0.60/0.25	0.508	0.430	0.590	0.504

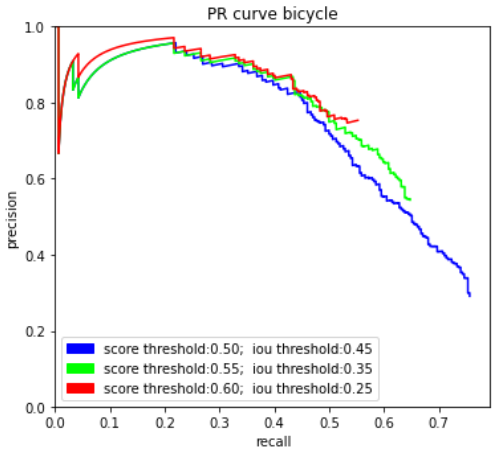
Le curve mostrate sono le precision-recall curve. La prima è la versione senza effettuare lo smoothing. Notiamo infatti che la curva non è monotona decrescente, per le ragioni già spiegate. Dopo lo smoothing tuttavia la curva diventa monotona decrescente, da queste sono state calcolate le average precision.

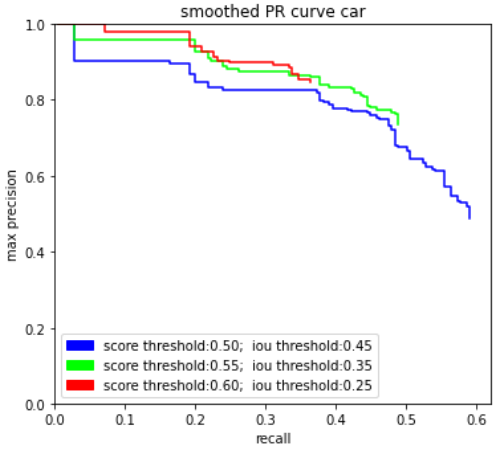
Osservando i grafici appare immediatamente chiaro perché la mean average precision diminuisce: lo scostamento tra le varie precision è piccolo. Sebbene la curva blu (threshold meno esigente) abbia precision generalmente più basse rispetto alla verde e alla rossa, i livelli di precision si mantengono più o meno sugli stessi valori. I livelli di recall che raggiunge la curva blu invece sono sensibilmente più alti sia della verde che della rossa, questo significa che l'area calcolata sarà più alta e quindi la mean average precision raggiungerà valori più alti. Questo ovviamente significa che ci saranno più predizioni doppie o triple per uno stesso oggetto, tuttavia non abbastanza da abbassare significativamente la precision.

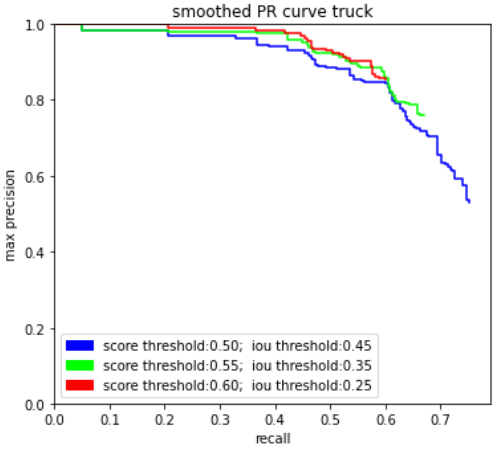
Notiamo inoltre che i grafici ottenuti sono perfettamente compatibili con i valori di precision e recall mostrati precedentemente, dove per valori di recall maggiori di 0.7 ci troviamo a precision di 0.4, per recall attorno a 0.6 ci troviamo a precision di 0.6, per recall di 0.5 e minori abbiamo precision dello 0.8 e superiori.

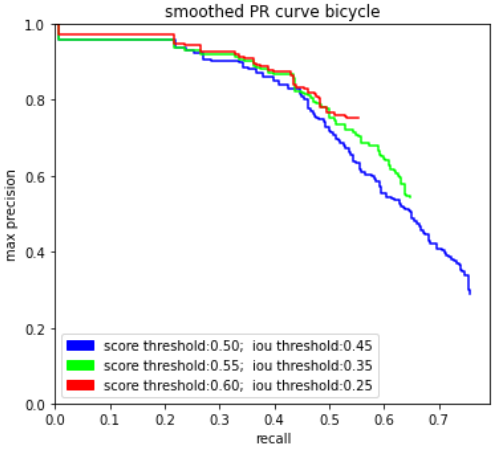




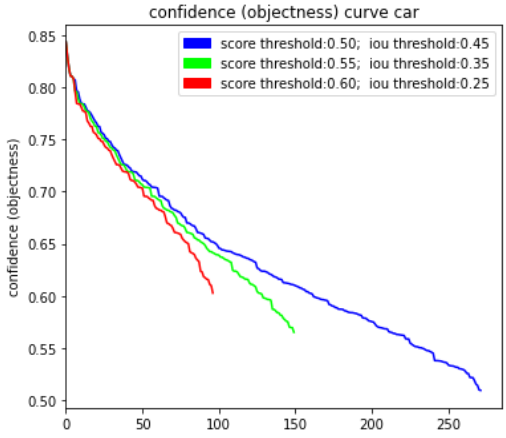


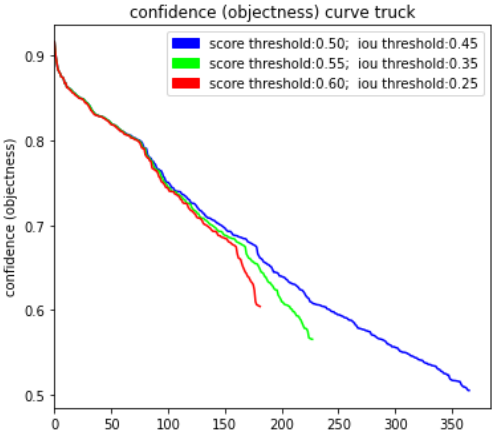


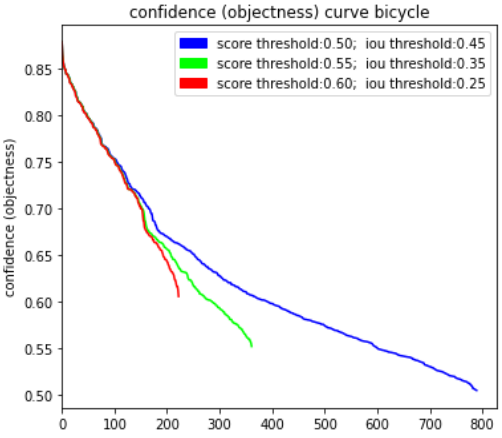




Quest'ultima curva invece mostra l'andamento dei confidence score, ovviamente decrescente, ma diverso a seconda dei threshold scelti. Threshold più stringenti rimuovono dalle predizioni molte bounding box e quindi corrispondono a confidence score che diminuiscono più velocemente.







I valori di average precision e mean average precision ottenuti inoltre sono compatibili con quelli di YOLOv2 (sebbene i dataset su cui vengono calcolati siano diversi). Sull'intero dataset COCO YOLOv2 ha la performance riportata in questa tabella (presa dalla pubblicazione scientifica di YOLOv3 [6]). Siamo interessati alla colonna AP_{50} che corrisponde alla nostra mAP@0.50, riga di YOLOv2.

	backbone	AP	AP_{50}	AP_{75}	AP_S	AP_M	AP_L
Two-stage methods							
Faster R-CNN+++ [5]	ResNet-101-C4	34.9	55.7	37.4	15.6	38.7	50.9
Faster R-CNN w FPN [8]	ResNet-101-FPN	36.2	59.1	39.0	18.2	39.0	48.2
Faster R-CNN by G-RMI [6]	Inception-ResNet-v2 [21]	34.7	55.5	36.7	13.5	38.1	52.0
Faster R-CNN w TDM [20]	Inception-ResNet-v2-TDM	36.8	57.7	39.2	16.2	39.8	**52.1**
One-stage methods							
YOLOv2 [15]	DarkNet-19 [15]	21.6	44.0	19.2	5.0	22.4	35.5
SSD513 [11, 3]	ResNet-101-SSD	31.2	50.4	33.3	10.2	34.5	49.8
DSSD513 [3]	ResNet-101-DSSD	33.2	53.3	35.2	13.0	35.4	51.1
RetinaNet [9]	ResNet-101-FPN	39.1	59.1	42.3	21.8	42.7	50.2
RetinaNet [9]	ResNeXt-101-FPN	**40.8**	**61.1**	**44.1**	**24.1**	**44.2**	51.2
YOLOv3 608 × 608	Darknet-53	33.0	57.9	34.4	18.3	35.4	41.9

La mAP@0.5 di 0.44 è compatibile con le nostre che vanno la 0.5 a 0.59.

Coco challenge

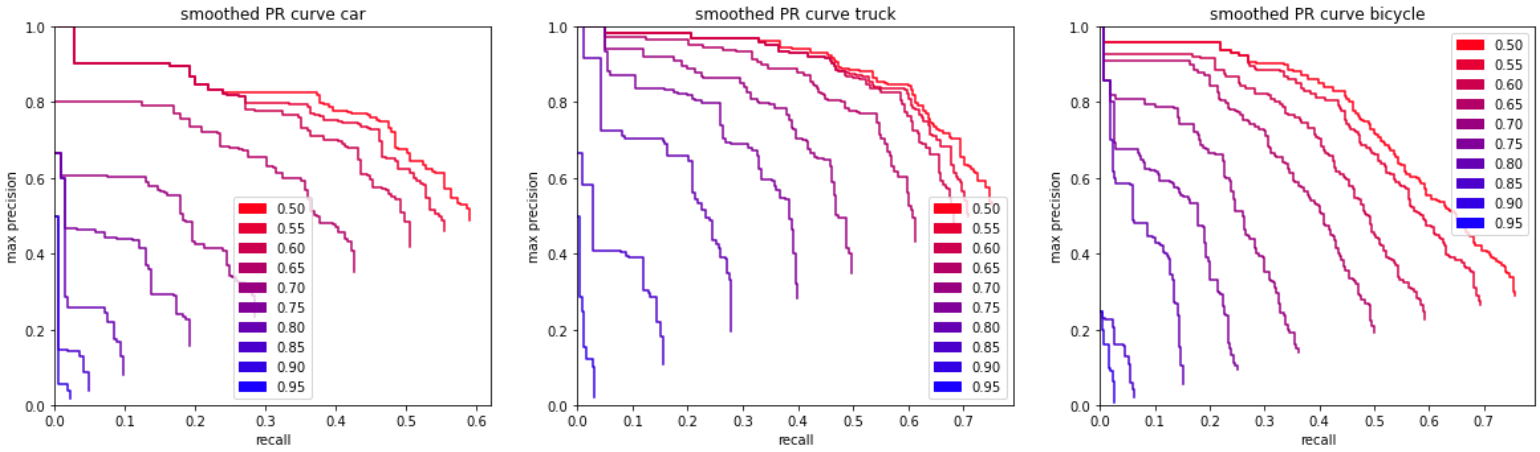
Per la COCO challenge si fissa lo score threshold e l'iou threshold usato per non max suppression ai valori di 0.5 e 0.45, mentre si effettua il calcolo della mean average precision per vari iou threshold da 0.5 a 0.95 con salti di 0.05. Infine si fa la media di questi risultati e si ottiene un valore, che è la mean average precision usata nella coco challenge. Per la coco challenge si calcola il valore delle average precision usando un interpolazione di 101 punti, mentre la tecnica qui usata è la semplice area under curve. Questi due metodi possono portare ad alcune piccole variazioni nel calcolo della mAP.

iou threshold ap	mAP	AP car	AP truck	AP bicycle
0.50	0.575	0.472	0.669	0.584
0.55	0.537	0.440	0.636	0.535
0.60	0.482	0.398	0.613	0.437
0.65	0.394	0.295	0.534	0.354
0.70	0.260	0.149	0.409	0.223

0.75	0.175	0.078	0.307	0.135
0.80	0.096	0.029	0.184	0.075
0.85	0.027	0.008	0.063	0.011
0.90	0.005	0.003	0.006	0.004
0.95	0.000	0.000	0.000	0.000

La mean average precision finale, calcolata come la media delle mean average precision per tutti i 10 IoU threshold è 0.255.

Per comprendere questi risultati è anche utile vedere quello che accade alla precision-recall curve al variare degli iou threshold. Se imponiamo iou threshold bassi abbiamo valori sia di recall che di precision più alti, stiamo infatti rilassando le condizioni per riconoscere come vera positiva una predizione (a differenza di quando visto invece nei risultati precedenti in cui cambiando threshold incorrevamo in un trade off tra precision e recall). Con threshold più alti invece la curva assume sia valori di precision che di recall più bassi.



Allo stesso modo confrontiamo i risultati ottenuti con quelli riportati nel sopra. Sebbene i dataset siano diversi e il calcolo delle average precision differisca (portando ad alcune variazioni nella mAP), notiamo che i valori ottenuti sono compatibili. Siamo interessati alle colonne 0.5:0.95, 0.5, 0.75 e la riga YOLOv2, i dati sono presi dalla pubblicazione scientifica di YOLOv2 [6].

		0.5:0.95	0.5	0.75	S	M	L	1	10	100	S	M	L
Fast R-CNN [5]	train	19.7	35.9	-	-	-	-	-	-	-	-	-	-
Fast R-CNN[1]	train	20.5	39.9	19.4	4.1	20.0	35.8	21.3	29.5	30.1	7.3	32.1	52.0
Faster R-CNN[15]	trainval	21.9	42.7	-	-	-	-	-	-	-	-	-	-
ION [1]	train	23.6	43.2	23.6	6.4	24.1	38.3	23.2	32.7	33.5	10.1	37.7	53.6
Faster R-CNN[10]	trainval	24.2	45.3	23.5	7.7	26.4	37.1	23.8	34.0	34.6	12.0	38.5	54.4
SSD300 [11]	trainval35k	23.2	41.2	23.4	5.3	23.2	39.6	22.5	33.2	35.3	9.6	37.6	56.5
SSD512 [11]	trainval35k	**26.8**	**46.5**	**27.8**	**9.0**	**28.9**	**41.9**	**24.8**	**37.5**	**39.8**	**14.0**	**43.5**	**59.0**
YOLOv2 [11]	trainval35k	21.6	44.0	19.2	5.0	22.4	35.5	20.7	31.6	33.3	9.8	36.5	54.4

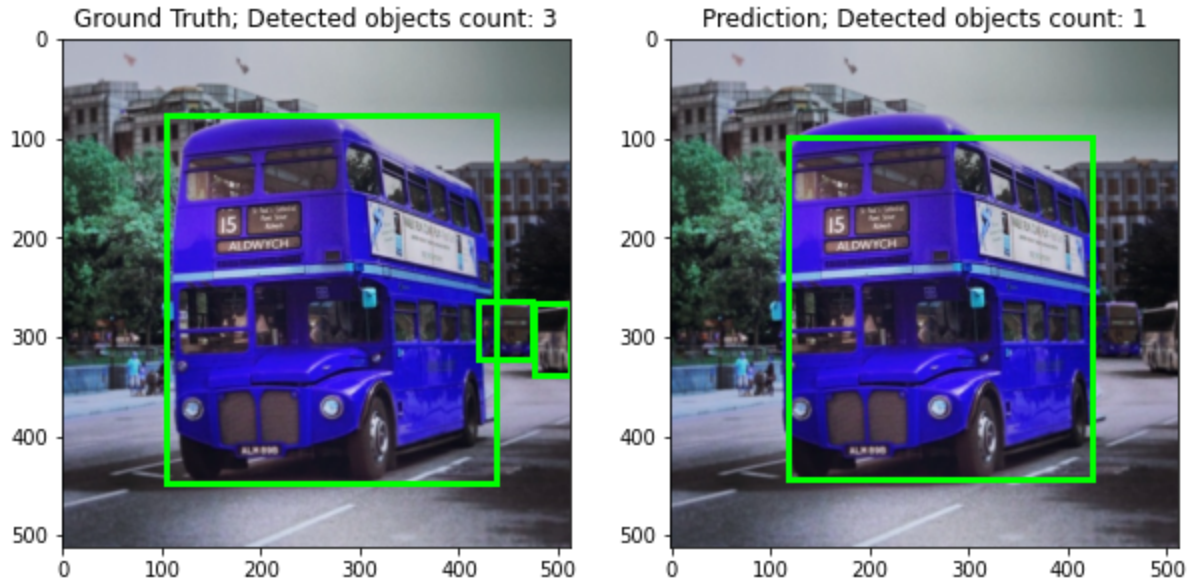
I valori ottenuti sono mAP 0.255, simile al mAP di 0.216, mentre per mAP@0.5 abbiamo 0.575 simile al 0.44 e per mAP@0.75 abbiamo 0.173 simile al 0.192.

Classificazioni

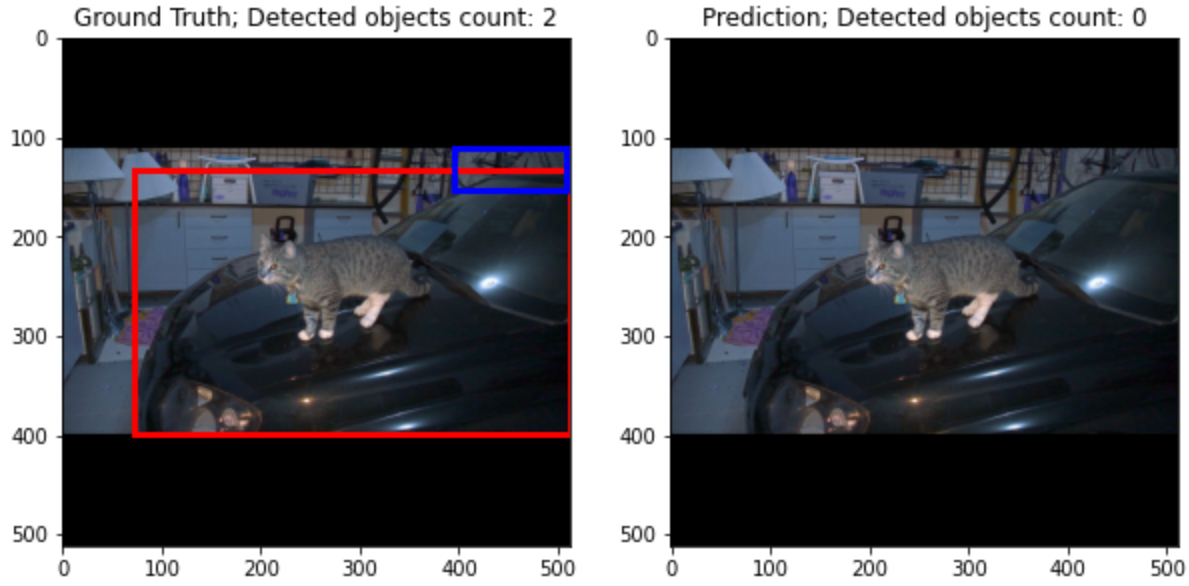
Nelle classificazioni ritroviamo il trade off visto precedentemente. Threshold più alti effettuano predizioni più precise ma a scapito di una recall. Threshold bassi invece hanno predizioni meno precise e recall più alta. Molto spesso la precisione diminuisce a causa di predizioni multiple di uno stesso oggetto.

Classificazioni per score threshold di 0.6 e iou threshold di 0.25

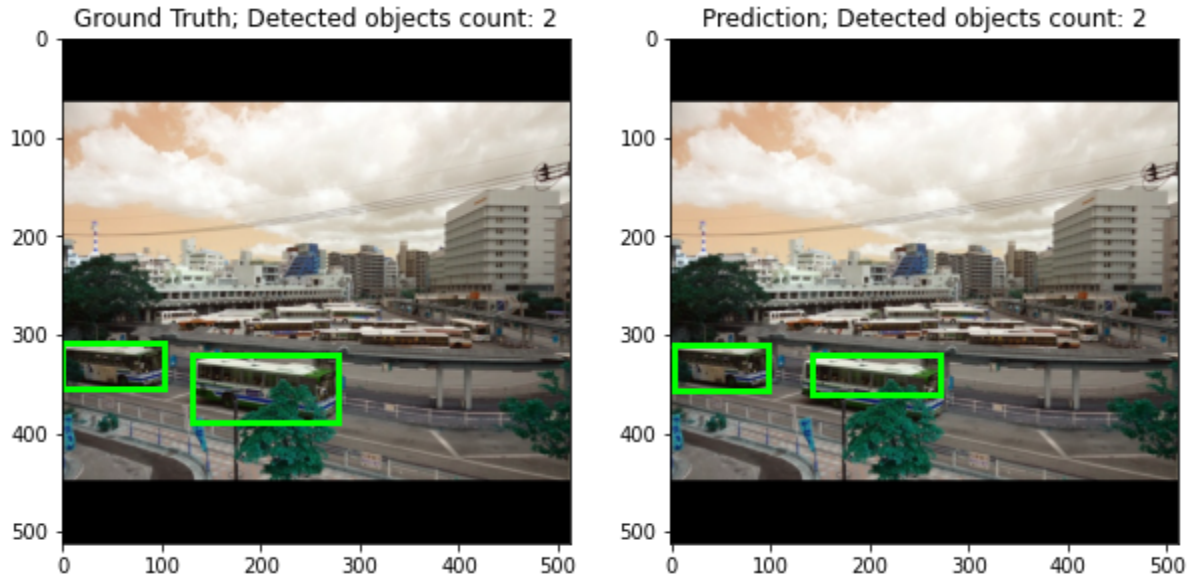
Con questi threshold notiamo come bounding box più piccole e più difficili a volte non vengono identificate

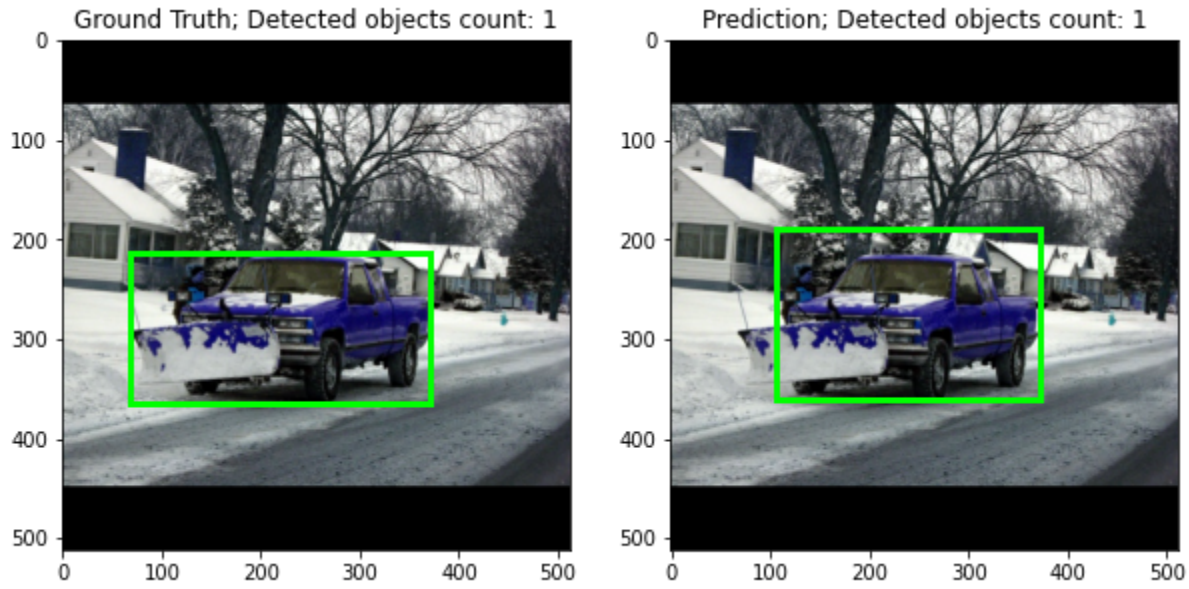


Allo stesso modo immagini difficilmente generalizzabili (ad esempio immagini troncate di parti di automobili) non vengono identificate



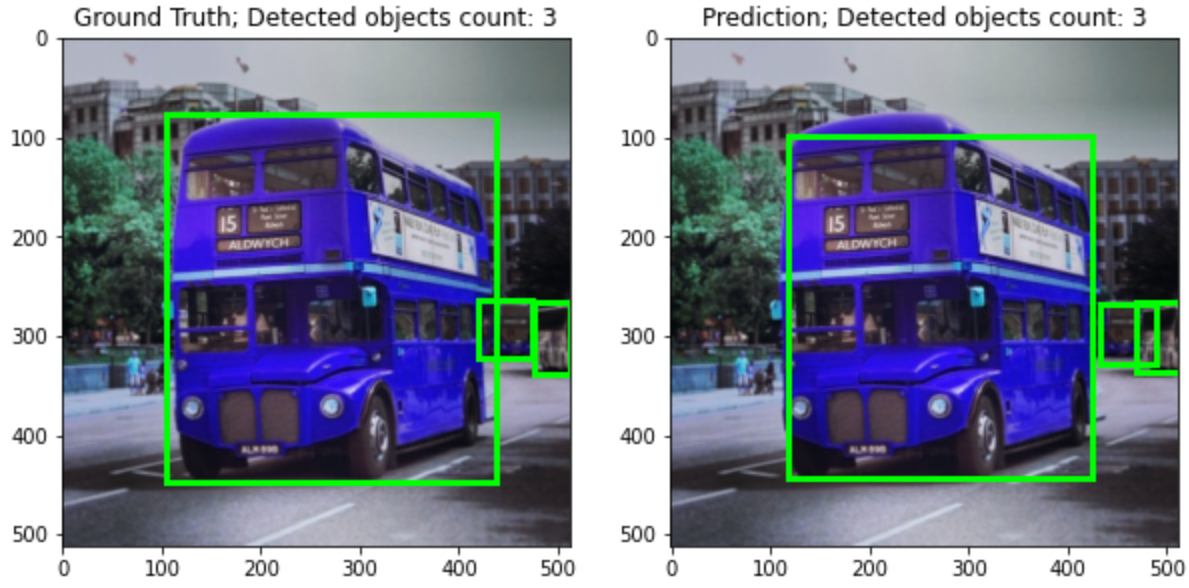
Tuttavia negli altri casi il modello è affidabile, inoltre raramente effettua predizioni multiple e questo rende le classificazioni più chiare



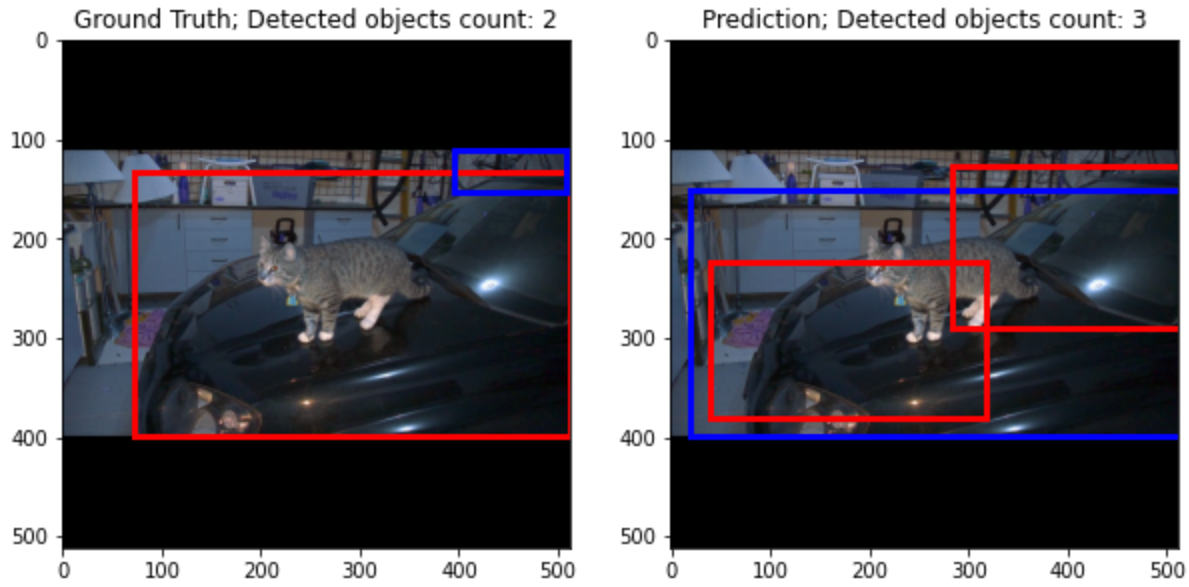


Classificazioni per score threshold di 0.5 e iou threshold di 0.45

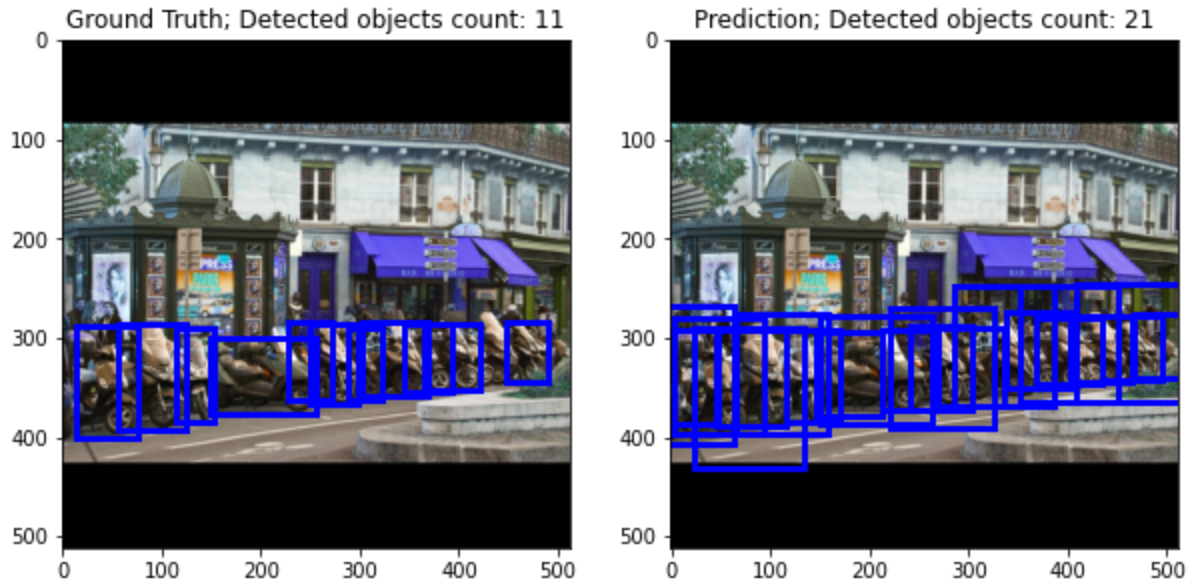
Con threshold più bassi invece abbiamo un vantaggio in una maggiore identificazione di bounding box meno certe



Abbiamo anche diverse predizioni di parti dell'automobile, sebbene non unificate



Un drawback è ovviamente la presenza di classificazioni multiple, come in questo caso in cui la presenza di molte motociclette genera numerosissime predizioni.



CONCLUSIONS

È stata addestrata la rete YOLOv2 su un dataset costruito a partire da immagini del dataset COCO. Sono state effettuate diverse misurazioni della performarce attraverso le metriche di precision e recall durante il training. Infine è stato implementato un

algoritmo per il calcolo della mean average precision, ottenendo risultati in linea con quelli presentati nelle pubblicazioni scientifiche di YOLO. Sono state anche disegnate precision-recall curve, che hanno permesso di comprendere le differenze e i trade off che si ottengono cambiando i threshold per filtraggio e non max suppression delle bounding box.

REFERENCES

1] Il dataset originale COCO
https://cocodataset.org/#home

2] Articoli sulle metriche usate
https://towardsdatascience.com/what-is-mean-average-precision-map-in-object-detection-8f893b48afd3

https://medium.com/@jonathan_hui/map-mean-average-precision-for-object-detection-45c121a31173

https://medium.com/@timothycarlen/understanding-the-map-evaluation-metric-for-object-detection-a07fe6962cf3

3] Articoli su YOLO
https://medium.com/@amrokamal_47691/yolo-yolov2-and-yolov3-all-you-want-to-know-7e3e92dc4899

https://medium.com/@jonathan_hui/real-time-object-detection-with-yolo-yolov2-28b1b93e2088

4] Come calcolare la mean average precision (mAP) in yolo

https://towardsdatascience.com/implementation-of-mean-average-precision-map-with-non-maximum-suppression-f9311eb92522

5] Articolo scientifico di YOLO
https://arxiv.org/pdf/1506.02640.pdf

6] Articoli scientifici di YOLOv2 e YOLOv3
https://arxiv.org/pdf/1612.08242.pdf

https://arxiv.org/pdf/1804.02767.pdf

7] Un'implementazione di YOLOv2

https://github.com/jmpap/YOLOV2-Tensorflow-2.0/

8] Libro Hands-On Machine Learning di Aurelien Geron

9] Il codice del progetto presentato
https://github.com/marvmarco/progetto-mqi