RELAZIONE PROGETTO DI METODI QUANTITATIVI PER L'INFORMATICA

Implementazione di Yolo sul dataset COCO 2017

Gioele Migno - 1795826

Data 25/08/2020

ABSTRACT

Il progetto consiste nell'addestramento di una rete YOLO V2 per eseguire object detection su tre diverse classi di oggetto: Bicycle, Car, e Person. L'obiettivo è quello di sviluppare la base per un sistema di guida assistita che segnali al conducente la presenza di tali elementi sulla strada.

PRELIMINARIES

Introduzione YOLOv2: Per eseguire object detection la rete suddivide l'immagine tramite una griglia composta da S x S celle, per ognuna di esse genera B boxes ed una sola distribuzione di probabilità per le classi. Successivamente seleziona le boxes migliori utilizzando **IOU** e **Score**, tale tecnica è chiamata Non-maximum Suppression.

Vediamo ora i parametri principali che andremo a modificare della rete YOLOv2, tra cui IOU e Score.

IOU: Intersection Over Union (IOU) permette di valutare la qualità di una predizione confrontando l'area della prediction con quella del ground truth (Area di intersezione / Area di unione). IOU assume valori compresi tra [0,1] dove 0 indica una prediction pessima, e 1 una prediction perfetta.

Score: (o confidenza) Tale grandezza indica il grado di confidenza che il modello ha rispetto ad una bounding box, lo score è calcolato tramite il prodotto tra IOU e la probabilità della classe scelta (il modello sceglie la classe con probabilità maggiore nella softmax).

YOLOv2 Loss Function: Nella loss function di YOLOv2 sono presenti quattro parametri che andremo successivamente a modificare. YOLOv2 utilizza come loss function la somma quadratica dell'errore tra la prediction e il ground truth, considerando tre diverse componenti:

- *Localization Loss*: Errore sulla posizione dell'oggetto
- *Classification Loss:* Errore sulla classificazione della classe
- *Confidence Loss:* Errore sulla confidenza del modello in riferimento alla predizione

In figura 1 sono riportate le tre diverse loss.

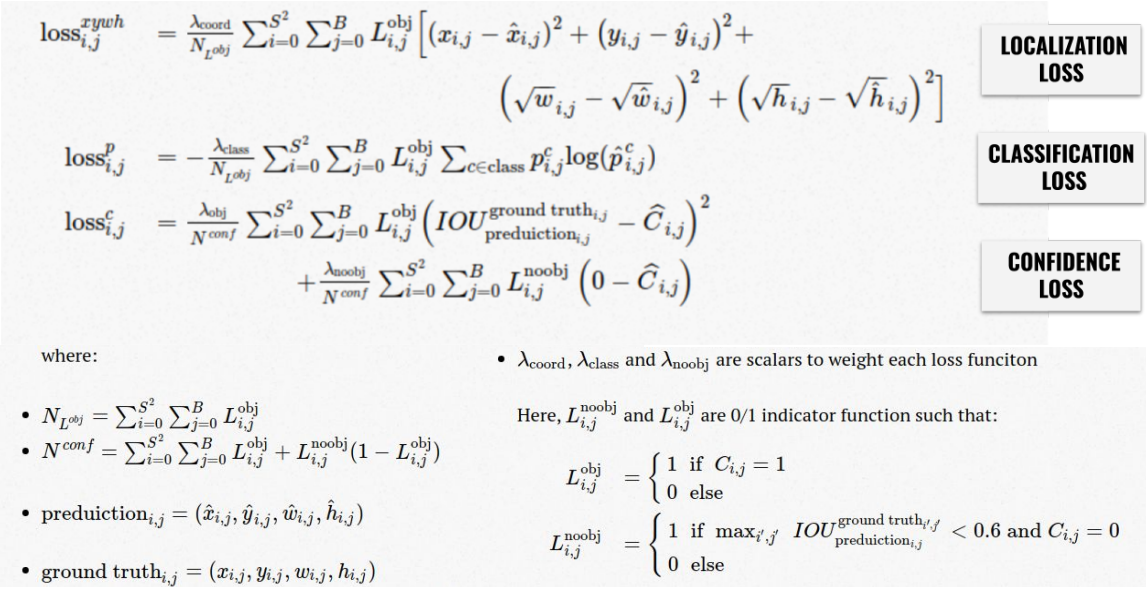


Figura 1: YOLOv2 Loss Function[1]

I quattro parametri di interesse sono: λ_{class} , λ_{coord}, λ_{obj}, λ_{noObj}

SELECTED DATASET

Il dataset scelto è COCO ristretto alle sole tre classi di interesse (bicycle, car, person). Come indicato dal nome COCO (Common Objects in COntext), tale dataset permette di eseguire object detection su oggetti presenti in contesti comuni (es strade), in una singola immagine possono quindi apparire contemporaneamente tutte e le tre le classi, in questo modo il modello viene addestrato con immagini simili a quelle di un contesto reale.

Il dataset è costruito a partire dal training e validation set COCO2017 uniti in un unico set in modo da eseguire successivamente una suddivisione bilanciata delle tre classi in tre subset (training, validation, e test set).

Tramite la COCO API[2] è stato possibile estrapolare 71211 immagini contenenti il seguente numero di oggetti: car: 45799, person: 273469, bicycle: 7429. A partire da queste sono state selezionate 2696 immagini contenenti: car: 7841, person: 7467, bicycle: 3757, non è stato possibile ottenere un migliore equilibrio per la classe bicycle poiché la maggior parte delle immagini contenti person contengono anche bicycle.

[1] Source: https://fairyonice.github.io/Part_4_Object_Detection_with_Yolo_using_VOC_2012_data_loss.html
[2] https://github.com/cocodataset/cocoapi

Il dataset è stato quindi suddiviso in training (80%), validation (15%) e test (5%) set, in figura 2 sono mostrati il numero di oggetti (boxes) presenti in ciascun subset.

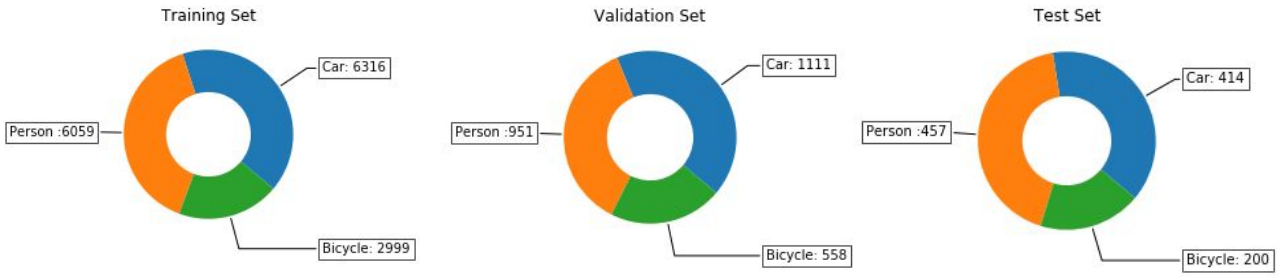


Figura 2: Boxes presenti in ogni subset

METHOD

Il modello si basa su un'implementazione di YOLOv2 in jupyter notebook (Github Repository: https://github.com/jmpap/YOLOV2-Tensorflow-2.0). Il codice include image augmentation (rotazione e variazione colori/luminosità).

L'utilizzo di un'implementazione tramite jupyter notebook rispetto ad una tramite Darknet permette di sfruttare facilmente risorse cloud come Google Colab, il codice è stato quindi modificato in modo da poter avere accesso a Google Drive.

Il notebook permette inoltre di modificare facilmente i livelli di input della rete in modo da variare le dimensioni dell'immagine in ingresso, il ratio deve essere unitario ed il valore iniziale è impostato a 512x512, la rete originale YOLOv2 utilizza invece in ingresso un'immagine 416x416. Come vedremo successivamente, la dimensione dell'immagine in ingresso influenza il risultato finale.

IMPLEMENTATION

La COCO API permette di ricavare facilmente le annotazioni delle singole immagini selezionando le sole classi di interesse. Le annotazioni sono state esportate nel seguente formato con coordinate normalizzate rispetto alla dimensione dell'immagine: (Formato utilizzato dal dataset Open Image)

<class> <Xmin> <Xmax> <Ymin> <Ymax>

(X_abs = Xmin * IMAGE_W, Y_abs = Ymin * IMAGE_W)

Le coordinate normalizzate consentono di poter utilizzare le medesime annotazioni anche su immagini ridimensionate aventi lo stesso ratio (IMAGE_W / IMAGE_H). Il notebook è stato quindi modificato in modo da utilizzare il precedente formato invece di quello originale (PASCAL VOC).

Il dataset COCO contiene immagini di dimensione e ratio variabile, è stato quindi necessario l'utilizzo di un zero padding centrato, poiché YOLO ha come input un'immagine con dimensione fissa e ratio unitario.

EXPERIMENTS / EXPERIMENTAL SETUP / EXPERIMENT PROCEDURES

Le diverse sessioni di training sono state eseguite su Google Colab con accelerazione hardware GPU, la piattaforma mette a disposizione diversi modelli di GPU: Tesla P100, Tesla K80, e Tesla T4. Tuttavia l'utente non può selezionare un determinato modello, ma l'assegnazione è eseguita in modo automatico, in alcuni casi quindi il tempo di training può variare a causa del diverso modello di GPU assegnato.

I parametri di interesse nel codice sono riportati in figura 3.

PARAMETRO	VALORE DEFAULT	PARAMETRO	VALORE DEFAULT
IMAGE_SIZE	512	LAMBDA_NOOBJECT	1
EPOCHS	800	LAMBDA_OBJECT	5
TRAIN_BATCH_SIZE	10	LAMBDA_CLASS	1
VAL_BATCH_SIZE	10	LAMBDA_COORD	1
SCORE_THRESHOLD	0.50	SCORE_TEST	SCORE_THRESHOLD
IOU_THRESHOLD	0.45	IOU_TEST	IOU_THRESHOLD

Figura 3: Parametri di training

I parametri `SCORE_TEST` e `IOU_TEST` indicano i valori utilizzati dal modello per le prediction sul test set. Il valore di default dei parametri si riferisce al setup iniziale della rete.

Per la valutazione del modello sono state utilizzate le metriche: PRECISION, RECALL, e mAP (Mean Average Precision), quest'ultima corrisponde alla media delle aree sottese ai grafici di precision e recall delle singole classi. Il codice utilizzato per il test è quello presente nella repository Github: https://github.com/Cartucho/mAP e si basa sull'adattamento in Python dei tool scritti in Matlab per la valutazione delle performance di object detection per la PASCAL VOC 2012 competition.

La procedura per gli esperimenti si basa sul partire dalla configurazioni di default (figura 3) e migliorare le performance variando gradualmente diversi parametri. Considerando il precision-recall tradeoff, siamo interessati maggiormente ad incrementare la recall, poiché in un sistema di assistenza alla guida è preferibile avere false positive piuttosto che false negative (oggetto presente ma non rilevato).

Il primo set di esperimenti (ESP_1) ha come obiettivo quello di osservare la variazione di performance andando a variare iou e score utilizzati durante il test (`SCORE_TEST` e `IOU_TEST`). Le prestazioni migliori si ottengono utilizzando (`SCORE_TEST=0.5 IOU_TEST=0.5`) e (`SCORE_TEST=0.5 e IOU_TEST=0.6`). Tenendo in considerazione tali risultati è stato eseguito un training con (`SCORE=0.5, IOU=0.55, SCORE_TEST=SCORE, IOU_TEST=IOU`), tuttavia tale variazione ha portato al peggioramento di circa 2% di recall, 2% di precision e 4% di mAP, si è quindi ripristinato tali grandezze ai valori iniziali.

Il secondo esperimento (ESP_2) ha come obiettivo quello di aumentare il numero di MATCH, ovvero oggetti rilevati con un sufficiente IOU rispetto al ground truth. Per farlo è stato necessario variare il parametro della localization loss (`LAMBDA_COORD = 2`). Tale variazione ha portato ad un aumento di 15 MATCH e di conseguenza ad un incremento di circa 2% di recall e 1% di mAP ed alla riduzione della precision del 2%.

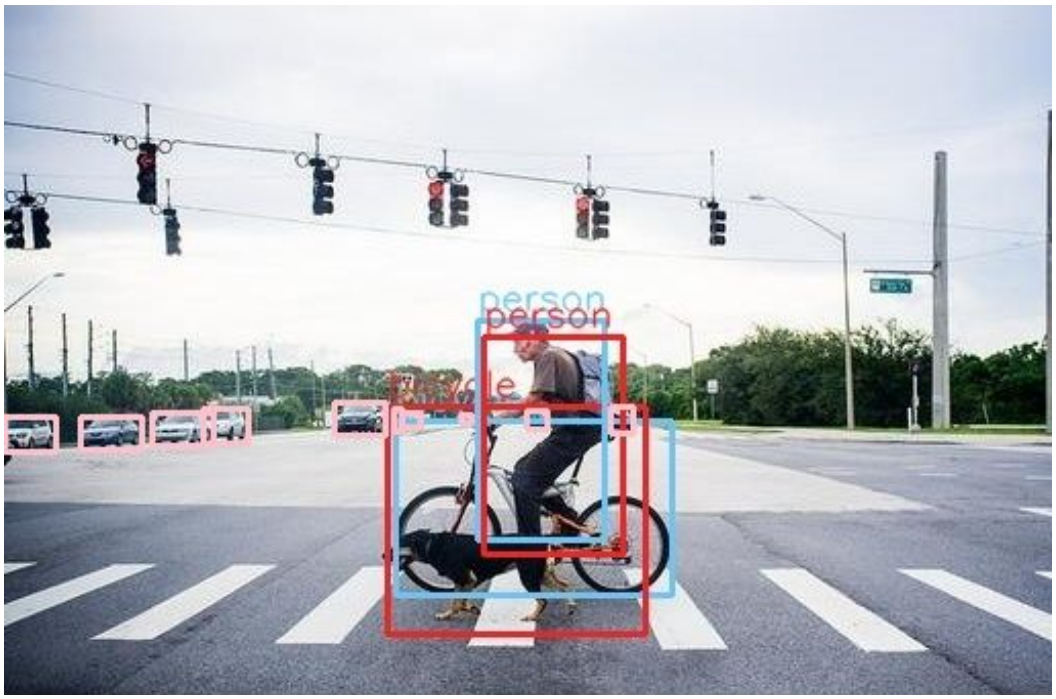
Il terzo set di esperimenti (ESP_3) è stato effettuato per osservare la variazione di performance al variare della dimensione delle immagini in ingresso, mantenendo invariati tutti gli altri parametri eccetto la dimensione dei batch ridotta a 8 a causa dell'esaurimento della RAM della GPU[3]. Si è ottenuto con dimensione 640x640: (+10% recall, +6% mAP e -3% precision), e con 800x800: (+19% recall, +8% mAP, e -11% precision). Tale miglioramento è dovuto principalmente all'ingrandimento degli oggetti più piccoli che a bassa risoluzione 512x512 non sono rilevabili dalla rete. Le figure 4, 5, 6 mostrano un esempio di oggetti non rilevati utilizzando immagini 512x512 (a) e rilevati invece utilizzando immagini 800x800 (b). In ciascuna immagine, la prima figura (a) rappresenta l'output del tool di test in cui le bounding box rosse sono le prediction del modello, quelle rosa gli oggetti non rilevati dalla rete e quelle blu gli oggetti correttamente rilevati dalla rete. La seconda figura (b) invece rappresenta l'output del modello, ovvero le sole prediction, i colori delle box indicano la classe e in ciascuna di esse è indicato lo SCORE (s).

[3] Durante il training con batch size = 8 risultano occupati circa 15GB della GPU.

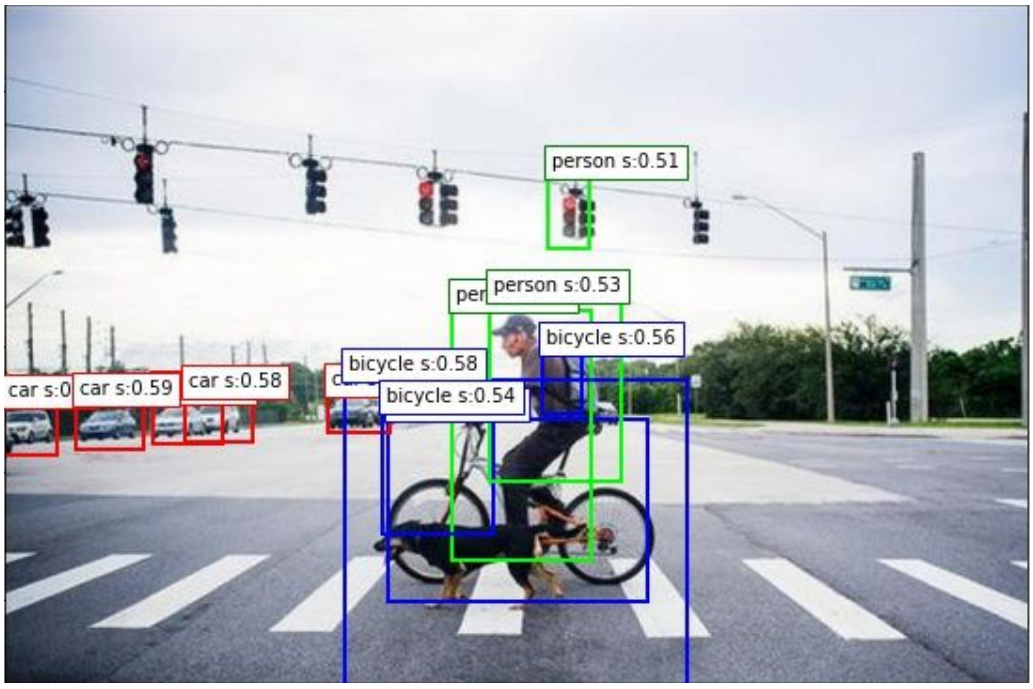
(a)

(b)

Figura 4: Confronto immagini 512x512 (a), 800x800 (b)



(a)



(b)

Figura 5: Confronto immagini 512x512 (a), 800x800 (b)

(a)

(b)

Figura 6: Confronto immagini 512x512 (a), 800x800 (b)

L'ultimo esperimento (ESP_4) consiste nell'eseguire nuovamente un test simile a ESP_1 utilizzando l'ultimo modello descritto (input immagini 800x800), in modo da ricavare il valore migliore per `SCORE_TEST` e `IOU_TEST`. La variazione di tali parametri consente di aumentare la recall fino al 2%, tuttavia ciò comporta la riduzione del 6% di mAP e dell'8% della precision, si è quindi scelto di lasciare invariati tali parametri.

I parametri del modello finale sono mostrati in figura 7.

PARAMETRO	VALORE DEFAULT	PARAMETRO	VALORE DEFAULT
IMAGE_SIZE	800	LAMBDA_NOOBJECT	1
EPOCHS	800	LAMBDA_OBJECT	5
TRAIN_BATCH_SIZE	8	LAMBDA_CLASS	1
VAL_BATCH_SIZE	8	LAMBDA_COORD	2
SCORE_THRESHOLD	0.50	SCORE_TEST	SCORE_THRESHOLD
IOU_THRESHOLD	0.45	IOU_TEST	IOU_THRESHOLD

Figura 7: Parametri modello finale. Le variazioni rispetto al modello di partenza sono evidenziate in giallo

RESULTS

Nel seguito sono riportati i risultati dei singoli esperimenti.

ESP_1: Variazione SCORE_TEST, IOU_TEST

	SIZE_IMAGE	EPOCHS	L_NO_OBJ	L_OBJ	L_CLASS	L_COORD	SCORE_IOU		TEST_SCORE	TEST_IOU	PRECISION	RECALL	AP
MODELLO INZIALE	512	800	1	5	1	1	0,5	0,45	0,5	0,45	0,4133	0,3367	0,2617
	512	800	1	5	1	1	0,5	0,45	0,55	0,45	0,51	0,3167	0,2512
	512	800	1	5	1	1	0,5	0,45	0,6	0,45	0,6133	0,29	0,234
	512	800	1	5	1	1	0,5	0,45	0,65	0,45	0,7233	0,24	0,2031
	512	800	1	5	1	1	0,5	0,45	0,7	0,45	0,7933	0,1633	0,1475
	512	800	1	5	1	1	0,5	0,45	0,5	0,3	0,5	0,3167	0,253
	512	800	1	5	1	1	0,5	0,45	0,5	0,4	0,44	0,33	0,261
	512	800	1	5	1	1	0,5	0,45	0,5	0,5	0,38	0,35	0,2605
	512	800	1	5	1	1	0,5	0,45	0,5	0,6	0,32	0,3633	0,3511
RISULTATI MIGLIORI	512	800	1	5	1	1	0,5	0,45	0,5	0,7	0,2633	0,3733	0,2108
MODELLO MODIFICATO	512	800	1	5	1	1	0,5	0,55	0,5	0,55	0,3933	0,31	0,2264

ESP_2: L_COORD=2

	SIZE_IMAGE	EPOCHS	L_NO_OBJ	L_OBJ	L_CLASS	L_COORD	SCORE_IOU		TEST_SCORE	TEST_IOU	PRECISION	RECALL	AP
MODELLO INIZIALE	512	800	1	5	1	1	0,5	0,45	0,5	0,45	0,4133	0,3367	0,2617
MODELLO MODIFICATO	512	800	1	5	1	2	0,5	0,45	0,5	0,45	0,3933	0,3567	0,278

ESP_3: Variazione dimensione immagine input

	SIZE_IMAGE	EPOCHS	L_NO_OBJ	L_OBJ	L_CLASS	L_COORD	SCORE_IOU		TEST_SCORE	TEST_IOU	PRECISION	RECALL	AP
MODELLO INIZIALE	512	800	1	5	1	2	0,5	0,45	0,5	0,45	0,3933	0,3567	0,278
	640	800	1	5	1	2	0,5	0,45	0,5	0,45	0,3633	0,46	0,3436
MODELLO MODIFICATO	800	800	1	5	1	2	0,5	0,45	0,5	0,45	0,28	0,55	0,36

ESP_4: Variazione SCORE_TEST, IOU_TEST modello finale

	SIZE_IMAGE	EPOCHS	L_NO_OBJ	L_OBJ	L_CLASS	L_COORD	SCORE	IOU	TEST_SCORE	TEST_IOU	PRECISION	RECALL	AP
MODELLO INIZIALE	800	800	1	5	1	2	0,5	0,45	0,5	0,45	0,28	0,55	0,36
	800	800	1	5	1	2	0,5	0,45	0,55	0,3	0,43	0,4467	0,3245
	800	800	1	5	1	2	0,5	0,45	0,55	0,35	0,4167	0,4734	0,3396
	800	800	1	5	1	2	0,5	0,45	0,55	0,4	0,38	0,4867	0,3438
	800	800	1	5	1	2	0,5	0,45	0,55	0,45	0,353	0,51	0,346
	800	800	1	5	1	2	0,5	0,45	0,6	0,45	0,46	0,46	0,3256
	800	800	1	5	1	2	0,5	0,45	0,65	0,45	0,56	0,39	0,2879
	800	800	1	5	1	2	0,5	0,45	0,7	0,45	0,647	0,3267	0,2478
	800	800	1	5	1	2	0,5	0,45	0,5	0,5	0,2533	0,5567	0,3436
	800	800	1	5	1	2	0,5	0,45	0,5	0,55	0,23	0,5667	0,3301
RISULTATI MIGLIORI	800	800	1	5	1	2	0,5	0,45	0,5	0,6	0,2033	0,57	0,3075

RISULTATI MODELLO FINALE

Il modello finale è stato addestrato, partendo dai pesi di YOLOv2, tramite un training di 800 epoche, il tempo richiesto tramite GPU è stato di 11658,262 secondi (circa 3h 20min), il valore minimo della validation loss (0.424) è stato raggiunto all'epoca 781. In figura 8 è mostrato l'andamento della training loss e della validation loss durante il training.

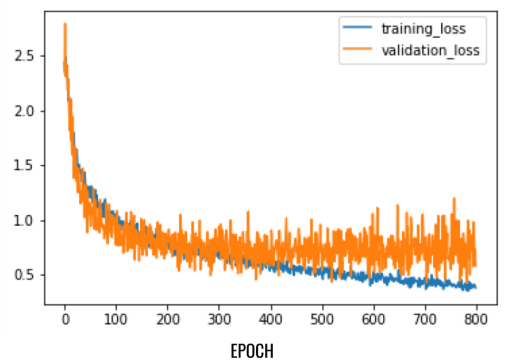


Figura 8: Andamento loss function durante il training

Osservando l'andamento della validation loss notiamo una forte oscillazione nelle epoche finali, la causa principale, in riferimento alla loss function descritta nell'equazione (1), è l'instabilità della componente CLASSIFICATION_LOSS, per ridurre tale oscillazione si è tentato di ridurre `LAMBDA_CLASS` impostandola a 0.5, tuttavia ciò non ha portato ad un miglioramento. Si ha comunque una relativa stabilità della training loss.

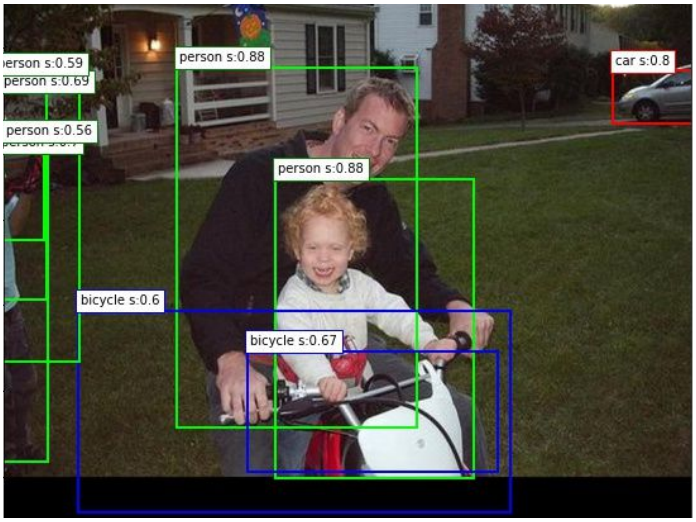
ESEMPIO OBJECT DETECTION SUL TEST SET

In figura 9 sono mostrati degli esempi di corretto object detection per ciascuna classe.

In figura 10 invece oggetti erroneamente classificati.

In tale figura, il modello rileva correttamente l'automobile con confidenza del 70%, lo score non è molto elevato a causa della dimensione ridotta dell'oggetto. Notiamo infatti nell'immagine (b) una maggiore confidenza (80%).

(a)



In questo caso il modello rileva correttamente le due persone al centro dell'immagine con elevata confidenza (88%), sulla sinistra tuttavia non rileva in modo preciso una persona poiché non completamente visibile.

Notiamo inoltre la classificazione della moto come una bicicletta, ciò è dovuto al fatto che la rete non conosce questa tipologia di oggetto.

(b)

In questo caso la bicicletta al centro dell'immagine viene rilevata con alta precisione e confidenza (85%). Sulla destra tuttavia riconosce erroneamente il murales come un gruppo di biciclette e persone.

(c)

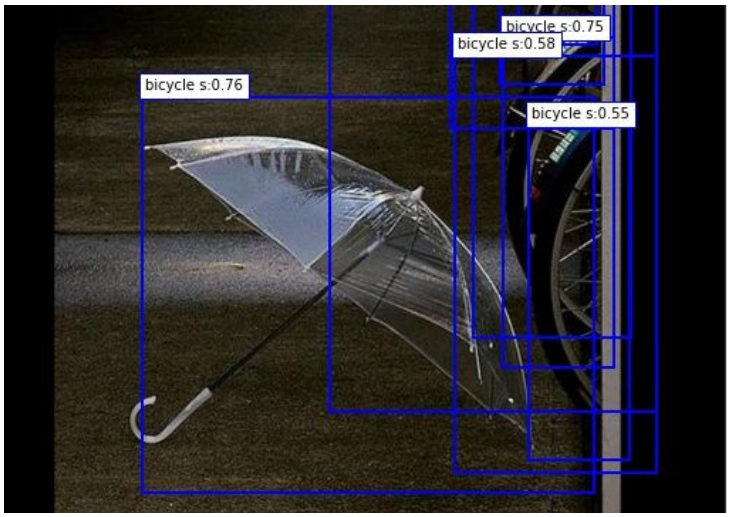
Figura 9: Esempio corretto object detection

Notiamo in questa figura la classificazione errata del cartello di STOP come un'automobile con una confidenza abbastanza alta (60%). Il veicolo sottostante viene tuttavia classificato correttamento con confidenza del 87%.

(a)

In questo caso il modello fa diverse classificazioni errate, la peggiore è riconoscere il motorino come una persona con confidenza del 61%

(b)



Oltre alla non precisa identificazione delle biciclette sul lato destro, causata dal fatto che non sono completamente visibili, il modello riconosce erroneamente un ombrello come una bicicletta.

(c)

Figura 10: Esempio errato object detection

In figura 11 è riportato un caso interessante in cui il modello si è comportato abbastanza bene nel riconoscere gli oggetti presenti nella strada, sono tuttavia presenti due falsi positivi: al centro dell'immagine identifica un idrante come una persona e in alto a sinistra una decorazione sempre come una persona.

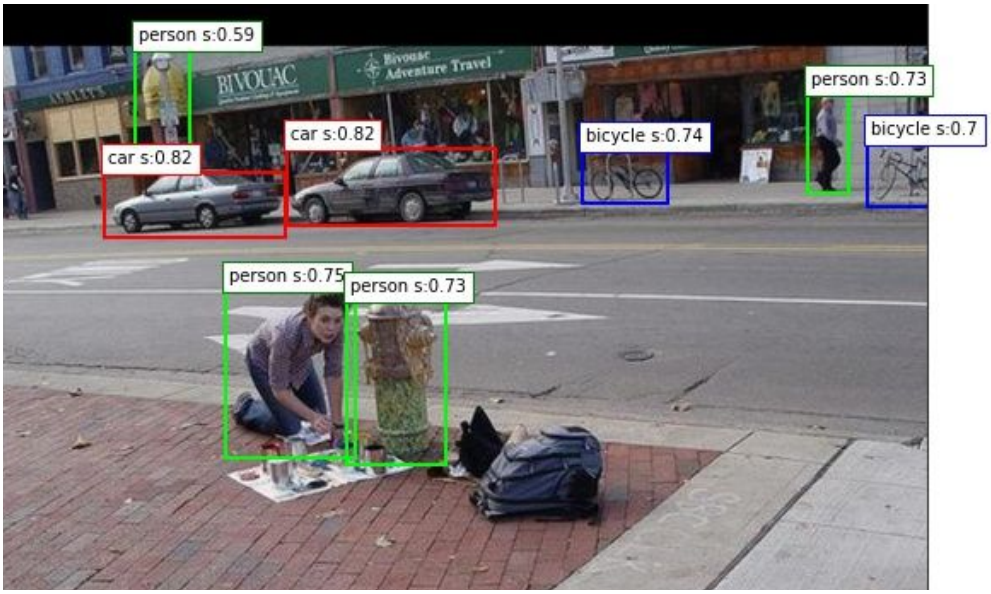
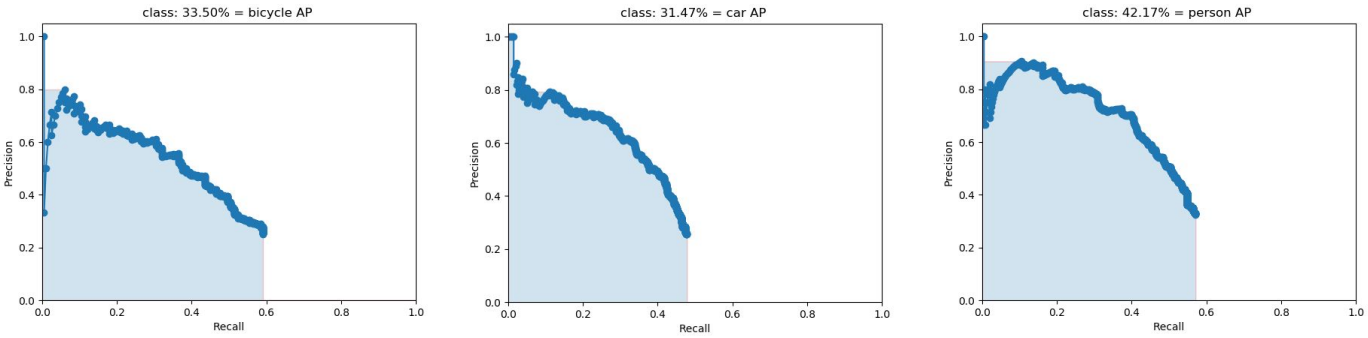


Figura 11: Esempio object detection con buoni risultati

RISULTATI PRECISION RECALL mAP

In figura 12 sono mostrati i grafici precision-recall di ciascuna classe.[4]



(a) (b) (c)

Figura 12: Grafici precision-recall per ciascuna classe. (a) bicycle, (b) car, (c) person

[4] Tool utilizzato: https://github.com/Cartucho/mAP

In figura 13 sono mostrati i confronti di precision, recall e AP tra il modello iniziale avente parametri mostrati in figura 3 e quello finale con parametri in figura 7. Notiamo un aumento medio di AP e RECALL ma anche una riduzione media della precision, come precedentemente detto tuttavia nel nostro caso siamo interessati maggiormente alle prime due metriche.

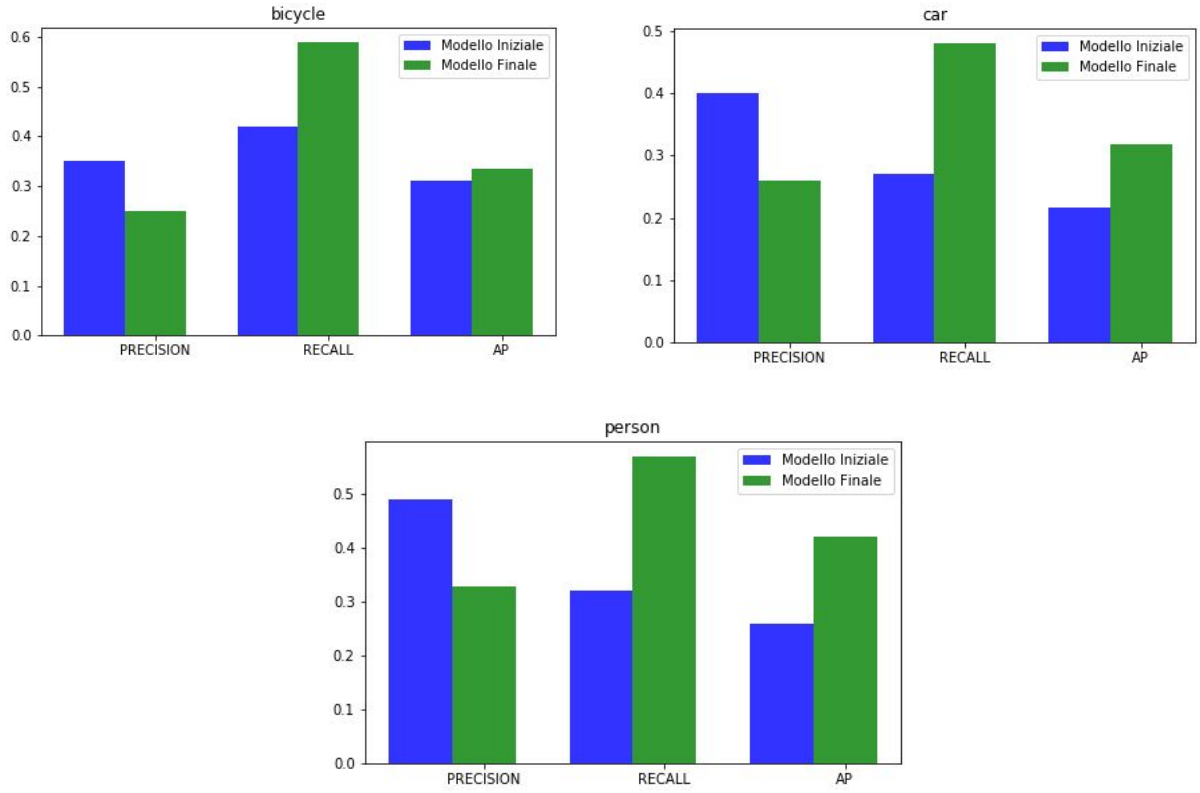


Figura 13:Confronto modello iniziale con quello finale

In figura 14 sono mostrati i valori delle metriche del modello finale per ciascuna classe.

CLASS	PRECISION	RECALL	AP
BICYCLE	0.25	0.59	0.335
CAR	0.26	0.48	0.3174
PERSON	0.33	0.57	0.4217

Figura 14: Risultati metriche modello finale

CONFRONTO RISULTATI

Dato che il dataset utilizzato è un subset di COCO2017 non è possibile effettuare un confronto accurato con risultati di terze parti, consideriamo tuttavia i risultati riportati sul sito di YOLO mostrati in figura 15.

Il nostro modello ottiene 35.8% mAP contro 48.1% riportato in figura utilizzando YOLOv2 608x608. Le prestazioni del modello sono quindi notevolmente inferiori tenendo in considerazione soprattutto il fatto che i risultati sul sito di YOLO si riferiscono all'intero dataset e quindi a tutte le classi disponibili, nel nostro caso invece ci siamo ridotti a sole tre classi.

Model	Train	Test	mAP
YOLOv2 608x608	COCO trainval	test-dev	48.1
Tiny YOLO	COCO trainval	test-dev	23.7
YOLOv3-320	COCO trainval	test-dev	51.5
YOLOv3-416	COCO trainval	test-dev	55.3
YOLOv3-608	COCO trainval	test-dev	57.9
YOLOv3-tiny	COCO trainval	test-dev	33.1

Figura 15: Risultati sito YOLO

CONCLUSIONS

Partendo da un implementazione di YOLOv2 su jupyter notebook, il modello è stato addestrato su un subset di COCO2017, ristretto alle classi bicycle, car e person, eseguendo diversi test in modo da migliorare il modello iniziale con l'obiettivo di aumentare soprattutto le metriche RECALL e mAP. Il miglioramento è stato ottenuto soprattutto grazie all'aumento della dimensione dell'immagine in ingresso portata da 512 a 800.

Considerando i risultati ottenuti, il modello necessita di ulteriori miglioramenti prima di poter essere impiegato come sistema di assistenza alla guida. I possibili miglioramenti sono: selezione accurata delle immagini del dataset, utilizzando soprattutto scenari stradali ed analisi del dataset al fine di ricavare le anchor box più frequenti.

REFERENCES

[1] YOLO WebSite:
https://pjreddie.com/darknet/yolo/

[2] YOLO, YOLOv2 and YOLOv3: All You want to know:
https://medium.com/@amrokamal_47691/yolo-yolov2-and-yolov3-all-you-want-to-know-7e3e92dc4899

[3] How YOLOv2 works in detail:
https://medium.com/@y1017c121y/how-does-yolov2-work-daaaa967c5f7

[4] Non-maximum Suppression (NMS):
https://towardsdatascience.com/non-maximum-suppression-nms-93ce178e177c

[5] YOLOv2 Loss Function:
https://fairyonice.github.io/Part_4_Object_Detection_with_Yolo_using_VOC_2012_data_loss.html

[6] COCO API:
https://github.com/cocodataset/cocoapi

[7] How to download specific classes from COCO dataset
https://github.com/cocodataset/cocoapi/issues/271

[8] Tool for mAP testing:
https://github.com/Cartucho/mAP#create-the-ground-truth-files