

# WALL-Y

*Relazione di progetto per l'esame di Intelligenza Artificiale*



**Anna Avena**

**Giulia Cantini**

**Matteo Del Vecchio**

**Simone Preite**

A.A. 2018/19

<b>OBIETTIVO DEL PROGETTO</b>	<b>2</b>
<b>OBJECT DETECTION</b>	<b>3</b>
<b>R-CNN</b>	<b>3</b>
Plain R-CNN	4
Fast R-CNN	4
Faster R-CNN	5
<b>DATASET</b>	<b>6</b>
COCO	6
WALDO	7
<b>PREPROCESSING</b>	<b>8</b>
Labeling	8
Data Augmentation	9
Cropping	9
<b>METRICHE</b>	<b>11</b>
Intersection over Union (IoU)	11
Precision	12
Recall	12
Recall-IoU Curve	13
Average Precision (AP) e Average Recall (AR)	14
<b>TRAINING</b>	<b>16</b>
<b>RISULTATI</b>	<b>19</b>
Inferenza	19
<b>CONCLUSIONI</b>	<b>20</b>

# OBIETTIVO DEL PROGETTO

“Where’s Wally?” (o Waldo in Nord-America) è una collezione di libri per bambini contenenti raccolte di immagini o *puzzle* molto popolate e create appositamente per rendere difficoltoso la localizzazione del personaggio principale, Wally, e di altri personaggi o oggetti.

In tutte le immagini vengono usati colori e pattern di personaggi predisposti per far sì che i protagonisti si mimetizzino tra la folla o che si confondano con l’ambiente.

È una sfida non indifferente anche per l’essere umano in sé perché non sempre questi personaggi compaiono allo stesso modo, infatti capita spesso che di Wally si veda solo la testa, che questa sia parzialmente coperta oppure delle altre in cui è presente tutto il corpo. In breve, queste immagini possono essere molto complicate.

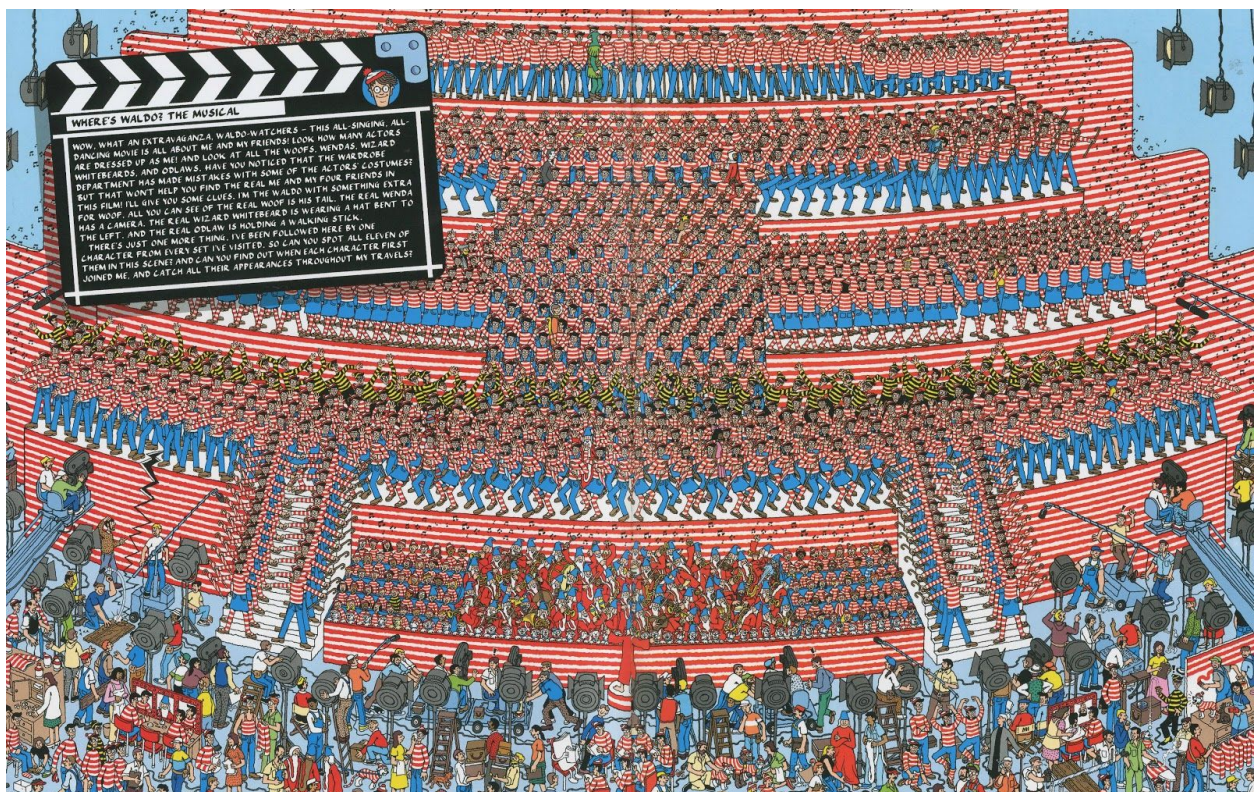


Figura: un esempio di scena in cui sono presenti Wally e tanti attori che lo impersonano.

Abbiamo deciso di affrontare il problema da un punto di vista di *object detection* su un'unica classe disponibile, “waldo” con l’ausilio di neural networks e della Tensorflow Object Detection API.<sup>1</sup>

<sup>1</sup> "models/research/object\_detection at master ...."

Considerando la difficoltà del task il nostro obiettivo era quello di creare un buon sistema di ritrovamento e che avesse un buon grado di accuratezza oltre ad essere relativamente veloce.

Per raggiungere questo scopo è stato svolto un lavoro di ricerca volto a individuare le tecnologie più adatte, le Faster R-CNN nel nostro caso, e le componenti di tali reti su cui intervenire per ottenere miglioramenti concreti. Inoltre sono state indagate le peculiarità dei dataset disponibili per arrivare infine alla creazione di un nostro dataset specifico per la ricerca di Wally.

## OBJECT DETECTION

*Object detection* è il problema di identificare oggetti appartenenti ad una certa classe all'interno di un'immagine e dare indicazioni su dove questi siano dislocati in essa. È un problema relativo al campo della *Computer Vision* che oggi giorno può essere risolto attraverso l'ausilio di diverse tipologie di reti neurali.

In particolare il problema si compone di due task che solitamente vengono divisi e analizzati separatamente da due diversi livelli fully connected, uno di *classificazione* che si occupa di etichettare gli oggetti ed uno di *localizzazione*, che attraverso regressione, traccia le cosiddette *bounding box* che identificano la posizione dell'oggetto all'interno dell'immagine.

Due famiglie di reti neurali vengono comunemente utilizzate in letteratura nella risoluzione del problema:

- Le R-CNN dalle quali derivano le FAST R-CNN e le FASTER R-CNN e che verranno spiegate più nel dettaglio nella sezione successiva in quanto per lo svolgimento del progetto in questione è stata usata una FASTER R-CNN RESNET 101.
- Le reti YOLO (*You Only Look Once*), più veloci delle R-CNN, che consentono di eseguire Object Detection in real-time.

## R-CNN

Per R-CNN, letteralmente “Regions with CNN”, si può intendere quasi una classe di reti neurali basate principalmente sull'uso di reti convoluzionali o fully connected. In particolare, i modelli più famosi sono sicuramente R-CNN, Fast R-CNN e Faster R-CNN, ognuno dei quali può essere visto come un miglioramento del precedente. Essi verranno descritti in poche parole per poi focalizzare l'attenzione sulla rete effettivamente utilizzata come base per questo progetto.

---

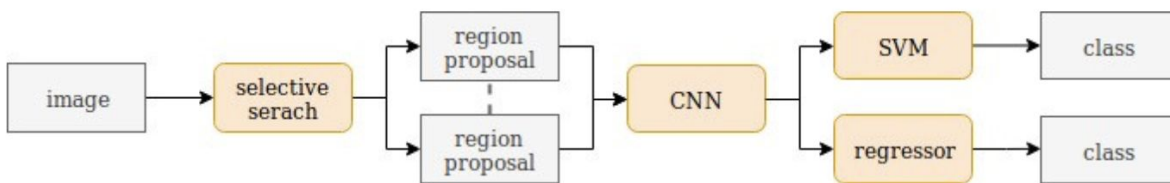
[https://github.com/tensorflow/models/tree/master/research/object\\_detection](https://github.com/tensorflow/models/tree/master/research/object_detection). Accessed 22 Feb. 2020.

## Plain R-CNN

La versione iniziale di questo tipo di reti neurali presuppone l'utilizzo di un algoritmo di *Selective-Search* che, data l'immagine in input, ha come obiettivo quello di generare *region proposals*, per l'individuazione ed il riconoscimento degli oggetti. L'idea di fondo dell'algoritmo consiste nell'analizzare i pixel dell'immagine ed unire quelli simili in una sola regione.

Una volta ottenute tutte le *region proposals*, esse vengono date in input ad una rete neurale convoluzionale per ottenere un vettore multidimensionale di features. Come ultimo passo, tale output viene utilizzato sia da un classificatore (SVM) per la predizione della classe dell'oggetto, che ad un regressore per ottenere le informazioni sul bounding box che circonda l'oggetto nell'immagine iniziale.

Tale approccio però, presenta due svantaggi molto pronunciati: l'algoritmo di *Selective-Search* per la generazione delle *region proposals* è molto lento; inoltre, ognuna delle regioni generate deve essere completamente elaborata dalla rete neurale convoluzionale.



Schema concettuale di una R-CNN

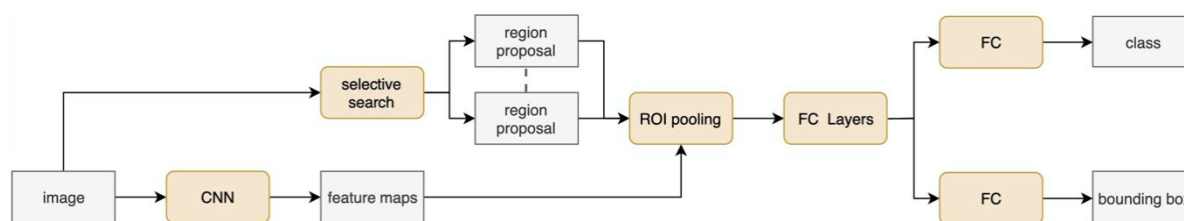
## Fast R-CNN

Anche in questo caso, le caratteristiche principali del modello sono molto simili alla Plain R-CNN: uso dell'algoritmo *Selective-Search* per la generazione delle *region proposals* e uso di reti neurali convoluzionali.

Ciò che effettivamente cambia è l'utilizzo di quest'ultima architettura: data l'intera immagine in input, la CNN restituisce una *feature map*. Il problema ora consiste nel mappare le regioni generate rispetto alla mappa di feature ottenuta. Entrambe queste informazioni vengono elaborate dal *ROI (Region of Interest) Pooling*, il quale sceglie le regioni più importanti direttamente dalla feature map. Tali regioni risultanti vengono quindi gestite da un *fully connected network* per preparare i dati ai task di classificazione e regressione.



Sebbene la situazione migliori rispetto alle Plain R-CNN, lo svantaggio principale di questo modello è comunque il tempo e le risorse necessarie all'algoritmo per generare le regioni.



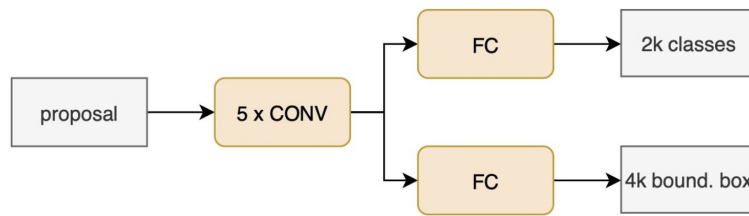
Schema concettuale di una Fast R-CNN

## Faster R-CNN<sup>2</sup>

Tale modello risulta essere lo stato dell'arte per quanto riguarda le reti neurali specializzate in object detection ma che separano il task di classificare un oggetto rispetto a quello di localizzarlo nell'immagine. Esso infatti, consiste in una Fast R-CNN la cui backbone è un *Region Proposal Network* (RPN). Come si può vedere dallo schema sottostante, l'algoritmo di Selective-Search è stato sostituito dal RPN, avendo come effetto principale quello di ridurre drasticamente il tempo di generazione delle regioni. Il RPN infatti, non lavora direttamente sull'immagine originale, bensì sulla *feature map* generata da una rete neurale convoluzionale. Come verrà descritto in seguito, la creazione delle *region proposals* ed il relativo training del RPN risultano essere abbastanza particolari, basandosi sul concetto di *anchor*.

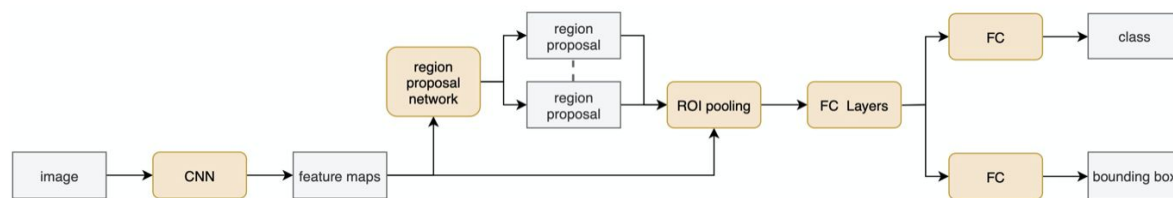
Un anchor è un bounding box fisso caratterizzato da un punto centrale e specifiche dimensioni. Tali dimensioni, insieme all'aspect ratio, sono anch'esse solitamente fissate, in modo tale da ottenere diversi bounding box relativamente allo stesso punto centrale. Strutturalmente, il RPN è un insieme di livelli convoluzionali con annessi dei livelli fully connected, al fine di classificare gli anchor e applicarci anche il task di regressione. Supponendo di avere in input  $k$  anchor, l'output consiste in  $2k$  *objectness score* e  $4k$  *regression values*: il primo indica la probabilità che quella anchor rappresenti un oggetto oppure lo sfondo, mentre il secondo è l'insieme dei valori di offset rispetto al bounding box vero e proprio.

<sup>2</sup> "Faster R-CNN: Towards Real-Time Object Detection with ...." 6 gen. 2016, <https://arxiv.org/abs/1506.01497>. Ultimo accesso: 22 feb. 2020.



Schema concettuale del Region Proposal Network (RPN)

Come step successivo, analogamente alla Fast R-CNN, il ROI Pooling estrapola le feature effettivamente utili e le elabora in modo da poterle utilizzare come input del classificatore e del regressore finali. Tale elaborazione consiste nel tagliare la feature map rispetto alle regioni di interesse ed applicare un max pooling su di esse. Infine, il classificatore restituisce  $N+1$  predizioni, una per ognuna delle  $N$  classi di oggetti più quella per il background, mentre il regressore ritorna  $4N$  predizioni, che rappresentano la tupla ( $x\ offset, y\ offset, width, height$ ) per ognuna delle  $N$  classi.



Schema concettuale di una Faster R-CNN

## DATASET

### COCO

Common Objects in Context (COCO)<sup>3</sup> è un dataset di grandi dimensioni ampiamente utilizzato per il riconoscimento visivo e l'identificazione di oggetti in un'immagine. COCO fornisce 91 categorie di oggetti comuni, ciascuna categoria con una media di 5.000 istanze etichettate. In totale il dataset conta 2.500.000 istanze su 328.000 immagini.

<sup>3</sup> "COCO dataset." <http://cocodataset.org/>. Accessed 23 Feb. 2020.

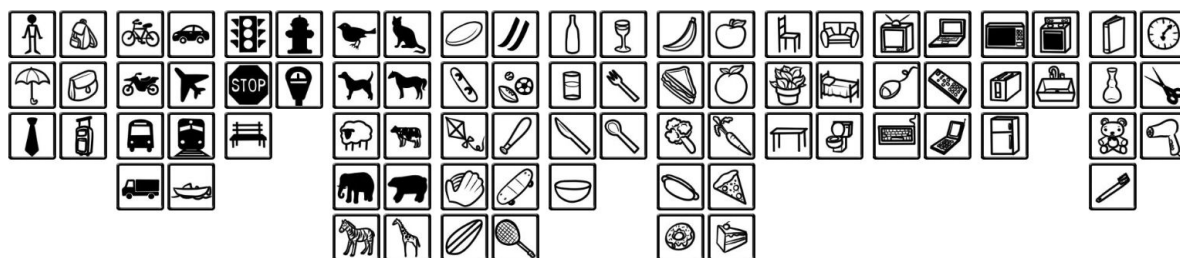


Figura: icone delle 91 categorie di oggetti presenti nel COCO dataset.

COCO è stato progettato per concentrare la ricerca sul rilevamento di oggetti mantenendo una particolare attenzione al contesto ed alla comprensione della scena, in particolare si focalizza sulla localizzazione di oggetti con una precisione a livello di pixel, rilevando oggetti anche in scene complesse.

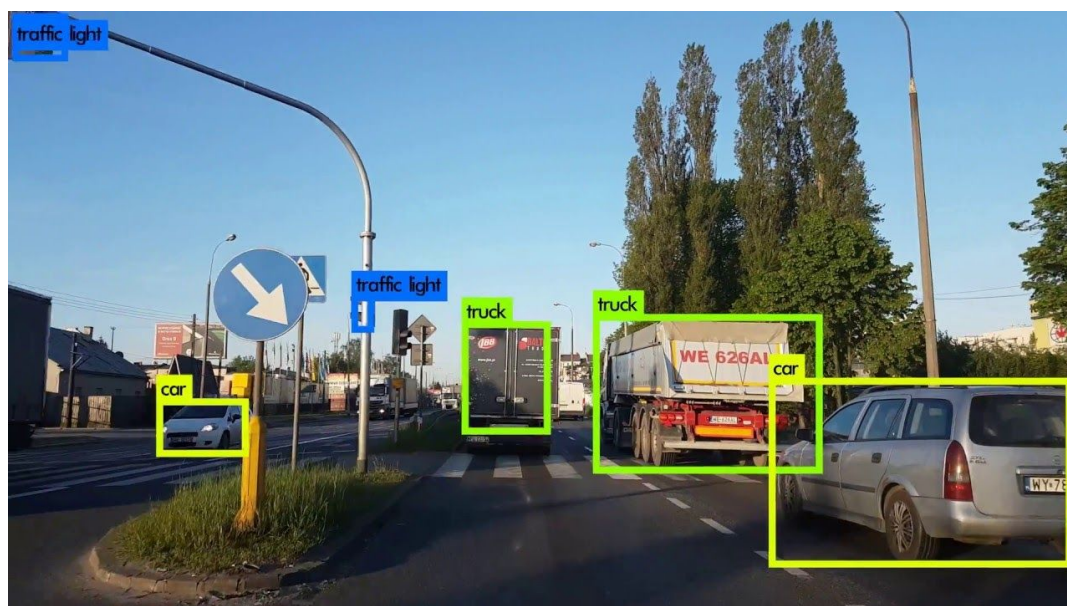


Figura: detection, localizzazione e labeling di vari oggetti.

COCO tratta le immagini a tre canali di colore (R,G,B) ed ogni colore viene trattato come una matrice. Sono inoltre considerate anche altre proprietà delle immagini come l'intensità e la trama.

Il modello che abbiamo scelto è stato pre-allenato proprio su COCO e successivamente specializzato tramite *transfer learning* sul dataset di Wally.

I modelli pre-allenati di Tensorflow si basano sul dataset COCO che pur essendo molto generico, offre una grande quantità di categorie. La Tensorflow Object Detection API offre modelli pre-allenati basati su alcuni dataset standard, tra cui COCO, mettendo a disposizione i pesi “congelati” di tutti gli strati eccetto del penultimo.



A partire da questi pesi, possiamo allenare la nostra rete ad imparare la rappresentazione del solo ultimo strato, utilizzando il dataset customizzato.

## WALDO

Per il nostro progetto abbiamo creato un dataset customizzato COCO-like.

Il dataset è stato così suddiviso:

- **Training set**, composto da 48 immagini su cui è stato allenato il modello;
- **Test set**, composto da 6 immagini su cui è stato effettuato il test.

Abbiamo deciso di utilizzare il 90% delle immagini per addestrare il modello ed il restante 10% per effettuare il test per ridurre il rischio di overfitting.

Il dataset fornisce le annotazioni per la classe “Waldo” con relativi riferimenti relativi alle bounding box:

`(x_min, x_max, y_min, y_max)`



Figura: alcune delle immagini del dataset.

## PREPROCESSING

### Labeling

Il primo step è preparare le annotazioni sulle immagini, per indicare al modello le posizioni degli oggetti e la loro classe.

A tal scopo si è utilizzato il tool labelImg<sup>4</sup>, che consente di creare annotazioni attraverso un'interfaccia grafica. Il risultato è stato esportato in .xml, e successivamente convertito in .csv per essere compatibile con la Tensorflow Object Detection API.

---

<sup>4</sup> "tzutalin/labelImg - GitHub." <https://github.com/tzutalin/labelImg>. Accessed 22 Feb. 2020.

Una annotazione così esportata è compatibile con il formato COCO ed è costituita da una tupla di 5 elementi:

(filename, width, height, class, x\_min, x\_max, y\_min, y\_max)

in cui *width* e *height* si riferiscono alle dimensioni dell'immagine *filename* annotata, gli altri quattro parametri indicano gli estremi della bounding box su un riferimento cartesiano.

Il file .csv contiene le annotazioni su tutte le immagini, di cui ogni riga contiene una tupla.

1	filename	width	height	class	xmin	ymin	xmax	ymax
2	wally_2020_2_9_787050_yejy.jpg	256	256	waldo	108	21	147	77
3	wally_2020_2_9_787050_yejy.jpg	256	256	waldo	105	20	151	149
4	wally_2020_2_9_787050_yejy.jpg	256	256	waldo	106	21	151	174
5	wally_2020_2_9_829475_tjbd.jpg	340	275	waldo	92	2	206	138
6	wally_2020_2_9_829475_tjbd.jpg	340	275	waldo	5	1	270	331

Figura: Un estratto del file .csv per le annotazioni

Le immagini sono state annotate seguendo due *policy* diverse, chiamate *onlyface* e *body+face*, nella prima si sono annotate solo le facce di Wally, nella seconda facce e figure intere. Tuttavia, da esperimenti successivi, si è concluso che tale divisione in due policy non è fondamentale per i risultati, e nelle sezioni successive si è considerata solamente la policy *body+face*.

## Data Augmentation

Visto che il dataset a disposizione è molto limitato, per evitare il rischio di *overfitting* si è fatta *data augmentation* utilizzando i tools messi a disposizione dalla API.

Tuttavia, dopo vari esperimenti, si è visto che queste tecniche non conducono a risultati rilevanti, il tipo di preprocessing più significativo è quello descritto nella sezione successiva.

## Cropping

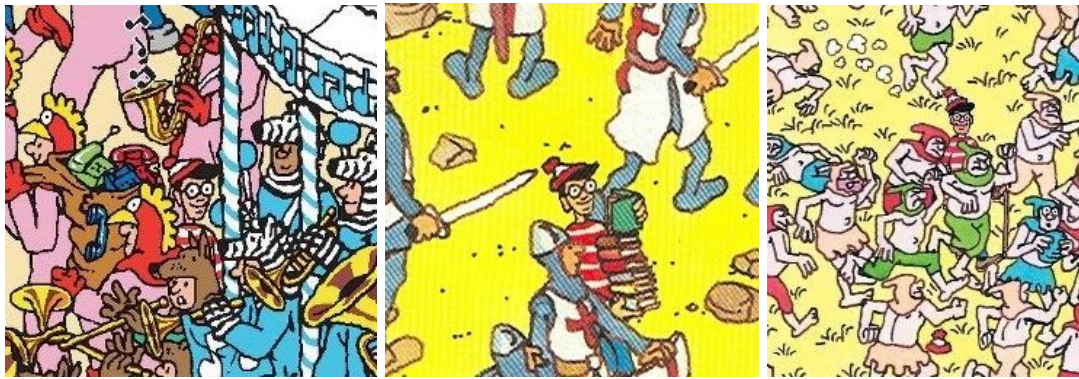
La struttura del dataset originale e la definizione del problema stesso pongono delle difficoltà aggiuntive rispetto ad un problema di object detection standard, in particolare abbiamo le seguenti considerazioni:

- Wally appare nelle immagini una volta sola, ovvero esiste una sola bounding box che è ground truth per ogni immagine;

- le immagini hanno risoluzioni diverse, e Wally appare in esse a dimensioni variabili (problema di *scaling*), e in generale le immagini sono “troppo grandi” (ad es. 2953x2088), mentre il modello è *pretrained* su COCO che usa immagini di dimensione molto minore;
- Wally, o la sua faccia, non appaiono quasi mai per intero (problema di *occlusion*);
- le immagini sono caratterizzate da molto *noise*, il task di trovare Wally è reso volutamente difficile in modo da confondere il giocatore, per questo motivo sono presenti *impostori* che assomigliano a Wally, o altre controparti (es. Wenda ).

Per questi motivi, attuare un preprocessing sul dataset originale risulta necessario per ottenere dei risultati soddisfacenti.

In particolare, per evitare il problema delle dimensioni troppo elevate delle immagini si è deciso di ridurre ciascuna delle immagini originali, producendo un dataset *cropped*, formato da tutte immagini di dimensione 256x256.



Le annotazioni sono state poi automaticamente aggiornate utilizzando la medesima libreria<sup>5</sup>, in modo da riflettere le dimensioni del dataset cropped.

<sup>5</sup> "lozuwa/imp: Impy is a Python3 library with features ... - GitHub." <https://github.com/lozuwa/imp>. Accessed 22 Feb. 2020.



# METRICHE

Le metriche utilizzate per la valutazione del modello sul test set sono basate su COCO.

## Average Precision (AP):

AP	% AP at IoU=.50:.05:.95 (primary challenge metric)
AP <sup>IoU=.50</sup>	% AP at IoU=.50 (PASCAL VOC metric)
AP <sup>IoU=.75</sup>	% AP at IoU=.75 (strict metric)

## AP Across Scales:

AP <sup>small</sup>	% AP for small objects: area < 32 <sup>2</sup>
AP <sup>medium</sup>	% AP for medium objects: 32 <sup>2</sup> < area < 96 <sup>2</sup>
AP <sup>large</sup>	% AP for large objects: area > 96 <sup>2</sup>

## Average Recall (AR):

AR <sup>max=1</sup>	% AR given 1 detection per image
AR <sup>max=10</sup>	% AR given 10 detections per image
AR <sup>max=100</sup>	% AR given 100 detections per image

## AR Across Scales:

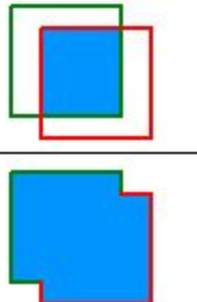
AR <sup>small</sup>	% AR for small objects: area < 32 <sup>2</sup>
AR <sup>medium</sup>	% AR for medium objects: 32 <sup>2</sup> < area < 96 <sup>2</sup>
AR <sup>large</sup>	% AR for large objects: area > 96 <sup>2</sup>

1. Unless otherwise specified, AP and AR are averaged over multiple Intersection over Union (IoU) values. Specifically we use 10 IoU thresholds of .50:.05:.95. This is a break from tradition, where AP is computed at a single IoU of .50 (which corresponds to our metric AP<sup>IoU=.50</sup>). Averaging over IoUs rewards detectors with better localization.
2. AP is averaged over all categories. Traditionally, this is called "mean average precision" (mAP). We make no distinction between AP and mAP (and likewise AR and mAR) and assume the difference is clear from context.
3. AP (averaged across all 10 IoU thresholds and all 80 categories) will determine the challenge winner. This should be considered the single most important metric when considering performance on COCO.
4. In COCO, there are more small objects than large objects. Specifically: approximately 41% of objects are small (area < 32<sup>2</sup>), 34% are medium (32<sup>2</sup> < area < 96<sup>2</sup>), and 24% are large (area > 96<sup>2</sup>). Area is measured as the number of pixels in the segmentation mask.
5. AR is the maximum recall given a fixed number of detections per image, averaged over categories and IoUs. AR is related to the metric of the same name used in [proposal evaluation](#) but is computed on a per-category basis.
6. All metrics are computed allowing for at most 100 top-scoring detections per image (across all categories).
7. The evaluation metrics for detection with bounding boxes and segmentation masks are identical in all respects except for the IoU computation (which is performed over boxes or masks, respectively).

Le metriche COCO utilizzano le seguenti definizioni.

## Intersection over Union (IoU)

È definita come area di *overlap* tra la bounding box che rappresenta la *ground truth* ( $B_{gt}$ ) e la bounding box predetta ( $B_p$ ) e serve a stabilire se la predizione è valida (*vero positivo TP*) o no (*falso positivo FP*). Per stabilirlo, si utilizza un valore di *threshold* (tipicamente 50%, 75%, 95%) che viene confrontato con l'IoU.

$$IOU = \frac{\text{area of overlap}}{\text{area of union}} = \frac{\text{area of overlap}}{\text{area of union}}$$


In particolare:

- True Positive (TP): una detection corretta, se  $IoU \geq \text{threshold}$
- False Positive (FP): una detection scorretta, se  $IoU < \text{threshold}$
- False Negative (FN): ground truth non trovata
- True Negative (TN): non viene normalmente utilizzata, rappresenta tutte le bounding box possibili che non rappresentano alcuna ground truth e che (in modo corretto) non sono state individuate.

## Precision

$$Precision = \frac{TP}{TP + FP} = \frac{TP}{\text{all detections}}$$

La *precision* misura l'abilità di un modello nell'identificare solo gli oggetti rilevanti. È il rapporto tra le predizioni corrette e quelle totali.

## Recall

La *recall* misura l'abilità di un modello ad individuare tutti i casi rilevanti, in object detection, tutte le bounding box che sono ground truth. È quindi il rapporto tra le predizioni corrette e tutte le ground truth individuabili.

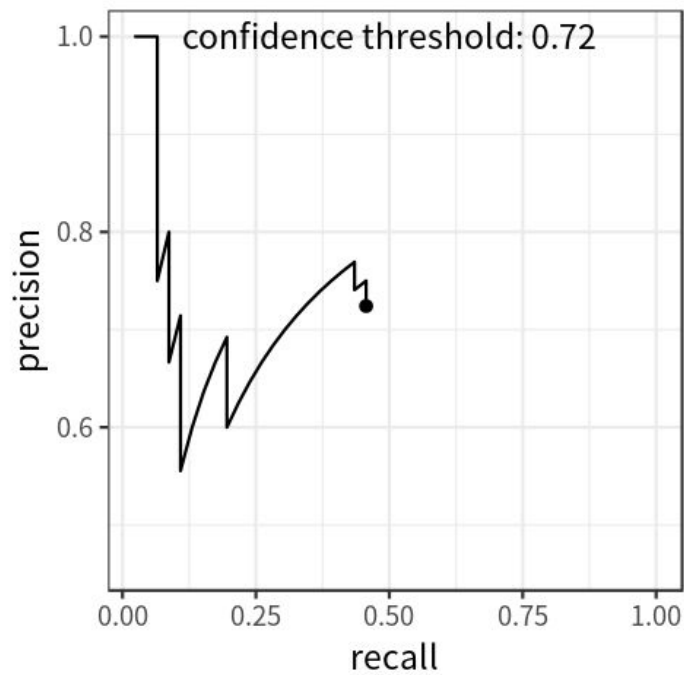
$$Recall = \frac{TP}{TP + FN} = \frac{TP}{\text{all ground truths}}$$

A partire da queste definizioni, si derivano le principali metriche usate in object detection, usate anche in COCO.

## Precision-Recall Curve

È la curva generata a partire da precision (asse y) e recall (asse x), per mostrare l'associazione tra le due metriche. I valori vengono poi plottati variando il livello di threshold.

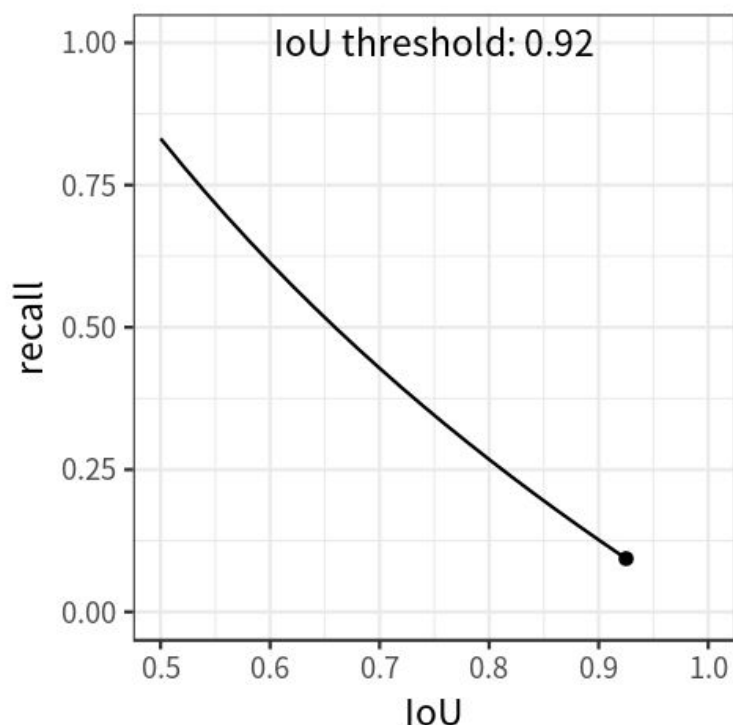




Al diminuire della threshold la recall aumenta monotonamente (intuitivamente, aumentano le detection TP basate su IoU), mentre la precision ha un andamento a *zig-zag* o in genere tende a diminuire. Per un buon object detector la precision resta elevata all'aumentare della recall.

## Recall-IoU Curve

È la curva generata da recall (asse y) e IoU (asse x) e mostra come la recall diminuisce all'aumentare di IoU (intuitivamente, diminuisce il numero di esempi TP identificati).



## Average Precision (AP) e Average Recall (AR) <sup>6</sup>

Sono misure numeriche basate sulla curva precision-recall, utili in caso la curva non sia particolarmente informativa (es. in presenza di intersezioni tra curve di diversi object detector).

AP è la precision media su tutti i livelli di recall unici (da 0 a 1). Il numero di livelli può essere scelto arbitrariamente; tradizionalmente sono scelti 11 livelli equidistanti (i.e., 0.1, 0.2 ... 1.0) e viene eseguita un'interpolazione della curva negli 11 punti su cui calcolare la precision.

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

AP quindi può essere definita come area sotto la curva interpolata, calcolata come:

---

<sup>6</sup> "An Introduction to Evaluation Metrics for Object Detection ...." 16 Dec. 2018, <https://blog.zenggyu.com/en/post/2018-12-16/an-introduction-to-evaluation-metrics-for-object-detection/>. Accessed 22 Feb. 2020.

$$AP = \sum_{i=1}^{n-1} (r_{i+1} - r_i) p_{interp}(r_{i+1})$$

dove  $r_i$  è la recall calcolata all'  $i$ -esimo livello.

AP è calcolata considerando una classe sola. In object detection, in cui solitamente sono presenti più classi, viene in realtà calcolata la mAP (mean Average Precision), definita come AP media su tutte le categorie di oggetti.

$$mAP = \frac{\sum_{i=1}^K AP_i}{K}$$

AR è la recall media su tutti i valori di IoU considerati e può essere calcolata come due volte l'area sotto la curva recall-IoU

$$AR = 2 \int_{0.5}^1 recall(o) do$$

in cui  $o$  è il valore di IoU e  $recall(o)$  è la recall corrispondente. AR è calcolata classe per classe.

Allo stesso modo si definisce la mAR (mean Average Recall) come media di AR su tutte le classi (K)

$$mAR = \frac{\sum_{i=1}^K AR_i}{K}$$

# TRAINING

Il training è suddiviso in alcune fasi principali come illustrato nella sezione Faster R-CNN che ne descrive l'allenamento: l'estrazione delle feature che viene fatta attraverso una rete convoluzionale chiamata *ResNet101* che a sua volta è stata pre-allenata sul famoso dataset ImageNet ed è in grado di classificare circa 1000 classi di oggetti in maniera generica, successivamente vengono calcolati gli anchor, che serviranno alla fase successiva per decidere quali sono le regioni effettivamente rilevanti all'interno di una certa immagine. Delle regioni ricavate allo step precedente ne consideriamo le 300 migliori usando una threshold come ausilio per la scelta.

Essendo la rete utilizzata allenata per classificazione generica, per completare il task è necessario eseguire uno step di fine-tuning per specializzare l'intera rete perché riconosca Wally, quindi in questa fase si agirà sull'ultimo livello, composto da classificatore e regressore. In particolare la parte che deve essere specializzata è il classificatore perché è quello che dovrà predire la classe *wally*.

La pipeline di training viene definita attraverso un file di config, nel nostro caso il file di chiama **faster\_rcnn\_resnet101\_coco.config**.

La rete scelta per completare il task è, appunto, una Faster R-CNN allenata su una sola classe, ovvero *wally* che è l'obiettivo della ricerca.

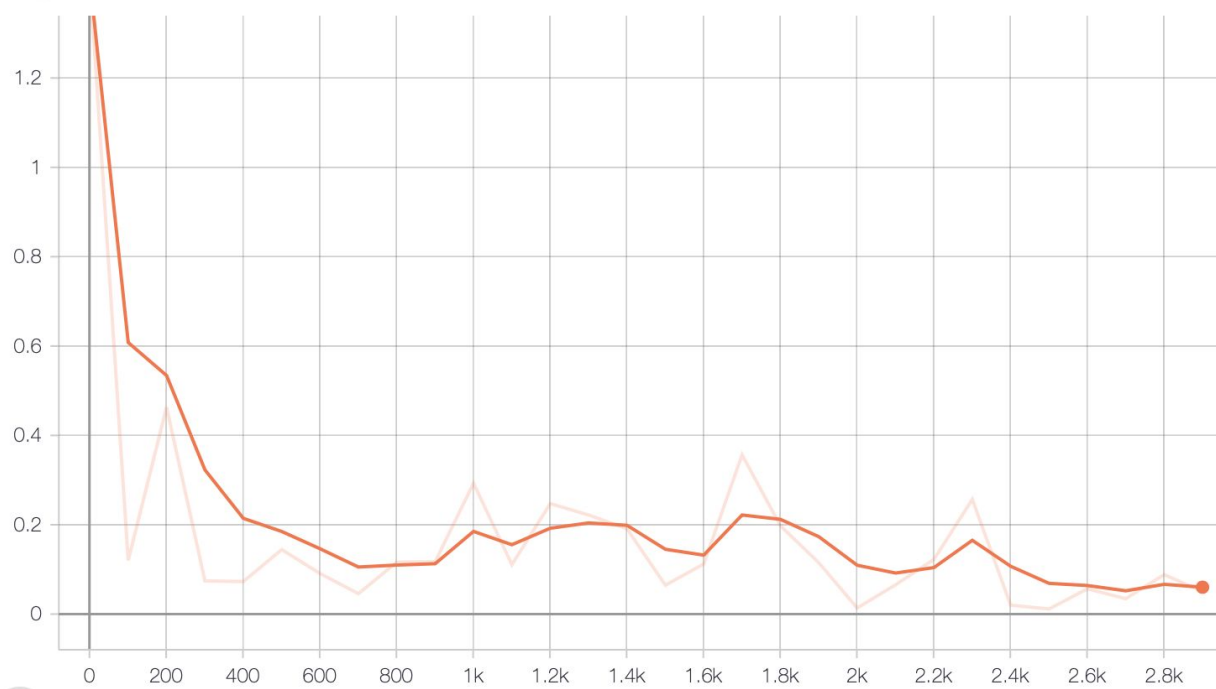
Sono specificati, nello stesso file, anche gli *hyperparameter* relativi a batch size, posto uguale a 1, il learning rate a  $3e-4$  ed il numero di step usato per specificare quanti passi di training eseguire, è impostato di default a 6000 ma è modificabile attraverso lo script di training.

Infine, nello stesso file sono indicati i percorsi relativi ai record di train e test generati dalla conversione delle annotazioni dal formato csv in Tensorflow record contenenti le informazioni che andranno ad addestrare la nostra rete per questo task specifico e le informazioni delle immagini su cui verranno eseguiti i test.

Durante la fase di training è possibile salvare dei checkpoint del modello per valutare quale riesce ad ottenere i risultati migliore, che non è detto essere l'ultimo.

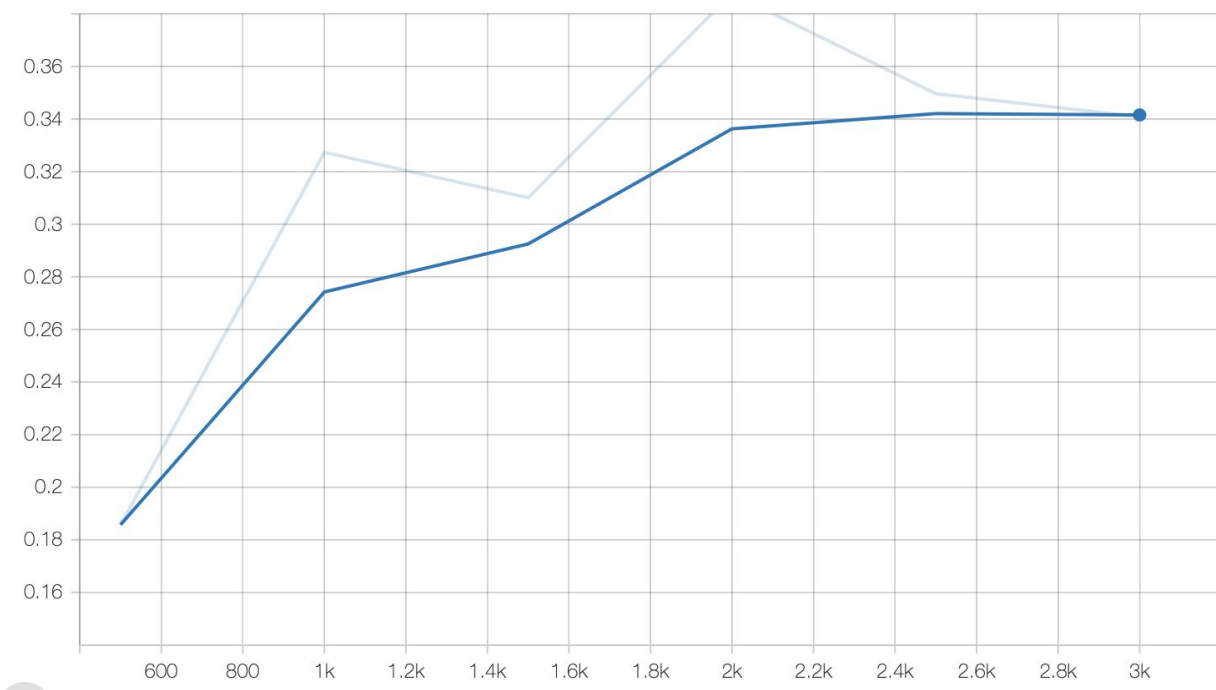
Nell'immagine seguente mostriamo l'andamento della funzione di *loss* ottenuta dall'unione di una cross entropy binaria per il classificatore ed una smooth *L1 loss* per il bounding box, opportunamente bilanciati e normalizzati.

loss\_1



mAP

tag: DetectionBoxes\_Precision/mAP

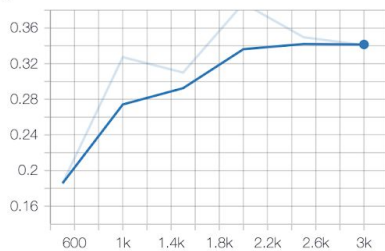




## DetectionBoxes\_Precision

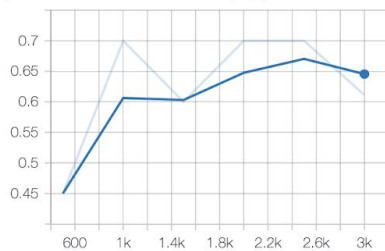
mAP

tag: DetectionBoxes\_Precision/mAP



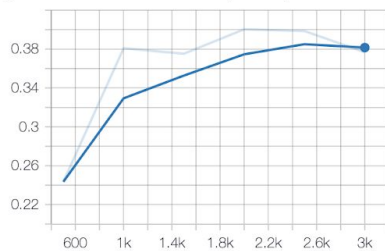
mAP (large)

tag: DetectionBoxes\_Precision/mAP (large)



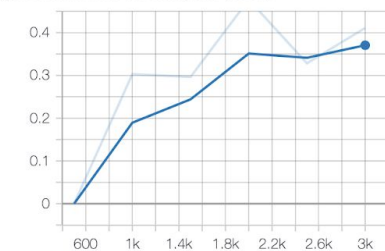
mAP (medium)

tag: DetectionBoxes\_Precision/mAP (medium)



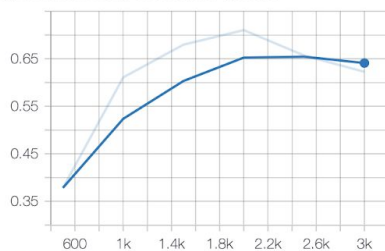
mAP (small)

tag: DetectionBoxes\_Precision/mAP (small)



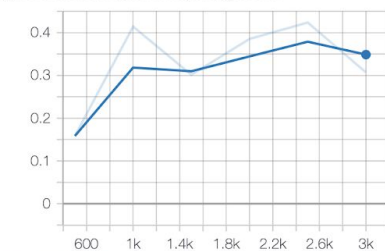
mAP@.50IOU

tag: DetectionBoxes\_Precision/mAP@.50IOU



mAP@.75IOU

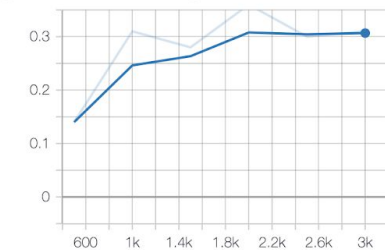
tag: DetectionBoxes\_Precision/mAP@.75IOU



## DetectionBoxes\_Recall

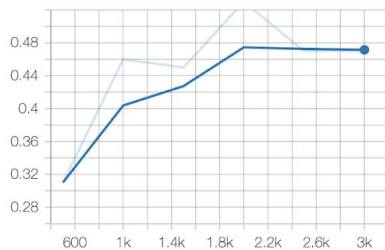
AR@1

tag: DetectionBoxes\_Recall/AR@1



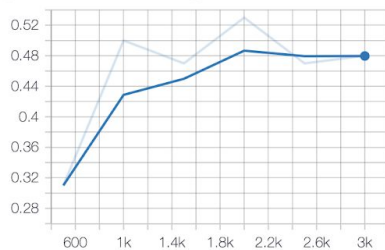
AR@10

tag: DetectionBoxes\_Recall/AR@10



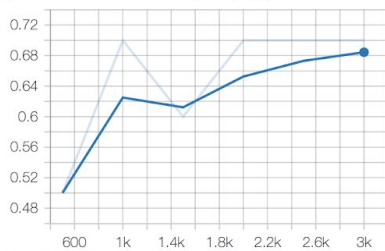
AR@100

tag: DetectionBoxes\_Recall/AR@100



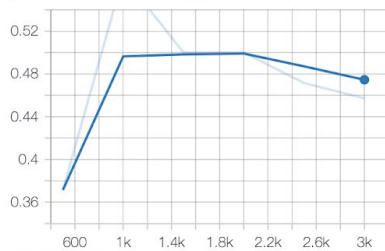
AR@100 (large)

tag: DetectionBoxes\_Recall/AR@100 (large)



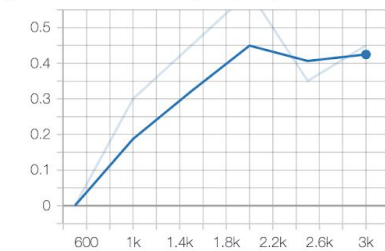
AR@100 (medium)

tag: DetectionBoxes\_Recall/AR@100 (medium)



AR@100 (small)

tag: DetectionBoxes\_Recall/AR@100 (small)



# RISULTATI

Riportiamo i risultati dello script di evaluation sul test set.

A sinistra si ha la predizione della rete, con bounding box e accuratezza, e a destra la ground truth.

Nella prima riga osserviamo come la rete riesca ad individuare Wally con un elevato grado di sicurezza, mentre nella seconda riga, in cui la scena appare più confusa (c'è una densità di personaggi più elevata), Wally viene individuato, ma insieme a lui anche altri falsi positivi.



## Inferenza

Per “inferenza” si intende l’esecuzione del modello di object detection su di un’immagine, in modo da avere come risultato una predizione.

Idealmente, la rete dovrebbe essere capace di individuare Wally all'interno di un'immagine complessa, ad esempio una scansione a grandezza intera presa da uno dei libri.

Per riprodurre questo scenario “reale” e verificare la performance del modello su di esso, si è elaborata una pipeline che aggiunge alla fase di detection vera e propria un pre- e post-processing.

Una tecnica che si è dimostrata efficace in letteratura in caso di object detection per oggetti di dimensione molto ridotta<sup>7</sup>, è suddividere l'immagine a dimensione reale in *tiles* di dimensione fissa su cui eseguire l'object detection singolarmente.

La pipeline di inferenza si suddivide quindi in 3 step: un *preprocessing*, in cui si divide l'immagine in sotto-immagini di dimensione fissa, una *detection*, in cui si disegnano le bounding box come overlay sulle immagini singole e un *post-processing*, che consiste nella ricombinazione delle tiles a formare l'immagine originale (che adesso contiene anche le bounding box).

## CONCLUSIONI

La performance del modello è stata ritenuta sufficiente per i nostri scopi, tuttavia, alcuni aspetti sono migliorabili:

- al momento la rete è allenata su immagini in cui Wally appare sempre al centro, mentre la suddivisione dell'immagine in tiles in fase di inference, produce sotto-immagini in cui Wally potenzialmente può apparire ai bordi, o comunque non in posizione centrale. Un miglioramento si potrebbe ottenere aggiungendo al preprocessing sul train set una fase di traslazione delle immagini, in modo che Wally non appaia più al centro;
- ancora durante l'inference, l'object detection ripetuta molteplici volte sulle singole tiles, può portare a produrre una serie di falsi positivi, chiaramente maggiore di quello che si avrebbe facendo object detection sull'immagine singola. Per migliorare i risultati si potrebbe pensare ad una ricombinazione “intelligente” delle tiles che cerca di minimizzare i falsi positivi.

Inoltre, i seguenti miglioramenti potrebbero essere avanzati:

- incrementare il numero di esempi nel dataset;

---

<sup>7</sup> "The Power of Tiling for Small Object Detection - CVF Open ...."

[http://openaccess.thecvf.com/content\\_CVPRW\\_2019/papers/UAVision/Unel\\_The\\_Power\\_of\\_Tiling\\_for\\_Small\\_Object\\_Detection\\_CVPRW\\_2019\\_paper.pdf](http://openaccess.thecvf.com/content_CVPRW_2019/papers/UAVision/Unel_The_Power_of_Tiling_for_Small_Object_Detection_CVPRW_2019_paper.pdf). Accessed 22 Feb. 2020.

- provare altre tecniche di data augmentation, da applicare successivamente alla fase di cropping;
- sperimentare con modelli *pretrained* diversi.