UNIVERSITÀ
DI TRENTO

Dipartimento di Ingegneria e Scienza dell'Informazione

Corso di Laurea in
Ingegneria dell'Informazione e Organizzazione d'Impresa

Elaborato finale

iTelescope

Riconoscimento automatico di immagini per applicazioni di sorveglianza territoriale tramite droni

Supervisore
Prof. Fabrizio Granelli

Laureando
Alessio Belli

Anno accademico 2020/2021

Ai miei genitori e a Martina,
che con il loro Amore hanno forgiato la persona che sono

A Marta,
che ogni giorno riempie e completa la mia vita

Ai miei nonni, zii e cugini,
che, per quanto lontani, sono molto vicini nel mio cuore

A Maria, Giuseppe, Elisa e Milena,
che rappresentano la mia seconda famiglia

A Flora, Roberto ed Antonio,
che da sempre mi accolgono con generosità ed affetto

A Matteo, Gianluca ed Alberto,
che mi dimostrano quotidianamente il vero valore dell'amicizia

A queste persone e a coloro che mi hanno sostenuto in questo percorso,
dedico il mio lavoro.

Indice

Introduzione

La seguente tesi propone l'analisi di un *object detector* che è stato impiegato per individuare dieci tipologie di oggetti all'interno di immagini urbane scattate da droni. Questi ultimi sono dispositivi sempre più utilizzati che, con l'ausilio dell'intelligenza artificiale, permettono di aprire nuovi scenari nell'ambito della sicurezza e del controllo cittadino automatizzato. Il drone è un velivolo radiocomandato in grado di sorvolare un'intera area cittadina in pochissimi minuti scattando diverse immagini e girando video. Una volta acquisiti i contenuti multimediali, è possibile applicare degli algoritmi che sono stati formulati al fine di individuare gli oggetti di interesse presenti nelle immagini.

Il primo capitolo propone una visione d'insieme nel campo dell'*image analysis* e della *computer vision*. L'analisi prosegue con un approfondimento sui principali *object detectors* presenti in letteratura, concentrando l'attenzione su "*You Only Look Once*" (*YOLO*). Quest'ultimo permette di riconoscere e localizzare i diversi oggetti, presenti nelle immagini e nei video, attraverso un solo passaggio nella rete neurale. Con le dovute ottimizzazioni e implementazioni, *YOLO* è uno dei migliori algoritmi per il rilevamento degli oggetti in *real-time*.

Nel secondo capitolo viene presentata l'applicazione desktop realizzata per consentire ad un utente di ricercare gli oggetti selezionati all'interno di fotografie e video. La *detection* è eseguita mediante il modello *YOLOv4*. Quest'ultimo è stato addestrato utilizzando un ampio *dataset* di immagini provenienti da droni e ritraenti dieci classi di oggetti (pedoni, macchine, motociclette, van, autobus, camion, persone, biciclette, tricicli e tricicli motorizzati). Il programma desktop è stato sviluppato applicando il linguaggio di programmazione *Python* e distribuito tramite la virtualizzazione offerta da *Docker*. In conclusione, si analizza una versione molto più leggera di *YOLO*, denominata *tinyYOLOv3*, che viene impiegata per l'analisi *on-board* su un drone delle immagini acquisite per il progetto "*NATO SPS G5428 DAVOSS*".

Nel terzo capitolo, infine, vengono presentati i risultati ottenuti in termini di accuratezza dell'*object detector*, sviluppato sia per *YOLOv4* che per *tinyYOLOv3*. In conclusione, si riporta il confronto tra le diverse configurazioni della *Convolutional Neural Network* che permette di individuare i parametri corretti da utilizzare al fine di creare il miglior modello per l'individuazione e il riconoscimento di oggetti all'interno di immagini fotografate da droni.

1 Analisi delle immagini

Nel corso degli anni l'*analisi delle immagini* ha rappresentato un elemento sempre più importante e centrale della nostra quotidianità. Per questo motivo è emersa la necessità di perfezionare gli studi al fine di rendere tale materia maggiormente pratica e dinamica. Recentemente, l'analisi delle immagini si è evoluta grazie anche all'utilizzo dell'intelligenza artificiale e di sistemi automatici. Tramite lo sviluppo di algoritmi dedicati è infatti possibile compiere un numero maggiore di operazioni in maniera sempre più efficiente ed efficace. Volendo semplificare con un esempio: l'addetto alla sicurezza che controlla le telecamere di sorveglianza può oggi essere sostituito con un sistema in grado di riconoscere i volti e segnalare le anomalie in modo automatico.

L'utilizzo di un sistema in grado di analizzare le immagini può trovare applicazione in numerosi altri campi come, ad esempio, nei laboratori diagnostici con lo scopo di riconoscere le varie cellule del sangue in un vetrino[24] o all'interno di un parcheggio al fine di segnalare i posti occupati dalle autovetture.

Tradizionalmente l'immagine studiata dal modello viene memorizzata in un momento antecedente a quello dell'analisi. In alcuni casi però, soprattutto nei sistemi più evoluti, è possibile elaborare i fotogrammi di un video in *real time*.

L'*analisi delle immagini* rientra all'interno di un campo molto più vasto denominato *image processing* [16]. Quest'ultimo si articola nelle seguenti fattispecie:

- *Image compression*
- *Image preprocessing*
- *Image analysis*

È importante precisare che l'*image analysis*, a differenza dell'*image compression* e dell'*image preprocessing*, non produce un'immagine bensì valori numerici che possono essere utilizzati per la classificazione ed il riconoscimento di oggetti all'interno dell'immagine di input.

1.1 Immagine digitale

Prima di affrontare l'argomento sull'analisi delle immagini e sulle applicazioni che possono derivare da essa, è opportuno definire il termine "*immagine digitale*". Quest'ultima non si riferisce solamente alla classica fotografia scattata per immortalare un momento speciale, ma anche a tutte quelle immagini ottenute grazie a sensori particolari in grado di convertire in immagine digitale qualsiasi grandezza fisica multidimensionale (mappa di calore, sorgente sonora...).

L'*immagine digitale* può essere definita come una matrice di elementi (*pixel*) che vengono identificati tramite l'utilizzo di coordinate *(x; y)*. La *risoluzione* si caratterizza per il numero di pixel che sono contenuti in ogni inch e viene espressa più comunemente con PPI[1].

L'esperimento del prisma di *Sir. Isaac Newton* mostrò come un raggio di luce bianca può essere diviso in un insieme di sette colori "puri" (rosso, arancione, giallo ecc.). Tale intuizione fu ripresa per definire i colori di un pixel. Più precisamente, quando si vuole determinare il colore di un pixel, la notazione più popolare è la RGB[2]. Quest'ultima grazie alla combinazione del rosso, del verde e del blu permette di ottenere qualsiasi colore. Molto spesso, le luci di questi colori sono rappresentate in un intervallo 0 - 255, dove 0 rappresenta la luminosità minima e il 255 la luminosità massima. Pertanto, è facile intuire che ogni colore è composto da una terna di valori. Per esempio, l'arancione è identificato dal codice RGB 255; 128; 0.

[1] Pixels Per Inch
[2] Red Green Blue

Nelle immagini digitali è possibile definire la *statistica dell'immagine* stessa, ossia il comportamento statistico dei valori assunti dai vari pixel. Una delle grandezze, che di solito viene presa in considerazione al momento dell'elaborazione di una immagine, è l'*istogramma*. Per la costruzione di quest'ultimo, è necessario calcolare la frequenza relativa di occorrenza dei differenti valori (livelli di grigio o colori) presenti all'interno dell'immagine. Nelle immagini a più componenti (immagini a colori per esempio) è possibile calcolare un istogramma per ogni componente cromatica.
La formula per calcolare l'istogramma di un'immagine MxN è la seguente:

$$hist(p) = \frac{numero\ di\ pixel\ I(x,y) = p}{MxN}$$

1.2 Computer vision

La *visione artificiale* iniziò la sua ascesa negli anni '60. Da allora tale campo si è evoluto fino ai giorni nostri, riuscendo a risolvere problemi sempre più frequenti e complessi[1].

La *Computer vision* è finalizzata a sviluppare tecniche capaci di "aiutare" un computer a vedere e a capire il contenuto di un'immagine digitale o di un video e di replicare il sistema visivo umano. Tali processi sono multidisciplinari in quanto appartengono ad un sottocampo del *machine learning* e dell'*intelligenza artificiale*[4].

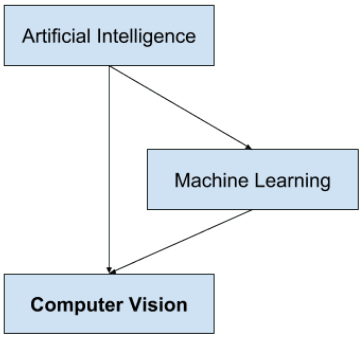


Figura 1.1: Computer vision[4]

Prima dell'adozione di sistemi di apprendimento, l'analisi delle immagini provenienti da raggi-X, risonanze magnetiche e immagini spaziali ad alta risoluzione doveva essere eseguita manualmente.

La *Computer vision* si differenzia dalla maggior parte delle funzioni dell'*image processing* in quanto non crea alcuna immagine, bensì si concentra sul contenuto dell'immagine stessa.

Sviluppare tecniche e algoritmi da utilizzare in questo settore non è un processo semplice, il computer è infatti tenuto a simulare il più possibile il cervello umano. Come si può intuire, questo compito è di ardua concretizzazione. Oltre a ciò, le teorie sulla computazione del cervello sono ridotte e di conseguenza si può solamente suppore il modo in cui il cervello "lavori"[17].

Attraverso la *computer vision*, è possibile sviluppare diverse applicazioni che mirano al riconoscimento di oggetti presenti nelle immagini. Tra queste vi sono: *object classification*, *object identification*, *object detection*, *object landmark detection* e *object recognition*[17].

Tali tecnologie possono essere sfruttate in vari ambiti come, ad esempio, per la guida autonoma degli autoveicoli, per la sorveglianza e per la gestione di aziende. In merito a quest'ultimo aspetto, le applicazioni possono analizzare le immagini provenienti dalle telecamere per visionare se un determinato articolo posizionato sullo scaffale sia esaurito o meno. Risulta evidente, perciò, che l'adozione della *computer vision* è in grado di migliorare l'attività d'impresa apportando notevoli vantaggi di carattere organizzativo.

In conclusione, è importante precisare che un notevole sviluppo della *computer vision* si è raggiunto con l'adozione del *deep learning*. Quest'ultimo permette di utilizzare delle reti neurali e degli algoritmi precostruiti e "allenarli" sulla tipologia di immagini che sono necessarie per la realizzazione di specifiche

applicazioni come l'*object detection*. Tra le diverse tipologie di reti neurali, le *CNN*[3] hanno raggiunto risultati migliori rispetto alle tecniche tradizionali.

Per creare un modello di *computer vision*, usando le *CNN*, bisogna rispettare 4 importanti step [2]:

- *Creazione del dataset* - Si crea un dataset al cui interno vi è un numero elevato di immagini target. Ogni immagine deve essere affiancata da una label che determina la classe, la posizione e la grandezza del box rispetto ad ogni oggetto significativo;
- *Estrazione features* - Si estraggono per ogni immagine delle *features* pertinenti al *task* da eseguire;
- *Training* – Si allena il modello di deep learning sulla base delle *features* estratte dalle immagini;
- *Testing* - Si valuta il modello creato attraverso l'analisi di nuove immagini. In questo step viene calcolata l'accuratezza del modello stesso.

1.2.1 Convolutional Neural Network

In questa sezione si illustrerà uno dei temi centrali della *computer vision* e soprattutto dell'*object detection* (sezione 1.3). La *CNN* è una classe del *deep neural network* comunemente applicata per l'analisi delle immagini e la sua architettura è molto simile allo schema di collegamento dei neuroni all'interno del cervello umano.

Tale tipologia di rete neurale permette di ridurre le immagini (anche di grandi dimensioni) in una "forma" processabile senza rinunciare alle caratteristiche fondamentali per la buona riuscita della predizione[26].

Le componenti principali di una *ConvNet* sono: *convolutional layers*, *pooling layers* e *fully connected layers*. Le prime due vengono utilizzate per le estrazioni delle *features* dell'immagine in input, mentre l'ultima per mappare tali caratteristiche al fine di ottenere l'output finale (per esempio la classificazione).

Convolutional Layers

Data un'immagine di input, i layers convoluzionali vengono utilizzati per l'estrazione delle caratteristiche target. Più precisamente, gli strati convoluzionali sono un insieme di operazioni lineari nelle quali degli *array* di numeri (denominati *kernel*) vengono applicati ad un'immagine composta a sua volta da un insieme di valori comunemente chiamati *tensor*.

Ipotizzando che non venga aggiunto il *padding* all'immagine, l'applicazione dei layers convoluzionali conduce alla creazione di una *feature map* con le dimensioni ridotte rispetto all'input.

In base alla tipologia di *kernel* è possibile estrarre una determinata *feature* dell'immagine. Per esempio, con l'applicazione di tale kernel $\begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}$ è possibile evidenziare i contorni in un'immagine. Queste procedure sono ripetute molte volte utilizzando matrici diverse (il numero delle matrici dipende dalle caratteristiche che si voglio analizzare).

Pooling Layers

Molto spesso, in seguito ad uno *layer convoluzionale*, è utile inserire un cosiddetto *pooling layer*. Tale componente viene utilizzata al fine di ridurre la dimensione spaziale della feature map che è stata creata precedentemente. Le motivazioni che spingono ad inserire layer di tale genere all'interno della *CNN* sono: la necessità di ridurre la potenza di calcolo richiesta per l'elaborazione dei dati e la volontà di estrarre le *features* predominanti. La soddisfazione di tali esigenze rende il processo di *training* del modello più efficace.

Esistono due tipologie di *pooling layer*:

- Il *max pooling* estrae il valore massimo della porzione di *feature map* coperta dal kernel del pooling. Tale layer è utilizzato molto spesso come soppressore di rumore;

[3]Reti Neurali Convoluzionali

- L'*average pooling* estrae il valore medio della porzione di *feature map* coperta dal kernel del pooling. Tale layer esegue semplicemente una riduzione della dimensionalità come meccanismo di soppressione del rumore.

Fully Connected Layers

I *Fully Connected Layers* rappresentano gli strati finali della *CNN*. Una volta eseguito l'ultimo layer convoluzionale o di pooling, l'immagine (matrice multidimensionale) deve essere trasformata in un vettore ad una dimensione. Successivamente, l'array ottenuto alimenta lo strato completamente connesso.

Il *FC layer* viene utilizzato al fine di imparare combinazioni non lineari a partire dalle *features* estratte dai layers precedenti. Dopo una serie di passaggi attraverso i *FC layers*, l'ultimo strato applica una funzione denominata *softmax activation*[4] che calcola la probabilità di appartenenza dell'input ad una particolare classe.

1.3 Object detection

Una delle applicazioni più utilizzate della computer vision è senza dubbio l'*object detection*. Tale pratica consiste nell'identificazione e nella localizzazione di oggetti presenti all'interno di immagini e video.

L'*object detection* può essere considerato come una combinazione di due tecniche: l'*image classification* e l'*object localization*. Come si può dedurre dalla loro denominazione, l'*image classification* riconosce gli oggetti presenti all'interno delle immagini, mentre l'*object localization* localizza gli oggetti stessi. Tramite l'utilizzo simultaneo di questi due approcci, si ottengono uno o più box (comunemente rettangoli) e una label identificante la classe dell'oggetto per ogni box.

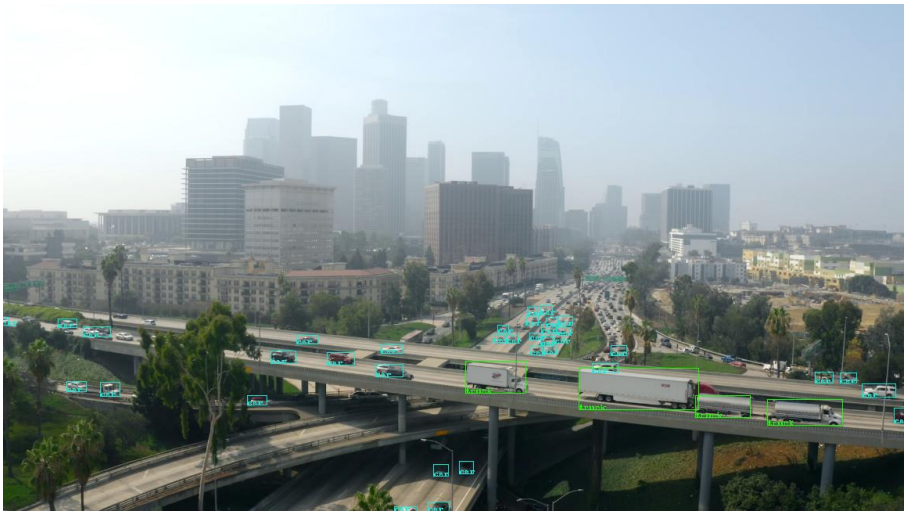


Figura 1.2: Esempio di *object detection* - creazione box e label per ogni oggetto identificato

Negli ultimi vent'anni l'*object detection* ha subito una profonda evoluzione. È possibile distinguere tra due principali periodi storici: gli *object detections* tradizionali (utilizzati prima del 2014) e gli *object detections* basati sul deep learning (sviluppati dopo 2014)[33]. Tale elaborato focalizzerà l'attenzione prevalentemente su quest'ultima fase in quanto rappresenta un'enorme rivoluzione rispetto al passato.

[4]Funzione utilizzata al fine di normalizzare l'output dei layer completamente connessi ad una distribuzione di probabilità sulle classi previste

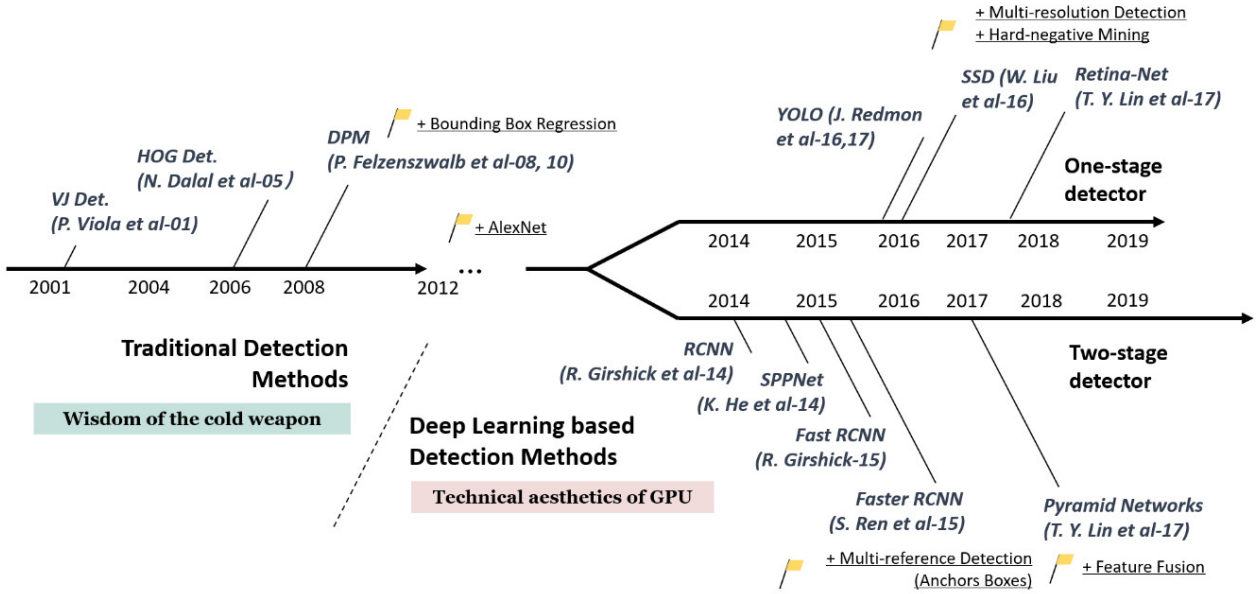


Figura 1.3: Evoluzione dei modelli di *object detection*[33]

In merito alla figura sopra riportata, è importante evidenziare che nell'ultimo periodo si sviluppano due principali ramificazioni: *one-stage detector* e *two-stage detector*. Di seguito si porteranno in esame i principali modelli di *object detection* e le loro caratteristiche fondamentali.

1.3.1 Two-stage detector

I modelli *two-stage detector* effettuano il rilevamento degli oggetti in due fasi. La prima consiste nel ricercare e nel proporre i diversi oggetti candidati (RoI[5]), la seconda nel classificare tali proposte e nel ridefinire con maggiore precisione la loro locazione all'interno dell'immagine[5].

R-CNN[9]

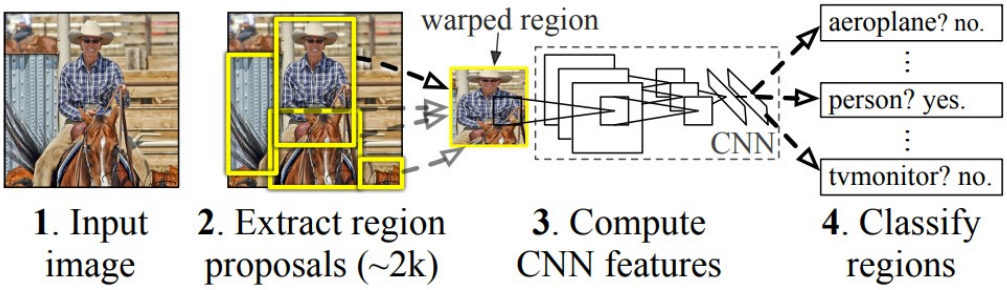


Figura 1.4: Funzionamento object detection con *R-CNN*

Il modello R-CNN[6] fu sviluppato e pubblicato nel 2014. Esso prende in analisi un'immagine di input, estrae all'incirca 2000 proposte di oggetti e per ogni area ritaglia un'immagine a dimensione fissa che viene successivamente analizzata da una rete neurale convoluzionale. Tale rete viene utilizzata per l'estrazione di determinate *features* e per la classificazione di ogni regione attraverso *SVM*[7].

R-CNN ha raggiunto una precisione media (*mAP*) del 53,7% nel *PASCAL VOC 2010*[8], mentre ha realizzato una mAP del 31,4% nel dataset *ILSVRC2013*[9] contenente 200 classi.

[5]Regions of Interest
[6]Region Based Convolutional Neural Networks
[7]Support Vector Machine - modelli di apprendimento supervisionato con algoritmi di apprendimento associati che analizzano i dati per la classificazione
[8]PASCAL Visual Object Classes 2010
[9]ImageNet Large Scale Visual Recognition Challenge 2013

Come si può intuire, l'*R-CNN* presenta delle limitazioni. Se per ogni immagine vengono classificate 2000 proposte di oggetti, il tempo impiegato per l'addestramento è considerevole. In aggiunta, il modello *R-CNN* non può essere utilizzato come metodo di riconoscimento in real time in quanto per ogni immagine di test impiega all'incirca 47 secondi. Come ultimo aspetto, l'algoritmo di ricerca selettiva è fisso e quindi non vi è nessun apprendimento in questa fase[7].

SPPNet[11]

SPPNet[10] è un modello utilizzato per la visual recognition a due stage. A differenza delle CNNs precedenti, che richiedevano una dimensione dell'immagine in input fissa (per esempio 224x224), nel modello *SPPNet* questa limitazione non è più presente in quanto tale requisito può comportare una riduzione dell'accuratezza nella fase di riconoscimento dell'immagine o di una sua parte.

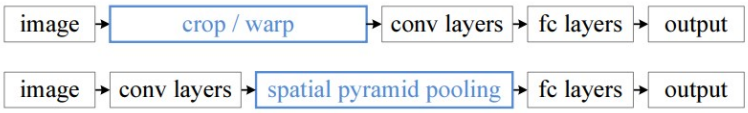


Figura 1.5: L'architettura sopra corrisponde ad una convenzionale *CNN*. Quella sotto corrisponde alla *SPPNet*.

Come si può evincere dalla figura 1.5, per consentire l'utilizzo di immagini a dimensione variabile è stato inserito un layer denominato *SPP*[11] tra i *layers* convoluzionali e quelli interamente connessi. In questo modo è possibile creare un vettore di lunghezza fissa anche al variare della dimensione dell'immagine. In un primo momento, le *features* vengono estratte dai primi livelli convoluzionali e successivamente vengono impiegate per costituire le rappresentazioni a lunghezza fissa date in input ai classificatori.

Il modello *SPPNet* risulta notevolmente più veloce rispetto al *R-CNN*, con una *mAP* del 59.2% nel *VOC07*[12].

Anche in questo modello sono presenti delle limitazioni. Per esempio, l'algoritmo di *fine-tuning* non può aggiornare gli strati convoluzionali che precedono lo *SPP*.

Fast R-CNN[8]

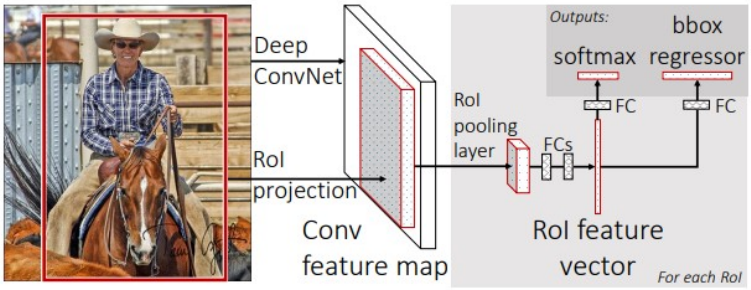


Figura 1.6: Architettura *Fast R-CNN*

Lo stesso autore del *R-CNN*, nel 2015 ha presentato un nuovo modello denominato *Fast R-CNN*.

Questa nuova versione corregge gli aspetti negativi dei due modelli precedenti aumentando anche la precisione e la velocità. I vantaggi del *Fast R-CNN* sono:

- Migliore *mAP* rispetto a *R-CNN* e *SPPnet*;
- Aggiornamento di tutti i *layers* della rete tramite l'addestramento;

[10]Spatial Pyramid Pooling in Deep Convolutional Networks
[11]Spatial Pyramid Pooling Layer
[12]Visual Object Classes 2007

- Non è richiesto spazio sul disco per il *caching* delle *features*

Per ridurre il tempo computazionale nel *Fast R-CNN*, non si esegue più la *CNN* all'incirca duemila volte per immagine ma solamente una volta per ottenere tutte le *RoI*. Al fine di eseguire l'*object detection*, si inserisce l'immagine nella *CNN* in modo tale da generare le regioni di interesse. Queste ultime sono rimodellate successivamente dal *RoI pooling layer* con lo scopo di renderle della stessa misura. In conclusione, queste regioni vengono fatte passare all'interno di una rete completamente connessa che classifica e individua i loro *boxes* di confine.

Con le modifiche apportate, il modello preso in esame è in grado di allenare una rete di rilevamento nove volte più veloce di *R-CNN* e tre volte più veloce di *SPPNet*. *Fast R-CNN* riesce a raggiungere nel *VOC 2007* un *mAP* del 70%[33].

Per rilevare gli oggetti presenti all'interno di una immagine, tale modello impiega all'incirca due secondi. Anche se rispetto al *R-CNN* è molto veloce, bisogna pensare che su grandi *dataset* tale tempo non sia molto soddisfacente. Inoltre, per individuare le *RoI*, si utilizza una ricerca selettiva con metodo di proposta che rende i tempi del processo molto più lunghi.

Faster R-CNN[23]

Faster R-CNN può essere considerato l'evoluzione del modello *Fast R-CNN*. La principale differenza risiede nel fatto che mentre quest'ultimo utilizza una ricerca selettiva per ricercare le *RoI*, la nuova versione utilizza una *RPN*[13].

Tale rete, prende in input un'immagine (di qualunque dimensione) e genera come output un insieme di rettangoli con proposte di oggetti, ognuno dei quali con un determinato punteggio *objectness*[14]. Questo nuovo modello creato da *Shaoqing Ren et al.* è composto da due principali moduli. Il primo, come anticipato precedentemente, è l'*RPN* mentre il secondo è il rilevatore del *Fast R-CNN*.

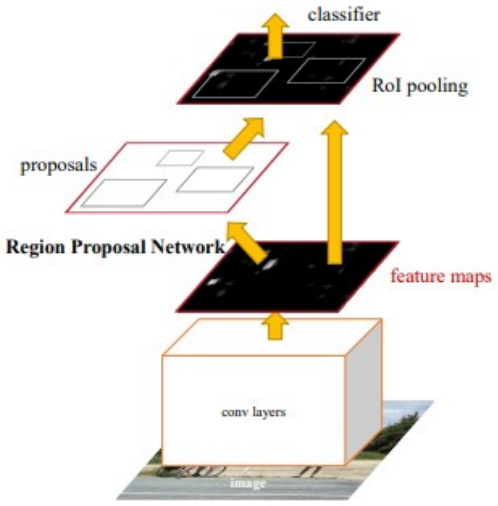


Figura 1.7: Architettura *Faster R-CNN*

Il riconoscimento degli oggetti, attraverso il *Faster R-CNN*, deve seguire determinati step: in prima battuta l'immagine viene inserita in input all'interno di *layers convoluzionali* che generano una mappa di *features*, successivamente l'*RPN* prende tale mappa e identifica le proposte di oggetti con i relativi punteggi, in seguito un *RoI pooling layer* viene applicato ad ognuna delle proposte per renderle della stessa dimensione ed infine le *RoI* vengono fatte passare attraverso una rete completamente connessa che classifica gli oggetti e crea i loro box.

Il modulo *RPN* viene utilizzato soprattutto al fine di ridurre la potenza computazionale e di aumentare la velocità di predizione dell'immagine (circa 0.2 secondi). Una volta che tale modulo

[13]Region Proposal Network
[14]Misura l'appartenenza ad un insieme di classi oggetti rispetto allo sfondo

riceve in input la *feature map* dalla *CNN*, utilizza una *sliding window* e genera per ogni finestra degli *anchor boxes* di differenti forme e dimensioni. Per tutti questi *boxes* creati, *RPN* predice due elementi: la probabilità che all'interno di questi vi sia un oggetto e la regolazione da effettuare al box stesso con lo scopo di adattarlo meglio all'oggetto. In seguito tutti questi valori vengono passati al *RoI pooling layer* che effettua le operazioni illustrate nei paragrafi precedenti[27].

Addestrando Faster R-CNN con i dataset COCO, VOC 2007 e VOC 2012, si ottiene una mAP del 78.8%[23].

Feature Pyramid Networks[12]

Feature Pyramid Networks è un estrattore di *features* che prende in input un'immagine di dimensioni arbitrarie e produce mappe di caratteristiche aventi dimensioni proporzionali a più livelli.

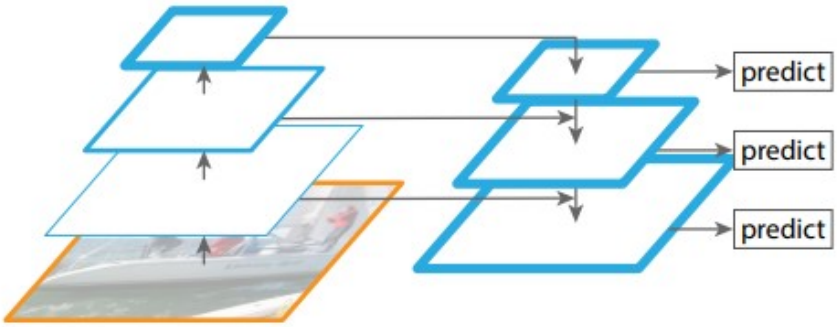


Figura 1.8: *Feature Pyramid Networks* - le *feature maps* sono rappresentate come piani con contorno azzurro. Quest'ultimo più è spesso e più il valore semantico delle *feature maps* è maggiore

La costruzione della piramide si struttura due percorsi: *bottom-up pathway* e *top-down pathway*. Il primo si riferisce ad una rete convoluzionale che estrae delle caratteristiche a diverse scale con un passo di scala di 2. È facile intuire che all'aumentare della vicinanza con il vertice della piramide, la risoluzione spaziale diminuisce, mentre il valore semantico cresce. Il percorso *top-down* invece, partendo da uno strato semantico ricco, costruisce degli strati dell'immagine a risoluzione più alta. Anche se le ricostruzioni sono semanticamente forti, le posizioni degli oggetti non sono molto precise, per tale motivo si aggiungono delle connessioni laterali al fine di aiutare il rilevatore a prevedere in modo più preciso la posizione degli oggetti. Ogni connessione laterale unisce mappe di caratteristiche della stessa dimensione spaziale dei due percorsi. Per riassumere, *FPN* combina *features* di bassa risoluzione, ma con elevato valore semantico, con *features* ad alta risoluzione, ma con basso valore semantico.

È opportuno sottolineare che *FPN* è solamente un estrattore di *features* che lavora con altri *object detectors*. Molto spesso tale modello viene combinato con il *Faster R-CNN*, ottenendo buoni risultati (*COCO13* mAP@.5 = 59.1%).

1.3.2 One-stage detector

I modelli one-stage detector contengono una singola rete convoluzionale che fornisce direttamente i box e la classificazione degli oggetti[5].

YOLO[20]

YOLO[15] è stato il primo modello presentato in ordine cronologico di tipo *one-stage*. Tale *object detector* permette di riconoscere diversi oggetti e contemporaneamente di individuare nello spazio la loro posizione attraverso un singolo passaggio dell'immagine. Per fare ciò, *YOLO* divide l'immagine in regioni, predice i *boundary boxes* e successivamente calcola la probabilità che tali elementi appartengano ad una determinata classe.

[15] You Only Look Once

Nel corso degli anni si sono sviluppate diverse versioni come *YOLOv2*[21], *YOLOv3*[22] e *YOLOv4*[3] (quest'ultima versione sarà illustrata con maggior approfondimento nella sezione 2.1).

YOLO risulta essere più veloce rispetto agli altri modelli. Il modello base può infatti raggiungere i 45 *frames per second* nel processare le immagini in real time. Inoltre, esiste una versione ancora più veloce (*fast YOLO*) che può raggiungere i 155 *frames per second*. Per la versione a 45 *fps* la mAP = 63.4% su VOC07.

SSD[15]

Al pari di *YOLO*, il modello *SSD*[16] è formato da una rete in grado di localizzare e di identificare un oggetto all'interno di un'immagine in un singolo passaggio.

La sua architettura presenta sostanzialmente due componenti: *backbone* e *SSD head*. La prima corrisponde ad una rete di classificazione preaddestrata in grado di estrarre il significato semantico dell'immagine, preservando la sua struttura spaziale anche ad una risoluzione inferiore. La seconda, invece, è composta da uno o più strati convoluzionali con i quali si riconoscono le classi degli oggetti e i loro *bounding boxes*.

SSD si basa su una rete convoluzionale *feed-forward*[17] che produce una collezione di *boundary boxes* di dimensioni fisse e un punteggio relativo alla presenza di un oggetto. Successivamente vi è una fase di soppressione che affina le rilevazioni. A tale struttura basica si possono aggiungere delle strutture ausiliarie, come per esempio quelle utilizzate per il rilevamento di oggetti su multi-scala.

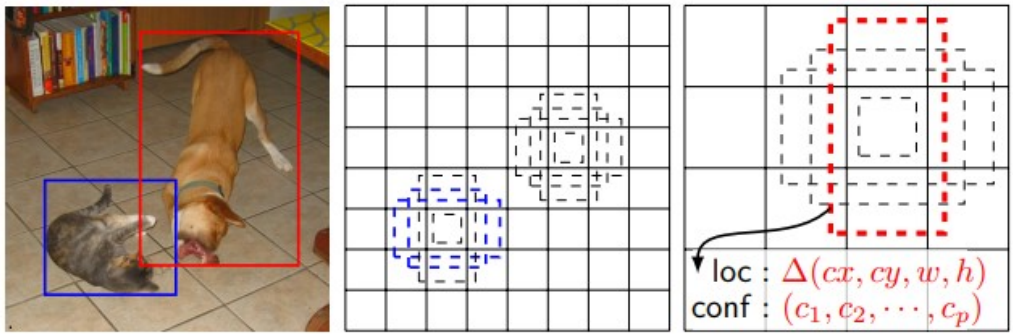


Figura 1.9: *SSD framework*[15]

Nella figura 1.9, è possibile notare la struttura di funzionamento di *SSD*. Durante l'addestramento sono necessari due elementi: un'immagine in input ed i box contenenti gli oggetti da individuare. L'*SSD* valuta un set di caselle predefinite con diverso rapporto forma in differenti scale (nel caso della figura le scale sono *8x8* e *4x4*). Per ogni casella, si prevedono sia gli *offset* di forma sia le confidenze di tutte le categorie degli oggetti. Durante l'addestramento queste caselle predefinite vengono confrontate con i *boxes* iniziali. In tal modo si riesce a calcolare la perdita di locazione e la perdita di fiducia.

I vantaggi apportati dall'*SSD* sono sicuramente la velocità (59 *fps*) e l'accuratezza (*VOC07* mAP=76.8% e *VOC12* mAP=74.9%) nel riconoscere gli oggetti.

RetinaNet[13]

Il modello *RetinaNet* fu sviluppato da *T.-Y. Lin et al.*. L'analisi della ricerca si focalizzò soprattutto sul divario esistente dal punto di vista dell'accuratezza tra i modelli *one-stage* e *two-stage*.

I ricercatori scoprirono che la causa principale si celava nell'addestramento. Per questo, è stata introdotta una nuova funzione denominata *focal loss* che permette al rilevatore di concentrarsi maggiormente sugli esempi difficili ed erroneamente classificati[33].
Grazie alla figura 1.10 è possibile illustrare l'architettura del modello *RetinaNet*. Quest'ultimo è composto da una piramide di *features* (*FPN*) posizionata sopra ad un'architettura *feed-forward*, *ResNet*

[16] Single Shot Detector
[17] rete nella quale le connessioni tra i nodi non formano un ciclo

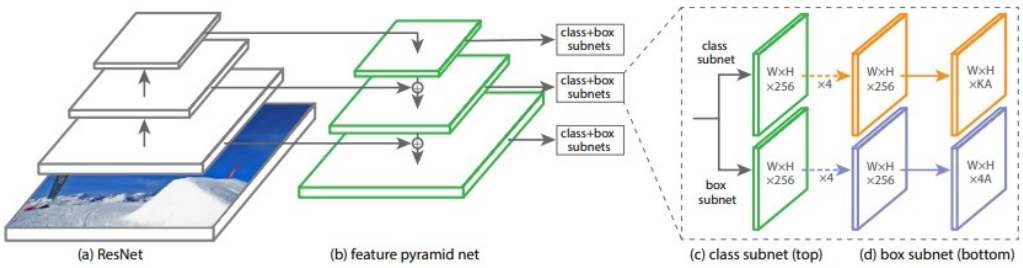


Figura 1.10: Architettura di *RetinaNet*[13]

(a), utilizzata per creare una piramide convoluzionale multi-scala, *feature pyramid net* (b). Successivamente, *RetinaNet* si avvale di due sottoreti: una in grado di classificare la *anchor box* e una in grado di definire i box di ogni oggetto.

Grazie alla semplicità di questa nuova rete, si è eliminato il divario in merito all'accuratezza dei rilevatori a due stadi. In aggiunta, il nuovo modello si connota di una velocità superiore. *RetinaNet* ha conseguito nel *COCO* un mAP = 59.1%.

1.4 YOLO

Come anticipato nella sezione 1.3.2, *YOLO* costituisce uno dei migliori modelli per l'*object detection* di tipo *one-stage*. *Joseph Redmon et al.* hanno sviluppato le prime tre versioni di *YOLO* (la prima di esse risale al 2016), mentre *Alexey Bochkovskiy et al.* hanno creato *YOLOv4*.

Prima del 2016 gli *object detector* prendevano l'immagine di input, la scomponevano in molte regioni ed eseguivano un classificatore per ogni regione da analizzare. Nel caso in cui il punteggio di tale classificatore fosse risultato elevato, si riteneva rilevato l'oggetto in questione. Considerata la numerosità delle regioni, l'applicazione di un rilevatore per ogni regione d'interesse rendeva l'*object detection* estremamente complesso e laborioso. Con *YOLO*, invece, l'intero processo viene svolto da una singola rete in grado di analizzare l'immagine solamente una volta (ecco spiegato il senso del nome *You Only Look Once*). Pertanto, è possibile aumentare la velocità di processazione dell'immagine in modo tale da riuscire ad adattare il modello anche per l'elaborazione di video in *real-time*.

In virtù delle sue caratteristiche, *YOLO* può essere applicato a qualunque dominio (veicoli a guida autonoma, riconoscimento cellule tumorali ecc.) e può essere eseguito anche solo utilizzando la potenza di calcolo di un cellulare (con specifiche ottimizzazioni e semplificazioni).

Le informazioni e le immagini riportate nelle sezioni successive sono state reperite all'interno del *paper* ufficiale del modello[20].

1.4.1 Detector

La rete di *YOLO* utilizza le caratteristiche dell'intera immagine per predire i *boundary boxes* attraverso un processo simultaneo che permette alla rete di ragionare sull'intera immagine e su tutti gli oggetti della stessa. Il modello preleva l'immagine in input e la divide in una griglia *S x S* (per esempio *19 x 19*).

Se il centro di un oggetto si trova all'interno di una cella, essa sarà responsabile del suo rilevamento. Ogni cella prevede *B bounding boxes* per ogni oggetto e calcola il punteggio di confidenza corrispondente. Tale punteggio fa riferimento alla probabilità che all'interno del *box* vi sia un oggetto e all'accuratezza del box predetto. Tale valore di confidenza è definito come $Pr(Oggetto) * \text{IOU}^{\text{truth}}_{\text{pred}}$. *IOU*[18] è una metrica di valutazione utilizzata per stimare la precisione di un *object detector* in un particolare dataset. Per calcolare tale metrica si applica la seguente formula:

$$IOU = \frac{Area\ di\ Overlap}{Area\ di\ unione}$$

Di norma, un punteggio dell'*intersection over union* > 0.5 è considerato un buon risultato[25].

[18] intersection over union

I *bounding boxes* contengono cinque predizioni ognuno *(x, y, w, h, confidence)*. In particolare, x e y sono le coordinate che rappresentano il centro del box all'interno della cella. Al contrario, la larghezza (w) e l'altezza (h) si riferiscono all'intera immagine. Inoltre, ogni cella predice C probabilità di appartenenza ad una determinata classe $Pr(Class_i \mid Oggetto)$. Al momento del test, si moltiplica la probabilità condizionata di appartenenza ad una classe per la confidenza individuale di predizione del box:

$$Pr(Classe_i \mid Oggetto) * Pr(Oggetto) * \text{IOU}^{\text{truth}}_{\text{pred}} = Pr(Classe_i) * \text{IOU}^{\text{truth}}_{\text{pred}}$$

Tale formula permette di calcolare la probabilità che una determinata classe appaia all'interno di un box e il livello precisione con cui il box delimita l'oggetto.

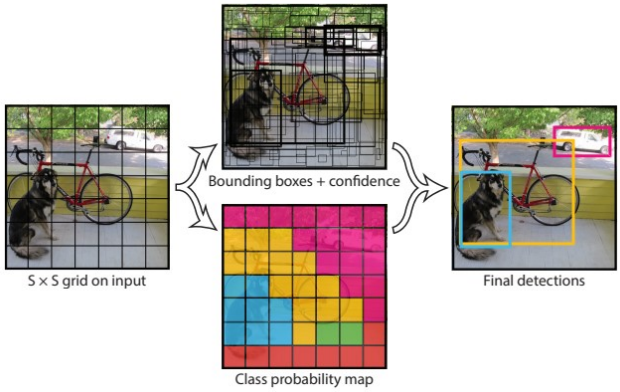


Figura 1.11: Detector *YOLO*

Un problema frequente consiste nell'identificazione di un oggetto molteplici volte. *YOLO* utilizza la tecnica *Non-Max Suppression* al fine di rilevare l'oggetto solamente una volta.

Per comprendere al meglio il suo funzionamento, è opportuno riportare un esempio:

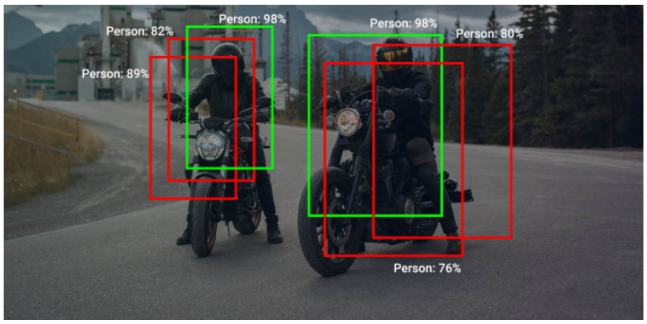


Figura 1.12: Rilevamento oggetto molteplici volte[28]

L'obiettivo è quello di scegliere il "miglior" *bounding box* (ossia quelli di colore verde nell'immagine) ed eliminare tutti gli altri (ossia quelli di colore rosso). Per riuscire nell'intento, vengono presi in considerazione due elementi: l'*IOU* tra i *boundary boxes* e l'*objectiveness score*. Quest'ultimo è in grado di mostrare quanto è sicuro che l'oggetto desiderato sia contenuto nel box.

La *non-max suppression* seleziona il box con il punteggio di *objectiveness* più alto (nel caso dell'immagine 1.12, i *boxes* di colore verde) e successivamente rimuove gli altri *boxes* (di colore rosso) che presentano un elevato *overlap* con quello principale. In tal modo, i *boxes* scelti alla fine della detection sono quelli che più si adattano all'oggetto in questione (nel caso dell'immagine, alle due persone)[28].

1.4.2 Architettura

La prima versione di *YOLO* presentava un'architettura di ventiquattro *layer convoluzionali* seguiti da due strati completamente connessi. Inoltre, è stata definita una versione denominata "*Fast YOLO*"

che presenta solamente nove layer convoluzionali e meno filtri nei rispettivi strati. Il livello di output finale, illustrato nella figura 1.13, è 7 x 7 x 30.

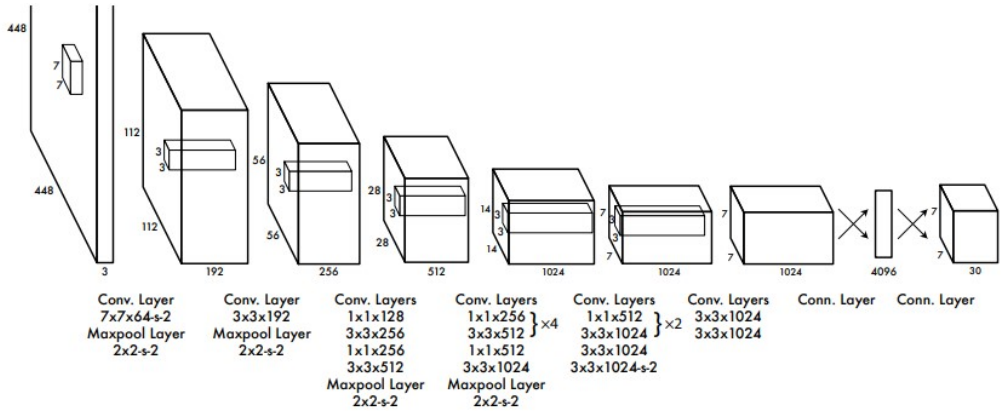


Figura 1.13: Illustrazione dell'architettura di *YOLO*

2 YOLO - immagini da un drone

In questo capitolo verranno illustrate, con maggiori dettagli, la versione *4* e la versione *tiny YOLOv3* dell'*object detector* denominato *YOLO*.

Questi modelli sono stati addestrati al fine di riconoscere diversi oggetti presenti in fotografie scattate da droni.

2.1 YOLOv4 [3]

YOLOv4 è un modello di *object detection* sviluppato da *Alexey Bochkovskiy et al.* nell'aprile del 2020. Il precedente sviluppatore di *YOLO*, *Joseph Redmon*, decise di abbandonare il progetto a causa delle implicazioni legate al suo utilizzo e alla presenza di problemi etici (per esempio l'adozione della tecnologia in campo militare).

Il modello *YOLOv4* presenta un miglioramento notevole delle performance rispetto alle sue versioni precedenti. La figura 2.1 dimostra che, ponendo a confronto *YOLOv4* con *YOLOv3* e altri modelli, l'accuratezza (*AP*) e i *frame per second* (*FPS*) di *YOLOv4* sono aumentati rispettivamente del 10% e del 12% rispetto a *YOLOv3*.

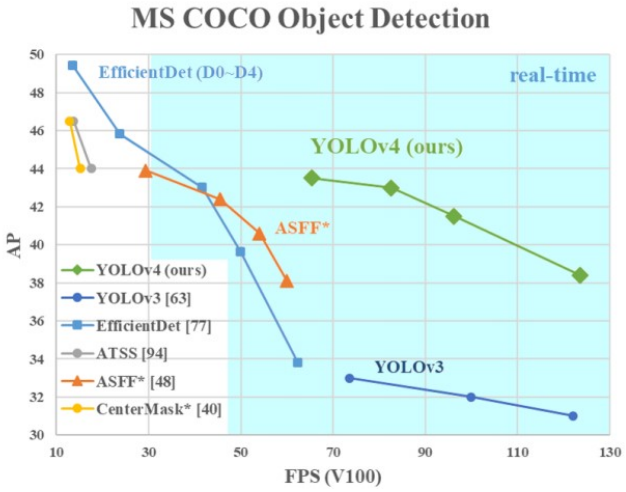


Figura 2.1: Performance di *YOLOv4* comparato ad altri *object detectors*

Il miglioramento delle *performance* si è ottenuto grazie all'introduzione di una nuova *Backbone* e alla modifica del *Neck*. Tali espedienti resero più agevole l'addestramento e facilitarono possibile il *training* anche con una singola *GPU*[1].

2.1.1 Backbone

La *backbone* è una rete neurale composta principalmente da *layers* convoluzionali ed è utilizzata al fine di estrarre le *features* essenziali da un'immagine.

Molto spesso la scelta di una determinata *backbone* rispetto ad un'altra consente di aumentare le prestazioni di un *object detector*.

La *backbone* di YOLOv4 è formata da 3 componenti: *bag of freesbies*, *bag of specials* e *CSPDarknet53*.

[1] Graphics processing unit

Bag of freesbies

La *bag of freesbies* è costituita da una serie di metodi di addestramento in grado di incrementare l'accuratezza del rilevatore senza aumentare il costo di inferenza.

Molti dei metodi, che rientrano all'interno della *bag of freesbies*, vengono impiegati per accrescere il numero dei dati. L'obiettivo primario di tale necessità è quello di aumentare la variabilità di un'immagine e quindi la robustezza del modello nel rilevamento degli oggetti contenuti nelle immagini.

Tra i processi di *data augmentation*, è opportuno menzionare il *photometric distortion* e il *geometric distorsion*. Il primo metodo, partendo da un'immagine in input, crea nuove immagini regolando la luminosità, il contrasto, la tonalità, la saturazione e il rumore. Il secondo, invece, viene utilizzato al fine di ruotare, capovolgere e ridimensionare l'immagine di partenza. Le regolazioni eseguite da questi due metodi sono di tipo *pixelwise*[2] e tutte le informazioni relative al *pixel* originale, presente nell'area elaborata, vengono mantenute.

Esistono in letteratura ulteriori soluzioni che, diversamente da quelle sopra citate, sono incentrate nella simulazione di problemi di occlusione e sono impiegate per migliorare i risultati nella classificazione e nel rilevamento degli oggetti. Tra queste è opportuno citare: *CutOut*, *MixUp*, *CutMix* e *Mosaic*.

La *CutOut*[6] è una tecnica focalizzata nel riempimento di una porzione dell'immagine, selezionata casualmente, con un valore random o complementare di zero. Attraverso il metodo *MixUp*[30] è possibile creare una nuova immagine attraverso l'interpolazione lineare ponderata di due immagini esistenti. Anche le *labels* degli oggetti presenti all'interno delle immagini sono regolate in maniera ponderata a seconda dei diversi coefficienti. La strategia denominata *CutMix*[29] è una combinazione delle due tecniche illustrate precedentemente e consiste nel posizionare all'interno di un'immagine la porzione di una fotografia presente nel *dataset*. Le relative *labels* sono mescolate proporzionalmente rispetto all'area della *patch*. In questo modo si è in grado di aumentare la robustezza del modello riuscendo a contrastare la corruzione dei dati e ad incrementare l'accuratezza nel rilevamento di immagini al di fuori della distribuzione. L'ultimo metodo di *data augmentation* è chiamato *Mosaic*. Quest'ultimo, a differenza della *CutMix*, unisce quattro porzioni di differenti immagini.

Un'altra tecnica che può essere inserita nella *bag of freesbies* è la *DropBlock*. Attraverso di essa, le caratteristiche di un blocco (regione contigua di una mappa di caratteristiche) sono scartate. Questo metodo viene utilizzato al fine di indurre la rete ad imparare da *features* su cui altrimenti non potrebbe fare affidamento. Per esempio, l'*object detector* deve essere in grado di riconoscere un cane sia dalla testa ma anche dal suo dorso. La tecnica *DropBlock* è largamente utilizzata anche per evitare l'*overffitting*[3].

Tale problematica può essere aggirata adoperando un ulteriore meccanismo chiamato *label smoothing*. Quest'ultimo viene spesso implementato al fine di aggiungere un rumore alle *labels*, in tal modo è possibile modificare il limite superiore della predizione ad un valore più basso. Per citare un esempio, avendo la foto di un cane con la relativa label `[0 1]` (assoluta certezza che vi è un cane), attraverso la *label smoothing* tale etichetta sarà modificata con i valori `[0:5 0:95]`.

Tutti i metodi illustrati fino ad ora possono essere inseriti all'interno di una *bag of freesbies* di un *object detector*. La *BoF*[4] della *backbone* di *YOLOv4* contiene quattro tecniche di addestramento: *CutMix*, *Mosaic*, *DropBlock regularization* e *label smoothing*.

Bag of Special

La *bag of specials* è un'altra importante componente della *backbone*. Come menzionato da *Alexey Bochkovskiy et al.*, quest'ultima consiste in un insieme di *plugins* e metodi di *post-processing* in grado di incrementare la precisione dell'*objects detection* solamente con un contenuto aumento del costo di inferenza.

[2] Un *pixel* alla volta

[3] Problematica nella quale un modello si adatta ai dati osservati (il campione) perché ha un numero eccessivo di parametri rispetto al numero di osservazioni

[4] Bag of Freesbies

Le tecniche presenti all'interno della *bag of specials* della *backbone* di *YOLOv4* sono: *Mish activation*[18], *MiWRC*[5] e *CSP*[6].

Quando si sviluppa la *backbone* di un *object detector*, è possibile utilizzare diverse *activation functions*. Queste ultime vengono implementante al fine di ottenere l'output di un nodo rispetto al valore in ingresso. La *ReLU*[7] è una delle funzioni di attivazione maggiormente utilizzata e viene definita come una funzione identità lineare per i valori positivi, mentre assume un valore pari a zero per tutti quelli negativi. Tali caratteristiche la rendono una componente a basso carico computazionale in grado di ridurre i tempi per l'addestramento. Una problematica che può sorgere con l'adozione della funzione *ReLU* riguarda la *dying ReLU* ossia la "morte" di alcuni neuroni che hanno prodotto valori negativi e che quindi sono mappati con il valore zero. Una volta che un neurone diventa negativo, è infatti improbabile che cambi il suo stato e che partecipi alla discriminazione dell'*input*. Per ovviare a tale problematica, sono state sviluppate diverse funzioni alternative. Una di queste, utilizzata all'interno del modello *YOLOv4*, è la *Mish activation* definita $f(x) = x \tanh(softplus(x))$. Le sue caratteristiche aiutano a prevenire il problema di *dying* in quanto permettono di conservare i valori negativi, di apprendere determinate *features* che altrimenti verrebbero perse e di evitare la saturazione (una delle cause principali che provoca il rallentamento dell'addestramento). Anche se la funzione *Mish* presenta un alto costo computazionale, l'utilizzo di quest'ultima consente di aumentare l'accuratezza rispetto alle funzioni classiche di attivazione (utilizzando *Mish* con *YOLOv4* e *CSPDarkNet53* si ha un'accuratezza maggiore del 2,1% rispetto all'utilizzo di *Leaky ReLU*).

All'interno della *bag of specials* di *YOLOv4* vi è un metodo chiamato *cross-stage partial connections* che è in grado di ridurre la complessità computazionale. Per raggiungere tale intento, il metodo in esame divide la *feature map* di *input* in due porzioni: una aggira il *DenseBlock*[8] e diventa l'*input* del successivo strato di transizione, mentre l'altra entra all'interno del *DenseBlock* e viene processata.

CSPDarknet53

La *CSPDarknet53* è una *backbone* e una rete neurale convoluzionale basata su *Darknet53*. Inizialmente gli sviluppatori di YOLOv4 si trovarono a scegliere tra due tipologie di architetture: *CSPDarknet53* e *CSPResNext50*. In seguito a numerosi studi, emerse che *CSPResNext50* era migliore in termini di classificazione di oggetti nel set di dati *ILSVRC2012* rispetto a *CSPDarknet53*. Quest'ultima, tuttavia, risulta più prestante in termini di rilevamento degli oggetti nel dataset *MS COCO*. La scelta finale ricadde su *CSPDarknet53*.

L'estrattore di *features Darknet53*, già in uso nella versione *YOLOv3*, presenta cinquantatré layers convoluzionali. La sua struttura è mostrata nella figura 2.2. Il modello *YOLOv4* sfrutta lo stesso *features extractor* della versione tre, ma con l'aggiunta del *CSP*. Come illustrato nella sezione precedente, la strategia *CSP* viene impiegata al fine di partizionare la *features map* in due porzioni per poi fonderle attraverso una gerarchia trasversale. In questo modo si possono eliminare i colli di bottiglia computazionali e si migliora l'apprendimento attraverso il trasferimento di una versione non modificata della mappa di caratteristiche allo strato di transizione.

2.1.2 Neck

Il *neck* è un componente di *YOLOv4* che integra e combina le *feature maps* provenienti dalla *backbone* al fine di prepararle alla fase di rilevamento.

All'interno del *neck* sono presenti due elementi: *SPP*[11] e *PAN*[14].

SPP

I diversi filtri presenti all'interno della *backbone* creano delle *feature maps*. Tali caratteristiche alimentano il modulo *SPP* che, prescindendo dalla dimensione delle *feature maps*, è in grado di sviluppare *features* di dimensione fissa. Per effettuare tale operazione, *SPP* utilizza un *max pooling layer* al fine di ottenere diverse rappresentazioni delle *feature maps* in *input*. I *kernel* del *pooling layer* in questione

[5]Multiinput weighted residual connections
[6]Cross-stage partial connections
[7]Rectified Linear Unit
[8]Modulo usato nelle reti neurali convoluzionali che collega tutti gli strati direttamente tra loro

	Type	Filters	Size	Output
	Convolutional	32	3 × 3	256 × 256
	Convolutional	64	3 × 3 / 2	128 × 128
1×	Convolutional	32	1 × 1	
	Convolutional	64	3 × 3	
	Residual			128 × 128
	Convolutional	128	3 × 3 / 2	64 × 64
2×	Convolutional	64	1 × 1	
	Convolutional	128	3 × 3	
	Residual			64 × 64
	Convolutional	256	3 × 3 / 2	32 × 32
8×	Convolutional	128	1 × 1	
	Convolutional	256	3 × 3	
	Residual			32 × 32
	Convolutional	512	3 × 3 / 2	16 × 16
8×	Convolutional	256	1 × 1	
	Convolutional	512	3 × 3	
	Residual			16 × 16
	Convolutional	1024	3 × 3 / 2	8 × 8
4×	Convolutional	512	1 × 1	
	Convolutional	1024	3 × 3	
	Residual			8 × 8
	Avgpool		Global	
	Connected		1000	
	Softmax			

Figura 2.2: Architettura *Darknet53*[22]

sono di quattro dimensioni: `1x1`, `5x5`, `9x9` e `13x13`. Utilizzando un kernel relativamente grande, si è dimostrato che l'efficacia del campo recettivo della *backbone* aumenta. Infine, le nuove *feature maps* create dai diversi *max pooling layers* sono combinate tra di loro creando mappe di caratteristiche a lunghezza fissa.

PAN

Nei primi metodi di *deep learning* venivano utilizzate delle reti in cui l'*input* passava in successione all'interno di diversi *layers*. I primi strati estraevano le informazioni semantiche necessarie per gli strati successivi. Tuttavia, man mano che si procedeva all'interno degli strati, alcune informazioni in grado di migliorare le predizioni venivano perse. Per ovviare a tale problematica, è stato sviluppato il modulo *PAN* in grado di consentire una migliore propagazione delle informazioni prodotte dai primi strati. Se nel classico *PAN* le informazioni del *layer* corrente sono sommate con quelle del livello precedente per formare un nuovo vettore, in *YOLOv4* tali informazioni vengono concatenate insieme.

2.1.3 Head

La *head* presente nei metodi *one-stage detector* è il componente che esegue la *dense prediction*. Quest'ultima corrisponde alla previsione finale dell'*object detector* ed è composta dalle coordinate del *bounding box*, dalla confidenza della predizione e dalla *label* contenente il nome dell'oggetto identificato.

YOLOv4 utilizza la stessa *head* della versione *YOLOv3* che è basata sul rilevamento degli *anchor boxes*, ossia un insieme di caselle predefinite con diverse dimensioni e *aspect ratios*. Tali caselle vengono impiegate per rilevare e localizzare gli oggetti all'interno delle immagini.

Anche la head di *YOLOv4* presenta la *bag of freesbies* e *la bag of specials*.

Bag of freesbies

Come accennato in precedenza, la *bag of freesbies* è composta da diversi metodi che vengono adoperati nell'*object detector* in questione. Di seguito vengono analizzati i metodi principali.

YOLOv4 utilizza la *CIoU-loss*[31] a discapito della semplice *IOU* (illustrata nella sezione 1.4.1).

La nuova funzione introduce due concetti: il primo è la distanza tra il centro del box predetto e il punto centrale del box reale, mentre il secondo è l'*aspect ratio*, il quale confronta le proporzioni del box reale con quello rilevato. Attraverso la distanza tra i punti centrali, l'*aspect ratio* e la *IOU* è possibile misurare la qualità del box predetto.

La formula per calcolare la *CIoU-loss* è definita nel seguente modo:

$$\mathcal{L}_{CIoU} = 1 - IoU + \frac{\rho^2(b, b^{gt})}{c^2} + \alpha v$$

dove b e b^{gt} corrispondono rispettivamente al centro del box predetto e a quello reale, ρ alla distanza Euclidea, c alla lunghezza della diagonale del più piccolo *box* coperto dai due *boxes* principali, $\alpha = \frac{v}{(1-IoU)+v}$ è un parametro di *trade-off* e $v = \frac{4}{\pi^2}(\arctan\frac{w^{gt}}{h^{gt}} - \arctan\frac{w}{h})^2$ rappresenta la coerenza tra gli *aspect ratios*.

Un altro metodo presente nella *bag of freesbies* della *head* è la *CmBN*[9]. Normalmente si sfrutta una *Batch normalization* al fine di calcolare la media e la varianza dei campioni all'interno di una *mini-batch* per ripulire l'input del livello. Nel caso in cui la dimensione della *mini-batch* sia ridotta, le stime calcolate rischiano di essere molto rumorose. Per risolvere tale problema, è stata inserita la *cross mini-batch normalization*. Quest'ultima impiega le stime delle *batches* recenti, allo scopo di migliorare la qualità delle stime di ogni batch. Di fatto si rilevano le statistiche dell'intera *batch* invece di raccoglierle all'interno delle *mini-batch*. Tuttavia, poiché i pesi cambiano ad ogni iterazione, le informazioni estratte dai *weights* possono diventare imprecise sotto i pesi più recenti. Al fine di ovviare tale complicazione, vengono utilizzati i polinomi di *Taylor*.

Un ulteriore metodo di *data augmentation* presente nella *head* è la *SAT*[10]. Tale tecnica, opera in due fasi ben distinte. Nella prima altera l'immagine originale invece che i *weight*, in modo tale da indurre la rete a pensare che non ci sia l'oggetto desiderato. Nella seconda fase, il modello viene addestrato con questa nuova immagine tramite il *boundary box* originale e l'etichetta. La *SAT* viene dunque utilizzata al fine di generalizzare il modello ed evitare l'*overfitting*.

Una diversa tecnica presente nel *detector* di *YOLOv4* è quella di predire differenti *anchors* per un singolo oggetto. Attraverso l'utilizzo della *features map* proveniente dagli strati convoluzionali, il *detector* crea svariati *anchor boxes* di diversi rapporti in modo tale da poter rappresentare oggetti di qualsiasi dimensione. Successivamente, utilizzando la *IoU*, il rilevatore seleziona i *boxes* migliori secondo la formula $IoU(verità, anchor) > soglia\ IoU$.

In conclusione, un metodo presente all'interno della *bag of freesbies* è la *Random training shapes*. Per migliorare la generalizzazione, il modello è allenato analizzando immagini con dimensioni diverse (*Multi-Scale Training*).

Bag of specials

All'interno della *bag of specials* del *detector* di *YOLOv4* sono presenti alcuni metodi come ad esempio il *mish activation*(citato precedentemente), il *SAM-block* e la *DIoU-NMS*.

Nei metodi di *deep lerning* è comune utilizzare strati denominati *layers of attention*. Nel caso di *YOLO*, tali strati sono impiegati al fine di evidenziare le caratteristiche più importanti (create dagli strati convoluzionali) e di rimuovere quelle trascurabili. *SAM*[11] consiste nell'applicare un *max pooling* e un *avg pooling* alla *features map* in uscita da uno strato convoluzionale. Successivamente queste due mappe di caratteristiche vengono combinate e fatte passare all'interno di uno strato convoluzionale. Infine, si applica una funzione *sigmoidea* con lo scopo di evidenziare le caratteristiche più rilevanti. In *YOLOv4*, tale modulo è stato modificato eliminando gli strati di *max* e *avg pooling*.

La *DIoU-NMS* è una tipologia di *non-maximum suppression* nella quale vengono prese in considerazione l'area di *overlap* e la distanza tra i punti centrali dei *boxes* in modo tale da verificare se i *boxes* sono ridondanti.

$$s_i = \begin{cases} IoU - \mathcal{R}_{DIoU}(\mathcal{M}, B_i) < \epsilon \\ IoU - \mathcal{R}_{DIoU}(\mathcal{M}, B_i) \geq \epsilon \end{cases}$$

Se il tasso di sovrapposizione al netto della distanza tra i due centri risulta essere inferiore ad una soglia si mantiene il box, altrimenti si elimina.

2.2 iTelescope

In questa sezione si procederà ad illustrare il progetto *iTelescope*. Quest'ultimo consiste nella creazione di un programma *desktop* che analizza e riconosce diverse classi di oggetti all'interno di immagini fotografate da droni.

[9] Cross mini-Batch Normalization
[10] Self-Adversarial Training
[11] Spatial Attention Module

Tale applicativo, con le dovute modifiche e autorizzazioni, può essere impiegato per scopi di sicurezza e controllo delle strade cittadine. Infatti, attraverso l'utilizzo di un drone in grado di scattare foto e di girare un video, sarà possibile in pochi istanti controllare se all'interno delle immagini vi è la presenza di determinati oggetti e la loro quantità.

La spiegazione di *iTelescope* è suddivisa in diverse sezioni: *training*, *testing* e creazione dell'applicativo.

2.2.1 Training

Molto spesso quando si vuole realizzare un'applicazione con un *object detector*, bisogna eseguire il *training* del modello sulla base delle immagini che devono essere processate.

La fase di addestramento è composta da diversi *step*, in ognuno dei quali il sistema analizza le immagini in input e produce delle predizioni. Ad ogni ciclo, i parametri di addestramento sono modificati al fine di aumentare l'accuratezza del modello.

In alcuni casi, tale processo può essere ignorato in quanto esistono in letteratura svariati *weights* precalcolati che consentono l'*object detection*. Nel caso di specie, tale processo non può essere ignorato perché le immagini provenienti dai droni sono molto specifiche.

Quando si vuole addestrare un *object detector*, è fondamentale ricercare un numero elevato di immagini finalizzate ad alimentare il *dataset* del modello. In *iTelescope* le fotografie sono state reperite dalla competizione *VisDrone2019*[32] e sono state impiegate per addestrare e testare il modello tramite l'utilizzo di un file di testo (*label*) che descrive tutti gli oggetti e la loro posizione in esse. *YOLOv4* ha un suo preciso standard per quanto riguarda le *labels*:

```
<id-object> <x> <y> <width> <height>
```

Per `id-object` si intente la classe dell'oggetto. Nel *dataset* preso in esame, le classi sono dieci (pedoni, persone, biciclette, macchine, van, camion, tricicli, tricicli motorizzati, bus, moto) e gli *id* sono compresi in un range che va da zero a nove. La `x` e la `y` si riferiscono alle coordinate omonime del centro del box contente l'oggetto, normalizzate tra zero e uno. La `width` e la `height` corrispondono alla larghezza e all'altezza del box, anch'esse normalizzate tra zero e uno.

Le *labels* associate alle immagini della *VisDrone2019* non possiedono in principio il formato di *YOLOv4*, quindi sono state modificate al fine di poterle adattare al modello in questione. Le suddette immagini presentano molteplici oggetti al loro interno, in media sessanta. Di seguito ne è riportato un esempio:

Figura 2.3: Esempio immagine dataset[32]

Come affermato in precedenza, l'obiettivo di *iTelescope* è quello di individuare il maggior numero di oggetti nelle immagini e di localizzarli all'interno di esse.

Tali fotografie vengono divise in due gruppi: *training set* e *testing set*. Il *training set* è composto dalle immagini impiegate per l'apprendimento del modello *YOLOv4*. In questo gruppo vengono inserite la maggior parte delle fotografie a disposizione (in *iTelescope* sono l'85%, ossia *6471* immagini). Tale insieme di immagini viene utilizzato per adattare i parametri (*weights*) al fine di migliorare

l'apprendimento e la rilevazione. All'interno del *set* in questione, è necessario che ogni oggetto venga rappresentato in molteplici situazioni diverse, così da rendere il *training* il più generalizzato possibile.

Dopo l'addestramento, si utilizza il *test set* (contenente nel caso in questione il restante 15% delle immagini, ossia *1142* fotografie) per calcolare l'accuratezza e l'errore del modello realizzato. Le *labels* di ogni immagine, in questa fase, vengono nascoste all'*object detector*. Ad ogni fotografia viene applicato il classificatore per effettuare la predizione che in seguito sarà confrontata con la *label* corretta. Così facendo, è possibile determinare l'accuratezza del modello.

Sviluppare algoritmi di *deep learning*, utilizzanti *dataset* di ingenti dimensioni, necessita una notevole potenza di calcolo. Ecco perché molto spesso è utile affidarsi a servizi in *cloud*. Nel caso di *iTelescope*, l'addestramento è stato realizzato con `Google Colaboratory`. Tale piattaforma consente di eseguire codice *Python* all'interno di blocchi di esecuzione. *Colab* viene eseguito su una macchina virtuale *Linux*, il che rende possibile l'esecuzione di comandi *Linux* direttamente all'interno dei blocchi. Questa piattaforma è molto sfruttata per le applicazioni di *machine learning* in quanto permette di definire la tipologia di *runtime*. Nel caso in questione è preferibile utilizzare la *GPU* per rendere l'addestramento più celere. *Colab* mette a disposizione una *12 GB NVIDIA Tesla P100* come scheda video.

Il modello *YOLOv4*, presente all'interno di *iTelescope*, viene addestrato utilizzando `Darknet`[19]. Quest'ultimo è un *framework* di rete neurale *open source* scritto in `C` e in `CUDA`[12] (di cui *Colab* ne è provvisto) che supporta il calcolo attraverso la *GPU*.

Per eseguire *Darknet* all'interno di *Colab* è sufficiente scaricare il *framework* dalla sua *repository* ufficiale *GitHub*[13] e compilarlo attraverso il comando *make* (è necessario in primo luogo impostare all'interno del *Makefile* la volontà di utilizzare la *GPU*).

All'interno *directory* di *Darknet*, risiedono diversi file che vengono utilizzati in fase di *training* e *testing*. Di seguito viene presentata la struttura base delle *directory* e dei file necessari per l'addestramento sul *dataset* prescelto.

```
Darknet/
├── ...
├── backup/
│   └── yolov4_last.weights
├── cfg/
│   ├── yolov4_train.cfg
│   └── yolov4_test.cfg
├── data/
│   ├── test/
│   │   ├── photo1.jpg
│   │   └── label1.txt
│   ├── train/
│   │   ├── photo4000.jpg
│   │   └── label4000.txt
│   ├── classes.name
│   ├── test.txt
│   ├── train.txt
│   └── yolov4.data
└── ...
```

Nella cartella `backup/` vengono salvati i *weights* intermedi durante la fase di *training*. *Darknet* è configurato per salvare tali pesi ogni cento iterazioni.

La *directory* `cfg/` contiene i file di configurazione di *YOLOv4* per l'addestramento e per il *testing*. Nello specifico, i file vengono utilizzati per definire la struttura della rete neurale e i suoi componenti

[12] Architettura hardware per l'elaborazione parallela creata da NVIDIA. Tramite di essa è possibile scrivere applicazioni capaci di eseguire calcolo parallelo sulle GPU delle schede NVIDIA

[13] GitHub è un servizio di hosting per progetti software. Il nome deriva dal fatto che GitHub è una implementazione dello strumento di controllo versione distribuito Git

(*layer* convoluzionali e di *yolo*). In `yolov4_train.cfg` è fondamentale modificare alcuni parametri coerentemente al dataset impiegato e al numero delle classi di oggetti.

Al fine di produrre il modello avente la massima accuratezza possibile, sono state realizzate diverse configurazioni modificando i parametri presenti nel file `yolov4_train.cfg`. All'interno di quest'ultimo, sono specificate diverse sezioni come `net`, `yolo`, `convolutional`.

Nel segmento `net` vengono espresse alcune informazioni della rete neurale e del *training*. Per esempio, i valori modificati con lo scopo di ottenere il modello migliore per *iTelescope*, sono: `batch = 64`, `subdivision = 32`, `width = 608 & height = 608`, `maxbatches = 20000` e `steps = 16000,18000`.

Nello specifico:

- La `batch` corrisponde al numero di immagini processate prima che il modello venga aggiornato;
- Le `subdivisions` sono i blocchi in cui le *batches* sono divise. Le immagini all'interno delle *subdivisions* vengono processate parallelamente nella *GPU*;
- I `width & height` si riferiscono alla dimensione della rete. Ogni immagine è ridimensionata secondo tali misure nella fase di *training* e *detection*;
- Le `max batches` sono il numero massimo di *batches* durante l'addestramento. Nel caso in questione tale valore deve essere calcolato con la seguente formula: $max\_batches = classi * 2000$;
- Gli `steps` indicano in quale numero di *batch* il *learning rate* viene regolato. Il primo valore viene definito come l'80% del *max batches*, mentre il secondo come 90%.

Nella sezione `convolutional`, al fine di addestrare *YOLOv4* sul *dataset* prescelto, vi è la necessità di modificare un parametro (solamente nei *layer* convoluzionali prima dei tre *layer yolo*) in base al numero di classi da rilevare nelle immagini. Tale parametro corrisponde al numero di *kernel* convoluzionali presenti nel *layer* in questione e viene calcolato con la formula $filters = (classes + 5) * 3$. Nel caso di iTelescope `filters = 45`.

L'ultima sezione da analizzare è quella di `yolo` che presenta diversi elementi, tra cui la definizione delle *anchors*, *truth tresh* e il numero di classi da ricercare.

La *directory* `data/` contiene altri file necessari per il *training* e per il *testing*. In essa sono infatti presenti le cartelle contenti le immagini e le *labels* utilizzate in queste due fasi. In aggiunta vi sono: il file `classes.name` avente un elenco delle classi da ricercare all'interno delle immagini, i files `test.txt` e `train.txt` contenenti i percorsi delle immagini e il file `yolov4.data` dotato delle informazioni per *Darknet* (tra cui il numero delle classi, i percorsi dei *files* `test.txt`, `train.txt`, `classes.name` e il percorso della cartella di `backup/`).

`Yolov4.data` può essere considerato come il nodo da cui si sviluppa tutto il processo di apprendimento e di *testing*.

Come accennato in precedenza, usufruendo dell'ambiente di sviluppo *Colab*, è possibile eseguire comandi *Linux* all'interno dei blocchi di esecuzione. Per avviare il *training* è sufficiente eseguire il comando:

```
darknet detector train yolov4.data yolov4_train.cfg yolov4_train.weights
```

In questo caso, è stato esplicitato un esempio di addestramento del modello a partire dai pesi calcolati in precedenza sullo stesso *dataset* e con la stessa configurazione.

Durante la fase di *training* è possibile visionare diversi parametri in grado di rappresentare lo stato di avanzamento dell'addestramento. L'*output* del processo mostra le informazioni riferite ad ogni *batch* e alle *subdivisions*.

In riferimento alle *batch*, vengono esplicitati: il numero attuale di iterazione, la *loss* totale, la *loss* media, il *learning rate* attuale e il numero di immagini processate.

Per quanto riguarda le suddivisioni, è opportuno precisare che ogni *subdivision* è divisa in tre regioni che rappresentano tre diverse scale. Inoltre, ciascuna regione fornisce diverse informazioni, tra cui: la *IoU* media e il tasso corretto di classificazione degli oggetti.

2.2.2 Testing

Nei modelli di *deep learning*, il *testing* è il processo finalizzato alla valutazione delle prestazioni. In *iTelescope*, si calcola l'accuratezza dell'*object detector* che ha lo scopo di ricercare dieci classi all'interno delle fotografie scattate da droni.

In questa fase, se da un lato si sfrutta il *framework Darknet* come per il *training*, dall'altra il file di configurazione è `yolov4_test.cfg` (anziché `yolov4_train.cfg`). All'interno di `yolov4_test.cfg`, i valori *batch* e *subdivision* sono impostati uguali ad uno.

Per avviare il processo di testing, si esegue il seguente comando:

```
darknet detector map yolov4.data yolov4_test.cfg yolov4_train.weights
```

Darknet, una volta eseguito il comando sopra citato, avvia il *testing* utilizzando le immagini del *testing set* indicate nel file `yolov4.data`, la configurazione della rete neurale esplicitata in `yolov4_test.cfg` e i *weights* precalcolati durante la fase di *training*.

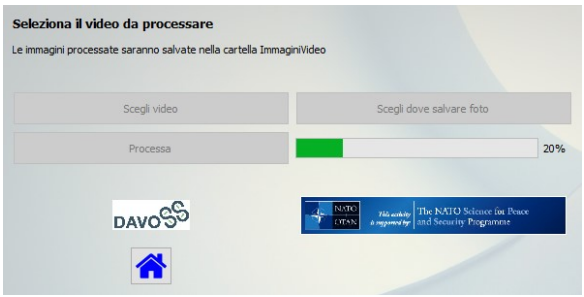
L'output di tale processo mostra la *mAP* del modello secondo diverse soglie (*thresholds*) e consente di visualizzare l'accuratezza riferita ad ogni classe di oggetto. In aggiunta, sono esplicitati i falsi positivi e i falsi negativi definiti durante tale fase.

2.2.3 Applicazione desktop

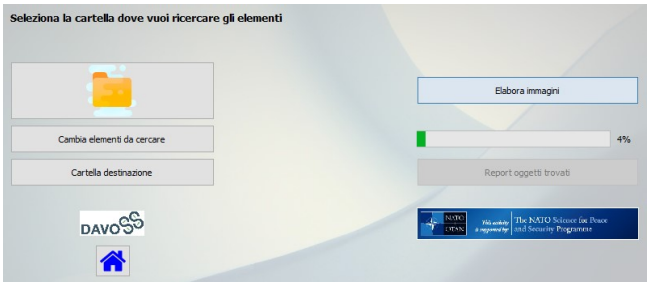
Dopo aver addestrato l'*object detector YOLOv4* con le immagini provenienti dai droni, è stata realizzata un'applicazione *desktop* finalizzata a ricercare le classi di oggetti (pedoni, automobili, moto...) presenti nelle fotografie aeree.

Il *software* viene sviluppato mediante il linguaggio di programmazione di alto livello *Python*. Per la definizione dell'interfaccia grafica si è reso necessario l'utilizzo di *PyQt5*, ossia di un modulo specializzato per l'interfaccia grafica che collega il *Qt C++ cross-platform framework* con *Python*. Tale libreria non è solamente un *GUI toolkit*, ma presenta anche diverse *features*. Tra di esse, si possono citare le *socket* di rete, *thread* e l'integrazione con i *database*. Questi tre elementi permettono di evidenziare il divario esistente tra *PyQt5* e *Tkinter* (*standard Python interface* per la *GUI*).

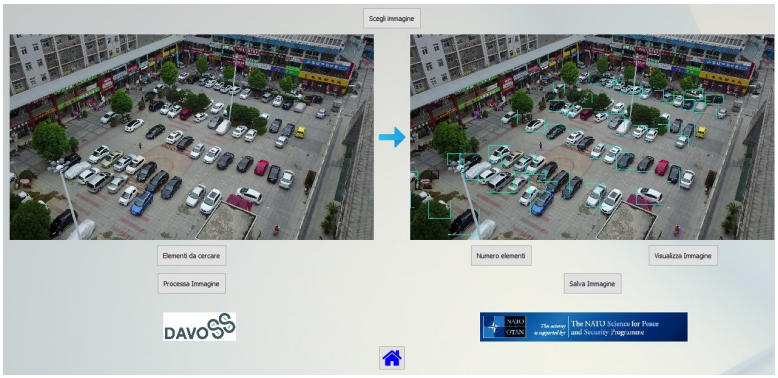
Di seguito verranno mostrare alcune immagini dell'applicazione.



(a)



(b)



(c)

Figura 2.4: Schermate applicazione iTelescope

L'applicazione, come mostrato dalla figura sopra riportata, si compone principalmente di tre finestre: *objects detection* di un video selezionato dall'utente (figura 2.4a), *objects detection* nelle immagini presenti in una determinata cartella (figura 2.4b) e *objects detection* in una singola immagine (figura 2.4c).

Nella finestra dove viene analizzata un'immagine singola, vi è la possibilità di scegliere quale tipologia di oggetto ricercare. Così facendo, una volta processata l'immagine di partenza, verranno visualizzati solamente i *boundary boxes* sugli elementi selezionati. In aggiunta, è possibile salvare l'immagine processata e verificare il numero di oggetti rilevati divisi per ogni categoria.

Nella sezione dedicata all'elaborazione delle immagini di una cartella, è nuovamente possibile selezionare la tipologia di oggetti da ricercare. Una volta eseguita l'elaborazione delle fotografie complessive, si ha la possibilità di creare un file di testo raffigurante il numero di oggetti trovati, divisi per categoria, all'interno delle varie immagini.

Infine, l'ultima finestra permette di analizzare un video. Il programma estrae un *frame* di quest'ultimo ogni secondo, lo analizza e salva l'immagine creata nella cartella selezionata dall'utente.

In *iTelescope* non viene utilizzato il *framework Darknet*, per tale motivo nella *detection* viene impiegata la libreria di *Python* `OpenCV` [14] che contiene l'istruzione:

```
net = cv2.dnn.readNet("yolov4_train.weights", "yolov4_test.cfg")
```

Quest'ultima viene sfruttata per caricare il modello della rete neurale di *YOLO* (stesso file del *testing*) e i *weights* calcolati durante il *training*. Nella funzione che esegue la *detection*, è possibile modificare la soglia con cui i *boundary boxes* vengono creati.

Per la realizzazione del programma, sono stati utilizzati i *thread*[15] di *PyQt5* in modo da rendere l'interfaccia grafica indipendente dall'elaborazione delle immagini e quindi offrire una miglior *user experience*.

L'applicazione *iTelescope* funziona in tutti i principali sistemi operativi (*Windows, Linux e MacOS*) in quanto *Python* e *PyQt5* sono multipiattaforma. In alcuni casi, però, vi possono essere delle problematiche relative alle librerie adoperate. Ecco perché si è reso necessario l'utilizzo di *Docker*, ossia un insieme di prodotti denominati "*platform as a service*" che sfruttano la virtualizzazione a livello di *OS* per distribuire software in pacchetti chiamati "*container*".

Al fine di rendere eseguibile *iTelescope* su tutti i dispositivi desktop, è stata creata un'immagine *Docker* a partire da *Ubuntu 20.04* nel quale sono state installate tutte le librerie necessarie per il funzionamento del programma di rilevamento degli oggetti.

Attraverso un semplice comando *docker* (`docker run iTelescope`) è possibile avviare il *container* e utilizzare il sistema operativo *Ubuntu* con all'interno *iTelescope*.

2.3 Tiny YOLOv3 [10]

Dal modello *YOLOv3* è stata sviluppata una versione semplificata denominata *tiny YOLOv3* che presenta un algoritmo di rilevazione degli oggetti adatto per i dispositivi *embedded* caratterizzati da limitata potenza di calcolo.

La struttura del modello è snella e per tale motivo l'*object detection* risulta essere estremamente veloce (a discapito della precisione).

Tiny YOLOv3 riduce la rete di rilevamento *Darknet-53* ad una rete composta da sette *layers* convoluzionali e sei *layers* di *max-pooling*.

Il modello *tiny YOLOv3* è stato addestrato utilizzando il dataset illustrato nel modello *YOLOv4*, *Colab* e *Darknet*. Inoltre il file di configurazione (`tiny-yolov3.cfg`) riflette la struttura snella del modello.

L'*object detector* prodotto sarà utilizzato per l'analisi *on-board* delle immagini acquisite da un drone per il progetto *NATO SPS G5428 DAVOSS*.

[14] Libreria contente funzioni di programmazione principalmente rivolte alla *computer vision*

[15] Esecuzione di una sequenza di istruzioni programmate che può essere gestita indipendentemente da uno scheduler

3 Risultati sperimentali

Nei capitoli precedenti è stato illustrato il funzionamento dei principali *object detector* con particolare riferimento al modello *YOLOv4*. Come già dimostrato, tale modello permette all'applicazione desktop *iTelescope* di riconoscere dieci tipologie di oggetti all'interno di immagini e video provenienti da droni.

Sono state eseguite diverse configurazioni delle reti neurali convoluzionali con differenti parametri allo scopo di ottenere un modello ben addestrato e affidabile.

3.1 Risultati YOLOv4

Di seguito verranno analizzate le configurazioni realizzate al fine di produrre il migliore modello *YOLOv4* in grado di adattarsi al dominio di immagini a disposizione.

3.1.1 YOLOv4 - Prima configurazione

Inizialmente il *dataset* era composto da dodici classi: pedoni, persone, biciclette, macchine, van, camion, tricicli, tricicli motorizzati, bus, moto, regioni non definite e altro.

In virtù di tali tipologie di oggetti, i parametri utilizzati per addestrare YOLOv4 sono stati: `max batches = 24000`, `width e height = 608`, `steps = (19200,21600)` e `subdivision = 64`.

Attraverso tale configurazione, l'*object detector* sviluppato, dopo aver terminato le 25000 *batches*, ha raggiunto una *mAP@0.50* = 25% e una *mAP@0.25* = 35%. Le classi che hanno presentato l'accuratezza migliore sono state quelle delle automobili (50,22%) e delle motociclette (35,07%). Viceversa, le classi che hanno registrato il livello più basso di precisione sono state quelle degli autobus (9%) e dei tricicli a motore (13%).

3.1.2 YOLOv4 - Seconda configurazione

Nella seconda configurazione sono stati ripresi gli stessi parametri della struttura precedente (3.1.1), ad eccezione di uno: il *learning rate*. Quest'ultimo corrisponde ad un iper-parametro impiegato per controllare la velocità con cui l'algoritmo aggiorna i *weights* in fase di *training*. Tale valore descrive quanto velocemente il modello si adatta al problema: più è basso il tasso di apprendimento e più epoche di allenamento sono richieste, mentre più è alto e meno epoche sono necessarie (in quanto i cambiamenti sono più rapidi).

La rete corrente è stata addestrata utilizzando il `learning rate = 0,0001` (invece di 0,001).

Al termine del *training*, le *performance* del modello creato sono risultate minori rispetto alla prima configurazione. La *mAP* è infatti risultata essere pari al 21% (*mAP@0.50*) e al 33% (*mAP@0.25*). In questo caso, le tipologie di oggetti rilevate con una maggiore precisione sono state le automobili (47,27%), le motociclette (26%) e i van (22,50%). Viceversa, le categorie con accuratezza peggiore sono state le biciclette (6,23%) e gli autobus (6,06%).

Tale configurazione, non avendo raggiunto un soddisfacente livello di accuratezza, non è stata presa in considerazione al momento della scelta della struttura da utilizzare.

3.1.3 YOLOv4 - Terza configurazione

La terza configurazione presenta anch'essa gli stessi valori della prima (3.1.1). In questo caso, però, all'avvio del *training* non sono stati utilizzati dei *weights* precalcolati, ossia dei pesi scaricati dalla *repository* ufficiale di *Darknet* (`yolov4.conv.137`).

Di frequente, poiché le immagini del *dataset* tendono ad essere molto differenti rispetto a quelle su cui si è addestrato in precedenza il modello (le immagini riprese dai droni hanno inquadrature molto particolari rispetto alle semplici fotografie), l'*object detector* può commettere diversi errori ed imprecisioni. Per risolvere questo problema, si è deciso di avviare il *training* non utilizzando i *weights*.

Tale struttura raggiunge una *mAP@0.50* = 23% e una *mAP@0.25* = 36%. Come si può notare, la *mean average precision* con soglia a 0,25 presenta una precisione maggiore rispetto ai due modelli

sopra elencati.

3.1.4 YOLOv4 - Quarta configurazione

La quarta configurazione ha raggiunto i risultati migliori ed è stata selezionata per essere utilizzata all'interno di *iTelescope*.

In questo caso, sono state eliminate due classi di oggetti non decisive ("regioni non definite" e "altro") dal dominio delle immagini analizzate dall'applicazione. È stata effettuata questa scelta poiché spesso il rilevatore veniva tratto in inganno e manifestava un aggravamento nella precisione media.

Si rimanda alla sezione 2.2 per la spiegazione dei parametri definitivi utilizzati per creare il file dei *weights* finali.

In quest'ultima configurazione, si è cercato di giungere ad un compromesso tra le istruzioni di *Alexey Bochkovskiy et al.* e le limitazioni imposte da *Google Colaboratory* che consentono di utilizzare solamente `13 GB` di memoria *RAM* nell'addestramento del modello selezionato. Per tale motivo, il numero di *subdivision* è stato impostato a trentadue anziché a sedici (valore consigliato dallo sviluppatore di *YOLOv4*). Questa modifica comporta una perdita di *mAP*. Anche la scelta della dimensione della rete per il *training* (*widht* e *height*) si riconduce alle limitazioni computazionali imposte.

Il modello ha raggiunto una *mAP@0.50* = 47,37% e una *mAP@0.25* = 53,65% con la dimensione della rete `608x608`.

Nella tabella 3.1, sono illustrati l'*average precision*, i *true positive* e i *false positive* divisi per ogni classe da ricercare nelle immagini. Il *true positive* corrisponde ad un rilevamento della classe corretto (in relazione alla *label* reale), mentre il *false positive* ad un rilevamento della classe errato. Tale tabella prende il nome di matrice di confusione e quest'ultima viene utilizzata con lo scopo di riassumere il successo nelle previsioni di un modello (in questo caso di un *object detector*).

CLASSE OGGETTO	AVERAGE PRECISION	TP	FP
Pedoni	38,67%	10221	8321
Persone	32,02%	3227	3753
Biciclette	24,80%	526	822
Macchine	84,61%	20232	5992
Van	52,49%	2102	2027
Camion	54,92%	812	561
Tricicli	37,27%	551	656
Tricicli a motore	21,17%	217	381
Bus	76,87%	874	185
Motociclette	50,92%	3863	3776

Tabella 3.1: Matrice di confusione *YOLOv4* con immagini di droni e dimensione della rete `608x608`

La tabella 3.1 dimostra che le classi di oggetti riconosciute con la maggior accuratezza sono le macchine e gli autobus, mentre gli oggetti che presentano il peggior livello di precisione sono le biciclette e tricicli a motore.

Secondo gli sviluppatori di *YOLOv4*, per migliorare la precisione del modello è opportuno, in fase di *testing*, aumentare la dimensione della rete (per esempio da `608x608` a `832x832` o `1024x1024`). Attraverso questo incremento, è possibile anche aumentare il riconoscimento degli oggetti di dimensione ridotta (questo è molto importante nelle immagini scattate dai droni).

Nella tabella seguente 3.2, è riportata la matrice di confusione con le dimensioni della rete aumentate (`832x832`).

Grazie all'aumento della dimensione della rete, l'*object detector* creato raggiunge una *mAP@0.50* = 52.08% e una *mAP@0.25* = 57,40%.

CLASSE OGGETTO	AVERAGE PRECISION	TP	FP
Pedoni	47,85%	12095	7811
Persone	37,85%	3551	3484
Biciclette	31,81%	664	827
Macchine	87,81%	20884	5200
Van	55,07%	2283	2012
Camion	59,56%	864	530
Tricicli	41,46%	625	668
Tricicli a motore	21,88%	233	384
Bus	80,16%	948	227
Motociclette	57,37%	4180	3421

Tabella 3.2: Matrice di confusione *YOLOv4* con immagini di droni e dimensione della rete `832x832`

Comparando le due tabelle (3.1 e 3.2), è possibile notare l'aumento della precisione nel modello in cui il *testing* viene effettuato con la dimensione della rete uguale a `832x832`. In quest'ultimo caso, l'accuratezza accresce particolarmente nel rilevamento degli oggetti più piccoli (pedoni e persone).

3.1.5 YOLOv4 - Esempio rilevazione

Di seguito, sono illustrati due esempi di rilevamento degli oggetti utilizzando i weights calcolati attraverso YOLOv4 - Quarta configurazione.

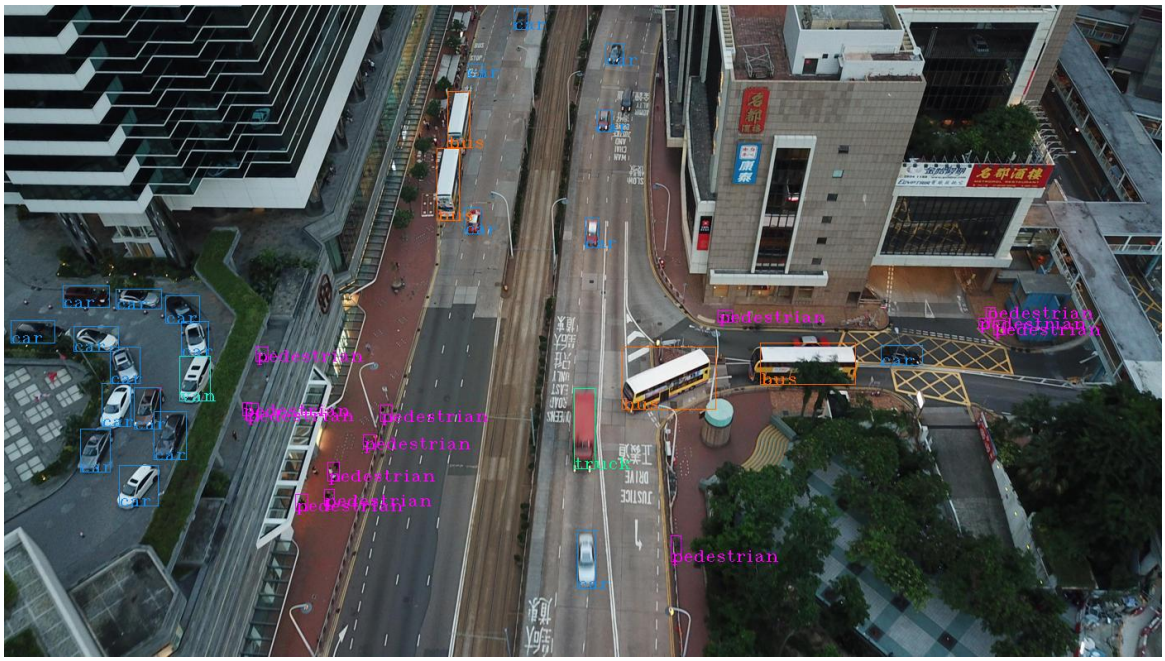


Figura 3.1: Esempio rilevamento. All'interno dell'immagine sono stati rilevati: pedoni, autobus, macchine e van

Figura 3.2: Rilevamento oggetti all'interno di un'immagine con numerosi elementi. In quest'ultima emergono alcune problematiche che verranno discusse successivamente

La rilevazione degli oggetti riesce a raggiungere un ottimo risultato anche quando viene applicata ad immagini con scarsa illuminazione. Tali risultati permettono di valutare positivamente anche l'adozione della *detection* nelle ore serali, così da garantire la sicurezza in tutte le ore giornaliere.

3.1.6 Problematiche riscontrate

Durante la fase di testing dell'applicazione *iTelescope*, sono emerse svariate problematiche riguardanti anche l'*object detector* creato.

Il processo di *object detection* presenta diversi problemi, e quindi un'accuratezza minore, se all'interno delle immagini vi è un numero elevato di elementi (si veda ad esempio la figura 3.2) e se gli oggetti si sovrappongono.

Nella configurazione di *YOLOv4* per *iTelescope*, un pedone si distingue da una persona in quanto quest'ultima o pedala o guida o è all'interno di un mezzo di trasporto. In alcuni casi, però, l'*object detector* confonde le due classi e questo è evidente nella figura 3.2, dove sul lato destro sono state identificate due persone invece che due pedoni. Tale problematica può essere anche verificata visionando i dati riguardanti i falsi positivi (tabella 3.1). Lo stesso errore di scambio tra classi si presenta anche tra biciclette e motociclette o tra semaforo e pedone (soprattutto se quest'ultimo è di ridotte dimensioni).

Per concludere, gli oggetti più piccoli all'interno delle immagini non sono rilevati. Tale problematica può essere causata dalla dimensione della rete nella fase di addestramento (nel caso corrente si è utilizzata come dimensione `608x608` per rispettare le limitazioni di *Colab*).

3.2 Risultati tiny YOLOv3

Anche nel caso di *tiny YOLOv3*, sono state eseguite diverse configurazioni al fine di eleggere la migliore.

3.2.1 Tiny YOLOv3 - Prima configurazione

Per la prima configurazione, il *training* è avvenuto applicando i seguenti parametri: `batch` = 64, `subdivision` = 16 e dimensione della rete uguale a 832. Inoltre, tramite una funzione di *Darknet*, è stato possibile ricalcolare le *anchor* in base alle immagini presenti all'interno del *dataset*.

Attraverso tali valori si è raggiunto una *mAP@50* = 8,49%, utilizzando come dimensione della rete `608x608`. Diversamente, con la *width* e *height* uguali a `832`, la *mAP@50* = 11,44%.

3.2.2 Tiny YOLOv3 - Configurazione finale

Tale configurazione è riuscita ad ottenere i risultati migliori al termine del *training*. Quest'ultima sfrutta gli *anchor* di *default*, `batch` = 64, `subdivision` = 32 e dimensione della rete uguale a 608.

Attraverso tali parametri, utilizzando come dimensione della rete di testing `608x608`, il modello *tiny YOLOv3* ha raggiunto una *mAP@0.50* = 20,18%. Mentre, con dimensione uguale a `832x832`, la *mAP@0.50* sale a 25,32%.

Nella tabella 3.3, sono illustrati la precisione media, i *true positive* e i *false positive* divisi per ogni categoria di oggetti (`832x832`).

CLASSE OGGETTO	AVERAGE PRECISION	TP	FP
Pedoni	11,74%	4704	9541
Persone	13,78%	1658	3323
Biciclette	4,59%	107	427
Macchine	66,27%	16360	8529
Van	33,54%	1263	1481
Camion	24,13%	359	504
Tricicli	15,10%	155	221
Tricicli a motore	7,51%	42	105
Bus	57,44%	569	124
Motociclette	19,10%	1849	3526

Tabella 3.3: Matrice di confusione *tiny YOLOv3* con immagini di droni e dimensione della rete `832x832`

3.2.3 Problematiche riscontrate

Come illustrato precedentemente, questa versione di *YOLO* se da un lato è estremamente veloce, dall'altra presenta diverse problematiche. Una tra queste è la bassa accuratezza. Tale problema si amplifica notevolmente quando gli oggetti all'interno delle immagini da analizzare sono elevati e sono di dimensioni ridotte (come nelle immagini del *dataset*). Per dimostrare ciò, di seguito si riporta un esempio di analisi di un'immagine (la medesima riportata nella configurazione finale di *YOLOv4*) che presenta le imprecisioni appena descritte.

Figura 3.3: Sono evidenti alcuni errori di rilevamento degli oggetti e la scarsa precisione. Inoltre gli oggetti rilevati sono di numero minore rispetto al *YOLOv4*

Conclusioni

Al termine dell'analisi condotta nel lavoro di tesi, è possibile trarre delle riflessioni conclusive.

Il primo capitolo, presentando il tema della *computer vision*, dimostra come tale campo scientifico interdisciplinare si sia evoluto dagli anni '60 fino ai giorni nostri. Un enorme sviluppo si è raggiunto grazie all'introduzione del *deep learning* che ha consentito la nascita e lo sviluppo di applicazioni sempre più complesse e sofisticate. Queste ultime sfruttano le reti neurali al fine di consentire la creazione di modelli che vengono utilizzati dal programma per effettuare le predizioni. Tra le principali tipologie di reti neurali è stata presentata la *Convolutional Neural Network* che, se applicata all'analisi delle immagini, consente di raggiungere ottimi risultati. L'esame procede con un approfondimento in merito all'*object detection*, ossia un'applicazione della *computer vision* che permette di ricercare e riconoscere oggetti all'interno di immagini e video. Tale obiettivo può essere raggiunto attraverso diverse tipologie di modelli, che sono stati esposti nell'elaborato elogiando i loro pregi ed esplicitando i loro difetti. Ad esempio, si è dimostrato che se da un lato gli algoritmi *two-stage detector* presentano una velocità limitata rispetto a quelli *one-stage*, dall'altro si connotano di una precisione migliore nella maggior parte delle situazioni. L'analisi prosegue con la rappresentazione delle caratteristiche e funzionalità di *YOLO* che lo rendono un ottimo candidato per la creazione di un *object detector* nelle immagini scattate dai droni.

Nel secondo capitolo, si esamina la creazione di un'applicazione desktop finalizzata al riconoscimento e alla localizzazione di oggetti in immagini e video provenienti da velivoli radiocomandati. Per effettuare l'*object detection*, si è scelto di analizzare il modello *YOLOv4* (quarta versione di *YOLO*) in quanto presenta accuratezza e *FPS* maggiorati rispetto alle sue versioni precedenti. Infine, sono stati evidenziati i passaggi necessari allo scopo di creare il modello di rilevamento. Come è possibile evincere dalla lettura del secondo capitolo, il *training* è una fase fondamentale poiché "insegna" all'*object detector* le caratteristiche univoche di ogni oggetto necessarie per il riconoscimento. Nel caso preso in esame, i parametri configuranti il *framework* di rete neurale *Darknet* vengono scelti effettuando un compromesso tra le indicazioni del creatore del modello *YOLOv4*, le caratteristiche del dataset e le limitazioni di *Google Colab*. Per rendere maggiormente intuitivo l'utilizzo dell'*object detector*, è stata creata un'applicazione desktop in grado di interfacciare l'utente con il modello di intelligenza artificiale.

Il terzo capitolo analizza, infine, i risultati sperimentali del modello di *object detection* che è stato sviluppato. Si possono così evidenziare le differenze, dal punto di vista dell'accuratezza, delle configurazioni create. Queste ultime differiscono in base ai diversi parametri utilizzati all'interno della configurazione di *Darkent*. Il modello prescelto riesce a raggiungere una *mAP@0.50 = 52,08%* applicando una dimensione della rete pari a *832x832*. In conclusione, è stata posta l'attenzione sul modello *tinyYOLOv3* il quale, nonostante sia caratterizzato da una precisione minore (*mAP@0.50=25,32%*), è in grado di raggiungere alte prestazioni dal punto di vista della velocità di elaborazione.

In merito all'applicazione sviluppata per tale progetto di tesi, è possibile affermare che il programma iTelescope soddisfa pienamente l'obiettivo per il quale è stato creato. Il programma è infatti di facile utilizzo e consente di elaborare numerose immagini e lunghi video in pochi secondi. Inoltre, l'applicazione è in grado di rilevare gli oggetti anche in immagini caratterizzate da scarsa luminosità, condizioni atmosferiche sfavorevoli e messa a fuoco non ottimale. Tali vantaggi sono fondamentali nell'ambito della sicurezza cittadina e della sorveglianza in quanto gli elementi della tempestività e della precisione ricoprono un ruolo di primaria importanza.

L'applicazione, anche se correttamente funzionante, in rari casi potrebbe causare dei problemi di

mismatch. Poiché tale progetto è stato sviluppato per favorire la vigilanza e il controllo all'interno delle città, in una prospettiva di sviluppi futuri è auspicabile aumentare la dimensione del *dataset* utilizzato per il *training*, incrementare il numero degli oggetti da riconoscere nelle immagini e sostituire *Google Colaboratory* (caratterizzato da molte limitazioni computazionali) con una valida alternativa. Inoltre, potrebbe essere utile modificare l'applicazione in modo tale da collegarla direttamente al drone che scatta le fotografie.

In conclusione, l'auspicio e la prospettiva di chi scrive è quello di incentivare l'utilizzo e il perfezionamento di questo tipo di applicazione al fine di rendere più semplice ed immediata la sorveglianza e la sicurezza delle nostre preziose città.

Bibliografia

[1] Anonymous. A brief history of computer vision and ai image recognition. https://www.pulsarplatform.com/blog/2018/brief-history-computer-vision-vertical-ai-image-recognition/. ultimo accesso 12/04/2021.

[2] Anonymous. An introductory guide to computer vision. https://tryolabs.com/resources/introductory-guide-computer-vision/. ultimo accesso 14/04/2021.

[3] Alexey Bochkovskiy, Chien-Yao Wang, and Hong-Yuan Mark Liao. Yolov4: Optimal speed and accuracy of object detection, 2020.

[4] Jason Brownlee. A gentle introduction to computer vision. `https://machinelearningmastery.com/what-is-computer-vision`, 2019. ultimo accesso 12/04/2021.

[5] Manuel Carranza-García, Jesús Torres-Mateo, Pedro Lara-Benítez, and Jorge García-Gutiérrez. On the performance of one-stage and two-stage object detectors in autonomous vehicles using camera data. *Remote Sensing*, 13(1), 2021.

[6] Terrance Devries and Graham W. Taylor. Improved regularization of convolutional neural networks with cutout. *CoRR*, abs/1708.04552, 2017.

[7] Rohith Gandhi. R-cnn, fast r-cnn, faster r-cnn, yolo — object detection algorithms. https://towardsdatascience.com/r-cnn-fast-r-cnn-faster-r-cnn-yolo-object-detection-algorithms-36d53571365e, 2018. ultimo accesso 19/04/2021.

[8] Ross Girshick. Fast r-cnn, 2015.

[9] Ross Girshick, Jeff Donahue, Trevor Darrell, and Jitendra Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. 2014.

[10] Xiaotian Gong, Li Ma, and Hangkong Ouyang. An improved method of tiny YOLOV3. *IOP Conference Series: Earth and Environmental Science*, 440:052025, mar 2020.

[11] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Spatial pyramid pooling in deep convolutional networks for visual recognition. *CoRR*, abs/1406.4729, 2014.

[12] Tsung-Yi Lin, Piotr Dollár, Ross Girshick, Kaiming He, Bharath Hariharan, and Serge Belongie. Feature pyramid networks for object detection, 2017.

[13] Tsung-Yi Lin, Priya Goyal, Ross B. Girshick, Kaiming He, and Piotr Dollár. Focal loss for dense object detection. *CoRR*, abs/1708.02002, 2017.

[14] Shu Liu, Lu Qi, Haifang Qin, Jianping Shi, and Jiaya Jia. Path aggregation network for instance segmentation. *CoRR*, abs/1803.01534, 2018.

[15] Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott Reed, Cheng-Yang Fu, and Alexander C. Berg. Ssd: Single shot multibox detector. *Lecture Notes in Computer Science*, page 21–37, 2016.

[16] Fernando Mendoza and Renfu Lu. *Basics of Image Analysis*, page 10. 01 2015.

[17] Ilija Mihajlovic. Everything you ever wanted to know about computer vision. https://towardsdatascience.com/everything-you-ever-wanted-to-know-about-computer-vision-heres-a-look-why-it-s-so-awesome-e8a58dfb641e, 2019. ultimo accesso 12/04/2021.

[18] Diganta Misra. Mish: A self regularized non-monotonic neural activation function. *CoRR*, abs/1908.08681, 2019.

[19] Joseph Redmon. Darknet: Open source neural networks in c. `http://pjreddie.com/darknet/`, 2013–2016.

[20] Joseph Redmon, Santosh Kumar Divvala, Ross B. Girshick, and Ali Farhadi. You only look once: Unified, real-time object detection. *CoRR*, abs/1506.02640, 2015.

[21] Joseph Redmon and Ali Farhadi. Yolo9000: Better, faster, stronger, 2016.

[22] Joseph Redmon and Ali Farhadi. Yolov3: An incremental improvement, 2018.

[23] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster r-cnn: Towards real-time object detection with region proposal networks, 2016.

[24] J Rodellar, S Alférez, A Acevedo, A Molina, and A Merino. Image processing and machine learning in the morphological analysis of blood cells. *Int. J. Lab. Hematol.*, 40 Suppl 1:46–53, 2018.

[25] Adrian Rosebrock. An introductory guide to computer vision. https://www.pyimagesearch.com/2016/11/07/intersection-over-union-iou-for-object-detection/. ultimo accesso 22/04/2021.

[26] Sumit Saha. A comprehensive guide to convolutional neural networks. https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53. ultimo accesso 22/04/2021.

[27] Pulkit Sharma. A step-by-step introduction to the basic object detection algorithms. https://www.analyticsvidhya.com/blog/2018/10/a-step-by-step-introduction-to-the-basic-object-detection-algorithms-part-1/, 2018. ultimo accesso 20/04/2021.

[28] Aishwarya Singh. Selecting the right bounding box using non-max suppression. https://www.analyticsvidhya.com/blog/2020/08/selecting-the-right-bounding-box-using-non-max-suppression-with-implementation/, 2020. ultimo accesso 01/05/2021.

[29] Sangdoo Yun, Dongyoon Han, Seong Joon Oh, Sanghyuk Chun, Junsuk Choe, and Youngjoon Yoo. Cutmix: Regularization strategy to train strong classifiers with localizable features. *CoRR*, abs/1905.04899, 2019.

[30] Hongyi Zhang, Moustapha Cissé, Yann N. Dauphin, and David Lopez-Paz. mixup: Beyond empirical risk minimization. *CoRR*, abs/1710.09412, 2017.

[31] Zhaohui Zheng, Ping Wang, Wei Liu, Jinze Li, Rongguang Ye, and Dongwei Ren. Distance-iou loss: Faster and better learning for bounding box regression. *CoRR*, abs/1911.08287, 2019.

[32] Pengfei Zhu, Longyin Wen, Xiao Bian, Ling Haibin, and Qinghua Hu. Vision meets drones: A challenge. *arXiv preprint:1804.07437*, 2018.

[33] Zhengxia Zou, Zhenwei Shi, Yuhong Guo, and Jieping Ye. Object Detection in 20 Years: A Survey. *arXiv e-prints*, page arXiv:1905.05055, May 2019.