Report di Progetto

Visione Artificiale e Riconoscimento - A.A. 2024/2025

Matteo Iorio - matteo.iorio2@studio.unibo.it - 0001112276
Stefano Furi - stefano.furi@studio.unibo.it - 0001125057
Fabio Vincenzi - fabio.vincenzi2@studio.unibo.it - 0001137554

23 aprile 2025

Indice

1 Introduzione al Problema

1.1 Individuazione di Segnali Stradali e Semafori

Il problema che abbiamo deciso di affrontare riguarda l'individuazione dei segnali stradali, e dei semafori a real time, al fine poi di realizzare un'ipotetica architettura in grado di assistere la guida di un conducente su strada. Per simulare l'installazione e l'utilizzo della rete neurale profonda selezionata su una macchina, ci siamo muniti del simulatore CARLA (approfondito nel corso di *Smart Vehicular Systems*). Questo simulatore permette di installare su una macchina qualsiasi un videocamera *RGB*, a tale telecamera è possibile definire un *handler* per ogni immagine che viene catturata. Non abbiamo fatto altro che prelevare l'immagine di input, darla in pasto alla migliore rete neurale individuata nel nostro esperimento e successivamente far visualizzare a schermo quali fossero gli eventuali oggetti identificati. Proprio per questo abbiamo dovuto esplorare varie tecniche per affrontare questo di problema, andando d individuare quali fossero le strade da seguire per raggiungere il nostro obiettivo finale. Dal momento in cui il sistema deve fornire una risposta a *real-time* si è scelto di utilizzare metodologie che prevedessero l'utilizzo di reti profonde, dal momento in cui un classico approccio di *template matching* non è fattibile a causa dei tempi di risposta. Ogni componente del gruppo ha deciso di scegliere una specifica architettura andando cosi ad esplorare i vantaggi e gli eventuali svantaggi.

1.2 Dataset

Dopo aver scelto le varie architetture da utilizzare per il nostro problema, abbiamo dovuto ricercare dei *dataset* che ci permettessero di raggiungere il nostro obiettivo. Per la ricerca dei dati di addestramento abbiamo utilizzato `roboflow` e `kaggle`, due siti web al cui interno è possibile cercare e scaricare *dataset*. Una comodità che forniscono questi due siti web, è la possibilità di scaricare il *dataset* selezionato tramite un'apposita libreria di `python`.

1.3 Training

Per quanto riguarda la fase più onerosa dell'intero sviluppo di questo sistema, ovvero la fase di training, abbiamo scelto di utilizzare le piattaforme che fornisce `kaggle`, le quali sono estremamente preformati disponendo di due GPU NVDIA Tesla T4. Ogni architettura proposta in Sezione 3 è stato sviluppata all'interno di questo ambiente. Per maggiori dettagli riguardanti il training delle reti si rimanda ai capitoli dedicati all'interno di questo documento, oppure ai contenuti dei notebook jupyter al repository del progetto[1].

1.4 Possibili Approcci

Numerosi sono gli approcci che possono essere impiegati per la risoluzione di questo problema; di seguito vengono analizzati gli approcci principali per la risoluzione del problema in questione.

1.4.1 Object Detection - *Handcrafted Feature-Based Methods*

Shape-Based Detection Tramite l'utilizzo della libreria *OpenCV*, è possibile individuare oggetti specifici tramite la loro forma geometrica all'interno di un'immagine. Come sappiamo dalla teoria, i segnali stradali presentano quattro macro-forme, riconosciute in tutto il mondo. A prima vista, questa strategia può sicuramente sembrare la più semplice. Purtroppo, non è così, poiché basta una piccola osservazione per capire che questo approccio è infattibile sotto ogni punto di vista. Per ogni frame, bisognerebbe cercare contemporaneamente tre o quattro forme specifiche, con un alto rischio di *match* con forme che non appartengono a un segnale stradale. Ad esempio, le ruote di un'auto potrebbero essere scambiate per un ipotetico segnale stradale. Inoltre, le distorsioni che possono derivare dall'acquisizione delle immagini potrebbero invalidare la ricerca degli oggetti stessi.

[1] https://github.com/S-furi/cv-traffic-tracking

Template Matching La seconda modalità attraverso la quale si può affrontare questo problema è tramite il *template matching*, dove un'immagine denominata *template* viene fatta scorrere su sotto finestre dell'immagine in input (a diverse risoluzioni), nel caso in cui il *template* dovesse fare *match* con una sotto porzione dell'immagine, allora potremmo aver individuato un segnale stradale, ma sfortunatamente anche in questo caso l'utilizzo di questo approccio è infattibile, tramite un banale calcolo. Solamente in Italia ci sono all'incirca più di 300 segnali stradali diversi (*lower bound*), e considerando la totalità dei paesi nel mondo possiamo effettuare questo calcolo:

$$TotalSigns = 195 \times 300 = 585000$$

Trecento viene preso come la media dei segnali stradali che si possono trovare in media in un paese, si può dunque capire che è impossibile fornire un risultato a real-time quando bisogna controllare più di cinquantotto mila cinquecento *template*. Anche se ad esempio il numero di cartelli stradali fosse estremamente basso (cento ad esempio), e si prenda come caso d'uso un solo paese, cercare all'interno di un'immagine anche solo cento *richiede* non è un'operazione che viene eseguita in *real-time*. Senza considerare possibili distorsioni nella camera.

1.4.2 Object Detection tramite Reti Neurali Profonde

Con la sempre maggiore disponibilità di calcolo, a partire da AlexNet [KSH17] si è dimostarto come l'impiego di reti neurali profonde possa produrre risultati eccezionali per una vasta gamma di problemi nell'ambito di computer vision. Ad oggi, l'impiego di reti pre-addestrate su ingenti quantità di immagini etichettate ha permesso la produzione di reti neurali con elevate capacità di discriminazione anche per scenari real-time (Sezione 2.1), o anche in contesti dove le capacità computazionali dei dispositivi sono limitate (Sezione 2.2). L'impiego di tali strutture per la task di object detection ha prodotto risultati eccellenti anche in presenza di dataset di fine-tuning non necessariamente grandi, poiché la rete di partenza presenta un grado di generalizzazione molto elevato.

2 Stato dell'Arte

2.1 Yolo

`YOLO` (You Only Look Once) è un sistema per il task di *object detection* in *run-time*. A differenza degli algoritmi tradizionali di *object detection*, che prevedono la generazione di proposte di regioni seguite da una fase di classificazione, `YOLO` affronta il problema come un semplice compito di regressione. In particolare, il modello prevede simultaneamente la *bounding box* e la probabilità della classe associata in un'unica iterazione della rete, rendendolo estremamente veloce. Proprio per questa caratteristica, YOLO è ampiamente utilizzato nei problemi di *detection* in tempo reale.

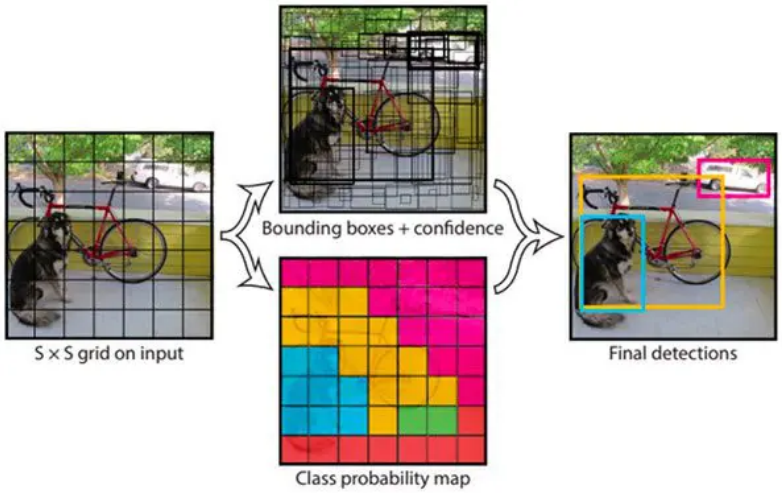

Figura 1: Come funziona YOLO

Architettura L'architettura di YOLO si basa principalmente su tre componenti principali:

- *Backbone*: Estrae dall'immagine di input le *feature* rappresentative;
- *Neck*: Aggrega le *features* estratte da immagini con scala diversa in modo da individuare oggetti con scale diverse;
- *Head*: Individua la *bounding box* e la classe di appartenenza.

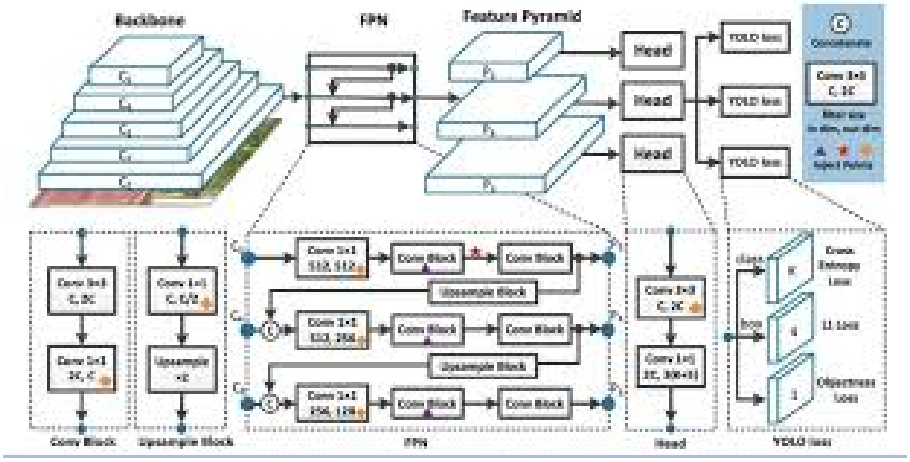

Figura 2: Architettura di YOLO

Come introdotto precedentemente, dal momento in cui `YOLO` fornisce l'individuazione della *bounding box* e la classe dell'oggetto in un tempo estremamente basso, questo ha fatto sii che divenisse una delle architetture più utilizzate per i problemi di *object detection* a *real-time*. La versione che si è scelto di utilizzare per condurre il nostro esperimento è la versione 11 dell'architettura di `YOLO`. `YOLO11` [JQ24] introduce significativi miglioramenti nelle capacità di estrazione delle caratteristiche grazie a un'architettura ottimizzata della backbone e del neck, consentendo un rilevamento degli oggetti più preciso e una gestione più efficace di compiti complessi. L'efficienza e la velocità del modello sono state ulteriormente perfezionate attraverso un design architetturale raffinato e pipeline di addestramento ottimizzate, garantendo un'elaborazione più rapida senza compromettere l'equilibrio tra accuratezza e prestazioni.

2.2 *Object Detection* per *Edge Computing*

Nel contesto dell'Edge Computing (e.g. dispositivi emdedded o mobile), dove le risorse computazionali sono limitate, è cruciale adottare modelli efficienti che bilancino accuratezza e velocità di inferenza. In questa sezione, verranno analizzati i principali modelli che hanno definito e definiscono lo stato dell'arte attuale per il task di object detection in contesti dove bassa latenza, e risorse limitate sono determinanti.

2.2.1 Region Proposal Convolutional Neural Network

Nell'ambito dell'*object detection*, l'impiego di algoritmi tradizionali i quali effettuano una ricerca esaustiva attraverso lo scorrimento di una finestra a diverse risoluzioni per definire delle regioni d'interesse, ha da sempre posto un grande limite in termini di prestazioni ed efficienza. Una prima implementazione di **Region Proposal Convolutional Neural Network** (R-CNN) [Gir+14] mira ad evitare cali di prestazioni dovuti ai ripetuti scorrimenti della finestra sull'immagine, grazie ad un algoritmo di *selective search*, il quale **propone** circa 2000 regioni per ogni immagine, le quali sono di input ad una CNN tradizionale (AlexNet nel caso della prima implementazione), la quale a sua volta restituisce un *feature vector* per ogni *region proposal*. A questo punto un classificatore (e.g. SVM) ha il compito di stabilire la classe di appartenenza, mentre un *bounding box regressor* effettua la localizzazione. Per via dell'alto numero di *region proposal* da classificare, i tempi di addestramento e classificazione tendono ad essere piuttosto onerosi. Inoltre, le *feature map* per ogni regione risultano in molto spazio occupato su disco. Per questi motivi viene proposta **Fast R-CNN** [Gir15], migliorando significativamente le prestazioni grazie ad un'architettura più ottimizzata (Figura 3): l'intera immagine viene elaborata *una sola volta* da una rete convoluzionale da cui si estraggono *feature map* condivise per tutte le *region proposal*. Un livello di *RoI Pooling* quindi uniforma le dimensioni delle regioni prima di essere di input ad un livello *fully connected* per le task di classificazione e regressione. Sebbene Fast R-CNN abbia ridotto drasticamente i tempi di inferenza

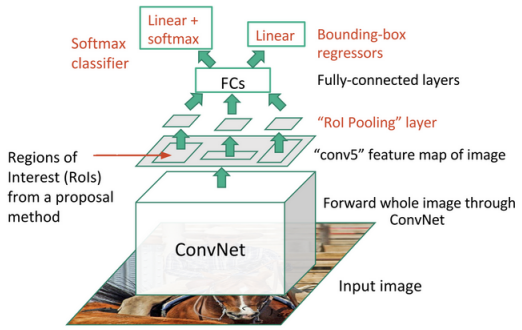

Figura 3: Architettura di *Fast R-CNN*, mostrando come interagiscono fra loro i livelli, specialmente il livello convoluzionale con il *RoI Pooling*, e quest'ultimo con il livello *fully-connected*

rispetto al primo modello, la dipendenza dall'algoritmo di *selective search* per la definizione iniziale

delle regioni di interesse costituisce un forte collo di bottiglia. Per questo motivo viene sviluppata **Faster Region-based Convolutional Neural Network (Faster R-CNN)** [Ren+16], la quale sostituisce *selective search* con una nuova rete neurale: *Region Proposal Network (RPN)*. Grazie a RPN le regioni proposte vengono generate direttamente in modo più rapido ed efficiente, il quale opera direttamente sulle *feature maps* estratte dal livello convoluzionale predicendo un insieme di *punti di ancoraggio*, i.e. *bounding box* predefinite di diverse dimensioni e proporzioni. Per ognuno di questi punti, viene classificato come elemento di *foreground* (contenente un oggetto di interesse) o di *background*. Lavorando direttamente e in collaborazione con il *backbone* (i.e. la CNN), Faster R-CNN risulta decisamente più veloce ed efficiente rispetto all'impiego dell'algoritmo *selective search*.

2.2.2 Mobile-Net

Mobile-Net [How+17] è una rete neurale leggera ed efficiente progettata per dispositivi *mobile* ed *embedded*. È una rete pensata principalmente per la classificazione e successivamente adattata per il task di object detection, ed ottiene buoni risultati considerando i vincoli di bassa latenza e prestazioni limitate caratteristiche dell'*edge computing*. Sin dalla prima versione l'architettura è caratterizzata dall'incorporamento di *depthwise separable convolutions* (Figura 4), ossia la decomposizione di una classica convoluzione in due

- Un livello di convoluzione che applica indipendentemente un filtro per ogni canale (profondità) dell'input;
- Un *pointwise convolutional layer* (convoluzione 1x1) il quale combina l'output della convoluzione.

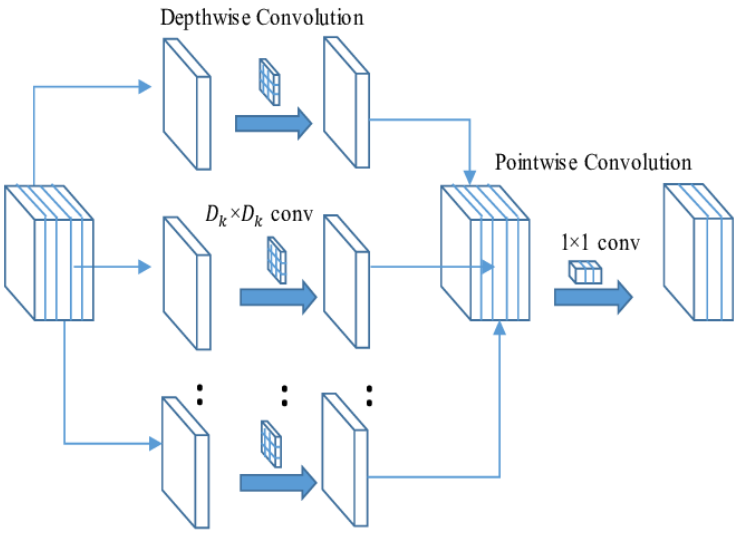

Figura 4: Schematizzazione *Depthwise Separable Convolution*, mostrando come dal volume di input si applica un *kernel* per ogni livello di profondità dell'input, e successivamente viene effettuata la *pointwise convolution*, ottenendo un volume di output come per una convoluzione classica, ma in maniera decisamente più efficiente.

Applicando in sequenza una *depthwise convolution* e una *pointwise convolution* è possibile ottenere un volume di output con le stesse dimensioni di quello ottenuto tramite una convoluzione classica.

Ad oggi (Marzo 2025), MobileNet ha subito quattro re-iterazioni le quali apportano notevoli miglioramenti nonostante le sempre maggiori prestazioni dei dispositivi per cui nasce. Ad esempio MobileNetV4 [Qin+24] introduce ulteriori componenti come *Universal Inverted Bottleneck* per

fornire maggiore flessibilità nel mescolare informazioni spaziali e di canale o *Mobile MQA* ossia un blocco di *attention* ottimizzato per l'accelerazione in dispositivi mobili (e.g. linea smartphone Google Pixel) che permette un incremento di velocità del 39% rispetto all precedenti implementazioni di *multi-head attention*.

2.2.3 EfficientDet

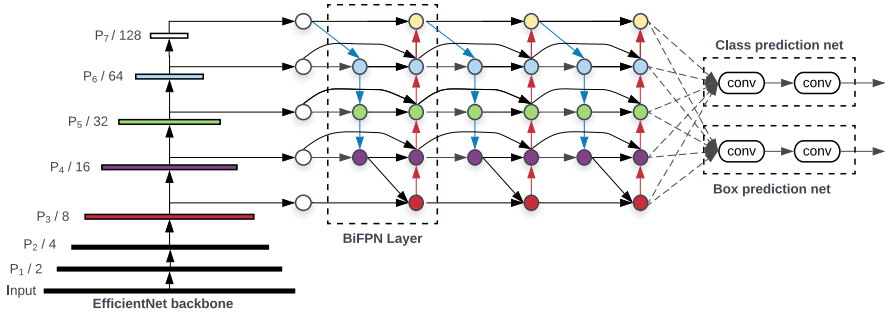

Figura 5: Architettura di *EfficientDet*, composto da *EfficientNet* come backbone, BiFPN come *feature network* e un *predictor* condiviso per le classi e bounding box.

EfficientDet [TPL19] nasce dallo sforzo di analisi ed ottimizzazione dei design delle architetture di reti neurali per la task di *object detection*, introducendo ottimizzazioni chiave per migliorare l'efficienza delle reti. Come mostrato in Figura 5 L'architettura di EfficientDet è principalmente composta da tre reti. La prima è rappresentata dal backbone, ossia l'estrattore di *feature* di alta qualità a varie scale, in questo caso *EfficientNet* [TL19]. Successivamente viene introdotta la rete *BiFPN* pesata, la quale facilita la fusione efficiente delle *feature* estratte su più scale dal backbone. Infine, le *feature* ottenute dalla loro fusione sono date in input ad una succesiva rete per la classificazione e la predizione delle *bounding box*. L'impiego di questa struttura porta a due importanti vantaggi:

- Raggiungere un'elevata precisione con un numero significativamente inferiore di parametri;
- Fornisce metodi di *riscalatura composita*, ossia meccanismi per poter dinamicamente aumentare uniformemente la risoluzione, la profondità e la larghezza della rete, ottimizzando prestazioni senza incrementare eccessivamente il costo computazionale.

2.3 RetinaNet

RetinaNet è un modello di object detection progettato per affrontare il problema dello squilibrio tra classi. Lo squilibrio tra classi si verifica quando il numero di esempi appartenenti alla classe di sfondo (regione senza oggetti) è significativamente superiore a quello delle classi di oggetti.

2.3.1 Innovazione della Focal Loss

La principal innovazione di RetinaNet è l'introduzione della *Focal Loss*, una funzione di loss modificata che reduce l'influenza degli esempi facilmente classificabili e aumenta il peso di quelli difficili, gestendo meglio la predominanza di sfondi negative rispetto agli oggetti di interesse, migliorando soprattutto le prestazioni nella rilevazione di oggetti piccolo o poco rappresentati nel dataset.

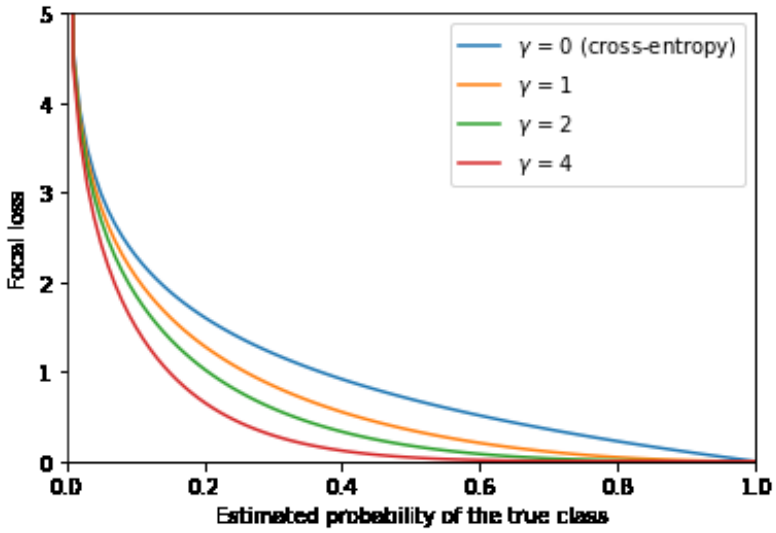

Figura 6: focal loss in funzione della probabilità prevista della classe reale, con valori variabili di γ

2.3.2 Architettura di RetinaNet

L'architettura di RetinaNet è costituita da:

- **Backbone**: una rete convoluzionale pre-addestrata, tipicamente ResNet o ResNeXt, utilizzata per estrarre feature da diversi livelli dell'immagine. In particolare, RetinaNet utilizza una rete backbone basata su Feature Pyramid Network (FPN), che sopra una ResNet feedforward, genera una ricca piramide di feature convoluzionali multi-scala.
- **Feature Pyramid Network (FPN)**: una struttura che consente di combinare informazioni multi-scala per il rilevamento di oggetti di diverse dimensioni. Questa struttura permette a RetinaNet di mantenere prestazioni elevate nel rilevamento su scale diverse.
- **Head per classificazione e regressione**: due sottoreti separate che si occupano rispettivamente della classificazione delle ancore (anchor boxes) e della regressione delle ancore per ottenere le vere bounding box degli oggetti.

L'architettura di RetinaNet è progettata intenzionalmente in modo semplice, permettendo di concentrarsi su una funzione di perdita innovativa, la *focal loss*, che elimina il gap di accuratezza tra il nostro rilevatore one-stage e i rilevatori two-stage di ultima generazione come Faster R-CNN con FPN, pur mantenendo una velocità di esecuzione superiore.

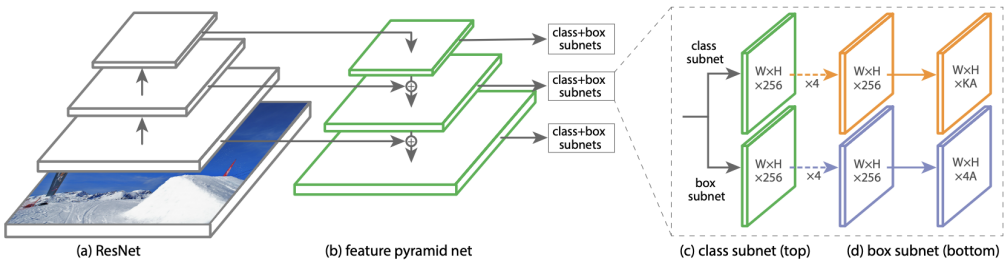

Figura 7: Architettura di RetinaNet

3 Approccio Sviluppato

3.1 Gestione delle ROI sovrapposte

3.1.1 Region Of Interest

La *Region of Interest (ROI)* è un'area specifica di un'immagine che viene selezionata per l'analisi, tipicamente nella *computer vision* e nell'*object detection*. Le *ROI* sono fondamentali per delimitare gli oggetti individuati da un modello di rilevamento. Tuttavia, un problema comune che si presenta è la sovrapposizione tra diverse *ROI*, specialmente quando più oggetti sono rilevati in prossimità o quando lo stesso oggetto viene individuato più volte con diverse *bounding box*.

3.1.2 Intersection Over Union

Per gestire la sovrapposizione delle *ROI* si utilizza il concetto di *Intersection over Union (IoU)*, che misura il grado di sovrapposizione tra due *bounding box*. L'*intersection over union* è definito come il rapporto tra l'area di intersezione delle due regioni e l'area della loro unione:

$$IoU = \frac{\text{Area di Intersezione}}{\text{Area di Unione}}$$

Siano B_1 e B_2 due *bounding box*, l'IoU si calcola come:

$$IoU = \frac{|B_1 \cap B_2|}{|B_1 \cup B_2|}$$

dove $|B_1 \cap B_2|$ rappresenta l'area comune tra le due *bounding box*, mentre $|B_1 \cup B_2|$ è l'area totale coperta da entrambe. L'IoU è ampiamente utilizzato nei metodi di Non-Maximum Suppression (NMS), che selezionano la *bounding box* con il punteggio più alto e rimuovono le altre che hanno un IoU superiore a una soglia predefinita. In questo modo, si elimina la ridondanza e si conserva solo la *bounding box* più rappresentativa per ogni oggetto rilevato, migliorando la precisione dell'*object detection*.

3.2 YOLO - Iorio Matteo

Qui di seguito andrò ad elencare le modalità con la quale ho strutturato il mio sistema, per ulteriori informazioni a riguardo consiglio vivamente di esplorare il file *jupyter* che contiene l'intero esperimento (link).

3.2.1 Design del sistema

Durante la fase di analisi e studio del dominio, più nello specifico nella fase di individuazione del possibile *dataset* da utilizzare, mi sono accorto che ogni singolo *dataset*, preso a se, non contenesse una buona combinazione di immagini di segnali stradali e di semafori. O per la maggior parte dei casi il singolo *dataset* non superava le mille immagini, e dal momento in cui l'utilizzo di reti profonde permette di ottenere buoni risultati con *dataset* corposi, ho deciso di spezzare e specializzare un singolo *object detector* YOLO su un solo specifico task, in pratica ho utilizzato un'architettura per i segnali stradali ed un'architettura per i semafori, in questo modo ero sicuro di avere le competenze separate e di utilizzare *dataset* molto ricchi in immagini per il singolo classificatore. La collezione di immagini che ho scelto per il classificatore dei segnali stradali è importato da *Roboflow* [usm24] il task dei semafori ho deciso di prendere un *subset* del *dataset* di *lisa-traffic* [Jen+16] dal momento in cui il *dataset* originale contiene più di cento mila immagini di semafori.

3.2.2 Fusione dei classificatori

Dal momento in cui il mio sistema utilizza due modelli `YOLO`, ho deciso di realizzare una classe denominata `Detectors`, la quale prende in input una lista di modelli ed espone una sola api,

denominata `detect` la quale prende in input un frame e restituisce in output tutte le *bounding box* e le varie classi individuate da ogni singolo modello. questo approccio permetterebbe anche una esecuzione parallela/distribuita della singola *detection*. Questo mi ha permesso di fondere l'utilizzo dei due modelli e ricercare nell'immagine di input sia segnali stradali *traffic signs* e sia i semafori *traffic ligths*.

3.2.3 Fase di Training

Per quanto riguarda invece la fase di training, ho deciso di optare per un numero di epoche che non fosse estremamente elevato, dal momento in cui le capacita computazionali fornite da *kaggle* sono limitate ad un breve periodo di tempo. Per ovviare a questo problema, dunque ho deciso di optare per un numero di epoche pari a trenta (per entrambe le architetture `YOLO`). Come ottimizzatore per entrambe le architetture, ho deciso di lasciarle entrambe su *auto*. Per quanto riguarda il *pre-processing* delle immagini ho lasciato le impostazioni di default, che prevedono *blurring* dell'immagine ed altre operazioni di *augmentation* e modifica sulle immagini di training. I *dataset* utilizzato erano nel formato `YOLOv11`, in questo modo non è stato necessario applicare delle modifiche all'intero *dataset*.

3.3 Faster R-CNN - Furi

Quanto discusso in Sezione 2.2 ha portato alla scelta del modello Faster R-CNN per la task di object detection. In particulare, il modello è composto da una "testa" Faster R-CNN ed un backbone: nel contesto di reti neurali efficienti per sistemi con risorse limitate è stato optato per MobileNetV3-FPN Large, un modello piuttosto leggero rispetto a reti come ResNet e AlexNet e reti SSD (eccetto l'alternativa SSD-Lite, la quale però performa peggio). Il modello di MobileNetV3-FPN non richiede input di dimensioni specifiche poiché riscala le immagini ad 800x800, da qui "Large" (la versione più leggera accetta solo input 320x320).

Le motivazioni dietro la scelta di questa rete, oltre ai vantaggi illustrati nelle sezioni precedenti, risiede anche nella facilità d'impiego di quest'ultima. L'implementazione e la manipolazione attraverso PyTorch ci risulta più semplice rispetto all'uso di Tensorflow o Keras, il quale fornisce API più intuitive e bene o male standard per molti modelli. D'altra parte, la scelta di EfficientDet è stata considerata e testata, risultando molto complicato effettuare fine tuning sulla rete per il nostro dominio applicativo, portando quindi alla scelta di Faster R-CNN.

Fine Tuning per il processo di fine tuning, è stato preso il backbone MobileNetV3-FPN pre-addestrato su COCO-V1 [2], al quale sono stati fissati i parametri non permettendo quindi una loro modifica in fase di training. Successivamente si sostituisce nella test FasterRCNN il *box predictor*, in quanto questo è oggetto di training nella fase successiva e necessita di essere adattato al nostro dominio applicativo, specificandone il numero di classi ed ereditando le *feature* estratte durante l'addestramento. Successivamente sono definiti gli *split* del dataset e una pipeline di data augmentation per il training set. I successivi dettagli di training sono specificati in Sezione 4.2.1.

Object Tracking Come per gli approcci dei miei colleghi, per Faster R-CNN è stato definito un 'FasterRCNNDetector' in grado di effettuare predizioni su singoli frame, impostando una soglia per l'accettazione o lo scarto della label e bounding box restituita in output. Attraverso il calcolo dell'Intersection over Union mostrato in Sezione 3.1.2, è così possibile creare un object tracker in grado di unire le singole predizioni del *detector* ed effettuare il tracking calcolando l'IoU tra il frame precedente e il corrente, in modo da mantenere e far scorrere eventuali bounding box che si riferiscono al medesimo oggetto nei due frame. Infine, tramite un video di test generato dal mio collega, viene salvato un nuovo video contenente l'object tracking effettuato dal modello.

[2] https://cocodataset.org/

3.4 Retina Net - Vincenzi

3.4.1 Architettura e Funzionamento

Retina Net è caratterizzato dall'utilizzo di una backbone profonda (in questo caso, la rete ResNet-101 in combinazione con Feature Pyramid Networks - FPN) che consente di estrarre rappresentazioni multiscala dall'immagine. Come riportato nella Sezione Sezione 2.3 l'elemento distintivo del modello è la focal loss, concentrando il training sulle difficoltà di classificazione degli oggetti meno rappresentati. Questo approccio si traduce in una maggiore robustezza nella rilevazione di oggetti di dimensioni e caratteristiche differenti.

3.4.2 Fase di Training e Integrazione nel Workflow

Il training del modello Retina Net è stato configurato utilizzando il framework Detectron2. In particolare:

- **Dataset e Preprocessing**: I dataset in formato COCO sono stati registrati e caricati, con il pre-processing delle immagini gestito tramite le pipeline standard offerte dalla libreria.
- **Configurazione degli Iperparametri**: Sono stati impostati parametri quali il learning rate, il numero di iterazioni (con una stima di 5 epoche), e il numero di classi.

3.4.3 Visualizzazione ed Object Tracking

Oltre alla fase di detection, è stato implementato anche un modulo di object tracking, simile a quello realizzato dai colleghi per YOLO e Faster R-CNN. Il modulo di tracking si basa sul calcolo dell'Intersection over Union (IoU) tra bounding box su frame consecutivi, e consente di assegnare un ID coerente agli oggetti nel tempo.

A partire dal modello RetinaNet fine-tuned, è stata realizzata la classe RetinaNetDetector per effettuare la detection su singoli frame video. Successivamente è stata sviluppata la classe IOUTracker che mantiene e aggiorna gli ID degli oggetti rilevati frame dopo frame.

Infine, è stato scritto uno script per processare un video in input e produrre un nuovo video annotato contenente le predizioni del modello e gli ID assegnati agli oggetti tracciati. Il video ottenuto mostra chiaramente la capacità del modello di rilevare e seguire oggetti, anche in presenza di più elementi nella scena.

4 Valutazione Sperimentale

4.1 Risultati YOLO - Iorio Matteo

La strategia di aver suddiviso i due compiti in due architetture separate mi ha permesso di migliorare nettamente le prestazioni dei due modelli, come si andrà a notare nei capitoli successivi. Per l'addestramento sono stati utilizzati i dataset nel formato YOLOv11.

4.1.1 Architettura per i segnali stradali

I primi risultati che andrò a descrivere riguardano l'architettura per i segnali stradali, il cui *dataset* era cosi suddiviso:

- *training set*: 7092 immagini ($\sim 71\%$)
- *validation set*: 1884 immagini ($\sim 19\%$)
- *test set*: 1024 immagini ($\sim 10\%$)

Come si puo notare dalla dimensione del *dataset*, abbiamo esattamente dieci mila immagini in totale, un quantitativo che pur sempre minimo permette di fornire una conoscenza di base alla rete neurale. I risultati dell'architettura specializzata nei segnali stradali, ha fornito degli ottimi risultati, come si può notare dall'immagine 8. Nello specifico, la seguente matrice di confusione dimostra chiaramente quanto accurato il nostro modelli riesca ad essere nonostante le pochissime epoche di addestramento che sono state fornite.

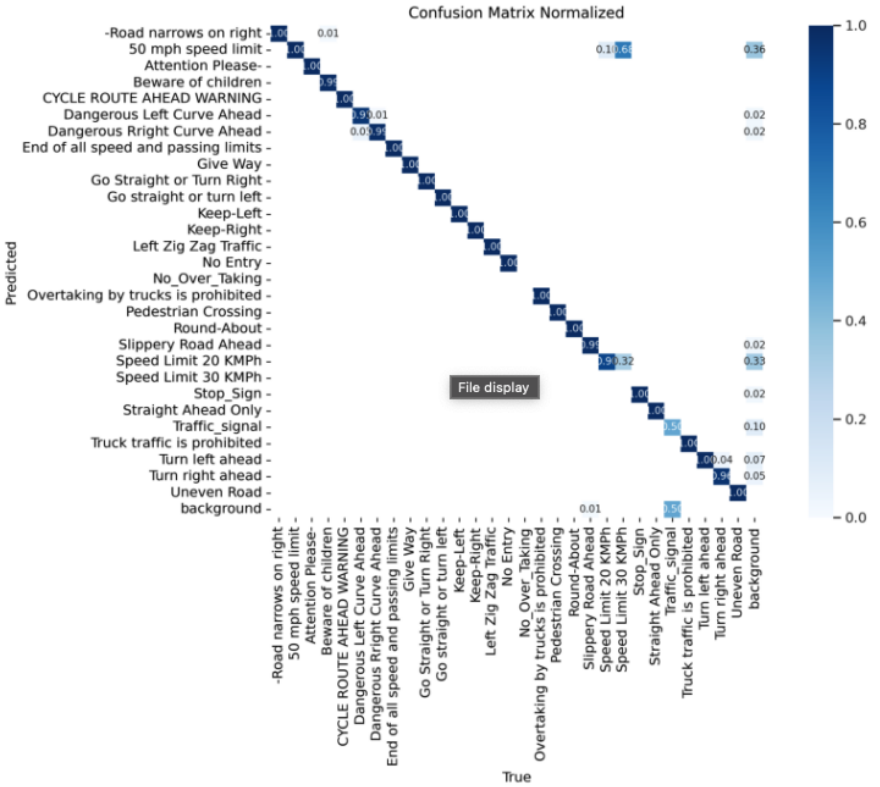

Figura 8: Matrice di confusione riguardante l'architettura dei segnali stradali

Per quanto riguarda il training effettivo sull'intero *dataset*, i risultati che sono stati ottenuti sono raggruppati nei grafici in Figura-9

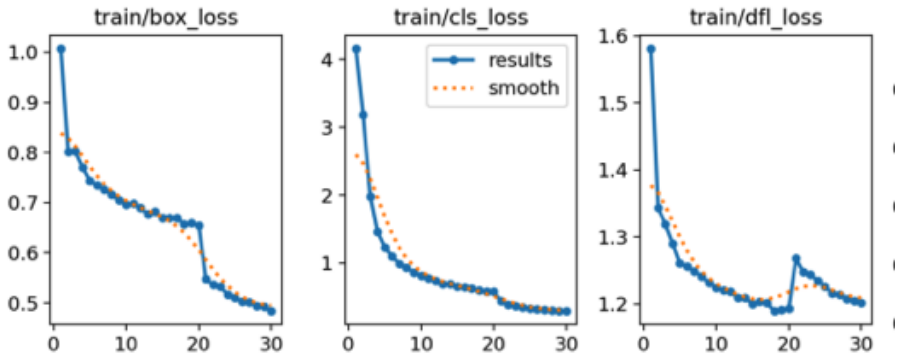

Figura 9: Fase di training sul dataset dei segnali stradali

Nonostante il numero di epoche fosse estremamente basso, siamo comunque riusciti a ottenere ottimi risultati sia per quanto riguarda la *box loss* che la *class loss*. È stato inoltre fondamentale verificare che le prestazioni sul *validation set* fossero soddisfacenti e che il modello non avesse manifestato *overfitting* (Figura 10).

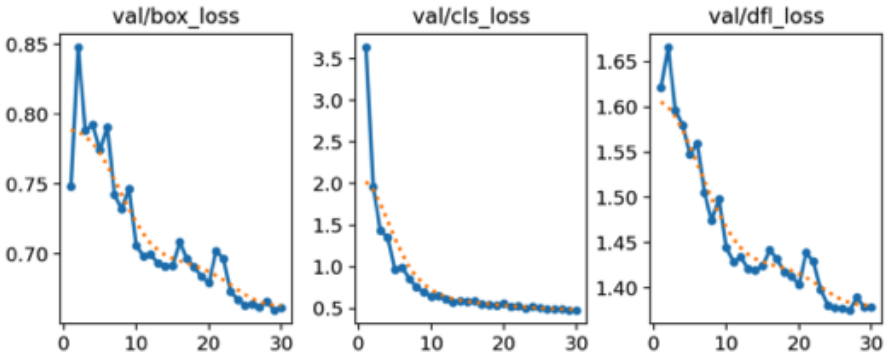

Figura 10: Grafici riguardanti il validation set dei segnali stradali

4.1.2 Osservazioni Finali

Qui di seguito verranno riportati i principali dati rilevati sul dataset di validation.

Class	Images	Instances	P	R	mAP@50	mAP@50-95
all	1884	1886	0.941	0.926	0.937	0.797
Road narrows on right	54	54	0.994	1.000	0.995	0.880
50 mph speed limit	62	62	0.690	0.984	0.896	0.731
Attention Please-	117	117	0.999	1.000	0.995	0.853
Beware of children	106	106	1.000	0.992	0.995	0.880
CYCLE ROUTE AHEAD WARNING	54	54	1.000	0.963	0.995	0.859
Dangerous Left Curve Ahead	42	42	1.000	0.967	0.995	0.855
Dangerous Rright Curve Ahead	72	72	0.989	0.986	0.995	0.833
End of all speed and passing limits	48	48	0.998	1.000	0.995	0.828
Give Way	107	107	1.000	0.998	0.995	0.841
Go Straight or Turn Right	79	79	1.000	0.992	0.995	0.855
Go straight or turn left	42	42	0.998	1.000	0.995	0.860
Keep-Left	64	64	0.999	1.000	0.995	0.850
Keep-Right	82	82	1.000	0.999	0.995	0.820
Left Zig Zag Traffic	64	64	0.999	1.000	0.995	0.853
No Entry	83	83	1.000	1.000	0.995	0.805
Overtaking by trucks is prohibited	47	47	0.997	1.000	0.995	0.834
Pedestrian Crossing	47	47	0.997	1.000	0.995	0.859
Round-About	67	67	0.999	1.000	0.995	0.873
Slippery Road Ahead	101	101	0.999	0.990	0.995	0.891
Speed Limit 20 KMPh	48	48	0.757	0.833	0.804	0.638
Speed Limit 30 KMPh	75	75	0.000	0.000	0.000	0.000
Stop_Sign	32	32	1.000	0.986	0.995	0.912
Straight Ahead Only	63	63	0.998	1.000	0.995	0.864
Traffic_signal	4	6	0.939	0.333	0.654	0.400
Truck traffic is prohibited	83	83	0.998	1.000	0.995	0.844
Turn left ahead	84	84	0.996	1.000	0.995	0.847
Turn right ahead	80	80	1.000	0.916	0.995	0.871
Uneven Road	77	77	0.999	1.000	0.995	0.884

Tabella 1: Risultati prodotti dalla *detection*, sono inclusi la *Precision* (P), la *Recall* (R), *mAP@50* e *mAP@50-95*.

La tabella sopra presenta le metriche di valutazione delle prestazioni del rilevamento degli oggetti su 29 diverse classi di segnali stradali. Le metriche includono *Precision* (P), *Recall* (R), *mAP@50* e *mAP@50-95*. Di seguito è riportato un riepilogo e un'analisi dei risultati:

- **Prestazioni Generali:** Il modello raggiunge prestazioni complessive elevate con una *Precision* di 0.941, *Recall* di 0.926, *mAP@50* di 0.937 e *mAP@50-95* di 0.797. Questi valori indicano che il rilevatore è sia accurato che affidabile nella maggior parte delle categorie.
- **Classi con Alte Prestazioni:** Diverse classi mostrano un rilevamento quasi perfetto, con valori di *Precision*, *Recall* e mAP molto vicini a 1. Esempi notevoli includono:
  - *Attenzione ai bambini*, *Dare precedenza*, *Segnale di stop*, *Svolta a destra*, *Attraversamento pedonale* e *Divieto di transito ai camion*, tutte con precisione e richiamo perfetti o quasi perfetti.
  - Questi segnali tendono ad essere visivamente distintivi e annotati in modo coerente, contribuendo a un'elevata performance di rilevamento.
- **Prestazioni Moderate:**
  - *Limite di velocità 50 mph* (P = 0.69) e *Limite di velocità 20 KM/h* (P = 0.757) mostrano una *Precision* ragionevole ma subottimale, suggerendo occasionali errori di classificazione.
  - Nonostante l'alto richiamo in alcuni casi, una precisione più bassa può indicare confusione con segnali visivamente simili.
- **Prestazioni Scarse:**
  - *Semaforo* presenta un *Recall* particolarmente basso (0.333) e un mAP@50 relativamente basso (0.654), probabilmente a causa del numero molto ridotto di campioni (4 immagini, 6 istanze), che potrebbe non essere sufficiente per un addestramento efficace.

– *Limite di velocità 30 KM/h* mostra punteggi pari a zero per tutte le metriche, indicando che il modello non è riuscito a rilevare affatto questa classe. Ciò è probabilmente dovuto a una mancanza di dati di addestramento, a problemi di annotazione o a un'elevata variabilità all'interno della classe.

- **Coerenza tra le Classi:** La maggior parte delle classi raggiunge valori mAP@50 superiori a 0.85, il che indica una capacità costante di localizzazione e classificazione da parte del rilevatore. I valori di mAP@50-95, sebbene più bassi a causa di soglie IoU più rigorose, rimangono comunque elevati per la maggior parte delle classi, riflettendo previsioni precise delle bounding box.

Conclusione: Il modello di rilevamento mostra prestazioni eccellenti nella maggior parte delle classi di segnali stradali, in particolare per quelle con un numero sufficiente di esempi di addestramento e caratteristiche visive distintive. Alcune classi richiedono ulteriore attenzione, sia aumentando i dati di addestramento che risolvendo eventuali incoerenze nelle annotazioni. I miglioramenti futuri potrebbero concentrarsi sulle classi con prestazioni inferiori ed esplorare tecniche di data *augmentation* per migliorare le performance in presenza di un numero limitato di campioni.

4.1.3 Architettura per i semafori

La seconda architettura che ha necessitato della fase di *training* riguardava esclusivamente i solo dati sui semafori. Il *dataset* utilizzato è cosi suddiviso:

- *training set*: 6251 immagini ($\sim 70\%$)
- *validation set*: 1784 immagini ($\sim 20\%$)
- *test set*: 900 immagini ($\sim 10\%$)

Per un totale di otto mila immagini. Anche l'architettura specializzata solamente sui semafori, ha fornito in output degli ottimi risultati, come si può vedere dalla matrice di confusione (Figura-11).

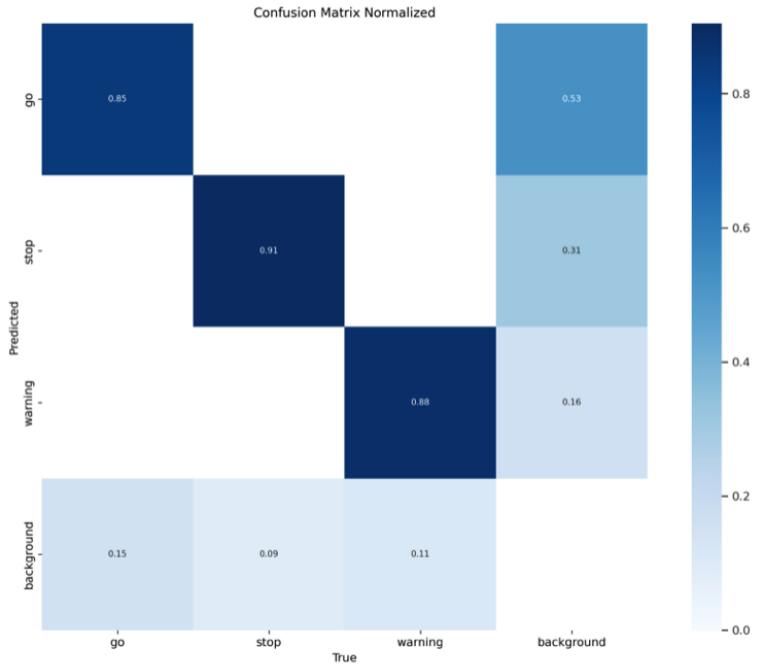

Figura 11: Matrice di confusione riguardante l'architettura dei semafori

Dal momento in cui però il *dataset* utilizzato presenta all'incirca diecimila immagini, si possono sicuramente ottenere risultati ancora migliori se si dovesse aumentare il numero di epoche per il training, in modo da fornire un migliore addestramento su tutte le immagini del *dataset*. Come si può notare dalla fase di training (Figura-12) abbiamo comunque raggiungo un buon livello di *loss*, ma sicuramente se avessimo fornito un numero maggiore di epoche, avremo potuto ottenere risultati sicuramente migliori come ci indica il grafico *train/box_loss*.

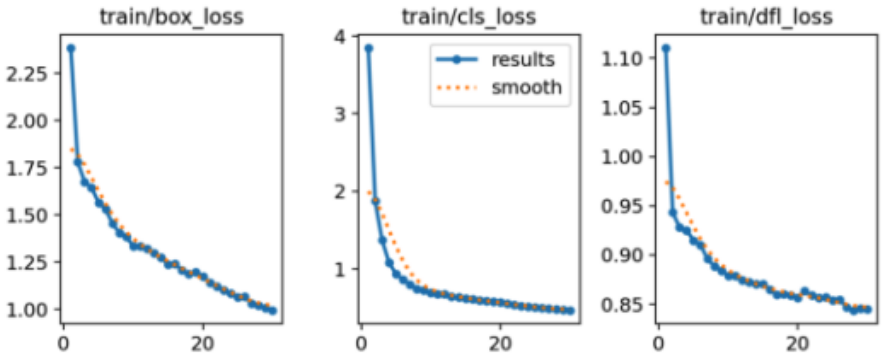

Figura 12: Training sul dataset dei semafori

Per quanto riguarda il *validation set* anche qui era importante non cadere nel rischio dell'*overfitting*. Anche in questo caso appunto possiamo notare che ci sia ancora un margine di miglioramento per quanto riguarda la fase del training.

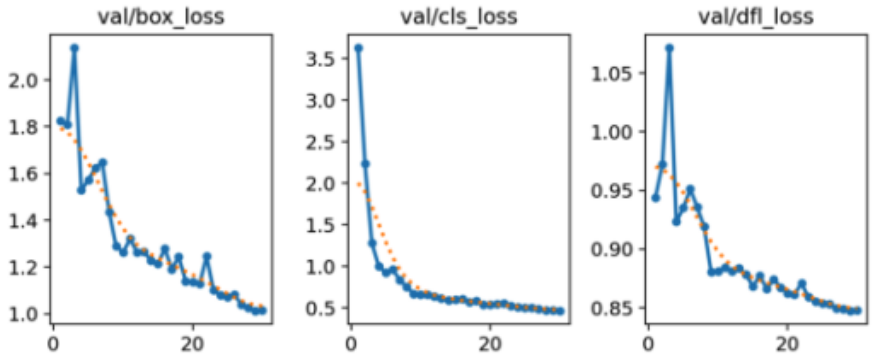

Figura 13: Grafici riguardanti il validation set dei semafori

4.1.4 Osservazioni Finali

Qui di seguito verranno riportati in formato tabulare i risultati che sono stati ottenuti sui dati del validation set.

Class	Images	Instances	P	R	mAP@50	mAP@50-95
all	1784	5607	0.930	0.860	0.922	0.607
go	921	2499	0.929	0.820	0.907	0.593
stop	815	2527	0.953	0.892	0.955	0.736
warning	244	581	0.907	0.869	0.903	0.491

Tabella 2: Performance nelle *detection*, sono utilizzate *Precision* (P), *Recall* (R), mean Average Precision at IoU 0.50 (mAP@50), and across 0.50–0.95 (mAP@50-95).

La tabella sopra presenta le prestazioni del rilevamento degli oggetti in tre categorie semantiche: *go*, *stop* e *warning*. Le metriche di valutazione includono *Precision* (P), *Recall* (R) e *mean Average Precision* (*mAP*) a soglie di IoU pari a 0.50 e da 0.50 a 0.95.

- **Prestazioni Generali:** Il modello mostra buone prestazioni complessive, con un'elevata *precision* (0.93), una solida *recall* (0.86) e un'ottima accuratezza di localizzazione (*mAP@50* = 0.922). Tuttavia, il punteggio *mAP@50-95* scende a 0.607, suggerendo che la localizzazione delle bounding box diventa meno precisa sotto condizioni IoU più restrittive.
- **Classe Stop:**
  - Questa classe mostra le prestazioni più elevate, con la massima *precision* (0.953), *recall* (0.892) e in particolare *mAP@50* (0.955), indicando un rilevamento molto affidabile e bounding box accurate.
  - Il punteggio *mAP@50-95* di 0.736 è significativamente più alto rispetto alle altre categorie, riflettendo un'eccellente precisione spaziale attraverso diverse soglie IoU.
- **Classe Go:**
  - La classe "go" mostra anch'essa buone prestazioni, con *precision* pari a 0.929 e una *recall* leggermente più bassa (0.820). Il calo nella *recall* suggerisce alcuni falsi negativi, potenzialmente dovuti a occlusioni o pattern visivi confusi.
  - Il punteggio *mAP@50-95* è di 0.593, che, pur essendo buono, indica margini di miglioramento nella localizzazione precisa delle bounding box.
- **Classe Warning:**
  - Sebbene *precision* (0.907) e *recall* (0.869) rimangano elevate, il punteggio *mAP@50-95* scende a 0.491—notevolmente inferiore rispetto alle altre classi.
  - Ciò può essere dovuto al numero più ridotto di istanze (581), a una minore coerenza visiva tra i segnali di avviso, o a una maggiore somiglianza tra classi.
  - Maggiori dati di addestramento o tecniche di data augmentation potrebbero aiutare questa categoria a migliorare la precisione di localizzazione.
- **Andamenti delle Prestazioni:**
  - Il fatto che la *precision* complessiva sia superiore alla *recall* suggerisce che il modello sia conservativo nelle sue rilevazioni—evitando falsi positivi, ma potenzialmente perdendo alcuni veri positivi.
  - Il calo da *mAP@50* a *mAP@50-95* indica che, sebbene il modello rilevi spesso l'oggetto corretto, la precisione della bounding box può ancora essere migliorata.

Conclusione: Il modello di rilevamento si comporta in modo affidabile su tutte le classi, con prestazioni particolarmente eccellenti nella classe *stop*. Sebbene anche le classi *go* e *warning* raggiungano buoni risultati, il miglioramento dell'accuratezza spaziale delle bounding box—specialmente per la classe warning—dovrebbe essere una priorità nei lavori futuri. La raccolta di campioni più variati e l'ottimizzazione con funzioni di perdita sensibili all'IoU potrebbero migliorare ulteriormente la qualità del rilevamento.

4.1.5 Conclusioni

La tecnica di fusione delle due reti neurali, ha permesso una migliore separazione delle mansioni, andando cosi ad utilizzare due modelli in parallelo per lavorare su classi completamente diverse, questo ha fatto sii che si potessero usare dataset ridotti ma molto piu specifici riguardanti problemi diversi (semafori e cartelli stradali). Ritengo che la mia scelta, date le circostanze, sia stata azzeccata, dal momento in cui specializzando due reti diverse e mettendole a lavorare assieme su ogni frame sono in grado di rilevare i soli loro oggetti di competenza.

4.2 Risultati Faster R-CNN - Furi Stefano

Il modello Faster R-CNN con backbone MobileNetV3 è stato addestrato sul dataset "*Traffic and Road Signs Dataset*" [usm24], composto da 10.000 stradali classificate secondo 30 classi tra cui limiti di velocità, segnali di precedenza, attraversamenti pedonali, segnale di stop, etc.

Il dataset viene adattato ad un oggetto `Dataset` di PyTorch, poiché è necessario applicare delle conversioni nella presentazione dei dati alla rete, la quale è pre-addestrata attraverso COCO-v1 (in contrasto con il formato "yolo" del dataset impiegato). La divisione in test, validation e training set è stata effettuata come segue:

- train: 7092 immagini
- validation: 1884 immagini
- test: 1024 immagini

4.2.1 Training

La rete, come detto in precendenza in Sezione 3.3, è pre-addestrata sul dataset COCO-v1. Risulta quindi necessaria una fase di *fine-tuning* per il nostro specifico problema. Questo avviene mantenendo invariato il backbone, fissando i suoi parametri (non verrano quindi riappresi parametri per il backbone in fase di fine-tuning), e sostituita la testa del modello con un `FatRCNNPredictor` adattato per dimensione al numero di classi del dataset (30), più un'ulteriore classe per la classificazione di elementi di *background* (i.e. nessuna classificazione). Successivamente definiamo una pipeline di image augmentation e data transformation rispettivamente per (*i*) applicare piccole rotazioni casuali (simulazione di rotazioni di camera), (*ii*) introdurre *shift* casuali sulle immagini di al massimo il 10%, (*iii*) applicare variazioni di luminosità e contrasto (simulazione diverse condizioni ambientali) e infine (*iv*) trasformare l'immagine in un tensore floating point di PyTorch. All'interno di questa pipeline è inoltre possibile specificare numerose ulteriori opzioni per la trasformazione delle immagini, ma ciò incrementa notevolmente le tempistiche di training non garantendo un vantaggio sostanzioso in termini prestazioni del modello. Per motivi di semplicità lasciamo questa opzione per miglioramenti futuri di questo modello. A questo punto il modello è stato addestrato per 15 epoche, con *SGD* come ottimizzatore impostando come valore iniziale un learning rate piuttosto basso (0.5e-3), avvalendoci inoltre di uno *scheduler* per quest'ultimo.

Il progresso di training è riassunto dai grafici in Figura 14. Come si evince dal primo grafico, la *loss* si stabilizza intorno alla quarta epoca, mostrando una netta convergenza verso valori prossimi allo 0.7 già dalla quinta epoca. Presi singolarmente i contributi della *loss*, si comprende come si ha un basso errore sia in classificazione sia per il *bounding box regressor*, rispettivamente al termine dell'ultime epoca di 0.3339 e di 0.1555.

Questi risultati si traducono nelle metriche esposte in Tabella 3. Per via anche delle dimensioni del dataset e anche dalla diversa composizione dei tre dataset, il modello ottiene risultati decisamente inferiori rispetto ai modelli esposti all'interno di questo documento. Le capacità del modello sono appena sufficienti sul dataset di validazione, con risultati migliori per la task di classificazione, ma solo su oggetti di medie e grandi dimensioni, d'altra parte presentando scarse prestazioni su oggetti piccoli. Questo può anche essere facilmente verificabile attraverso il video generato dalla rete al termine del notebook.

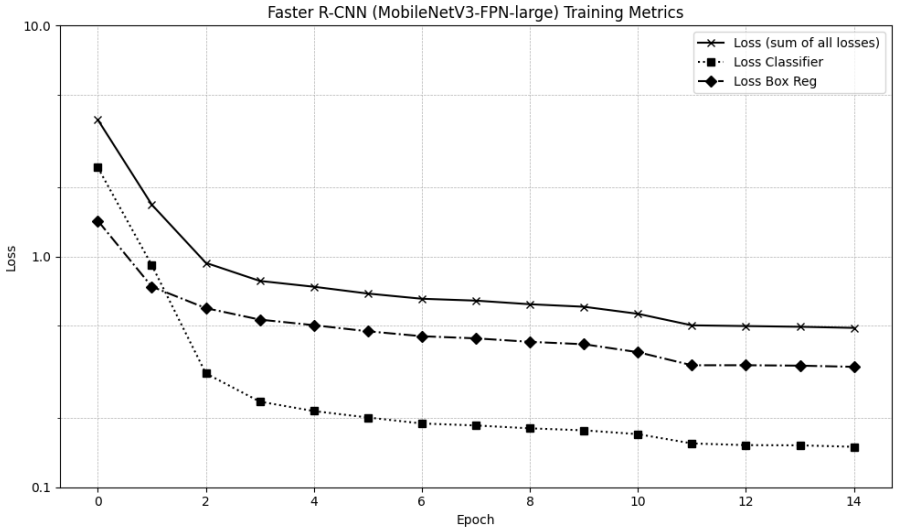

Figura 14: Metriche di training per la rete *Faster R-CNN* con MobileNetV3-FPN come backbone.

Metric	Value	Interpretation
Mean Average Precision (mAP)		
Overall mAP	0.7274	Average precision across all classes
mAP @ 0.50 IoU	0.8967	Precision at 50% overlap threshold
mAP @ 0.75 IoU	0.8786	Precision at 75% overlap threshold
Average Recall (AR)		
AR @ 1 detection	0.7692	Recall with top 1 detection
AR @ 10 detections	0.7739	Recall with top 10 detections
AR @ 100 detections	0.7751	Recall with top 100 detections

Tabella 3: Metriche di valutazione per il modello *Faster R-CNN* con MobileNetV3-FPN come backbone. La rete presenta discrete capacità di discriminazione per quanto riguarda la task di classificazione, mentre presenta scarsi risultati in presenza di oggetti piccoli e risultati nettamente migliori con oggetti di medie e grandi dimensioni. Infine, i valori di *recall* suggeriscono che ulteriori *detection* oltre le 10 dimostrano come il modello non predice un numero superiore di elementi all'interno dell'immagine rispetto al *ground truth*, dimostrando quindi una certa consistenza.

È inoltre interessante ai fini delle motivazioni che hanno spinto all'esplorazione di reti neurali per l'*object detection* per sistemi embedded o mobile, illustrare le metriche esposte in Tabella 4. Queste sono state ispirate dal grafico mostrato in [TPL19] per la comparazione delle prestazioni dei modelli in rispetto al dataset COCO-v1 e il il numero di *floating point operations* (FLOPs) espressi in miliardi. A confronto delle versioni stato dell'arte di EfficientDet esposte nel paper, *Faster R-CNN* con MobileNetV3 come backbone è circa paragonabile in termini di numero di parametri addestrabili ad EfficientDet D4. La rete proposta in questo elaborato presenta un quarto del numero di FLOPs (11.9 miliardi contro 55 miliardi) presentando inoltre un tempo inferiore in fase di *inference* (14.6ms contro 42.8ms), considerando però la diversità della GPU utilizzate (NVIDIA Tesla T4 contro NVIDIA Titan V, quest'ultima generalmente più lenta in fase di inferenza ma migliore per training). Infine considerando queste tempistiche è interessare capire in linea teorica le possibili capacità di *real-time* di questa rete, pensando a scenari di sistemi a basse prestazioni montati on-board. Senza l'impiego di accelerazione attraverso schede grafiche, è possible processare 2.92 frame per secondo, risultando subottimale in contesti critici; d'altra parte, un limite massimo di circa 68.39 frame per secondo risulta un numero più che accettabile per sistemi real-time, considerando come *frame-rate* di riferimento 24/30fps per sistemi di cattura

video moderni.

Metric	Value	Additional Information
Computational Complexity		
MACs	5.95 B	Multiply-Accumulate Operations
FLOPs	11.9 B	Floating Point Operations
Parameters	19.01 M	Model Size
Inference Performance		
CPU Average Time	0.3426 s	Average of 100 Runs
GPU Average Time	0.0146 s	Average of 100 Runs
Maximum CPU FPS	2.92	Theoretical frames processible per second
Maximum GPU FPS	68.39	Theoretical frames processible per second

Tabella 4: Caratteristiche e risultati pratici della rete *Faster R-CNN* con MobileNetV3-FPN come backbone. La rete presenta un modesto numero di parametri con una relativament limitata complessità computazionale rispetto ad altri modelli per la task di *object detection*. Attraverso l'accelerazione con GPU si ottengono miglioramenti in termini di tempistiche di inferenza di circa 23 volte, equivalenti a circa 65 frame in più processabili per ogni secondo.

4.2.2 Possibili cause d'errore

Per poter stabilire le possibili cause d'errore è interessante mostrare la matrice di confusione sul validation set in Figura 15 insieme alla Sezione 4.2.2 di mAP e mAR per ogni classe, per poter indicare possibili classi problematiche. Per quanto la rete mostri buoni risultati per la maggior parte dei casi, con una mAP media di 0.72 su tutte le classi e un mAR di 0.77, esistono classi che impattano significativamente le prestazioni del modello, come la classe *road narrows on right*, *no overtaking*, e, com'era prevedibile, le classi dei limiti di velocità rispettivamente di 30 e 20 km/h. Questo è dovuto sicuramente dalla forte somiglianza dei due, ma è anche interessante notare come il limite dei 50 abbia un'accuratezza ben più ampia, con solo 0.09% identificati erroneamente per limite dei 20 km/h. In questo caso quindi sarebbe necessario potenziare il dataset su questi casi, e indagare attentamente sugli esempi di test e training durante la fase di fine tuning. Inoltre, è possibile che il modello dopo 15 epoche, nonostante una convergenza dopo solo 5 epoche, potrebbe presentare *overfitting*, poiché le perdite nel training risultano migliori del modello Yolo studiato all'interno di questo documento, ma con risultati in fase di validazione nettamente migliori. In generale, anche per i casi di *straight ahead only* e *go straight or turn left*, le forti similitudini nei colori, le forme e le componenti dei cartelli gioca un ruolo fondamentale.

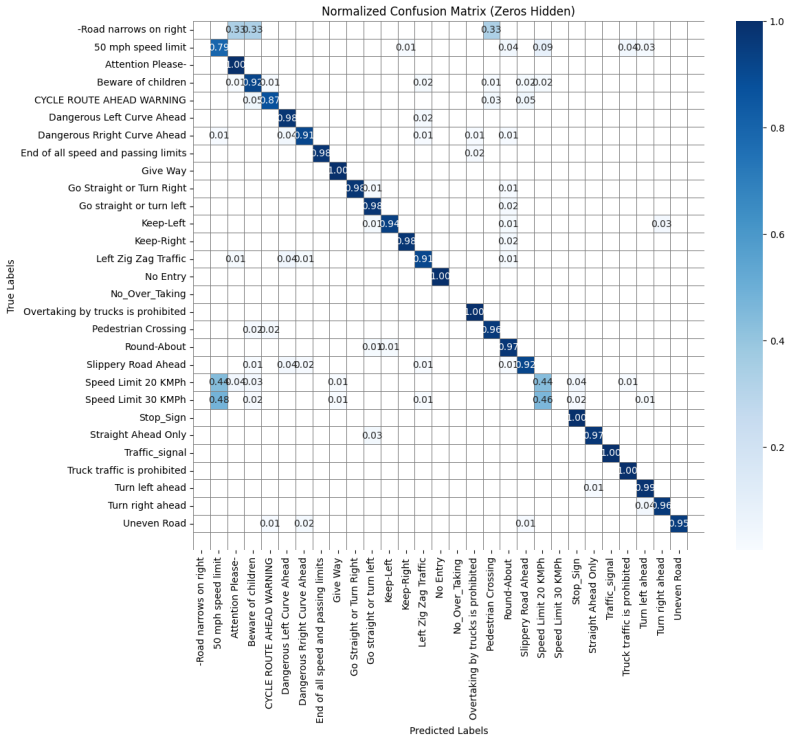

Figura 15: Matrice di confusione sul validation set per *Faster R-CNN* con MobileNetV3-Large come backbone.

Label	mAP	mAR @ 100
Round-About	0.8636	0.8891
Turn right ahead	0.8585	0.8857
Overtaking by trucks is prohibited	0.8571	0.8745
Speed Limit 30 KMPh	0.8515	0.8656
Beware of children	0.8470	0.8679
Pedestrian Crossing	0.8466	0.8806
Stop_Sign	0.8379	0.8714
Left Zig Zag Traffic	0.8352	0.8641
Turn left ahead	0.8270	0.8587
Attention Please-	0.8225	0.8590
Truck traffic is prohibited	0.8150	0.8488
Go straight or turn left	0.8136	0.8595
Dangerous Left Curve Ahead	0.8114	0.8476
CYCLE ROUTE AHEAD WARNING	0.8083	0.8426
End of all speed and passing limits	0.8072	0.8354
Keep-Left	0.8072	0.8531
Go Straight or Turn Right	0.8069	0.8506
Traffic_signal	0.8068	0.8506
Dangerous Rright Curve Ahead	0.7983	0.8347
Give Way	0.7905	0.8364
No_Over_Taking	0.7886	0.8255
Keep-Right	0.7854	0.8305
No Entry	0.7672	0.8108
50 mph speed limit	0.7380	0.8274
Slippery Road Ahead	0.5909	0.8000
Straight Ahead Only	0.1846	0.4333
Road narrows on right	0.0000	0.0000
Speed Limit 20 KMPh	0.0000	0.0000
Mean	0.7274	0.7751

Tabella 5: Mean Average Precision e Mean Average Recall per ogni classe del validation set per il modello Faster R-CNN con MobileNetV3-FPN come backbone.

5 Risultati RetinaNet - Vincenzi Fabio

5.1 Dataset e Configurazione

Per addestrare il modello **RetinaNet** con backbone **ResNet-101-FPN**, è stato utilizzato il dataset **"Self-Driving Cars"**, un insieme di immagini di ambienti stradali annotate secondo il formato COCO. Questo dataset include 16 classi diverse, rappresentative di oggetti comunemente presenti in scenari di guida autonoma.

Set	Numero immagini
Training set	3530
Validation set	801
Test set	638

5.2 Training e Convergenza

Durante l'addestramento, il modello è stato ottimizzato utilizzando l'algoritmo SGD con un valore di momentum pari a 0.9. Il learning rate iniziale era impostato a 0.005 e veniva ridotto del 90

Il batch size è stato fissato a 4, a causa di limitazioni hardware, mentre il numero totale di epoche è stato pari a 5.

5.3 Valutazione delle Prestazioni

I risultati delle valutazioni sul test set mostrano prestazioni complessive soddisfacenti con alcune differenze significative tra categorie.

Tabella 6: Metriche di valutazione COCO

Metrica	**Valore**	**Note**
mAP @[.50:.95]	0.751	Metrica principale
AP @0.50	0.896	
AP @0.75	0.831	
AP piccoli oggetti	0.418	Area $< 32^2$ px
AP medi oggetti	0.808	Area $32^2 - 96^2$ px
AP grandi oggetti	0.906	Area $> 96^2$ px

Tabella 7: Recall a diverse soglie

Metrica	**Valore**	**Dettaglio**
AR @1	0.780	1 detection per immagine
AR @10	0.833	10 detection per immagine
AR @100	0.834	100 detection per immagine

5.3.1 Analisi per Categoria

Tabella 8: Performance per categoria (AP)

Categoria	**AP**	**Categoria**	**AP**	**Categoria**	**AP**
Green Light	0.506	Speed Limit 10	0.336	Speed Limit 80	0.856
Red Light	0.431	Speed Limit 20	0.865	Speed Limit 90	0.701
Speed Limit 30	0.834	Speed Limit 40	0.855	Speed Limit 100	0.819
Speed Limit 50	0.802	Speed Limit 60	0.848	Speed Limit 110	0.793
Speed Limit 70	0.826	Speed Limit 120	0.862	Stop	0.924

5.3.2 Osservazioni Principali

L'analisi dei risultati ottenuti con RetinaNet ha permesso di individuare sia punti di forza rilevanti che diverse criticità su cui intervenire. Il modello ha dimostrato buone capacità di rilevamento, in particolare su oggetti di dimensioni medie e grandi. Tra i risultati più significativi, spiccano le eccellenti performance nella classificazione dei segnali di Stop, per cui è stata raggiunta una precisione media (AP) pari a 0.924. Anche la rilevazione dei cartelli relativi ai limiti di velocità si è dimostrata solida, con valori di AP generalmente superiori a 0.800. A livello globale, il modello ha ottenuto un Average Recall pari a 0.834 (AR@100), evidenziando una buona capacità di copertura delle istanze rilevanti nelle immagini.

Tuttavia, sono emerse alcune limitazioni da tenere in considerazione. Il modello ha mostrato difficoltà nel distinguere classi visivamente simili, come i semafori verdi e rossi, con valori di AP piuttosto bassi (0.506 e 0.431 rispettivamente). Questo tipo di errore può essere attribuito sia alla

ridotta dimensione dell'oggetto nelle immagini, sia alla sensibilità del modello alle condizioni di illuminazione, fattori che rendono più complessa la discriminazione tra colori.

Anche la rilevazione di oggetti molto piccoli (con dimensioni inferiori a 32 pixel) ha rappresentato una sfida: la precisione media in questo caso è scesa a 0.418, confermando la difficoltà dei modelli di object detection nel trattare oggetti poco visibili o scarsamente definiti. Inoltre, si è osservata una certa instabilità nelle prestazioni tra classi simili, come i segnali con limiti di velocità differenti, dove dettagli minimi possono generare confusione.

Per affrontare queste problematiche, si propongono diverse strategie di miglioramento. Un primo passo riguarda l'ampliamento del dataset, includendo immagini acquisite in condizioni ambientali meno frequenti o più difficili, come la guida notturna o situazioni di pioggia e nebbia. L'utilizzo di tecniche avanzate di data augmentation, come la simulazione di condizioni atmosferiche avverse o effetti di motion blur, può aumentare la capacità del modello di generalizzare a scenari reali più complessi.

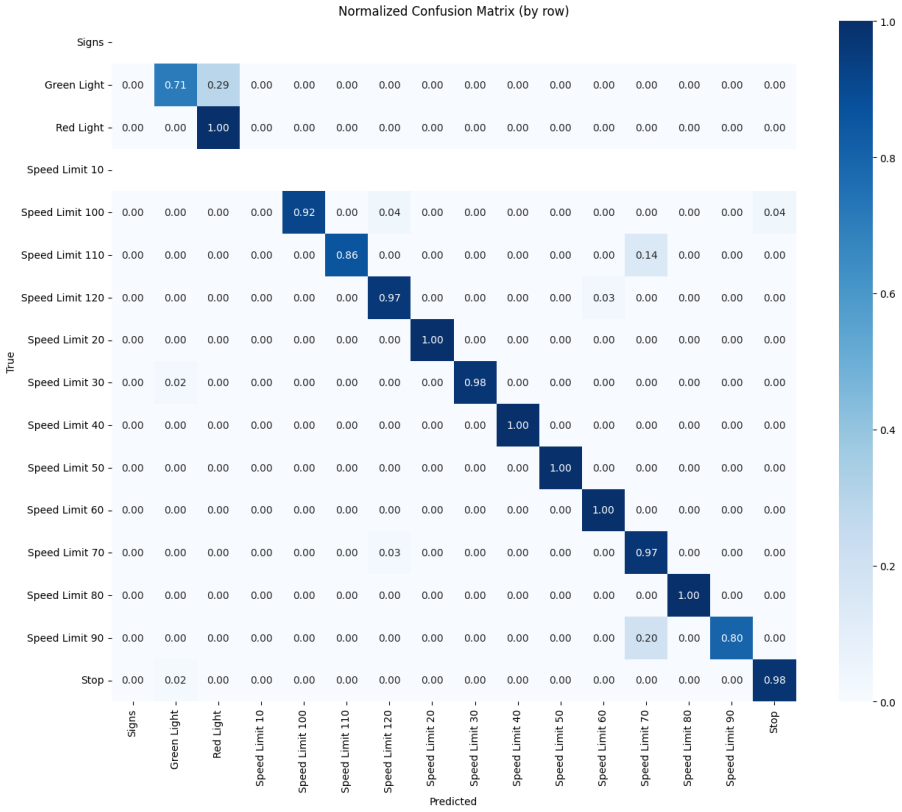

Figura 16: Matrice di confusione normalizzata

Tabella 9: Metriche di valutazione COCO

Metrica	**Valore**	**Soglia IoU**
mAP	0.727	0.50:0.95
mAP	0.897	0.50
mAP	0.879	0.75
AR	0.769	-

5.4 Object Tracking

Per il tracciamento degli oggetti tra fotogrammi consecutivi è stata implementata una strategia basata sull'IoU (Intersection over Union). Gli oggetti rilevati vengono associati mantenendo lo stesso ID se la sovrapposizione tra bounding box supera la soglia di 0.5. Inoltre, per garantire maggiore robustezza, sono stati considerati solo oggetti con un livello di confidenza superiore a 0.7. Questo approccio si è rivelato semplice ma efficace per mantenere la coerenza nell'identificazione degli oggetti nel tempo.

6 Implementazione con Simulatore Carla

Per testare il nostro modello in un contesto controllato e realistico, abbiamo utilizzato il simulatore CARLA, una piattaforma open-source per la simulazione di ambienti urbani destinata alla ricerca su veicoli autonomi. L'obiettivo era generare video realistici di guida autonoma contenenti segnali stradali e semafori, da utilizzare come input per il nostro modello.

Abbiamo avviato l'esperimento creando uno scenario di guida autonoma: un veicolo viene generato in un punto casuale della mappa e dotato di una fotocamera RGB collegata alla parte anteriore. Il veicolo è poi impostato in modalità autopilota, così da muoversi in autonomia nell'ambiente simulato, mentre la fotocamera registra continuamente ciò che il veicolo "vede".

La creazione del veicolo e della fotocamera avviene tramite due funzioni distinte: `spawn_vehicle` e `spawn_camera`.

```python
def spawn_vehicle(vehicle_index=0, spawn_index=0, pattern='vehicle.*'):
    blueprint_library = world.get_blueprint_library()
    vehicle_bp = blueprint_library.filter(pattern)[vehicle_index]
    spawn_point = world.get_map().get_spawn_points()[spawn_index]
    vehicle = world.spawn_actor(vehicle_bp, spawn_point)
    return vehicle

def spawn_camera(attach_to=None, transform=carla.Transform(carla.Location(x=1.2,
↪ z=1.2), carla.Rotation(pitch=-10)), width=800, height=600):
    camera_bp = world.get_blueprint_library().find('sensor.camera.rgb')
    camera_bp.set_attribute('image_size_x', str(width))
    camera_bp.set_attribute('image_size_y', str(height))
    camera = world.spawn_actor(camera_bp, transform, attach_to=attach_to)
    return camera
```

La funzione `spawn_vehicle` genera un veicolo selezionato dalla blueprint library e lo posiziona in un punto specifico della mappa. La funzione `spawn_camera` invece crea una fotocamera RGB con risoluzione configurabile e la collega al veicolo generato. Il posizionamento e l'orientamento della fotocamera sono definiti tramite una trasformazione che simula la visuale frontale del guidatore.

Elaborazione delle Immagini in Tempo Reale Una volta avviata, la fotocamera invia in tempo reale i frame acquisiti a una funzione di callback. Ogni immagine viene convertita in un array NumPy e processata dal modello YOLO precedentemente addestrato, il quale rileva e classifica eventuali oggetti di interesse presenti nel frame.

```python
def camera_callback(image):
    global last_message

    img_array = np.frombuffer(image.raw_data, dtype=np.uint8)
    img_array = img_array.reshape((image.height, image.width, 4))
    img_array = img_array[:, :, :3]
    results = model.predict(img_array, conf=0.6)

    output_image = results[0].plot()
    video_output[:] = output_image

    detections = []
    for box in results[0].boxes:
        label = model.names.get(box.cls.item())
        detection = {"label": label}
        detections.append(detection)

    if not detections:
```

```
        detections.append({"label": "No Detection"})
```

L'immagine elaborata viene anche visualizzata a schermo tramite OpenCV, permettendo così di osservare in tempo reale i risultati della predizione del modello durante la simulazione. Il ciclo principale continua finché l'utente non interrompe l'esecuzione manualmente (ad esempio premendo il tasto 'q'). Al termine, tutti gli attori del simulatore vengono rimossi per liberare le risorse.

```
vehicle = spawn_vehicle()
world.tick()
camera = spawn_camera(attach_to=vehicle)
video_output = np.zeros((600, 800, 3), dtype=np.uint8)
vehicle.set_autopilot(True)
cv2.namedWindow('RGB Camera', cv2.WINDOW_AUTOSIZE)

try:
    while True:
        if cv2.waitKey(1) == ord('q'):
            break
        cv2.imshow('RGB Camera', video_output)
        world.tick()
finally:
    cv2.destroyAllWindows()
    camera.destroy()
    vehicle.destroy()
```

7 Conclusioni e Possibili Sviluppi Futuri

Le reti esplorate dimostrano vari approcci su come affrontare il ben noto problema del riconoscimento di ambientazioni stradali, in particlare dei cartelli e dei semafori. Ad oggi questi sistemi sono sempre più comuni all'interno delle autovetture (*Tesla Traffic Light and Stop Sign Control*), ma è interessante indagare le prestazioni di questi sistemi con strumenti stato dell'arte, sia per architetture capaci di carichi di lavoro sostenuti, sia per dispositivi embedded o mobile, o in generale con risorse limitate.

Prestazioni Ottenute

7.1 Sviluppi Futuri

- Studio e applicazione di Vision Language Models (VLM) per una possibile interazione con un language agent per scenari di guida autonoma assistita da LLM. L'architettura quindi comprenderebbe un sistema di object detection/object tracking, integrata con un *language agent* in grado di percepire lo stato del veicolo (e.g. attraverso il simulatore CARLA), elaborare informazioni multimodali (vantaggio di utilizzare VLM) e definire un ciclo di ragionamento in tempi brevi per prendere decisioni che agiscano sull'ambiente.
- Creazione di un *dataset* migliorato andando ad utilizzare il simulatore di *CARLA* per raccogliere immagini stradali, etichettare ogni singolo cartello/semaforo e utilizzare successivamente tale *dataset* per addestrare una rete profonda, cosi facendo il modello sarà in grado di andare ad individuare gli oggetti in immagini molto più complesse;
- Utilizzo di reti neurali più profonde con un epoche di addestramento maggiorate, cosi facendo si potrà andare ad utilizzare tutta la capacità computazionale del modello utilizzato, facendo in modo che sia estremamente preciso grazie anche all'utilizzo di *dataset* molto più corposi (versioni di YOLOv11 non nano);

Riferimenti bibliografici

[Gir+14] Ross Girshick et al. *Rich feature hierarchies for accurate object detection and semantic segmentation*. 2014. arXiv: 1311.2524 [cs.CV]. URL: https://arxiv.org/abs/1311.2524.

[Gir15] Ross Girshick. *Fast R-CNN*. 2015. arXiv: 1504.08083 [cs.CV]. URL: https://arxiv.org/abs/1504.08083.

[How+17] Andrew G. Howard et al. «MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications». In: *CoRR* abs/1704.04861 (2017). arXiv: 1704.04861. URL: http://arxiv.org/abs/1704.04861.

[Jen+16] Morten Bornø Jensen et al. «Vision for looking at traffic lights: Issues, survey, and perspectives». In: *IEEE Transactions on Intelligent Transportation Systems* 17.7 (2016), pp. 1800–1815. DOI: 10.1109/TITS.2015.2509509.

[JQ24] Glenn Jocher e Jing Qiu. *Ultralytics YOLO11*. Ver. 11.0.0. 2024. URL: https://github.com/ultralytics/ultralytics.

[KSH17] Alex Krizhevsky, Ilya Sutskever e Geoffrey E. Hinton. «ImageNet classification with deep convolutional neural networks». In: *Commun. ACM* 60.6 (mag. 2017), 84–90. ISSN: 0001-0782. DOI: 10.1145/3065386. URL: https://doi.org/10.1145/3065386.

[Qin+24] Danfeng Qin et al. *MobileNetV4 – Universal Models for the Mobile Ecosystem*. 2024. arXiv: 2404.10518 [cs.CV]. URL: https://arxiv.org/abs/2404.10518.

[Ren+16] Shaoqing Ren et al. *Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks*. 2016. arXiv: 1506.01497 [cs.CV]. URL: https://arxiv.org/abs/1506.01497.

[TL19] Mingxing Tan e Quoc V. Le. «EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks». In: *CoRR* abs/1905.11946 (2019). arXiv: 1905.11946. URL: http://arxiv.org/abs/1905.11946.

[TPL19] Mingxing Tan, Ruoming Pang e Quoc V. Le. «EfficientDet: Scalable and Efficient Object Detection». In: *CoRR* abs/1911.09070 (2019). arXiv: 1911.09070. URL: http://arxiv.org/abs/1911.09070.

[usm24] usmanchaudhry622@gmail.com. *Traffic and Road Signs Dataset*. https://universe.roboflow.com/usmanchaudhry622-gmail-com/traffic-and-road-signs. Open Source Dataset. visited on 2025-03-26. 2024. URL: https://universe.roboflow.com/usmanchaudhry622-gmail-com/traffic-and-road-signs.