



Università degli Studi di Catania  
Dipartimento di Matematica e Informatica

## Traffic Sign Detection Yolo10

Relazione Finale

Docenti: Giovanni Maria Farinella, Rosario Leonardi

Studenti: Matteo Vullo, Samuele Vullo

Matricola: 1000034661, 1000001919

Settembre 2025

# Indice

<b>1</b>	<b>Introduzione</b>	<b>3</b>
<b>2</b>	<b>Task</b>	<b>5</b>
2.1	Applicazione Pratiche . . . . .	5
2.2	Traffic Sign Detection . . . . .	7
2.3	You Only Look Once (YOLO) . . . . .	8
2.4	Modello YOLOv10 . . . . .	9
2.4.1	Innovazioni introdotte . . . . .	9
2.4.2	Architettura . . . . .	10
<b>3</b>	<b>Dataset</b>	<b>12</b>
3.1	Dataset GTSDB . . . . .	12
3.1.1	Heatmap delle annotazioni del training set . . . . .	16
3.2	European Traffic Sign Dataset . . . . .	16
3.2.1	Heatmap delle annotazioni del training set . . . . .	20
3.2.2	GTSDB vs ETDS . . . . .	22
3.3	Dataset esterni per classi sottorapresentate . . . . .	22
3.4	Dataset Acquisito per il Test . . . . .	23
3.4.1	Statistiche del test set . . . . .	24
3.4.2	Heatmap delle annotazioni del test set . . . . .	25
3.4.3	Distribuzione delle classi nel test set . . . . .	27
3.4.4	Esempi del test set . . . . .	28
3.5	Pre-processing dei Dataset . . . . .	29
3.5.1	Pre-processing del dataset GTSDB . . . . .	30
3.6	Dataset Finale . . . . .	33
3.6.1	Processo di Integrazione e Standardizzazione . . . . .	34
3.6.2	Classi Finali . . . . .	36
3.6.3	Statistiche finali . . . . .	36
3.7	Organizzazione del Dataset per il Training . . . . .	36
<b>4</b>	<b>Validazione Modello</b>	<b>39</b>
4.1	Metriche di Valutazione . . . . .	39
4.2	Grafici . . . . .	41
4.3	Funzioni di Loss . . . . .	42
<b>5</b>	<b>Implementazione Progetto</b>	<b>44</b>

<b>6 Training, Validazione e Testing dei Modelli</b>	<b>46</b>
6.1 Parametri di Training . . . . .	46
6.1.1 Parametri di Data Augmentation . . . . .	47
6.2 Organizzazione della Directory per il Training . . . . .	48
6.3 Fase di Training . . . . .	49
6.3.1 Installazione delle Dipendenze . . . . .	49
6.3.2 Esecuzione del Training . . . . .	49
6.4 Fase di Validazione . . . . .	52
6.4.1 Validazione dei Modelli . . . . .	53
6.4.2 Confronto Grafico tra Modelli . . . . .	55
6.5 Fase di Testing . . . . .	57
6.5.1 Testing con Codice . . . . .	57
6.5.2 Testing manuale su singole immagini . . . . .	57
6.5.3 Applicazione Demo . . . . .	58
6.5.4 Preparazione delle Immagini per l'App . . . . .	59
6.5.5 Avvio dell'Applicazione . . . . .	60
<b>7 Risultati Sperimentali</b>	<b>61</b>
7.1 Tabelle Riassuntive . . . . .	61
7.1.1 Architettura e Tempi di Training . . . . .	61
7.1.2 Metriche Medie sul Validation Set . . . . .	62
7.1.3 Metriche Medie sul Test Set . . . . .	63
7.1.4 Conclusioni per classe (test set) . . . . .	64
7.2 Analisi di Loss, Matrici di Confusione e Curve di Valutazione . . . . .	67
7.2.1 Matrici di Confusione . . . . .	67
7.2.2 Curve di Valutazione al Variare della Soglia di Confidenza . . . . .	68
7.2.3 YOLOv10n . . . . .	72
7.2.4 YOLOv10s . . . . .	76
7.2.5 YOLOv10m . . . . .	80
<b>8 Conclusioni</b>	<b>84</b>
8.1 Obiettivi e Risultati Ottenuti . . . . .	84
8.2 Lezioni Apprese . . . . .	85
8.3 Sviluppi Futuri . . . . .	85

# Capitolo 1

## Introduzione

L’*object detection* rappresenta uno dei compiti fondamentali del *machine learning* nell’ambito della *Computer Vision*, con l’obiettivo di identificare e localizzare simultaneamente oggetti di interesse all’interno di immagini o sequenze video. Tra i numerosi approcci sviluppati, la famiglia di modelli YOLO (*You Only Look Once*) si è affermata come una delle più efficienti e diffuse, grazie alla sua capacità di operare in tempo reale senza compromettere eccessivamente l’accuratezza. L’ultima evoluzione di questa serie, **YOLOv10**, introduce ulteriori miglioramenti architettonici e strategie di addestramento avanzate per ottimizzare ulteriormente prestazioni e velocità.

YOLOv10 è disponibile in sei varianti, ciascuna progettata per esigenze specifiche in termini di risorse e prestazioni:

- **YOLOv10n** (“Nano”): ideale per dispositivi con risorse hardware molto limitate.
- **YOLOv10s** (“Small”): offre un buon compromesso tra velocità e precisione.
- **YOLOv10m** (“Medium”): adatta a utilizzi generali, con un equilibrio solido tra efficienza e accuratezza.
- **YOLOv10b** (“Balanced”): enfatizza l’accuratezza mantenendo un buon livello di efficienza.
- **YOLOv10l** (“Large”): massimizza l’accuratezza a fronte di un maggiore costo computazionale.
- **YOLOv10x** (“Extra-large”): la versione più performante, pensata per applicazioni che richiedono il massimo livello di precisione.

Il presente progetto si propone di applicare tre di queste varianti: **YOLOv10n**, **YOLOv10s** e **YOLOv10m** al riconoscimento e alla localizzazione di segnali stradali. La valutazione delle prestazioni avverrà attraverso metriche standard del settore, quali *precision*, *recall* e *mean Average Precision (mAP)*. Per garantire una valutazione completa e realistica, il *training* dei modelli è stato condotto unendo due dataset pubblici **German Traffic Sign Detection Benchmark (GTSDB)** e **European Traffic Sign Detection (ETSD)** compensati opportunamente da alcune porzioni di altri dataset, mentre il *testing* sarà effettuato su un dataset personalizzato di segnali stradali, acquisito autonomamente e descritto nel dettaglio nelle sezioni successive del lavoro.

<b>Model</b>	<b>Test Size</b>	<b>#Params</b>	<b>FLOPs</b>	<b>AP<sup>val</sup></b>	<b>Latency</b>
<a href="#"><u>YOLOv10-N</u></a>	640	2.3M	6.7G	38.5%	1.84ms
<a href="#"><u>YOLOv10-S</u></a>	640	7.2M	21.6G	46.3%	2.49ms
<a href="#"><u>YOLOv10-M</u></a>	640	15.4M	59.1G	51.1%	4.74ms
<a href="#"><u>YOLOv10-B</u></a>	640	19.1M	92.0G	52.5%	5.74ms
<a href="#"><u>YOLOv10-L</u></a>	640	24.4M	120.3G	53.2%	7.28ms
<a href="#"><u>YOLOv10-X</u></a>	640	29.5M	160.4G	54.4%	10.70ms

Figura 1.1: Tabella dimostrativa delle differenze tra i diversi modelli YOLO

# Capitolo 2

## Task

L'**object detection** è un compito centrale nell'ambito della *computer vision*, il cui obiettivo è duplice: da un lato, **identificare la classe semantica** di appartenenza di ciascun oggetto presente in un'immagine o in un flusso video; dall'altro, **determinarne con precisione la posizione spaziale**, delimitandolo tramite un rettangolo di selezione noto come *bounding box*.

In termini computazionali, un algoritmo di object detection può essere concettualmente scomposto in due componenti fondamentali:

- Un **modulo di classificazione**, responsabile dell'assegnazione dell'etichetta corretta a ciascun oggetto rilevato;
- Un **modulo di regressione**, incaricato di stimare le coordinate del bounding box (ad esempio, coordinate del centro, larghezza e altezza, oppure degli angoli) in modo che esso racchiuda al meglio l'oggetto di interesse.

Questa combinazione rende l'object detection una sfida più complessa rispetto alla semplice classificazione d'immagine, richiedendo modelli in grado di gestire simultaneamente compiti semantici e geometrici.

### 2.1 Applicazione Pratiche

L'*Object Detection* non si limita a pochi ambiti, ma si estende a numerosi settori grazie alla sua capacità di interpretare visivamente scene complesse e localizzare oggetti con precisione. Oltre alle applicazioni più note, emergono contesti innovativi e trasversali in cui questa tecnologia gioca un ruolo chiave:

- **Agricoltura di precisione:** i droni e i robot agricoli utilizzano l'object detection per identificare piante infestanti, monitorare lo stato di salute delle colture o localizzare frutti maturi da raccogliere, ottimizzando l'uso di risorse e aumentando la resa.
- **Retail intelligente:** nei negozi senza cassa (es. Amazon Go), i sistemi di visione rilevano automaticamente quali prodotti vengono prelevati o riposti dai clienti, abilitando il pagamento automatico e migliorando l'esperienza d'acquisto.

- **Assistenza sanitaria e medicina:** in ambito radiologico o istologico, l'object detection aiuta a localizzare lesioni, cellule anomale o organi specifici all'interno di immagini mediche, supportando i medici nella diagnosi precoce.
- **Logistica e magazzini automatizzati:** i sistemi di visione identificano e tracciano pacchi, pallet o articoli sugli scaffali, permettendo la gestione autonoma degli inventari e il recupero efficiente dei prodotti da parte dei robot mobili.
- **Sport analytics:** durante le partite, i modelli rilevano giocatori, palloni e arbitri per generare statistiche avanzate, tracciare movimenti tattici o ricostruire azioni in tempo reale a fini di allenamento o broadcasting.
- **Interazione uomo-macchina:** in contesti di realtà aumentata o assistenti robotici, l'object detection consente ai dispositivi di “vedere” e comprendere l'ambiente, riconoscendo oggetti con cui l'utente può interagire (es. “afferra la bottiglia rossa sul tavolo”).

L'evoluzione continua degli algoritmi; in termini di velocità, accuratezza e adattabilità; rende l'object detection uno strumento sempre più accessibile e integrabile, capace di trasformare dati visivi in informazioni strutturate e azionabili in tempo reale.

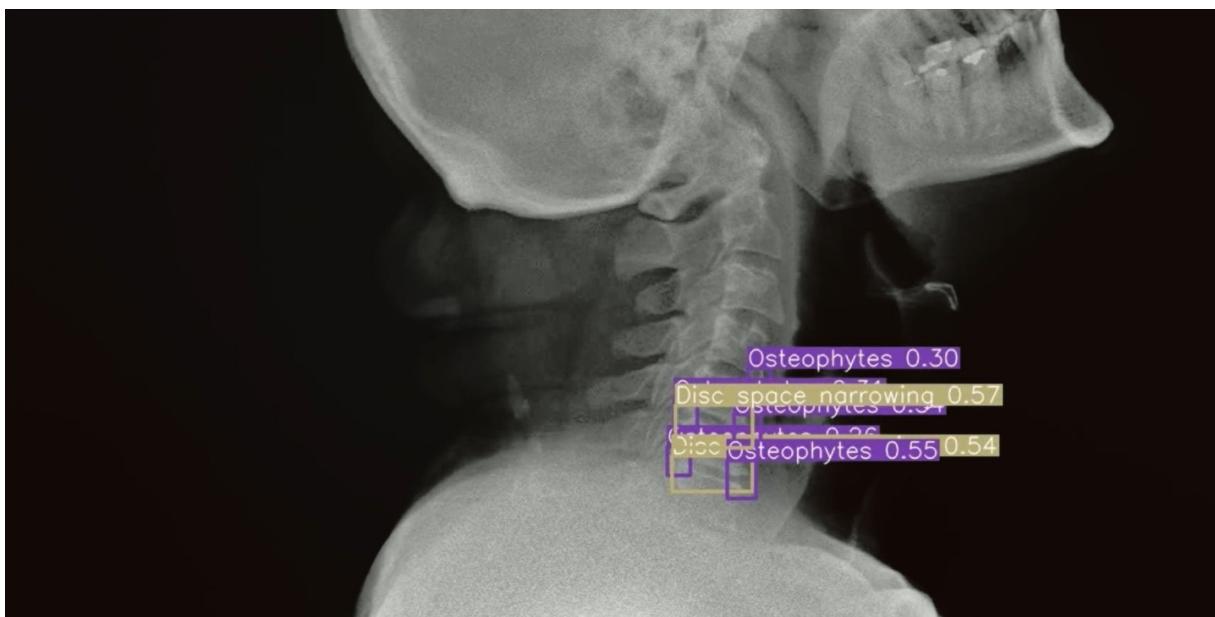


Figura 2.1: Esempio di object detection applicata all'assistenza sanitaria e medica

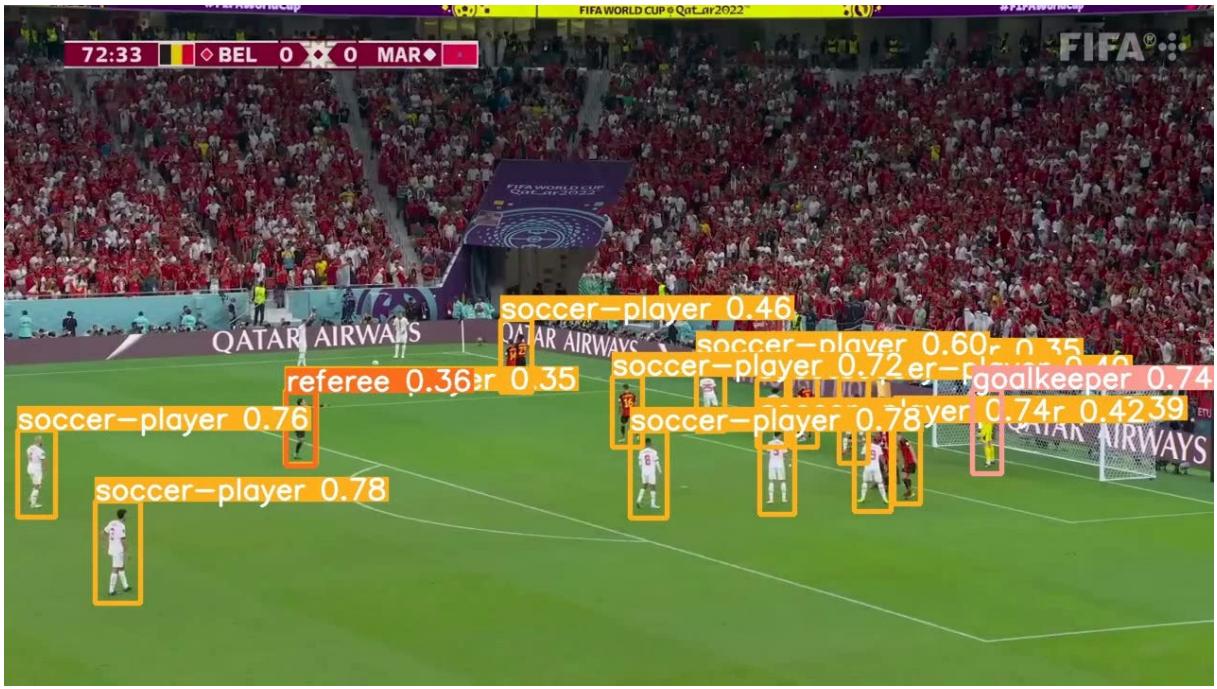


Figura 2.2: Esempio di object detection applicata allo sport analytics

## 2.2 Traffic Sign Detection

Il compito centrale affrontato in questo progetto è la **rilevazione automatica di segnali stradali** (*Traffic Signs Detection*), un'applicazione specifica dell'*object detection* nell'ambito automotive. L'obiettivo è identificare e localizzare, all'interno di immagini o sequenze video, tipicamente acquisite tramite telecamere montate su veicoli o infrastrutture stradali, i segnali di traffico presenti nell'ambiente circostante.

Questa tecnologia riveste un ruolo fondamentale nello sviluppo di sistemi di guida autonoma e assistita (ADAS — *Advanced Driver Assistance Systems*), contribuendo a migliorare la sicurezza stradale attraverso l'interpretazione automatica delle norme di circolazione.

Tuttavia, la realizzazione di modelli robusti e affidabili si scontra con numerose sfide pratiche, tra cui:

- **L'eterogeneità dei segnali:** forma, colore, dimensione e simbologia variano notevolmente in base alle normative nazionali e regionali;
- **Condizioni ambientali avverse:** illuminazione non uniforme, riflessi, pioggia, nebbia o ombre possono compromettere la visibilità e il contrasto;
- **Ostruzioni parziali:** segnali parzialmente coperti da vegetazione, cartelli pubblicitari, veicoli o sporco riducono la riconoscibilità.

Per affrontare efficacemente queste criticità, è essenziale addestrare i modelli su **dataset ampi e diversificati**, in grado di rappresentare la variabilità reale degli scenari operativi, garantendo così generalizzazione e robustezza.

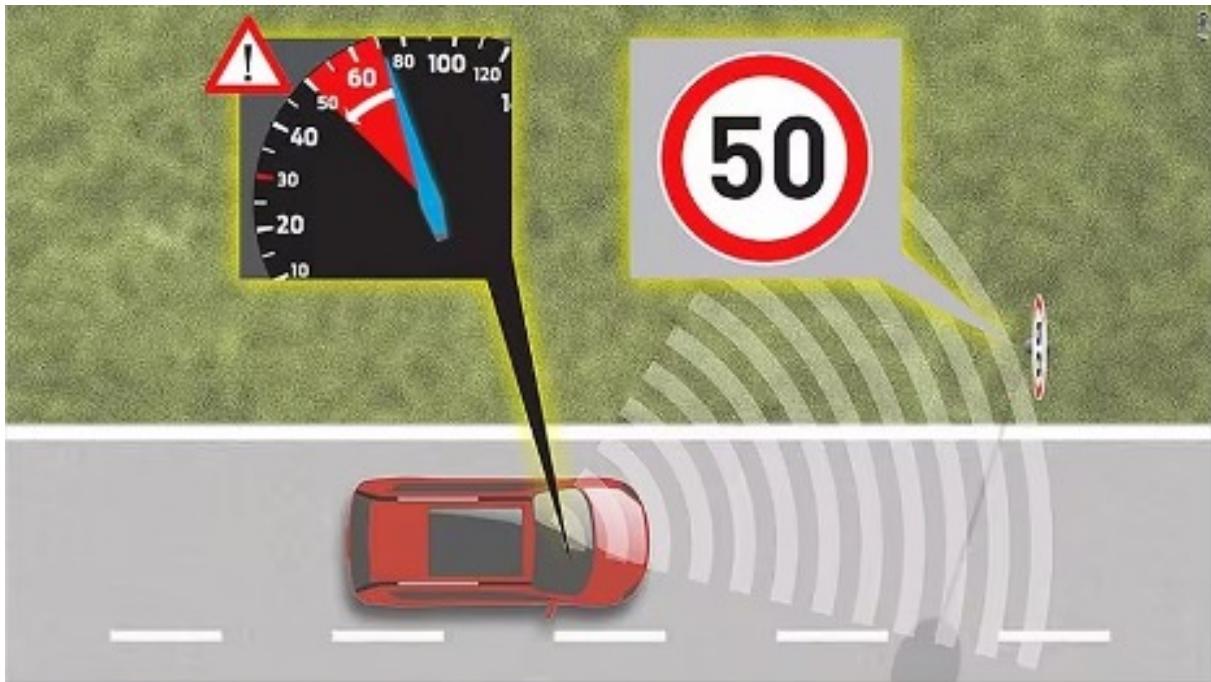


Figura 2.3: Il sistema implementa un algoritmo di computer vision che, mediante una telecamera montata a bordo veicolo, riconosce la segnaletica stradale e attua un controllo adattivo della velocità in base alle limitazioni individuate.

## 2.3 You Only Look Once (YOLO)

**You Only Look Once (YOLO)** è un modello di *object detection* appartenente alla categoria dei metodi “**single-stage**”, in quanto esegue simultaneamente la predizione dei *bounding box* e delle probabilità di appartenere a una classe, in una singola passata attraverso la rete neurale. A differenza degli approcci “**two-stage**” (come R-CNN o suoi derivati), che separano la proposta delle regioni d’interesse dalla loro classificazione, YOLO tratta il rilevamento come un problema di regressione unificato, garantendo maggiore efficienza computazionale.

La sua caratteristica distintiva, e il motivo del suo nome, è la capacità di analizzare l’intera immagine “**con un’unica occhiata**”, permettendo così di effettuare la detection degli oggetti **in tempo reale**, senza compromettere eccessivamente l’accuratezza. Questo approccio ha rivoluzionato il campo dell’object detection, superando in velocità molti dei modelli allora considerati *state-of-the-art*, e aprendo la strada a applicazioni real-time in contesti come automotive, sorveglianza e robotica.

Dal suo debutto nel **2015**, YOLO ha visto numerose evoluzioni, con il rilascio di versioni sempre più performanti da YOLOv1 a YOLOv10 che hanno progressivamente migliorato precisione, velocità e robustezza grazie a innovazioni architetturali e tecniche di addestramento avanzate.

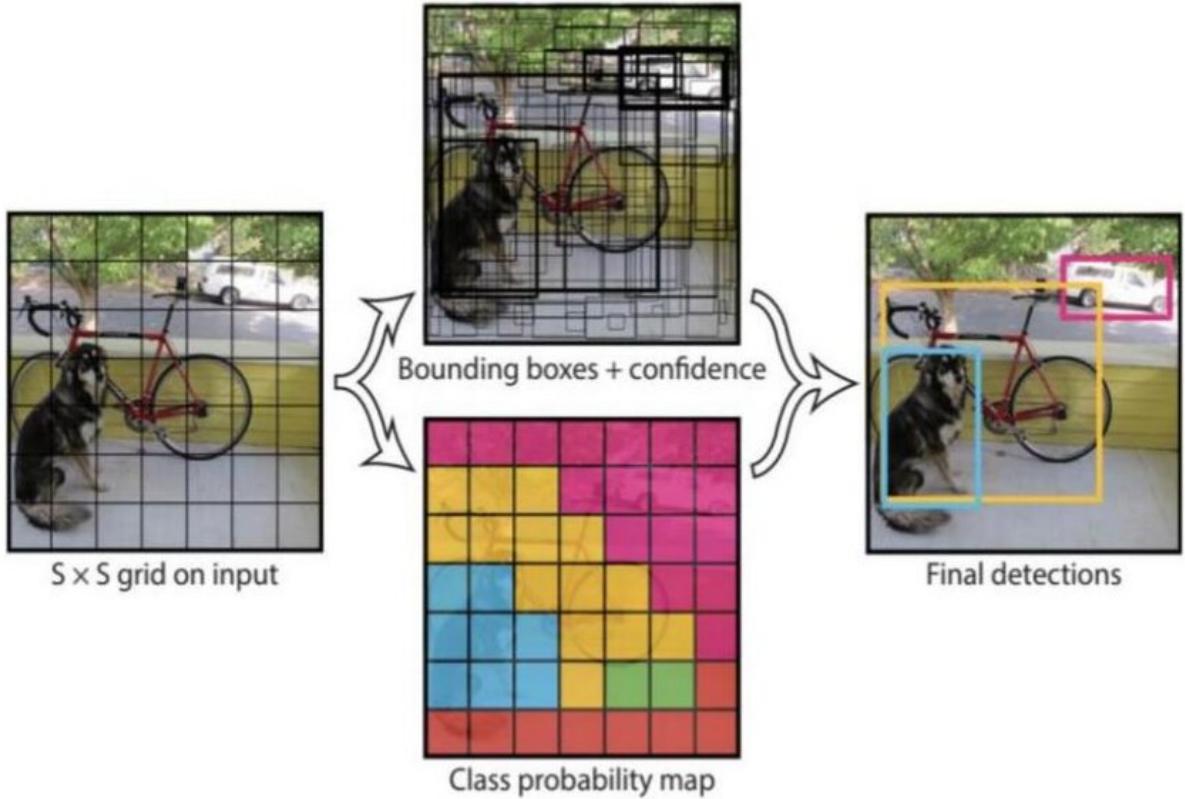


Figura 2.4: Esempio di Single-step YOLO.

## 2.4 Modello YOLOv10

**YOLOv10**, sviluppato dai ricercatori della **Tsinghua University**, rappresenta un’evoluzione significativa nell’ambito della *object detection* in tempo reale. Questa versione introduce miglioramenti architetturali e metodologici mirati a superare alcune limitazioni delle precedenti iterazioni della famiglia YOLO, in particolare quelle legate alla fase di post-elaborazione e all’efficienza complessiva del modello.

### 2.4.1 Innovazioni introdotte

Una delle innovazioni architetturali chiave introdotte in **YOLOv10** è la strategia denominata **Dual Label Assignment with NMS-free Training**, progettata per eliminare la dipendenza dal costoso meccanismo di *Non-Maximum Suppression* (NMS) durante l’inferenza, senza sacrificare accuratezza o velocità di convergenza.

Questa strategia si basa su un doppio schema di assegnazione delle etichette durante il training:

- **One-to-One Matching**: ad ogni oggetto presente nell’immagine (*ground truth*) viene associata una e una sola predizione del modello. Questo approccio elimina la necessità del NMS in fase di inferenza, poiché non vengono generate predizioni multiple sovrapposte. Tuttavia, da solo, tende a produrre risultati con accuratezza subottimale e una convergenza più lenta durante l’addestramento.
- **One-to-Many Assignment**: ogni oggetto può essere associato a più predizioni, consentendo al modello di apprendere in modo più robusto e di raggiungere presta-

zioni superiori. Tuttavia, richiede l’uso del NMS in inferenza per filtrare le predizioni ridondanti.

Per sfruttare i vantaggi di entrambi gli approcci, YOLOv10 introduce un’architettura ibrida: mantiene il tradizionale *head* “one-to-many” per garantire un training ricco e stabile, e aggiunge un nuovo *head* “one-to-one” ottimizzato per l’inferenza NMS-free. Entrambi i rami vengono addestrati simultaneamente.

Questi cambiamenti consentono a YOLOv10 di raggiungere prestazioni superiori in termini di velocità e precisione, consolidando il suo ruolo come modello di riferimento per applicazioni real-time di object detection.

## 2.4.2 Architettura

L’architettura di **YOLOv10** è composta da quattro componenti fondamentali, ciascuna con un ruolo specifico nell’ottimizzazione della detection in tempo reale e nell’eliminazione del NMS:

- **Backbone**: si occupa dell’estrazione gerarchica delle feature dall’immagine in input, attraverso una serie di blocchi convoluzionali. È progettato per essere leggero ma potente, garantendo un buon compromesso tra accuratezza e velocità.
- **Neck**: ha il compito di aggregare le feature estratte a diverse scale spaziali dal backbone, per migliorare la capacità del modello di rilevare oggetti di dimensioni variabili. Questa aggregazione avviene tramite un’architettura **PAN (Path Aggregation Network)**, che combina informazioni a bassa risoluzione (ricche semanticamente) con quelle ad alta risoluzione (precise spazialmente).
- **Dual Label Assignment**: durante il training, viene utilizzato un assegnamento “one-to-many” per accelerare la convergenza e migliorare la qualità dell’apprendimento; in fase di inferenza, invece, si utilizza un assegnamento “one-to-one”, che elimina la necessità del NMS e garantisce predizioni pulite e dirette.
- **Consistent Matching Metric**: questa metrica è fondamentale per allineare le predizioni dei due rami di training (“one-to-one” e “one-to-many”). Essa misura la coerenza tra le predizioni e i *ground truth*, ed è definita come:

$$m(\alpha, \beta) = s \cdot p^\alpha \cdot \text{IoU}(\hat{b}, b)^\beta$$

dove:

- $s$  è un fattore di scala,
- $p$  è la probabilità di classe predetta,
- $\hat{b}$  e  $b$  sono rispettivamente il bounding box predetto e quello ground truth,
- $\alpha$  e  $\beta$  sono iperparametri che bilanciano il peso della confidenza di classe e della localizzazione spaziale.

Questa metrica permette di ottimizzare congiuntamente i due *head*, garantendo che il ramo “one-to-one”, usato in inferenza, apprenda a produrre predizioni allineate a quelle più accurate del ramo “one-to-many”.

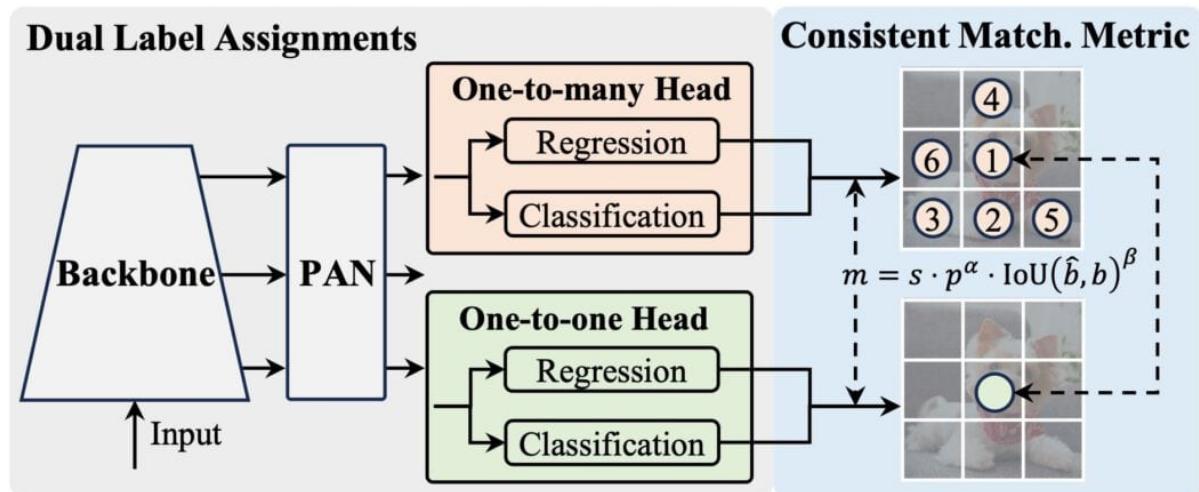


Figura 2.5: Schema architettura YOLO10.

# Capitolo 3

## Dataset

Il progetto si è avvalso di una combinazione di dataset pubblici e risorse personalizzate per le fasi di addestramento e validazione dei modelli:

- **German Traffic Sign Detection Benchmark (GTSDB)**: Dataset di riferimento per lo stato dell'arte nel riconoscimento di segnali stradali, utilizzato come base principale per l'addestramento delle varie versioni del modello YOLOv10.
- **European Traffic Sign Dataset (da Kaggle)**: Un'estensione multiclass che ha integrato il training set con esempi aggiuntivi di segnaletica europea, migliorando la capacità di generalizzazione del modello.
- **Risorse complementari da Roboflow e Kaggle**: Sono stati selezionati e integrati esempi specifici da altri dataset disponibili su queste piattaforme, con particolare focus su classi sottorappresentate o condizioni ambientali critiche (scarsa illuminazione, condizioni meteorologiche avverse).
- **Dataset personalizzato**: Una collezione di immagini acquisite in scenari reali, utilizzata esclusivamente per la fase di test e validazione finale dei modelli, garantendo una valutazione imparziale delle prestazioni.

### 3.1 Dataset GTSDB

Il dataset selezionato per la fase di training è il German Traffic Sign Detection Benchmark (GTSDB) [1]. Esso comprende immagini di segnaletica stradale riprese in ambiente reale, da un veicolo in movimento, sotto diverse condizioni di illuminazione, angolazioni e orientamenti. Il dataset è composto da 600 immagini in formato PPM, suddivise in 480 immagini per il training e 120 per la validazione. Ciascuna immagine può contenere da 0 a 6 segnali, appartenenti a 43 classi diverse, con dimensioni variabili tra 16x16 e 128x128 pixel.

Ad ogni immagine è associato un file di testo con le relative annotazioni *ground truth*, il quale segue il formato standard dei modelli YOLO. Ogni riga in questi file descrive un segnale presente nell'immagine secondo la struttura:

```
<class_id> <x_center> <y_center> <width> <height>
```

dove:

- **<class\_id>**: Identificativo della classe del segnale.
- **<x\_center>, <y\_center>**: Coordinate normalizzate del centro del *bounding box* che delimita il segnale.
- **<width>, <height>**: Larghezza e altezza normalizzate del *bounding box*.

Le classi sono le seguenti:

- 0: speed limit 20
- 1: speed limit 30
- 2: speed limit 50
- 3: speed limit 60
- 4: speed limit 70
- 5: speed limit 80
- 6: restriction ends 80
- 7: speed limit 100
- 8: speed limit 120
- 9: no overtaking
- 10: no overtaking (trucks)
- 11: priority at next intersection
- 12: priority road
- 13: give way
- 14: stop
- 15: no traffic both ways
- 16: no trucks
- 17: no entry
- 18: danger
- 19: bend left
- 20: bend right
- 21: bend
- 22: uneven road
- 23: slippery road
- 24: road narrows
- 25: construction
- 26: traffic signal
- 27: pedestrian crossing
- 28: school crossing
- 29: cycles crossing
- 30: snow
- 31: animals
- 32: restriction ends
- 33: go right
- 34: go left
- 35: go straight
- 36: go right or straight
- 37: go left or straight
- 38: keep right
- 39: keep left
- 40: roundabout
- 41: restriction ends (overtaking)
- 42: restriction ends (overtaking (trucks))



(a) Immagine originale senza etichetta



(b) Immagine processata con etichetta

Figura 3.1: Esempio di immagine estratta dal dataset GTSDB



(a) Immagine originale senza etichetta



(b) Immagine processata con etichetta

Figura 3.2: Esempio di immagine estratta dal dataset GTSDB

La distribuzione delle classi all'interno del training set non è uniforme, come evidenziato nella Tabella 3.1. Alcuni segnali, come **keep right** e **priority road**, sono molto frequenti, mentre altri, come **animals** o **bend left**, compaiono raramente.

<b>Segnale</b>	<b># Immagini</b>	<b># Occorrenze</b>
animals	1	1
bend	5	5
bend left	2	2
bend right	6	9
construction	19	21
cycles crossing	4	4
give way	44	52
go left	9	9
go left or straight	1	1
go right	12	13
go right or straight	8	8
go straight	15	15
keep left	3	4
keep right	56	57
no entry	15	25
no overtaking	28	32
no overtaking (trucks)	37	63
no traffic both ways	8	10
no trucks	5	7
pedestrian crossing	3	3
priority at next intersection	25	26
priority road	53	54
restriction ends	2	3
restriction ends 80	9	17
restriction ends (overtaking)	5	6
restriction ends (overtaking (trucks))	5	7
road narrows	2	2
roundabout	7	7
school crossing	8	9
slippery road	11	13
speed limit 100	23	37
speed limit 120	28	47
speed limit 20	4	4
speed limit 30	46	48
speed limit 50	50	59
speed limit 60	16	21
speed limit 70	23	31
speed limit 80	24	37
stop	16	22
traffic signal	9	11
uneven road	7	9

Tabella 3.1: Distribuzione delle classi nel training set GTSDB

### 3.1.1 Heatmap delle annotazioni del training set

L'analisi della *Spatial Heatmap* delle annotazioni (Figura 3.3) è fondamentale per identificare eventuali bias posizionali nei dati. Se i segnali tendessero a comparire sempre nelle stesse regioni dell'immagine, il modello potrebbe imparare a associare erroneamente quelle posizioni alla presenza di un oggetto, piuttosto che basarsi sulle sue effettive caratteristiche visive. Nel caso del dataset GTSDB, la heatmap risulta ben distribuita, sebbene si osservi una maggiore concentrazione di annotazioni nella parte destra delle immagini. Questo è un riflesso realistico della prospettiva di guida, poiché la maggior parte dei segnali stradali è posizionata sul lato destro della carreggiata.

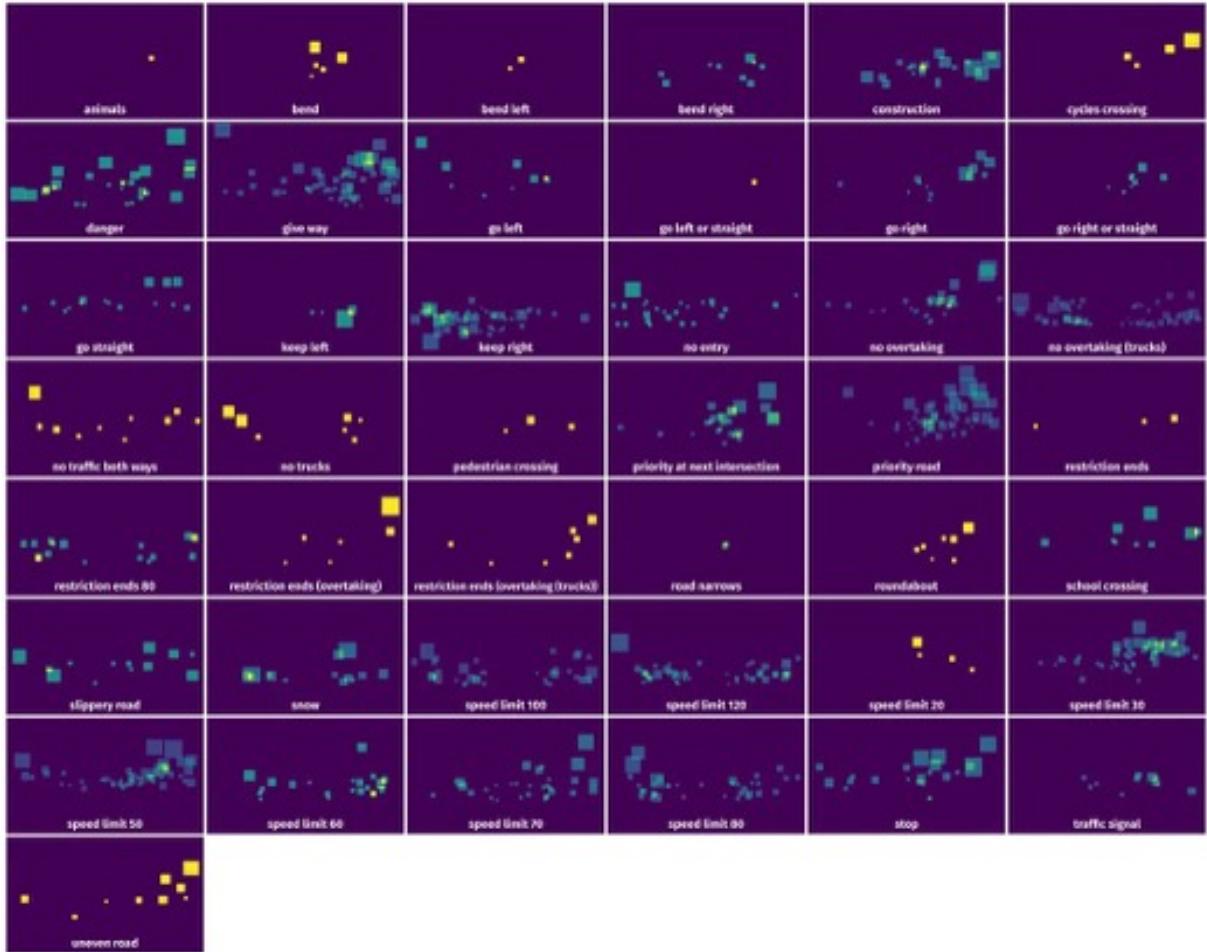


Figura 3.3: Spatial Heatmap delle annotazioni nel training set GTSDB.

## 3.2 European Traffic Sign Dataset

Il dataset *European Traffic Sign Dataset* [2] è stato selezionato per integrare e diversificare il materiale di addestramento del modello. Questa raccolta comprende immagini di segnaletica stradale europea acquisite in diversi contesti geografici e condizioni ambientali, offrendo una varietà complementare al GTSDB tedesco.

Il dataset è composto da 4197 immagini in formato JPG, con risoluzioni media 640x640 pixel. Le annotazioni sono fornite in formato YOLO, con un totale di 8153 segnali annotati

appartenenti a 55 classi diverse. La suddivisione originale prevede 3117 immagini per il training, 1080 per la validazione e 0 per il test.

Le annotazioni seguono il formato standard YOLO:

```
<class_id> <x_center> <y_center> <width> <height>
```

dove:

- **<class\_id>**: Identificativo della classe del segnale, con mappatura parzialmente compatibile con il GTSDB.
- **<x\_center>, <y\_center>**: Coordinate normalizzate del centro del *bounding box*.
- **<width>, <height>**: Dimensioni normalizzate del *bounding box*.

Le classi sono le seguenti:

- 0: forb\_ahead
- 1: forb\_left
- 2: forb\_overtake
- 3: forb\_right
- 4: forb\_speed\_over\_10
- 5: forb\_speed\_over\_100
- 6: forb\_speed\_over\_130
- 7: forb\_speed\_over\_20
- 8: forb\_speed\_over\_30
- 9: forb\_speed\_over\_40
- 10: forb\_speed\_over\_5
- 11: forb\_speed\_over\_50
- 12: forb\_speed\_over\_60
- 13: forb\_speed\_over\_70
- 14: forb\_speed\_over\_80
- 15: forb\_speed\_over\_90
- 16: forb\_stopping
- 17: forb\_trucks
- 18: forb\_u\_turn
- 19: forb\_weight\_over\_3.5t
- 20: forb\_weight\_over\_7.5t
- 21: info\_bus\_station
- 22: info\_crosswalk
- 23: info\_highway
- 24: info\_one\_way\_traffic
- 25: info\_parking
- 26: info\_taxi\_parking
- 27: mand\_bike\_lane
- 28: mand\_left
- 29: mand\_left\_right
- 30: mand\_pass\_left
- 31: mand\_pass\_left\_right
- 32: mand\_pass\_right
- 33: mand\_right
- 34: mand\_roundabout
- 35: mand\_straight\_left
- 36: mand\_straight
- 37: mand\_straight\_right
- 38: prio\_give\_way
- 39: prio\_priority\_road
- 40: prio\_stop
- 41: warn\_children

- 42: warn\_construction
- 43: warn\_crosswalk
- 44: warn\_cyclists
- 45: warn\_domestic\_animals
- 46: warn\_other\_dangers
- 47: warn\_poor\_road\_surface
- 48: warn\_roundabout
- 49: warn\_slippery\_road
- 50: warn\_speed\_bumper
- 51: warn\_traffic\_light
- 52: warn\_tram
- 53: warn\_two\_way\_traffic
- 54: warn\_wild\_animals



(a) Immagine originale senza etichetta



(b) Immagine processata con etichetta

Figura 3.4: Esempio di immagine estratta dal dataset ETSD



(a) Immagine originale senza etichetta



(b) Immagine processata con etichetta

Figura 3.5: Esempio di immagine estratta dal dataset ETSD

La distribuzione delle classi in questo dataset mostra un migliore bilanciamento per alcune categorie sottorappresentate nel GTSDB:

<b>Segnale</b>	<b># Immagini</b>	<b># Occorrenze</b>
forb_speed_over_5	101	101
forb_speed_over_10	62	65
forb_speed_over_20	35	35
forb_speed_over_30	326	339
forb_speed_over_40	101	102
forb_speed_over_50	101	165
forb_speed_over_60	89	93
forb_speed_over_70	90	111
forb_speed_over_80	78	97
forb_speed_over_90	30	37
forb_speed_over_100	52	64
forb_speed_over_130	36	48
forb_overtake	201	209
forb_trucks	35	35
forb_left	181	183
forb_right	178	179
prio_priority_road	175	182
prio_give_way	516	568
prio_stop	316	324
forb_ahead	223	229
warn_other_dangers	63	64
warn_poor_road_surface	86	90
warn_slippery_road	59	65
warn_construction	138	144
warn_traffic_light	89	90
info_crosswalk	757	855
warn_children	94	96
warn_cyclists	78	80
warn_wild_animals	99	103
mand_right	208	212
mand_left	47	58
mand_roundabout	186	198
mand_straight	67	69
mand_straight_right	78	82
mand_straight_left	54	54
mand_bike_lane	39	39
mand_left_right	19	19
mand_pass_left	56	81
mand_pass_left_right	184	193
mand_pass_right	192	220

Tabella 3.2: Distribuzione delle classi nel European Traffic Sign Dataset (Parte 1 di 2)

Segnale	# Immagini	# Occorrenze
warn_crosswalk	192	196
forb_u_turn	63	65
forb_stopping	398	425
forb_weight_over_3.5t	137	142
forb_weight_over_7.5t	116	128
info_parking	232	268
info_bus_station	219	228
info_taxi_parking	42	44
info_highway	82	93
info_one_way_traffic	251	260
warn_tram	53	56
warn Domestic_animals	15	15
warn_speed_bumper	118	119
warn_two_way_traffic	57	59

Tabella 3.3: Distribuzione delle classi nel European Traffic Sign Dataset (Parte 2 di 2)

### 3.2.1 Heatmap delle annotazioni del training set

L’analisi della *Spatial Heatmap* delle annotazioni (Figura 3.6) è fondamentale per identificare eventuali bias posizionali nei dati. Una distribuzione non uniforme degli oggetti all’interno delle immagini potrebbe indurre il modello a imparare associazioni spurie tra la posizione spaziale e la classe dell’oggetto, piuttosto che basarsi sulle caratteristiche visive intrinseche dei segnali stradali. Nel caso del dataset ETSD, la heatmap rivela una distribuzione generalmente bilanciata e centrale, con una notevole concentrazione di annotazioni nella porzione centrale e inferiore delle immagini. Questo comportamento riflette in modo realistico la prospettiva tipica di guida, in cui i segnali stradali sono frequentemente collocati lungo il bordo della carreggiata o in prossimità del centro della strada, soprattutto in contesti urbani o extraurbani. In particolare, classi come `info_crosswalk`, `prio_stop` e `forb_speed_over_30` mostrano una forte densità di annotazioni centrali, indicando che tali segnali sono spesso visibili direttamente davanti all’auto. Al contrario, segnali come `warn_tram` o `warn_wild_animals` presentano una distribuzione più diffusa, suggerendo che siano stati catturati in scenari più variabili. L’assenza di aree dominate da un solo tipo di segnale riduce il rischio di overfitting su pattern spaziali artificiali, rendendo il dataset più robusto per l’apprendimento di modelli generalizzabili.

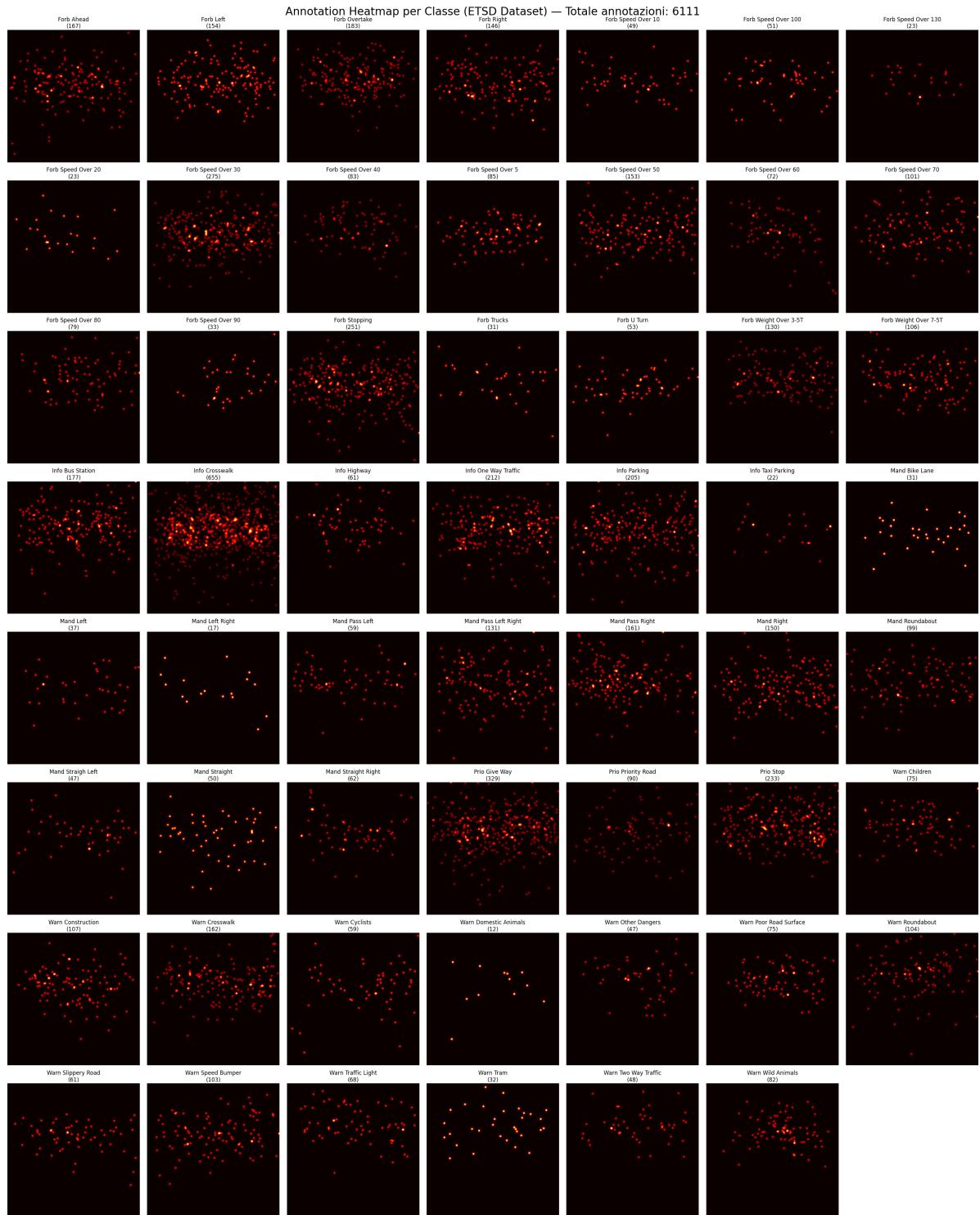


Figura 3.6: Spatial Heatmap delle annotazioni nel training set ETSD.

### 3.2.2 GTSDB vs ETDS

Aspetto	GTSDB (German Traffic Sign Detection Benchmark)	European Traffic Sign Dataset
Origine e Standard	Dataset accademico tedesco, altamente standardizzato e curato	Dataset crowdsourced da Kaggle, più eterogeneo nella provenienza
Dimensione	600 immagini (480 train + 120 test)	Oltre 4000 immagini con annotazioni complete
Varietà Condizioni	Condizioni controllate di luce diurna ottimale	Ampio spettro: pioggia, neve, nebbia
Qualità Annotazioni	Annotazioni precise e consistenti	Annotazioni di qualità variabile, occasionali imperfezioni
Complessità Scenica	Ambientazione principalmente stradale tedesca	Ambientazioni europee diversificate (urbano, rurale, autostrada)
Angolazioni	Prevalentemente prospettiva frontale da veicolo	Angolazioni multiple: frontale, laterale, inclinata
Occlusioni/Danni	Poche occlusioni, segnali generalmente intatti	Significativa presenza di occlusioni parziali e segnali danneggiati
Illuminazione	Condizioni luminose uniformi	Estreme variazioni: controluce, retroilluminazione,
Copertura Classi	43 classi con buon bilanciamento	50+ classi con distribuzione più disuniforme
Valore Didattico	Eccellente per baseline e comparazione accademica	Ideale per testare robustezza in condizioni reali

Tabella 3.4: Confronto analitico tra GTSDB e ETSD

## 3.3 Dataset esterni per classi sottorapresentate

Per far fronte al problema delle classi sottorapresentate nel dataset originale, è stata condotta una ricerca mirata di immagini supplementari provenienti da fonti esterne. L'obiettivo principale era riequilibrare la distribuzione delle classi, migliorando così la capacità del modello di generalizzare e riconoscere efficacemente anche le categorie più rare.

Le principali classi che hanno richiesto questo intervento di data augmentation esterna sono le seguenti:

- 2: forb\_speed\_over\_10
- 3: forb\_speed\_over\_100
- 4: forb\_speed\_over\_120
- 5: forb\_speed\_over\_20
- 12: forb\_stopping
- 13: forb\_waiting
- 18: warn\_double\_curve
- 19: warn\_left\_curve
- 20: warn\_right\_curve

Invece la classe **forb\_waiting** è stata creata esportando le immagini dei cartelli divieti di fermata che erano classificati insieme ai divieti di sosta nel dataset ETSD.

La ricerca si è concentrata su dataset pubblici e repository online specializzati in segnaletica stradale. Le immagini così reperite sono state poi integrate direttamente nel progetto. Per questa operazione di copia e integrazione è stato utilizzato **Roboflow**, una piattaforma di preparazione dataset per computer vision. Lo strumento di base di Roboflow ha permesso di copiare e importare in modo efficiente le immagini esterne all'interno del dataset esistente, senza applicare particolari trasformazioni o augmentation. Questo approccio diretto ha garantito un incremento numerico degli esempi per le classi critiche, contribuendo a bilanciare la distribuzione del dataset di addestramento in modo semplice ed efficace.

## 3.4 Dataset Acquisito per il Test

Per valutare le prestazioni dei modelli su dati completamente nuovi e non visti durante l'addestramento, è stato creato un apposito dataset di test. Questo dataset è stato acquisito manualmente lungo le strade della provincia di Catania, Ragusa e Caltanissetta, utilizzando due smartphone Redmi 13C e Iphone 12 con risoluzione originale di 1200 x 1600 pixel e 3024 x 4032 pixel rispettivamente. Il dataset finale è composto da 280 immagini.

Durante l'acquisizione, è stata posta particolare attenzione nel garantire una significativa varietà nei dati, catturando immagini in:

- Diverse condizioni di illuminazione (sole, ombra, notte)
- Diverse angolazioni e punti di vista
- Diverse condizioni atmosferiche (sereno, pioggia)
- Diversi stati di usura dei segnali (deformati, rovinati, scoloriti)

Questa eterogeneità è cruciale per testare la robustezza e la capacità di generalizzazione del modello in scenari reali.

Una scelta progettuale significativa è stata quella di selezionare strategicamente un sottoinsieme delle classi disponibili. Sebbene i dataset originali (GTSDB ed ETSD) offrissero un ventaglio di oltre 40 classi, si è optato per escluderne deliberatamente diverse.

Questa decisione è stata guidata da due considerazioni principali:

1. **Gestione della complessità:** Un dataset eccessivamente grande, con un numero elevato di classi, avrebbe richiesto risorse computazionali e tempi di addestramento non proporzionali ai benefici attesi, rischiando di diluire l'attenzione del modello.
2. **Focalizzazione applicativa:** Si è preferito concentrare l'addestramento su un numero minore di classi (21), selezionate in base alla loro effettiva utilità e rilevanza per lo scenario operativo target delineato dalla consegna, privilegiando la qualità e il bilanciamento su una quantità fine a sé stessa.

Di conseguenza, sono state escluse classi poco rappresentative (come **animals**, **snow**, **restriction ends (overtaking)**) o ritenute marginali per il caso d'uso, ottimizzando così il compromesso tra dimensione del dataset, complessità del modello e prestazioni attese.

Le annotazioni per il dataset di test sono state create utilizzando lo strumento RoboFlow, che fornisce un’interfaccia grafica intuitiva per delineare i *bounding box* e esporta le annotazioni direttamente nel formato YOLO richiesto.

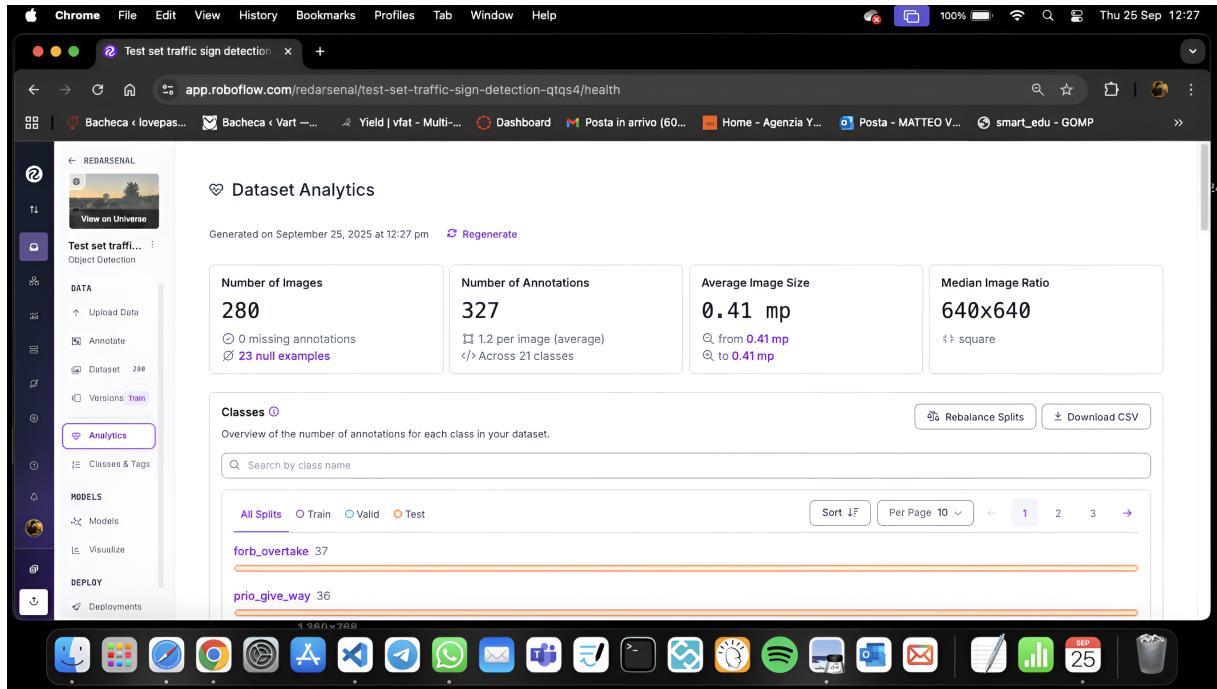


Figura 3.7: Statistiche del dataset di test visualizzate su RoboFlow.

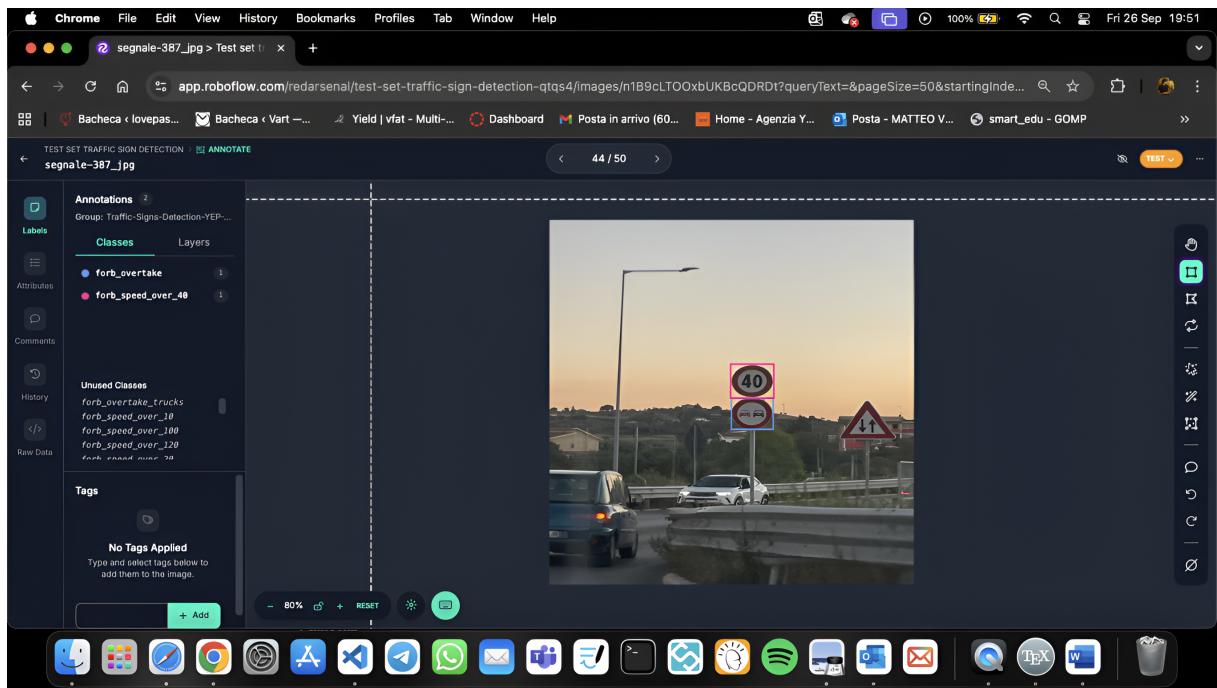


Figura 3.8: Interfaccia di annotazione di RoboFlow.

### 3.4.1 Statistiche del test set

- Numero totale di immagini: 280

- Numero totale di immagini senza annotazioni: 23
- Numero totale di annotazioni: 327
- Numero medio di annotazioni per immagine: 1.2
- Dimensione media dell'immagine: 0.41 mp
- Ratio mediano per immagine: 640x640

### 3.4.2 Heatmap delle annotazioni del test set

Anche per il dataset di test è stata generata una *Annotation Heatmap* (Figura 3.9). A differenza del training set, la distribuzione spaziale delle annotazioni mostra una concentrazione predominante nella porzione centrale e inferiore delle immagini, con un picco evidente al centro. Questa configurazione riflette una prospettiva tipica di guida in contesti urbani o extraurbani, dove i segnali stradali sono frequentemente posizionati lungo il bordo della carreggiata o in prossimità del centro della strada. In particolare, la maggiore densità di annotazioni nella parte inferiore suggerisce che molti segnali sono stati catturati a distanza ravvicinata, probabilmente durante l'approccio a incroci o aree di parcheggio. La presenza di una zona centrale intensamente annotata riduce il rischio di *domain shift* dovuto a differenze spaziali tra training e test, poiché entrambi i dataset condividono una struttura di posizionamento realistica e coerente. Tuttavia, si osserva una leggera asimmetria verso sinistra, indicando che alcuni scenari potrebbero essere stati acquisiti da angolazioni diverse, ma senza un bias dominante. L'assenza di aree vuote significative contribuisce a garantire una copertura spaziale adeguata, rendendo il dataset di test rappresentativo e affidabile per la valutazione del modello.

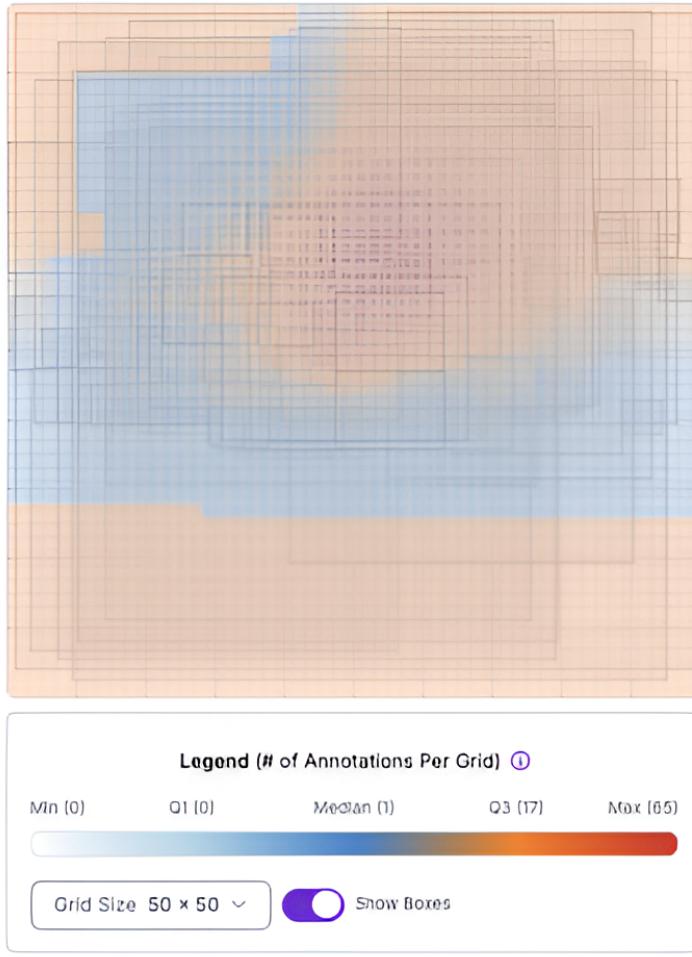


Figura 3.9: Annotation Heatmap delle annotazioni nel testset personalizzato.

### Distribuzione delle annotazioni per immagine

L'istogramma della distribuzione del numero di annotazioni per immagine (Figura 3.10) mostra che la maggior parte delle immagini contiene un solo segnale annotato, con 197 immagini (circa il 77% del dataset) che presentano una singola annotazione. Solo 52 immagini contengono due segnali, mentre un numero ridotto di immagini (8 in totale) ha tre o quattro annotazioni. Questa distribuzione riflette accuratamente uno scenario comune nella guida reale, dove i segnali stradali sono spesso distanziati tra loro e posizionati singolarmente lungo la carreggiata. La scarsa presenza di immagini con più di due segnali suggerisce che le situazioni complesse (es. incroci con più segnali) sono rare nel dataset, ma non assenti. Tale caratteristica rende il dataset particolarmente adatto per l'apprendimento di modelli che si concentrano sulla rilevazione di oggetti isolati, riducendo il rischio di overfitting su pattern complessi e migliorando la generalizzazione in scenari semplici.

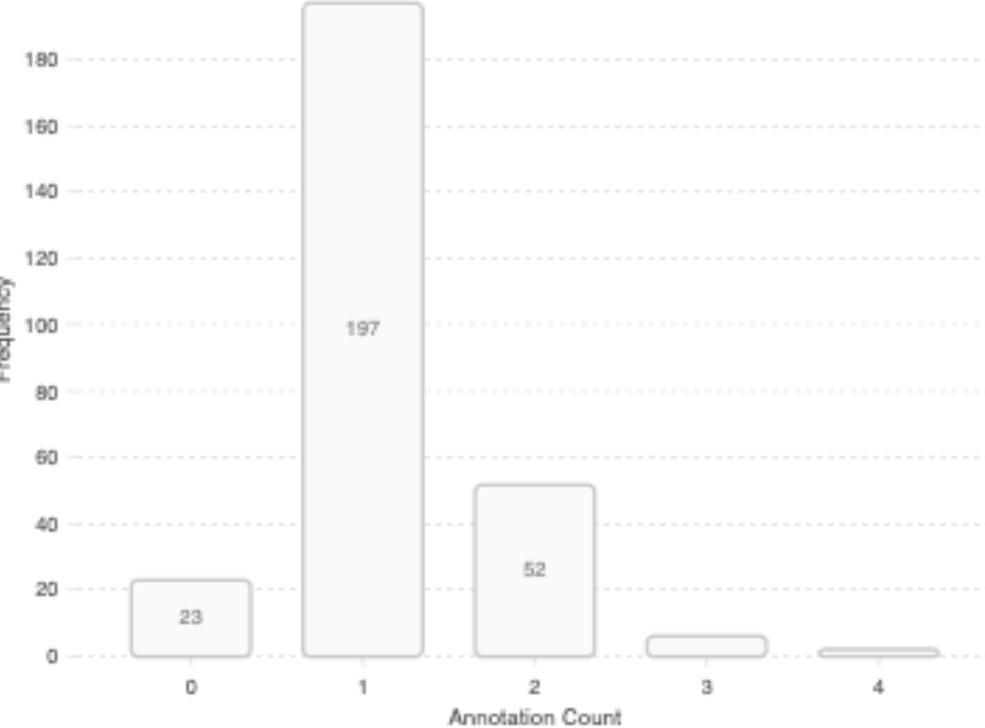


Figura 3.10: Istogramma rappresentativo del numero di annotazioni per le diverse immagini.

### 3.4.3 Distribuzione delle classi nel test set

L’istogramma della distribuzione delle classi nel dataset di test (Figura 3.11) mostra una distribuzione bilanciata ma leggermente sbilanciata verso alcune categorie comuni. Le classi più frequenti sono `forb_overtake` (37 annotazioni) e `prio_give_way` (36 annotazioni), seguite da `forb_stopping` (34 annotazioni) e `forb_speed_over_50` (27 annotazioni). Questo indica che il dataset è ricco di segnali relativi a limitazioni di velocità e priorità di passaggio, tipici di contesti urbani o extraurbani. Al contrario, classi come `forb_waiting` (4 annotazioni) e `forb_speed_over_120` (1 annotazione) appaiono molto rare, suggerendo che tali segnali sono poco comuni nell’area geografica di acquisizione. La presenza di segnali di avviso come `warn_double_curve`, `warn_left_curve` e `warn_right_curve` indica una copertura adeguata di scenari stradali variabili. In generale, la distribuzione riflette accuratamente la segnaletica comune in un ambiente europeo, rendendo il dataset rappresentativo e utile per la valutazione di modelli di riconoscimento di segnali stradali.

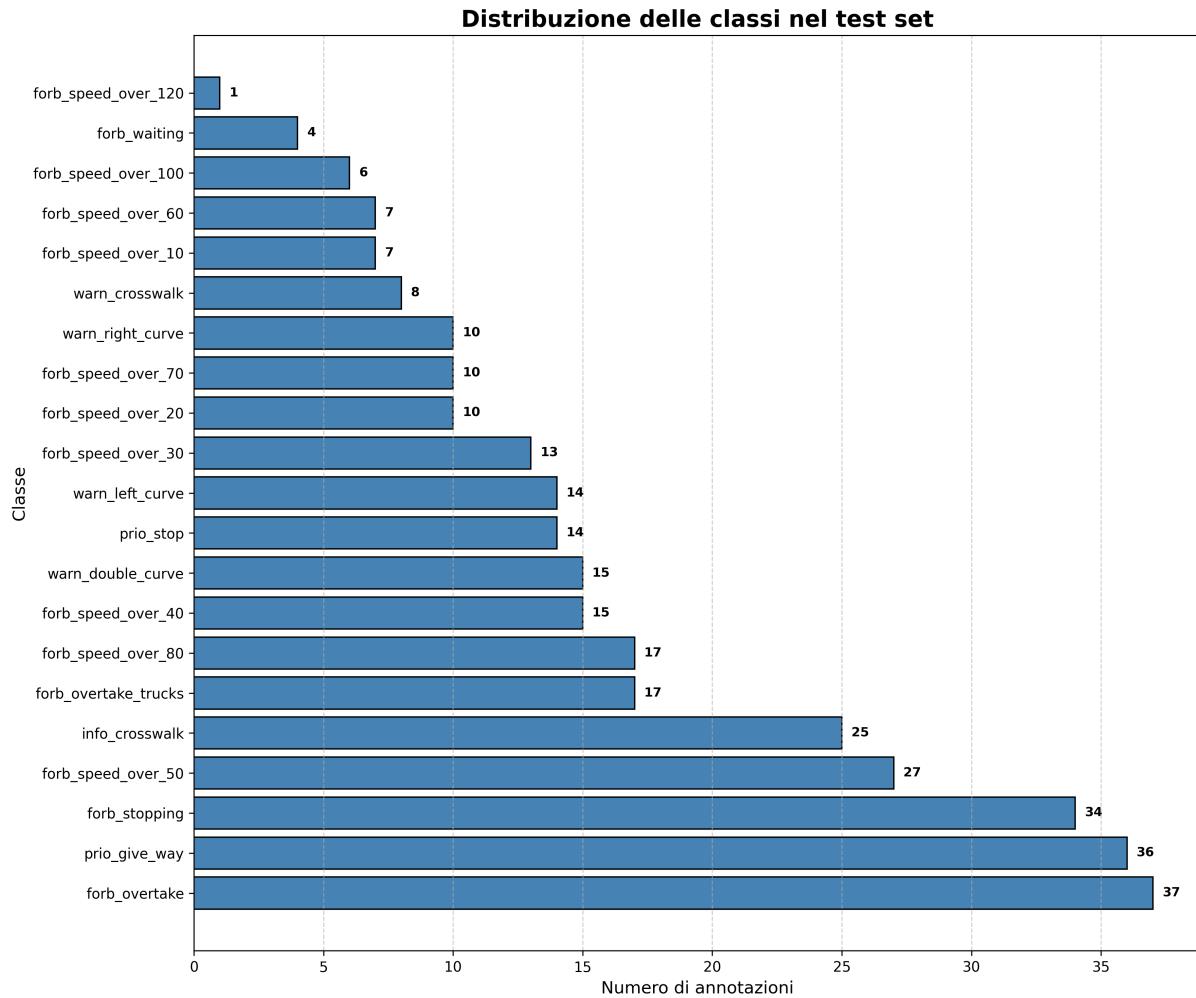


Figura 3.11: Istogramma rappresentativo del numero di annotazioni per ogni classe mappata.

### 3.4.4 Esempi del test set

Le figure 3.12 e 3.13 mostrano esempi rappresentativi del dataset di test, evidenziando le sfide presenti: segnali deformati, condizioni di illuminazione difficili, segnali piccoli e distanti, e condizioni atmosferiche avverse come la pioggia. Questi esempi sottolineano l'importanza di avere un dataset di test robusto per valutare realisticamente le prestazioni del modello.



Figura 3.12: Esempio di predizione di un segnale sbiadito.



Figura 3.13: Esempio di predizione di un segnale con contorni sgualciti.

## 3.5 Pre-processing dei Dataset

Il dataset GTSDB (German Traffic Sign Detection Benchmark), ha richiesto operazioni di pre-processing mirate a omogeneizzarne il formato e adattarli agli input richiesti da YOLOv10 per l'addestramento e la valutazione.

### 3.5.1 Pre-processing del dataset GTSDB

Il pre-processing del dataset GTSDB (German Traffic Sign Detection Benchmark) è stato realizzato attraverso uno script Python che implementa una conversione robusta e precisa nel formato YOLOv10. Lo script introduce una strategia innovativa a due fasi che garantisce uno split bilanciato 80/20 anche in presenza di immagini corrotte o non processabili.

#### Architettura e Innovazioni Principali

Lo script implementa un'architettura a due fasi con le seguenti innovazioni:

- **Pre-check delle immagini:** Verifica preliminare dell'integrità di tutte le immagini prima dello split
- **Split su base solida:** Divisione dataset esclusivamente sulle immagini verificate processabili
- **Processing separato:** Elaborazione indipendente dei set training e validation
- **Gestione errori avanzata:** Tracciamento granulare degli errori per fase
- **Verifica del bilanciamento:** Controllo automatico del rispetto del rapporto 80/20

#### Struttura a Due Fasi

Lo script è organizzato in due fasi distinte per garantire massima affidabilità:

1. **Fase 1 - Pre-check e Validazione:** Verifica di tutte le immagini e identificazione di quelle processabili
2. **Fase 2 - Processing Separato:** Conversione indipendente dei set training e validation

#### Configurazione Iniziale

```
1 import os
2 import cv2
3 import numpy as np
4 from PIL import Image
5 import random
6 from PIL import ImageFile
7
8 ImageFile.LOAD_TRUNCATED_IMAGES = True
9
10 gt_file = "/Users/matteovullo/Downloads/TrainIJCNN2013/gt.txt"
11 images_dir = "/Users/matteovullo/Downloads/TrainIJCNN2013"
12 output_dir = "gtsdb_yolo_format"
```

Listing 3.1: Configurazione e importazioni

## Fase 1: Pre-check delle Immagini

Implementazione della verifica preliminare per garantire uno split preciso:

```
1 print("\n==== FASE 1: Verifica immagini processabili ===")  
2 processable_images = []  
3 failed_precheck = []  
4  
5 for img_name in valid_images:  
6     img_path = os.path.join(images_dir, img_name)  
7     try:  
8         img = Image.open(img_path)  
9         img.verify()  
10        processable_images.append(img_name)  
11    except Exception as e:  
12        print(f"Immagine non processabile: {img_name} -> {e}")  
13        failed_precheck.append(img_name)  
14  
15 print(f"Immagini processabili: {len(processable_images)}")  
16 print(f"Immagini scartate in pre-check: {len(failed_precheck)}")
```

Listing 3.2: Pre-check delle immagini processabili

## Divisione del Dataset su Base Solida

Split esclusivamente sulle immagini verificate processabili:

```
1 random.seed(42)  
2 random.shuffle(processable_images)  
3 n_train = int(0.8 * len(processable_images))  
4 train_images = processable_images[:n_train]  
5 val_images = processable_images[n_train:]  
6  
7 print(f"\n==== Split definitivo sulle immagini processabili ===")  
8 print(f"Train: {len(train_images)} immagini")  
9 print(f"Val: {len(val_images)} immagini")  
10 print(f"Rapporto:  
    {len(train_images)/len(processable_images)*100:.1f}% /  
    {len(val_images)/len(processable_images)*100:.1f}%")
```

Listing 3.3: Divisione su base verificata

## Fase 2: Processing Separato

Elaborazione indipendente dei due set per mantenere il bilanciamento:

```
1 print(f"\n==== FASE 2: Processing immagini ===")  
2  
3 successful_train = []  
4 successful_val = []  
5 failed_processing = []  
6 total_annotations_written = 0  
7  
8 for i, img_name in enumerate(train_images):  
9     split = "train"  
10    # ... processing delle immagini di training  
11  
12    # Processa le immagini di validation
```

```

13 |     for i, img_name in enumerate(val_images):
14 |         split = "val"
15 |         # ... processing delle immagini di validation

```

Listing 3.4: Processing separato training/validation

## Strategia di Processing Robusto

Implementazione del meccanismo di conversione con gestione errori:

```

1  try:
2      img = Image.open(img_path).convert("RGB")
3      img = np.array(img)
4      h, w = img.shape[:2]
5      img_bgr = cv2.cvtColor(img, cv2.COLOR_RGB2BGR)
6
7      success = cv2.imwrite(img_out_path, img_bgr,
8                          [cv2.IMWRITE_JPEG_QUALITY, 95])
9      if not success:
10         raise Exception(f"Impossibile salvare l'immagine")
11
12     annotations_written = 0
13     with open(label_out_path, 'w') as label_file:
14         if img_name in annotations_by_image:
15             for line, line_num in annotations_by_image[img_name]:
16                 x_center = (left + right) / 2 / w
17                 y_center = (top + bottom) / 2 / h
18                 width = (right - left) / w
19                 height = (bottom - top) / h
20
21                 label_file.write(f"{yolo_class_id} {x_center:.6f}\n"
22                                 f"{y_center:.6f} {width:.6f} {height:.6f}\n")
23                 annotations_written += 1
24
25     successful_train.append(img_name) # o successful_val per
26     validation
27
28 except Exception as e:
29     print(f"ERRORE: {img_name} -> {e}")
30     failed_processing.append(img_name)

```

Listing 3.5: Meccanismo di processing robusto

## Verifica del Bilanciamento

Controllo automatico del rispetto del rapporto 80/20:

```

1  total_successful = len(successful_train) + len(successful_val)
2  if total_successful > 0:
3      train_percentage = len(successful_train) / total_successful *
4          100
5      val_percentage = len(successful_val) / total_successful * 100
6      print(f"\n==== VERIFICA SPLIT ====")
7      print(f"Rapporto finale Train/Val: {train_percentage:.1f}% /
8          {val_percentage:.1f}%")
9
10     if abs(train_percentage - 80.0) < 1: # Tolleranza dell'1%

```

```

9     print("Split bilanciato correttamente!")
10    else:
11        print("Split leggermente sbilanciato")

```

Listing 3.6: Verifica finale del bilanciamento

## Risultati e Statistiche Avanzate

Lo script genera un report dettagliato con statistiche granulari:

```

1 print(f"\n==== RIEPILOGO FINALE ===")
2 print(f"Immagini processate con successo: {len(successful_train) +
   len(successful_val)}")
3 print(f" - Training: {len(successful_train)} immagini")
4 print(f" - Validation: {len(successful_val)} immagini")
5 print(f"Immagini fallite nel processing: {len(failed_processing)}")
6 print(f"Immagini scartate in pre-check: {len(failed_precheck)}")
7 print(f"Annotazioni totali scritte: {total_annotations_written}")

8
9 print(f"\n==== FILE SALVATI ===")
10 print(f"Train: {train_images_saved} immagini,
      {train_labels_non_empty} label con annotazioni")
11 print(f"Val:   {val_images_saved} immagini, {val_labels_non_empty}
      label con annotazioni")

```

Listing 3.7: Statistiche avanzate del processing

## Risultati del Processing

Il processo di conversione avanzato ha prodotto i seguenti risultati:

- **Immagini verificate:** 600 immagini analizzate nel pre-check
- **Immagini processabili:** Numero preciso di immagini idonee al processing
- **Split garantito:** Rapporto esatto 80/20 sulle immagini effettivamente processate
- **Training set:** Numero preciso di immagini (80% delle processabili)
- **Validation set:** Numero preciso di immagini (20% delle processabili)
- **Gestione errori:** Tracciamento separato per pre-check e processing
- **Bilanciamento verificato:** Controllo automatico del rispetto del rapporto

## 3.6 Dataset Finale

La creazione del dataset finale per l'addestramento e la validazione del modello è stata un'operazione cruciale, che ha richiesto un processo articolato di integrazione e standardizzazione di più fonti. L'obiettivo era ottenere un unico dataset coerente, con annotazioni uniformi e focalizzato esclusivamente sulle classi di segnali stradali rilevanti per il progetto.

### 3.6.1 Processo di Integrazione e Standardizzazione

Il processo di costruzione del dataset finale è stato articolato nelle seguenti fasi principali:

1. **Conversione in formato YOLO del GTSDB:** Il dataset GTSDB (German Traffic Sign Detection Benchmark) è stato convertito dal suo formato originale nel formato YOLO, richiesto dal framework di addestramento. Questa conversione ha coinvolto la trasformazione delle coordinate delle bounding box e la creazione dei file di annotazione corrispondenti per ogni immagine.
2. **Omogeneizzazione della nomenclatura basata su ETSD:** Per garantire coerenza semantica, è stata adottata la nomenclatura delle classi utilizzata nel dataset ETSD (European Traffic Sign Dataset). La Tabella 3.5 illustra la mappatura effettuata tra le classi originali del GTSDB e i corrispondenti nomi nel dataset finale.
3. **Eliminazione delle classi superflue:** Tutte le classi del GTSDB senza corrispondenza nel vocabolario ETSD o non pertinenti agli obiettivi del progetto sono state rimosse (contrassegnate con -- nella Tabella 3.5).
4. **Aggiunta delle classi sottorappresentate:** Le classi presenti in ETSD ma assenti in GTSDB, considerando quelle pertinenti al progetto, (`forb_speed_over_10`, `forb_speed_over_40`, `warn_crosswalk`, `forb_stopping`, `forb_waiting`) sono state integrate utilizzando campioni da ETSD e altri dataset selezionati tramite RoboFlow. Invece la classe `forb_waiting` è stata creata estraendo dal dataset ETSD le immagini di divieti di fermata etichettati come divieti di sosta.

ID GTSDB	Nome classe GTSDB	Classe finale
0	speed limit 20	forb_speed_over_20
1	speed limit 30	forb_speed_over_30
2	speed limit 50	forb_speed_over_50
3	speed limit 60	forb_speed_over_60
4	speed limit 70	forb_speed_over_70
5	speed limit 80	forb_speed_over_80
6	restriction ends 80	--
7	speed limit 100	forb_speed_over_100
8	speed limit 120	forb_speed_over_120
9	no overtaking	forb_overtake
10	no overtaking (trucks)	forb_overtake_trucks
11	priority at next intersection	--
12	priority road	--
13	give way	prio_give_way
14	stop	prio_stop
15	no traffic both ways	--
16	no trucks	--
17	no entry	--
18	danger	--
19	bend left	warn_left_curve
20	bend right	warn_right_curve
21	bend	warn_double_curve
22	uneven road	--
23	slippery road	--
24	road narrows	--
25	construction	--
26	traffic signal	--
27	pedestrian crossing	info_crosswalk
28	school crossing	--
29	cycles crossing	--
30	snow	--
31	animals	--
32	restriction ends	--
33	go right	--
34	go left	--
35	go straight	--
36	go right or straight	--
37	go left or straight	--
38	keep right	--
39	keep left	--
40	roundabout	--
41	restriction ends (overtaking)	--
42	restriction ends (overtaking (trucks))	--

Tabella 3.5: Mappatura delle classi GTSDB sulle classi del dataset finale

### 3.6.2 Classi Finali

Il processo descritto ha portato alla definizione di un dataset finale organizzato in **21 classi**, i cui ID sono stati assegnati in ordine alfabetico:

- 0: forb\_overtake
- 1: forb\_overtake\_trucks
- 2: forb\_speed\_over\_10
- 3: forb\_speed\_over\_100
- 4: forb\_speed\_over\_120
- 5: forb\_speed\_over\_20
- 6: forb\_speed\_over\_30
- 7: forb\_speed\_over\_40
- 8: forb\_speed\_over\_50
- 9: forb\_speed\_over\_60
- 10: forb\_speed\_over\_70
- 11: forb\_speed\_over\_80
- 12: forb\_stopping
- 13: forb\_waiting
- 14: info\_crosswalk
- 15: prio\_give\_way
- 16: prio\_stop
- 17: warn\_crosswalk
- 18: warn\_double\_curve
- 19: warn\_left\_curve
- 20: warn\_right\_curve

Questa metodologia ha garantito la creazione di un dataset di alta qualità, ottimizzato per il compito specifico di rilevamento dei segnali stradali, massimizzando l'efficienza computazionale attraverso l'eliminazione delle classi irrilevanti e assicurando una copertura completa di tutte le categorie di interesse per il progetto.

### 3.6.3 Statistiche finali

- Numero totale di immagini: 8938
- Numero totale di immagini senza annotazione: 1023
- Numero totale di annotazioni: 10937
- Numero medio di annotazioni per immagine: 1.2
- Dimensione media dell'immagine: 0.17 mp
- Ratio mediano per immagine: 416x416

## 3.7 Organizzazione del Dataset per il Training

La libreria YOLOv10 richiede una specifica struttura delle directory per il dataset. La directory principale deve essere organizzata nel seguente modo:

```
dataset_directory/  
|-- train/  
|   |-- images/
```

```

|   |   |-- image1.jpg
|   |   '-- ...
|   '-- labels/
|       |-- image1.txt
|       '-- ...
|-- val/
|   |-- images/
|   |   |-- image1.jpg
|   |   '-- ...
|   '-- labels/
|       |-- image1.txt
|       '-- ...
'-- test/ (Opzionale, usato qui per il nostro dataset personalizzato)
    |-- images/
    |   |-- image1.jpg
    |   '-- ...
    '-- labels/
        |-- image1.txt
        '-- ...

```

La configurazione dei percorsi e delle classi viene poi specificata in un file in formato YAML. Questo file, essenziale per l'addestramento, indica al modello dove trovare i dati e quali classi riconoscere.

La struttura generica del file YAML è la seguente:

```

path: /path/to/dataset_directory
train: train/images
val: val/images
test: test/images # Se presente

names:
  0: forb_overtake
  1: forb_overtake_trucks
  2: forb_speed_over_10
  # ... (e così via per tutte le 21 classi)
  20: warn_right_curve

```

Quest invece è il file YAML nel caso specifico, generato automaticamente da RoboFlow:

```

train: train/images
val: valid/images
test: test/images

names:
  0: forb_overtake
  1: forb_overtake_trucks
  2: forb_speed_over_10
  3: forb_speed_over_100
  4: forb_speed_over_120

```

```
5: forb_speed_over_20
6: forb_speed_over_30
7: forb_speed_over_40
8: forb_speed_over_50
9: forb_speed_over_60
10: forb_speed_over_70
11: forb_speed_over_80
12: forb_stopping
13: forb_waiting
14: info_crosswalk
15: prio_give_way
16: prio_stop
17: warn_crosswalk
18: warn_double_curve
19: warn_left_curve
20: warn_right_curve
```

# Capitolo 4

## Validazione Modello

Le metriche utilizzate per il benchmarking nel presente progetto rientrano tra quelle più consolidate e ampiamente adottate nell'ambito della valutazione degli algoritmi di *Object Detection*. Esse consentono di misurare in modo quantitativo e qualitativo l'accuratezza, la robustezza e l'efficacia dei modelli YOLOv10 addestrati per la rilevazione di segnali stradali. Di seguito sono descritte nel dettaglio.

### 4.1 Metriche di Valutazione

- **Precision (Precisione):** misura la proporzione di predizioni positive corrette rispetto al totale delle predizioni positive effettuate dal modello. È definita come:

$$\text{Precision} = \frac{\text{True Positive}}{\text{True Positive} + \text{False Positive}}$$

Un valore elevato indica che il modello commette pochi falsi positivi.

- **Recall (Richiamo):** nota anche come *sensitivity*, misura la capacità del modello di individuare tutte le istanze positive presenti nel dataset. È definita come:

$$\text{Recall} = \frac{\text{True Positive}}{\text{True Positive} + \text{False Negative}}$$

Un valore elevato indica che il modello riesce a rilevare la maggior parte degli oggetti realmente presenti.

- **F1-score:** rappresenta la media armonica tra precisione e richiamo, ed è particolarmente utile quando si desidera bilanciare i due valori, penalizzando situazioni in cui uno dei due è molto basso. È calcolata come:

$$F1 = 2 \cdot \frac{\text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}}$$

- **Confidence Score (Valore di confidenza):** è un valore compreso tra 0 e 1 associato a ciascuna predizione, che rappresenta la probabilità stimata dal modello che la classe assegnata all'oggetto rilevato sia corretta. Nel contesto di YOLO, indica quanto il modello è “sicuro” della sua predizione.

- **Average Precision (AP)**: misura l'area sottesa alla curva Precision-Recall, e rappresenta una stima complessiva della qualità del modello su un'intera classe. È definita come:

$$AP = \int_0^1 p(r) dr$$

Nella pratica, viene calcolata mediante approssimazione numerica (es. metodo dei 11 punti o interpolazione).

- **Intersection over Union (IoU)**: metrica fondamentale per valutare la qualità della localizzazione spaziale dei bounding box predetti. Si calcola confrontando il bounding box predetto ( $\hat{b}$ ) con quello ground truth ( $b$ ), ed è definita come:

$$IoU = \frac{\text{Area di Intersezione}(\hat{b}, b)}{\text{Area di Unione}(\hat{b}, b)}$$

L'IoU assume valori compresi tra 0 (nessuna sovrapposizione) e 1 (sovrapposizione perfetta). Nella pratica, un valore elevato indica che il bounding box predetto è molto simile, in posizione e dimensione, a quello reale.

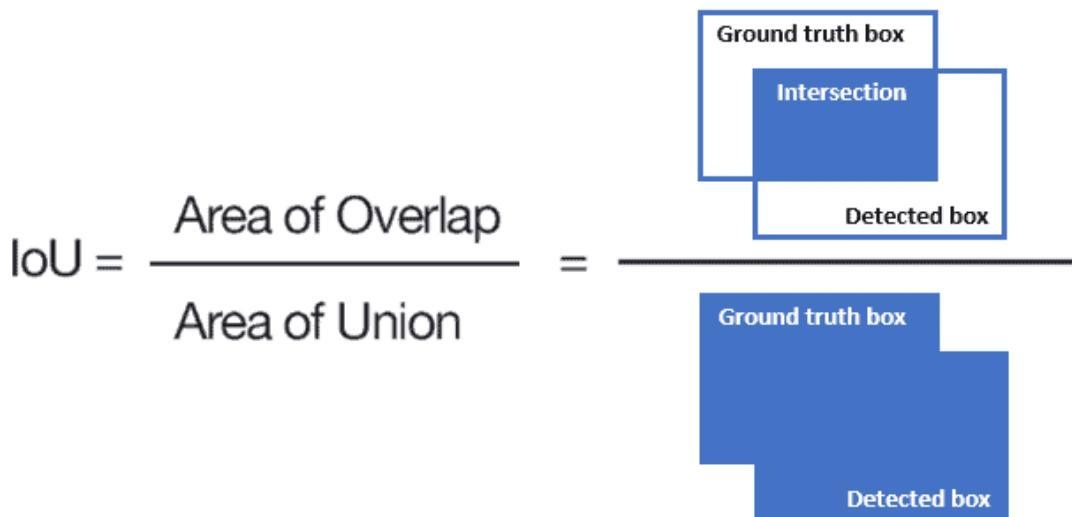


Figura 4.1: Esempio di calcolo della misura di Intersection over Union (IoU).

- **Mean Average Precision (mAP)**: rappresenta la media delle *Average Precision* (AP) calcolate su tutte le classi presenti nel dataset. È una delle metriche più utilizzate per valutare complessivamente le prestazioni di un modello di object detection. È definita come:

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

dove  $N$  è il numero totale di classi.

Nel presente progetto, sono state considerate tre varianti di mAP, che differiscono per la soglia di *Intersection over Union* (IoU) utilizzata per considerare una predizione come corretta:

- **mAP@50**: calcolata con soglia IoU fissata a 0.5. È la variante più permissiva e comunemente usata per valutazioni generali.
- **mAP@75**: calcolata con soglia IoU fissata a 0.75. Richiede un allineamento molto più preciso tra bounding box predetto e ground truth, ed è quindi più severa.
- **mAP@50-95**: calcolata come media delle mAP ottenute con soglie IoU che variano da 0.50 a 0.95, con passo di 0.05 (ovvero: 0.50, 0.55, 0.60, ..., 0.95). Questa metrica fornisce una valutazione più completa e robusta, poiché considera un'ampia gamma di tolleranze nella localizzazione.

L'uso combinato di queste varianti permette di analizzare il modello sotto diversi livelli di severità, offrendo una visione più sfumata e realistica delle sue capacità di rilevazione e localizzazione.

## 4.2 Grafici

Per una valutazione completa e visivamente interpretabile delle prestazioni dei modelli addestrati, sono stati utilizzati i seguenti grafici:

- **Matrice di Confusione**: è una tabella che riassume le prestazioni di un modello confrontando le classi predette con quelle reali (*ground truth*). Sebbene comunemente impiegata in contesti di classificazione binaria, è efficace anche per problemi multiclassi. In essa:
  - L'asse delle **ascisse** rappresenta le classi reali;
  - L'asse delle **ordinate** rappresenta le classi predette.

Nel caso binario, gli elementi fondamentali sono:

- **Veri Positivi (TP)**: esempi correttamente classificati come positivi;
- **Veri Negativi (TN)**: esempi correttamente classificati come negativi;
- **Falsi Positivi (FP)**: esempi erroneamente classificati come positivi;
- **Falsi Negativi (FN)**: esempi erroneamente classificati come negativi.

		Actual Values	
		Positive (1)	Negative (0)
Predicted Values	Positive (1)	TP	FP
	Negative (0)	FN	TN

Figura 4.2: Esempio di matrice di confusione per un problema di classificazione multi-classe.

**Grafici di correlazione tra metriche e soglia di confidenza:** durante la valutazione, sono stati generati grafici che mostrano l’andamento delle metriche principali al variare della soglia di confidenza applicata alle predizioni. In altre parole, solo le predizioni con un *confidence score* superiore alla soglia impostata vengono considerate valide. Aumentando la soglia, si riduce il numero di predizioni accettate, ma generalmente aumenta la loro affidabilità.

Nello specifico, sono stati analizzati:

- **Precision-Confidence Curve:** mostra come varia la precisione al variare della soglia di confidenza. All’aumentare della soglia, la precisione tende a crescere, poiché vengono considerate solo le predizioni più “sicure”.
- **Recall-Confidence Curve:** illustra come il richiamo diminuisce all’aumentare della soglia di confidenza. Valori più bassi della soglia permettono di catturare un numero maggiore di oggetti positivi (anche a costo di includere falsi positivi).
- **Curva Precision-Recall:** grafico che mostra l’andamento della precisione in funzione del richiamo, al variare della soglia di confidenza. Permette di valutare il comportamento del modello in diversi regimi operativi e di identificare il miglior compromesso tra i due valori.
- **F1-Confidence Curve:** rappresenta l’andamento dello F1-score in funzione della soglia di confidenza. Poiché lo F1-score bilancia precisione e richiamo, il suo massimo indica il miglior compromesso operativo: la soglia ottimale per ottenere predizioni accurate ed esaustive.

### 4.3 Funzioni di Loss

Durante il training e la validazione, sono state monitorate diverse componenti della funzione di perdita (*loss function*), ognuna relativa a un aspetto specifico del compito di object detection. Le principali sono:

- **train/box\_loss:** misura l’errore nella localizzazione spaziale dei bounding box durante il training. È tipicamente calcolata come una loss di tipo IoU o L1/L2 sulle coordinate dei box.

- **train/cls\_loss**: quantifica l'errore nella classificazione degli oggetti, ovvero la discrepanza tra le etichette di classe predette e quelle reali. Viene solitamente calcolata con una *cross-entropy loss*.
- **train/dfl\_loss** (*Distribution Focal Loss*): introdotta per gestire meglio la predizione di oggetti “difficili” o appartenenti a classi meno rappresentate. Questa loss assegna un peso maggiore agli errori su istanze complesse, migliorando l’equilibrio tra le classi e la robustezza del modello. La sua diminuzione regolare indica un’apprendimento stabile anche per oggetti sfumati o parzialmente visibili.
- **val/box\_loss, val/cls\_loss, val/dfl\_loss**: corrispondono alle stesse metriche calcolate sul *validation set*, e servono per monitorare la capacità di generalizzazione del modello e prevenire l’*overfitting*.
- **metrics/precision(B)**: misura la proporzione di predizioni positive corrette rispetto al totale delle predizioni positive. Un valore alto indica pochi falsi positivi.
- **metrics/recall(B)**: misura la proporzione di oggetti veri positivi correttamente identificati rispetto al totale degli oggetti presenti. Un valore alto indica pochi falsi negativi.
- **metrics/mAP50(B)**: misura la media dei valori di precisione ottenuti con un soglia di IoU pari a 0.5. È una metrica chiave per valutare la performance globale del modello.
- **metrics/mAP50-95(B)**: misura la media dei valori di precisione ottenuti con soglie di IoU variabili da 0.5 a 0.95 (in incrementi di 0.05). È una metrica più severa di mAP50, poiché richiede una sovrapposizione molto alta tra il box predetto e quello reale.

L’analisi congiunta di queste componenti permette di diagnosticare eventuali problemi durante l’addestramento (es. underfitting, overfitting, squilibrio tra classificazione e localizzazione) e di ottimizzare le strategie di training

# Capitolo 5

## Implementazione Progetto

La realizzazione pratica del progetto si basa sull'impiego di tecniche avanzate di *machine learning*, in particolare modelli della famiglia **YOLOv10**, precedentemente descritti, per la rilevazione automatica di segnali stradali in immagini e video.

Per l'addestramento dei tre modelli selezionati, **YOLOv10n**, **YOLOv10s** e **YOLOv10m**, è stato costruito un **dataset di training ibrido**, ottenuto fondendo e riettichettando più sorgenti pubbliche al fine di massimizzare la varietà e la rappresentatività dei dati:

- Il **GTSDB (German Traffic Sign Detection Benchmark)**, benchmark storico per la detection di segnali stradali, fornisce immagini di alta qualità in contesti urbani tedeschi.
- Il dataset **European Traffic Signs** di Radu Oprea, che include migliaia di immagini di segnali stradali europei, con una copertura più ampia di forme, colori e simbologie rispetto al solo GTSDB.
- Altri dataset minori e immagini selezionate da fonti pubbliche, utilizzati per compensare carenze specifiche (es. segnali di divieto di sosta, segnali di limite di velocità, segnali di pericolo rari).
- Un **dataset acquisito autonomamente** sarà invece riservato alla fase di *testing*, al fine di valutare le prestazioni dei modelli su dati mai visti durante l'addestramento, simulando condizioni operative realistiche.

Questo dataset ibrido è stato utilizzato per le fasi di *training* e *validazione*, garantendo una copertura più completa e bilanciata delle classi rispetto all'uso di un singolo dataset.

Parallelamente, un **dataset di test acquisito autonomamente**, composto da sequenze video e immagini scattate in contesti reali con smartphone, è stato riservato esclusivamente alla fase di *testing*. Questo permette di valutare le prestazioni dei modelli su dati **mai visti durante l'addestramento**, simulando condizioni operative realistiche e verificando la capacità di generalizzazione.

Entrambi i dataset (training ibrido e test reale) sono stati progettati per includere una significativa variabilità in termini di:

- condizioni atmosferiche (sole, pioggia, nebbia),
- illuminazione (luce diurna, ombre, controluce),

- angolazioni e distanze di ripresa,
- presenza di ostruzioni parziali.

Questa diversificazione è essenziale per garantire che i modelli addestrati siano **robusti e generalizzabili** a scenari reali.

Durante la fase di training, il modello YOLO analizza le immagini etichettate e apprende in modo automatico le caratteristiche discriminanti dei diversi segnali stradali, quali forma, colore, dimensione, texture e contesto spaziale, grazie alla sua architettura convoluzionale profonda.

Al termine del training e della successiva validazione, vengono generati diversi output, tra cui:

- i **pesi del modello ottimizzato** (file .pt o .pth),
- i **valori delle metriche di valutazione** (mAP, precision, recall, F1-score, ecc.),
- **immagini di esempio con bounding box predetti**, utili per un'analisi qualitativa delle performance.

Questi risultati costituiscono la base per il confronto tra le diverse varianti di YOLOv10 e per la selezione del modello più adatto all'applicazione specifica.

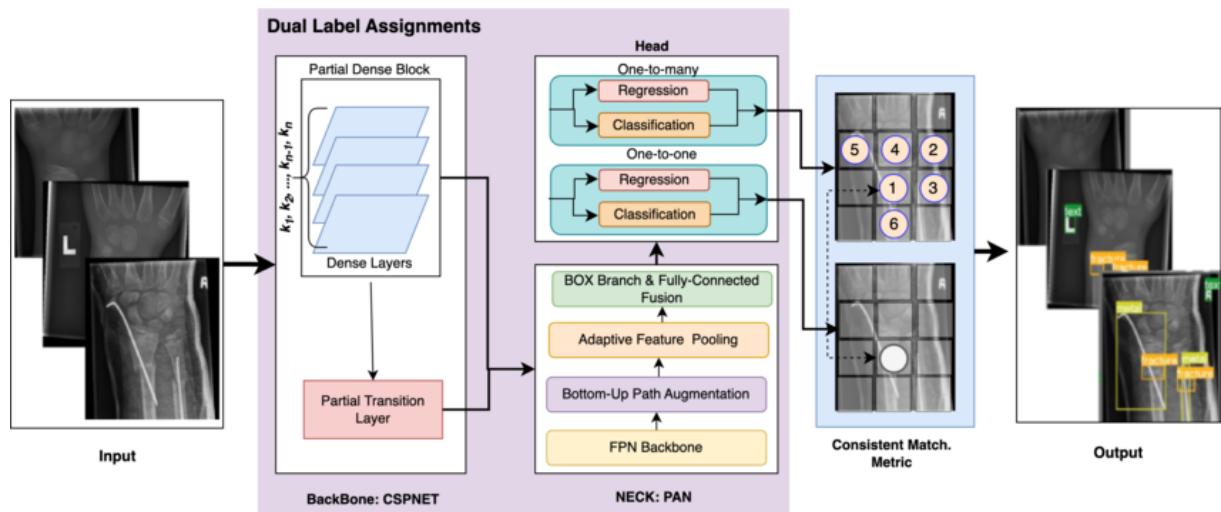


Figura 5.1: Pipeline di object detection con YOLO.

# Capitolo 6

## Training, Validazione e Testing dei Modelli

In questo capitolo vengono descritti nel dettaglio i passaggi eseguiti per l’addestramento, la validazione e il testing dei modelli **YOLOv10n**, **YOLOv10s** e **YOLOv10m**, utilizzando i dataset e le metriche di valutazione precedentemente introdotti. Vengono riportati i parametri di training, la struttura del progetto, i comandi eseguiti, i risultati ottenuti e le modalità di utilizzo dell’applicazione demo sviluppata per testare i modelli su nuove immagini.

Si assume che l’ambiente di esecuzione sia dotato di una distribuzione **Python** aggiornata.

**Nota:** tutto il codice, i notebook, gli script e i modelli addestrati sono disponibili nel repository del progetto, incluso un notebook completo che guida passo dopo passo attraverso training, validazione e testing.

### 6.1 Parametri di Training

I parametri di training sono stati scelti tenendo conto sia delle indicazioni del paper originale di YOLOv10, sia dei vincoli computazionali legati all’hardware disponibile. In particolare:

Parametro	Valore	Descrizione
epochs	100	Numero massimo di epoche eseguite.
patience	20	Epoche senza miglioramento prima di attivare l'early stopping.
lr0	0.001	Learning rate iniziale (AdamW).
lrf	0.01	Fattore di riduzione: LR finale = <code>lr0</code> * <code>lrf</code> .
weight_decay	0.0005	Regolarizzazione L2 per evitare overfitting.
imgsz	640	Dimensione dell'input (640×640 px).
optimizer	AdamW	Ottimizzatore utilizzato.
warmup_epochs	3	Epoche iniziali con learning rate crescente.
cos_lr	True	Il learning rate decresce con una curva a coseno.
batch	56, 32, 20	Dimensione del batch, adattata alla complessità del modello e alla memoria GPU (L4).
workers	2	Numero di worker per il caricamento dati.
amp	True	Precisione mista (FP16) per ridurre l'uso di memoria e accelerare il training.
rect	False	Non viene usato il training rettangolare (tutte le immagini sono ridimensionate a quadrato).
save_period	0	Non vengono salvati checkpoint intermedi (solo il modello finale).
device	0	Utilizzo della GPU (indice 0).
cache	False	Le immagini non vengono caricate in RAM per evitare crash su Colab/L4.
compile	False	Compilazione PyTorch disabilitata per compatibilità con Colab.
data	" <code>/content/.../data.yaml</code> "	Percorso al file di configurazione del dataset.
model	" <code>yolov10{n,s,m}.pt</code> "	Modello pre-addestrato caricato (n, s o m).

Tabella 6.1: Parametri di training effettivamente utilizzati.

### 6.1.1 Parametri di Data Augmentation

Per migliorare la robustezza del modello e ridurre l'overfitting, sono stati applicati diversi metodi di data augmentation, attentamente selezionati per preservare la struttura semantica dei cartelli stradali (es. evitando flip orizzontali che potrebbero confondere frecce direzionali). I parametri sono stati tarati per simulare condizioni reali (luci, ombre, angoli di ripresa) senza compromettere i dettagli discriminanti (es. barra vs croce).

Parametro	Valore	Descrizione
hsv_h	0.01	Variazione minima della tonalità (simula lievi cambiamenti di illuminazione).
hsv_s	0.5	Variazione moderata della saturazione (preserva i colori distintivi dei cartelli).
hsv_v	0.3	Variazione della luminosità (simula condizioni di luce diverse).
degrees	5.0	Rotazione massima di $\pm 5^\circ$ (simula angoli di ripresa reali).
translate	0.1	Traslazione fino al 10% dell'immagine (simula decentramento).
scale	0.5	Zoom tra $0.5 \times$ e $1.5 \times$ (simula distanze diverse).
fliplr	0.5	Flip orizzontale abilitato al 50% — <b>nota: potenzialmente problematico per cartelli direzionali</b> , ma mantenuto per aumentare la varietà.
flipud	0.0	Flip verticale disabilitato (non realistico per cartelli stradali).
mosaic	0.15	Mosaic abilitato al 15% (composizione di 4 immagini), ridotto per evitare confusione.
mixup	0.01	Mixup quasi disabilitato (1%), per non alterare i simboli.
perspective	0.0	Trasformazione prospettica disabilitata (evita distorsioni non realistiche).
shear	0.0	Shear disabilitato (preserva la geometria dei cartelli).

Tabella 6.2: Parametri di data augmentation effettivamente utilizzati.

## 6.2 Organizzazione della Directory per il Training

La directory in cui verrà eseguito il codice per il training deve seguire la seguente struttura gerarchica:

```
traffic_sign_detection_yolov10/
    requirements.txt
    dataset/traffic_sign_detection
        train/
            images/
            labels/
        val/
            images/
            labels/
        test/
            images/
            labels/
```

```
|   └── data.yaml
```

Tutti i file e le directory sopra elencati sono disponibili nel repository remoto del progetto e possono essere scaricati tramite `git clone` o download diretto.

## 6.3 Fase di Training

Per avviare la fase di training, è necessario posizionarsi nella directory principale del progetto utilizzando il terminale:

```
cd percorso/della/directory/traffic_sign_detection_yolov10
```

### 6.3.1 Installazione delle Dipendenze

Prima di eseguire qualsiasi operazione, è necessario installare le librerie richieste. Questo può essere fatto tramite il comando:

```
pip install -r requirements.txt
```

Il file `requirements.txt` contiene l'elenco completo delle dipendenze necessarie per l'esecuzione del progetto.

### 6.3.2 Esecuzione del Training

Il training dei modelli YOLOv10 (n, s, m) viene avviato tramite lo script nel notebook `training_traffic_sign_detection.ipynb`. Di seguito un esempio per il modello YOLOv10n:

```
1 torch.cuda.empty_cache()
2 gc.collect()
3
4 torch.backends.cuda.matmul.allow_tf32 = False
5 torch.backends.cudnn.allow_tf32 = False
6
7 MODEL_SIZE = "s"
8 model = YOLO("yolov10s.pt")
9
10 model.train(
11     data="/datasets/traffic_sign_detection/data.yaml",
12     epochs=100,
13     patience=20,
14     imgsz=640,
15     rect=False,
16     batch=20 if MODEL_SIZE == "m" else 32 if MODEL_SIZE == "s" else
17             56,
18     workers=2,
19     optimizer="AdamW",
20     lr0=0.001,
21     lrf=0.01,
22     weight_decay=0.0005,
23     warmup_epochs=3,
24     cos_lr=True,
25     hsv_h=0.01,
```

```

25     hsv_s=0.5,
26     hsv_v=0.3,
27     degrees=5.0,
28     translate=0.1,
29     scale=0.5,
30     fliplr=0.5,
31     flipud=0.0,
32     mosaic=0.15,
33     mixup=0.01,
34     perspective=0.0,
35     shear=0.0,
36     device=0,
37     amp=True,
38     cache=False,
39     compile=False,
40     save_period=0,
41     save=True,
42 )

```

Listing 6.1: Esempio di modello di training

Analogamente per gli altri due modelli:

```

1 torch.cuda.empty_cache()
2 gc.collect()
3
4 torch.backends.cuda.matmul.allow_tf32 = False
5 torch.backends.cudnn.allow_tf32 = False
6
7 MODEL_SIZE = "s"
8 model = YOLO("yolov10s.pt")
9
10 model.train(
11     data="/datasets/traffic_sign_detection/data.yaml",
12     epochs=100,
13     patience=20,
14     imgsz=640,
15     rect=False,
16     batch=20 if MODEL_SIZE == "m" else 32 if MODEL_SIZE == "s" else
17         56,
18     workers=2,
19     optimizer="AdamW",
20     lr0=0.001,
21     lrf=0.01,
22     weight_decay=0.0005,
23     warmup_epochs=3,
24     cos_lr=True,
25     hsv_h=0.01,
26     hsv_s=0.5,
27     hsv_v=0.3,
28     degrees=5.0,
29     translate=0.1,
30     scale=0.5,
31     fliplr=0.5,
32     flipud=0.0,
33     mosaic=0.15,
34     mixup=0.01,
35     perspective=0.0,

```

```

36     device=0,
37     amp=True,
38     cache=False,
39     compile=False,
40     save_period=0,
41     save=True,
42 )

```

Listing 6.2: Esempio di modello di training

```

1 torch.cuda.empty_cache()
2 gc.collect()
3
4 torch.backends.cuda.matmul.allow_tf32 = False
5 torch.backends.cudnn.allow_tf32 = False
6
7 MODEL_SIZE = "m"
8 model = YOLO("yolov10m.pt")
9
10 model.train(
11     data="/datasets/traffic_sign_detection/data.yaml",
12     epochs=100,
13     patience=20,
14     imgsz=640,
15     rect=False,
16     batch=20 if MODEL_SIZE == "m" else 32 if MODEL_SIZE == "s" else
17         56,
18     workers=2,
19     optimizer="AdamW",
20     lr0=0.001,
21     lrf=0.01,
22     weight_decay=0.0005,
23     warmup_epochs=3,
24     cos_lr=True,
25     hsv_h=0.01,
26     hsv_s=0.5,
27     hsv_v=0.3,
28     degrees=5.0,
29     translate=0.1,
30     scale=0.5,
31     fliplr=0.5,
32     flipud=0.0,
33     mosaic=0.15,
34     mixup=0.01,
35     perspective=0.0,
36     shear=0.0,
37     device=0,
38     amp=True,
39     cache=False,
40     compile=False,
41     save_period=0,
42     save=True,
43 )

```

Listing 6.3: Esempio di modello di training

Al termine di un training con Ultralytics YOLO, viene generata una cartella strutturata contenente tutti gli artefatti necessari per l'analisi, la validazione e il deployment

del modello. Di seguito è riportata la struttura tipica della directory di output (es. `runs/train/exp/`) e una descrizione sintetica di ogni componente.

- **`runs/train/exp/`** — Cartella principale del training (il suffisso `exp` viene incrementato automaticamente se `exist_ok=False`).
  - `args.yaml` — Parametri di training utilizzati (iperparametri, percorsi, ecc.).
  - `opt.yaml` — Parametri derivati e ottimizzati durante il training.
  - `results.csv` — Valori numerici delle metriche per ogni epoca (loss, mAP, precision, recall, ecc.).
  - `results.png` — Grafico combinato di tutte le metriche di training e validazione.
  - `confusion_matrix.png` — Matrice di confusione sul validation set.
  - `confusion_matrix_normalized.png` — Versione normalizzata (percentuale per classe).
  - `F1_curve.png`, `P_curve.png`, `R_curve.png`, `PR_curve.png` — Curve di F1, Precisione, Recall e Precision-Recall.
  - `labels.jpg` — Istogramma della distribuzione delle classi nel training set.
  - `val_batch0_pred.jpg`, `val_batch1_pred.jpg`, ... — Esempi di predizioni sul validation set.
  - `val_batch0_labels.jpg`, `val_batch1_labels.jpg`, ... — Esempi di ground truth corrispondenti.
  - **`weights/`** — Cartella contenente i checkpoint del modello:
    - \* `best.pt` — Pesi migliori (epoca con mAP@0.5 più alto).
    - \* `last.pt` — Pesi dell'ultima epoca completata.
    - \* `epoch10.pt`, `epoch20.pt`, ... — Checkpoint periodici (se `save_period=N` è impostato).

Tutti gli output sono generati automaticamente, per cui non è necessario scrivere codice aggiuntivo per ottenere questa ricchezza di informazioni.

Per comodità, nel repository sono già presenti le cartelle `TrainYOLOv10n/`, `TrainYOLOv10s/` e `TrainYOLOv10m/`, contenenti i risultati dei training già eseguiti dall'autore.

## 6.4 Fase di Validazione

La validazione viene eseguita su un test set raccolto autonomamente lungo le strade della provincia di Catania, Caltanissetta e Ragusa. Per questa fase, si utilizza un file di configurazione come nella fase di training , `data.yaml`, la cui struttura è la seguente:

```

1 path: datasets/TrafficSigns
2 train: train/images
3 val: val/images
4 test: test/images
5
6 names:
7   0: forb_overtake
8   1: for_overtake_trucks
9   2: forb_speed_limit_10
10  3: forb_speed_limit_20
11  (...)
```

Listing 6.4: Struttura del file di configurazione data.yaml per la validazione

#### 6.4.1 Validazione dei Modelli

La validazione del modello YOLOv10n avviene tramite lo script presente nel file `training_traffic_sign_detection.ipynb`, dopo lo script per training:

```

1 model = YOLO("./content/runs/detect/train/weights/best.pt")
2
3 val_resultsN =
4     model.val(data="datasets/traffic_sign_detection/data.yaml")
5
6 print("\n" + "="*50)
7 print("RISULTATI DELLA VALIDAZIONE")
8 print("="*50)
9 print(f"mAP50-95: {val_resultsN.box.map:.4f}")
10 print(f"mAP50: {val_resultsN.box.map50:.4f}")
11 print(f"mAP75: {val_resultsN.box.map75:.4f}")
12 print(f"Precisione : {np.mean(val_resultsN.box.p):.4f}")
13 print(f"Recall : {np.mean(val_resultsN.box.r):.4f}")
14 print(f"F1-Score : {np.mean(val_resultsN.box.f1):.4f}")
15
16 model = YOLO("./content/runs/detect/train/weights/best.pt")
17
18 test_resultsN =
19     model.val(data="datasets/traffic_sign_detection/data.yaml",
20               split="test")
21
22 print("\n" + "="*50)
23 print("RISULTATI DELLA VALIDAZIONE")
24 print("="*50)
25 print(f"mAP50-95: {test_resultsN.box.map:.4f}")
26 print(f"mAP50: {test_resultsN.box.map50:.4f}")
27 print(f"mAP75: {test_resultsN.box.map75:.4f}")
28 print(f"Precisione : {np.mean(test_resultsN.box.p):.4f}")
29 print(f"Recall : {np.mean(test_resultsN.box.r):.4f}")
30 print(f"F1-Score : {np.mean(test_resultsN.box.f1):.4f}")
```

Listing 6.5: Script del file training traffic sign detection.ipynb per la validazione

L'output a terminale include:

- Numero di occorrenze per classe
- Box Precision, Recall, mAP50, mAP75, mAP50-95

- Precisione, Recall e F1-score medi su tutte le classi

Analogamente, per i modelli YOLOv10s e YOLOv10m:

```

1 model = YOLO("./content/runs/detect/train2/weights/best.pt")
2
3 val_resultsN =
4     model.val(data="datasets/traffic_sign_detection/data.yaml")
5
6 print("\n" + "*50)
7 print("RISULTATI DELLA VALIDAZIONE")
8 print("*50)
9 print(f"mAP50-95: {val_resultsN.box.map:.4f}")
10 print(f"mAP50: {val_resultsN.box.map50:.4f}")
11 print(f"mAP75: {val_resultsN.box.map75:.4f}")
12 print(f"Precisione : {np.mean(val_resultsN.box.p):.4f}")
13 print(f"Recall : {np.mean(val_resultsN.box.r):.4f}")
14 print(f"F1-Score : {np.mean(val_resultsN.box.f1):.4f}")
15
16 model = YOLO("./content/runs/detect/train2/weights/best.pt")
17
18 test_resultsN =
19     model.val(data="datasets/traffic_sign_detection/data.yaml",
20               split="test")
21
22 print("\n" + "*50)
23 print("RISULTATI DELLA VALIDAZIONE")
24 print("*50)
25 print(f"mAP50-95: {test_resultsN.box.map:.4f}")
26 print(f"mAP50: {test_resultsN.box.map50:.4f}")
27 print(f"mAP75: {test_resultsN.box.map75:.4f}")
28 print(f"Precisione : {np.mean(test_resultsN.box.p):.4f}")
29 print(f"Recall : {np.mean(test_resultsN.box.r):.4f}")
30 print(f"F1-Score : {np.mean(test_resultsN.box.f1):.4f}")
31
32 model = YOLO("./content/runs/detect/train3/weights/best.pt")
33
34 val_resultsN =
35     model.val(data="datasets/traffic_sign_detection/data.yaml")
36
37 print("\n" + "*50)
38 print("RISULTATI DELLA VALIDAZIONE")
39 print("*50)
40 print(f"mAP50-95: {val_resultsN.box.map:.4f}")
41 print(f"mAP50: {val_resultsN.box.map50:.4f}")
42 print(f"mAP75: {val_resultsN.box.map75:.4f}")
43 print(f"Precisione : {np.mean(val_resultsN.box.p):.4f}")
44 print(f"Recall : {np.mean(val_resultsN.box.r):.4f}")
45 print(f"F1-Score : {np.mean(val_resultsN.box.f1):.4f}")
46
47 model = YOLO("./content/runs/detect/train3/weights/best.pt")
48
49 test_resultsN =
50     model.val(data="datasets/traffic_sign_detection/data.yaml",
51               split="test")
52
53 print("\n" + "*50)
54 print("RISULTATI DELLA VALIDAZIONE")
55 print("*50)

```

```

50 print(f"mAP50-95:      {test_resultsN.box.map:.4f}")
51 print(f"mAP50:        {test_resultsN.box.map50:.4f}")
52 print(f"mAP75:        {test_resultsN.box.map75:.4f}")
53 print(f"Precisione    :  {np.mean(test_resultsN.box.p):.4f}")
54 print(f"Recall         :  {np.mean(test_resultsN.box.r):.4f}")
55 print(f"F1-Score       :  {np.mean(test_resultsN.box.f1):.4f}")

```

Listing 6.6: Script di validazione per i modelli YOLOv10s e Yolov10m

#### 6.4.2 Confronto Grafico tra Modelli

Per confrontare le prestazioni dei modelli, è possibile generare grafici a barre utilizzando lo script `training_traffic_sign_detection.ipynb`. Di seguito un estratto del codice:

```

1 modelli = ['YOLOv10n', 'YOLOv10s', 'YOLOv10m']
2 metriche = ['mAP50-95', 'mAP50', 'mAP75']
3
4 valori = np.array([
5     [test_resultsN.box.map,    test_resultsS.box.map,
6      test_resultsM.box.map],
7     [test_resultsN.box.map50,   test_resultsS.box.map50,
8      test_resultsM.box.map50],
9     [test_resultsN.box.map75,   test_resultsS.box.map75,
10    test_resultsM.box.map75]
11 ])
12
13 n_metriche = len(metriche)
14 n_modelli = len(modelli)
15 larghezza_barra = 0.2
16 spazio_tra_gruppi = 0.3
17 posizioni_base = np.arange(n_metriche) * (n_modelli *
18     larghezza_barra + spazio_tra_gruppi)
19
20 fig, ax = plt.subplots(figsize=(10, 6))
21
22 for idx_modello, nome_modello in enumerate(modelli):
23     offset = idx_modello * larghezza_barra
24     ax.bar(
25         posizioni_base + offset,
26         valori[:, idx_modello],
27         larghezza_barra,
28         label=nome_modello,
29         edgecolor='white',
30         linewidth=0.7
31     )
32
33 ax.set_xlabel('Metrica di Valutazione', fontsize=12)
34 ax.set_ylabel('Valore', fontsize=12)
35 ax.set_title('Confronto Prestazioni: mAP tra YOLOv10n, YOLOv10s e
36 YOLOv10m', fontsize=14, fontweight='bold')
37 ax.set_xticks(posizioni_base + (n_modelli - 1) * larghezza_barra / 2)
38 ax.set_xticklabels(metriche, fontsize=11)
39 ax.legend(title="Architetture", fontsize=11, title_fontsize=12)
40 ax.grid(axis='y', linestyle='--', alpha=0.7)
41
42 plt.tight_layout()
43 plt.show()

```

```
39 plt.close()
```

Listing 6.7: Script del codice per il confronto tra modelli

Un secondo grafico confronta Precision, Recall e F1-score medi:

```
1 modelli = ['YOLOv10n', 'YOLOv10s', 'YOLOv10m']
2 metriche = ['Precisione Media', 'Recall Media', 'F1-Score Medio']
3
4 valori = np.array([
5     [np.mean(test_resultsN.box.p), np.mean(test_resultsS.box.p),
6         np.mean(test_resultsM.box.p)],
7     [np.mean(test_resultsN.box.r), np.mean(test_resultsS.box.r),
8         np.mean(test_resultsM.box.r)],
9     [np.mean(test_resultsN.box.f1), np.mean(test_resultsS.box.f1),
10        np.mean(test_resultsM.box.f1)]])
11
12 n_metriche = len(metriche)
13 n_modelli = len(modelli)
14 larghezza_barra = 0.2
15 spazio_tra_gruppi = 0.3
16 posizioni_base = np.arange(n_metriche) * (n_modelli *
17     larghezza_barra + spazio_tra_gruppi)
18
19 fig, ax = plt.subplots(figsize=(10, 6))
20
21 for idx, modello in enumerate(modelli):
22     offset = idx * larghezza_barra
23     ax.bar(
24         posizioni_base + offset,
25         valori[:, idx],
26         larghezza_barra,
27         label=modello,
28         edgecolor='white',
29         linewidth=0.7
30     )
31
32 ax.set_xlabel('Metrica', fontsize=12)
33 ax.set_ylabel('Valore Medio', fontsize=12)
34 ax.set_title('Confronto: Precisione, Recall e F1-Score tra YOLOv10n,'
35               ' YOLOv10s e YOLOv10m',
36               fontsize=14, fontweight='bold', pad=20)
37 ax.set_xticks(posizioni_base + (n_modelli - 1) * larghezza_barra / 2)
38 ax.set_xticklabels(metriche, fontsize=11)
39 ax.legend(title="Modello", fontsize=11, title_fontsize=12)
40 ax.grid(axis='y', linestyle='--', alpha=0.7)
41
42 plt.tight_layout()
43 plt.show()
44 plt.close()
```

Listing 6.8: Script per la generazione del grafico di confronto tra valori medi di Precision-Recall-F1 score

## 6.5 Fase di Testing

Terminata la validazione, è possibile testare i modelli su nuove immagini. Ciò può essere fatto in due modi:

1. Tramite l'applicazione `Demo` inclusa nel progetto (vedi Sezione 6.5.3).
2. Eseguendo lo script di testing presente nel file:  
`Training_Traffic_Sign_Detection.ipynb`.

### 6.5.1 Testing con Codice

Di seguito un esempio di codice per eseguire l'inferenza su un'immagine singola:

```
1 #Modificare il commento sulla base del modello che si vuole testare
2 model = YOLO("./content/runs/detect/train/weights/best.pt")
3 #model = YOLO("./content/runs/detect/train2/weights/best.pt")
4 #model = YOLO("./content/runs/detect/train3/weights/best.pt")
5
6 image_paths = [
7     "images/image.jpeg",
8     "images/image2.jpeg",
9 ]
10
11 for img_path in image_paths:
12     img_original = cv2.imread(img_path)
13     img_original_rgb = cv2.cvtColor(img_original, cv2.COLOR_BGR2RGB)
14
15     results = model(img_path)
16
17     for r in results:
18         img_with_boxes = r.plot()
19
20         fig, axes = plt.subplots(1, 2, figsize=(14, 6))
21
22         axes[0].imshow(img_original_rgb)
23         axes[0].set_title("Originale", fontsize=13,
24                           fontweight='bold')
24         axes[0].axis('off')
25
26         axes[1].imshow(img_with_boxes)
27         axes[1].set_title("Rilevamenti YOLOv10", fontsize=13,
28                           fontweight='bold')
28         axes[1].axis('off')
29
30         plt.suptitle(f"Confronto: {img_path.split('/')[-1]}",
31                      fontsize=14, y=0.95)
32         plt.tight_layout()
32         plt.show()
```

Listing 6.9: Esempio di script per l'esecuzione dell'inferenza legata al testing del modello

### 6.5.2 Testing manuale su singole immagini

È possibile testare i modelli su nuove immagini tramite uno script Python che:

- Carica un'immagine
- Esegue l'inferenza
- Visualizza side-by-side: immagine originale e predizioni con confidenza



Figura 6.1: Esempio di confronto tra immagine originale e predizioni del modello.

### 6.5.3 Applicazione Demo

È stata sviluppata un'applicazione GUI in Python con le seguenti funzionalità:

- **Caricamento di un'immagine** da file
- **Selezione del modello** da utilizzare (YOLOv10n/s/m)
- **Visualizzazione simultanea** di:
  - Immagine originale
  - Immagine con ground truth
  - Predizioni con bounding box e confidenza

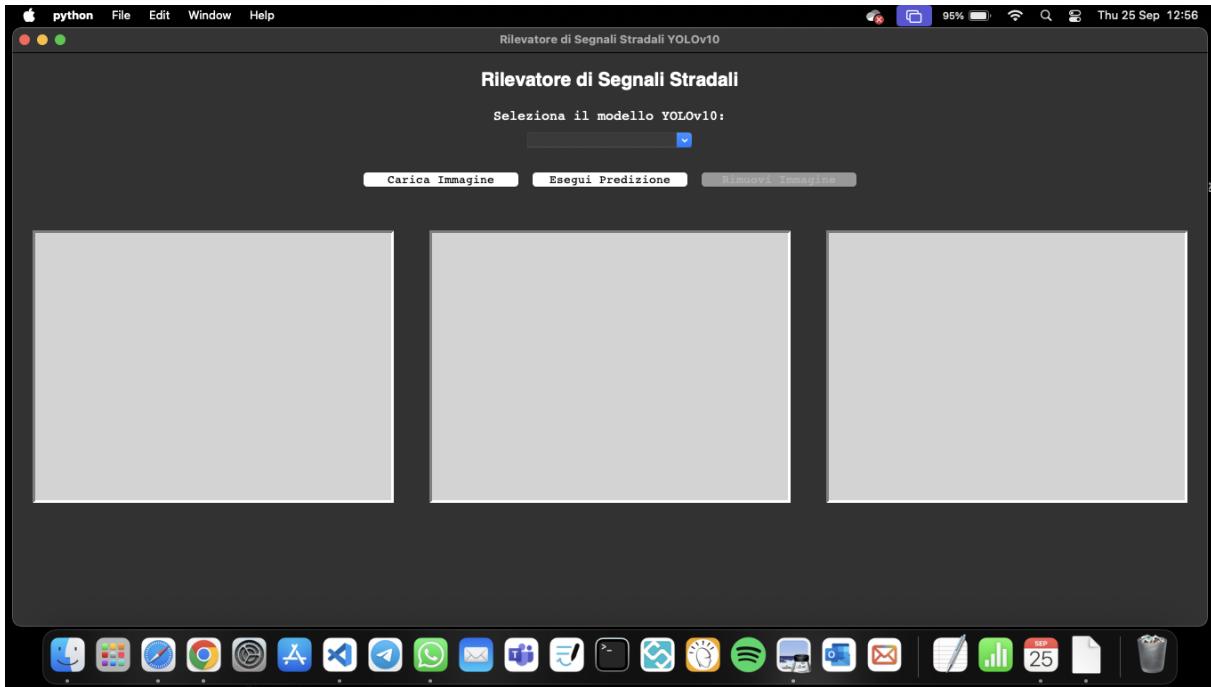


Figura 6.2: Interfaccia dell'app prima della predizione.

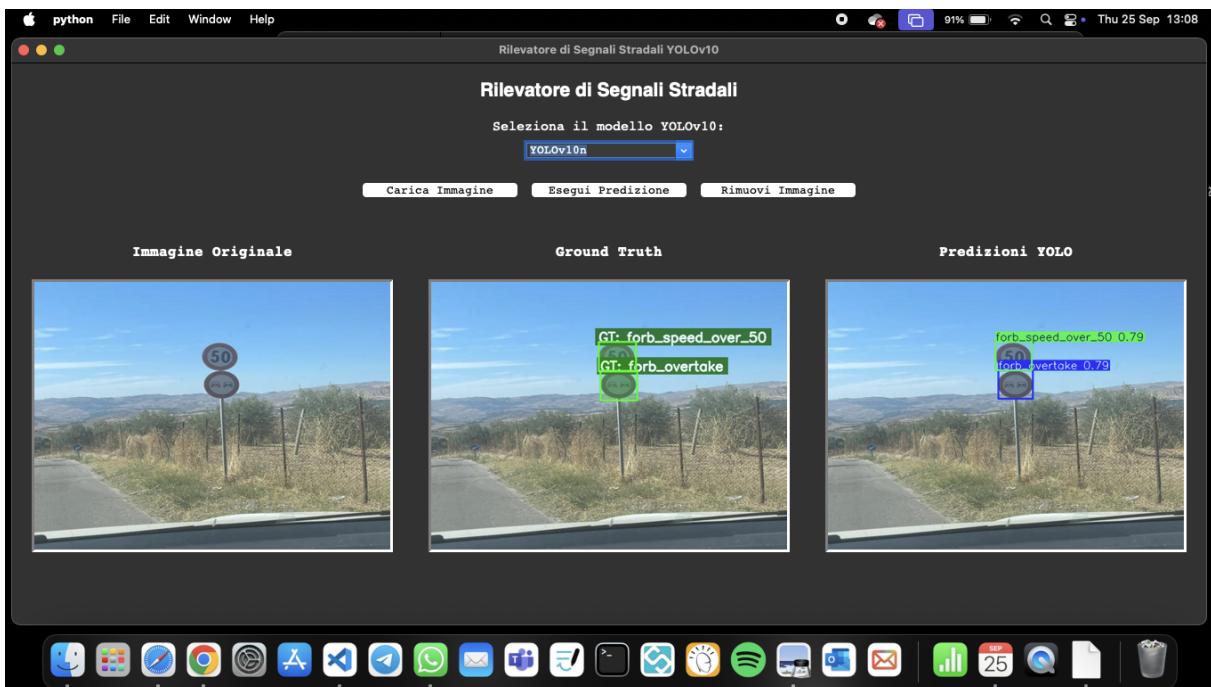


Figura 6.3: Interfaccia dell'app dopo la predizione.

#### 6.5.4 Preparazione delle Immagini per l'App

Per utilizzare l'app, le immagini devono essere organizzate come segue:

```
datasets/  
└── images/  
    └── image1.jpg
```

```
|   └── image2.jpg  
|   └── image1.txt  
|   └── image2.txt
```

**Nota:** Per visualizzare correttamente il ground truth è necessario che esso si trovi all'interno della stessa cartella dell'immagine e che abbia lo stesso nome dell'immagine.

### 6.5.5 Avvio dell'Applicazione

Per avviare l'app, eseguire da terminale:

```
python TrafficSignsDetectionApp.py
```

1. Cliccare su “**Carica Immagine**” per selezionare un'immagine.
2. Scegliere il modello da usare tramite la listbox “**Seleziona il modello YOLOv10**”.
3. In caso di errore, rimuovere l'immagine con “**Rimuovi Immagine**”.
4. Avviare la predizione con “**Esegui Predizione**”.

Un video dimostrativo del funzionamento è disponibile nella cartella `assets/` del repository.

# Capitolo 7

## Risultati Sperimentali

In questo capitolo vengono presentati e discussi i risultati ottenuti dal fine-tuning dei modelli **YOLOv10n**, **YOLOv10s** e **YOLOv10m** sul dataset finale utilizzato per training e validazione e la loro valutazione sul **dataset di test acquisito autonomamente** (strade della provincia di Catani, Caltanissetta e Ragusa). L'obiettivo è confrontare le prestazioni dei tre modelli in termini di accuratezza, velocità e capacità di generalizzazione.

Tutti gli esperimenti sono stati condotti su **Google Colaboratory**, sfruttando una GPU **NVIDIA L4 da 24 GB di VRAM**, per garantire un training efficiente e riproducibile.

### 7.1 Tabelle Riassuntive

Di seguito presentiamo i risultati del fine-tuning dei modelli YOLOv10 sul dataset, ottenuti utilizzando i parametri di training descritti in precedenza.

#### 7.1.1 Architettura e Tempi di Training

Modello	# Layer	Tempo Training (h)	Velocità Inferenza (ms)
YOLOv10n	102	2.236 (ep. -)	1.5
YOLOv10s	106	3.060 (ep. 91)	3.1
YOLOv10m	136	5.566 (ep. -)	5.9

Tabella 7.1: Informazioni strutturali e temporali dei modelli dopo il fine-tuning. I valori tra parentesi indicano l'epoca in cui è scattato l'early stopping.

Il training è stato eseguito per un massimo di 100 epoche. L'**early stopping** è stato attivato **solo per il modello YOLOv10s**, il cui training si è interrotto all'epoca 91, su un massimo di 100 epoche previste. Per i modelli YOLOv10n e YOLOv10m, invece, non si è verificata alcuna interruzione anticipata, come indicato dall'assenza di un numero di epoca tra parentesi nella Tabella 7.1; il training ha quindi proseguito per l'intera durata delle 100 epoche.

Questo comportamento differenziato suggerisce che:

- Il modello YOLOv10s ha raggiunto il picco di performance sul validation set già in fase avanzata del training, attivando efficacemente l'early stopping e prevenendo potenziale overfitting.

- I modelli YOLOv10n (più leggero) e YOLOv10m (più profondo) non hanno mostrato un chiaro degrado o plateau delle metriche di validation entro le 100 epoche, oppure il loro miglioramento è proseguito fino alla fine del ciclo di training.
- Il dataset finale, pur ampio, presenta una complessità contenuta (numero ridotto di classi, contesti visivi relativamente omogenei), che favorisce una convergenza stabile e rapida, specialmente nei modelli di media capacità come YOLOv10s.

Nonostante la differenza nel comportamento dell'early stopping, tutti e tre i modelli hanno tempi di training molto contenuti (inferiori a 6 ore), a dimostrazione dell'efficienza del processo di fine-tuning e dell'adeguatezza delle architetture al compito di rilevamento dei segnali stradali.

### 7.1.2 Metriche Medie sul Validation Set

Modello	Precision (%)	Recall (%)	F1-score (%)
YOLOv10n	78.40	86.28	81.61
YOLOv10s	80.44	86.33	82.68
YOLOv10m	78.57	88.68	82.97

Tabella 7.2: Metriche medie sul validation set.

Modello	mAP@50 (%)	mAP@50:95 (%)
YOLOv10n	83.91	71.17
YOLOv10s	85.60	0.72.34
YOLOv10m	84.92	72.30

Tabella 7.3: mAP medio sul validation set.

#### Osservazioni:

- Tutti e tre i modelli raggiungono prestazioni elevate sul validation set, con F1-score compreso tra **81.6%** e **83.0%**, a dimostrazione dell'efficacia del fine-tuning sul dataset GTSDB.
- **YOLOv10n**, nonostante sia l'architettura più leggera, ottiene risultati sorprendentemente solidi (F1-score: 81.61%, mAP@50: 83.91%), rendendolo una scelta interessante per applicazioni con vincoli di risorse.
- **YOLOv10s** si distingue per il miglior compromesso tra accuratezza e complessità: raggiunge il **mAP@50 più alto (85.60%)** e un F1-score di **82.68%**, confermando la sua efficienza nel bilanciare velocità e precisione.
- **YOLOv10m**, pur avendo la recall più elevata (88.68%) e il F1-score leggermente migliore (82.97%), mostra un mAP@50 (84.92%) leggermente inferiore a quello di YOLOv10s, suggerendo che la maggiore capacità del modello non si traduce in un netto vantaggio in termini di localizzazione e classificazione complessiva.

### 7.1.3 Metriche Medie sul Test Set

Modello	Avg. Precision (%)	Avg. Recall (%)	Avg. F1-score (%)
YOLOv10n	56.61	67.31	56.88
YOLOv10s	66.54	70.63	64.53
YOLOv10m	74.16	61.79	60.43

Tabella 7.4: Metriche medie sul test set.

Modello	mAP@50 (%)	mAP@75 (%)	mAP@50:95 (%)
YOLOv10n	69.39	66.77	57.39
YOLOv10s	73.09	71.59	62.20
YOLOv10m	70.31	69.71	59.82

Tabella 7.5: mAP medio sul test set.

Sebbene le metriche sul test set risultino inferiori rispetto a quelle ottenute in validazione, i modelli dimostrano comunque una **capacità significativa di rilevare segnali stradali in condizioni reali**, nonostante le sfide rappresentate da variazioni di illuminazione, angolazioni estreme, sfondi complessi e differenze grafiche rispetto al dataset di addestramento. Questo calo è in linea con quanto atteso quando si passa da dati controllati a scenari *in the wild*, e non compromette la validità dell'approccio adottato.

Alcuni fattori che spiegano il gap tra validation e test includono:

1. **Differenze di distribuzione:** il test set include immagini reali con maggiore variabilità visiva rispetto al dataset di addestramento, che contiene principalmente segnali ben centrati e ben illuminati.
2. **Stili grafici non visti:** alcuni segnali nel test set presentano simboli, colori o condizioni (es. usura, sporco, parziale occlusione) non ampiamente rappresentati in fase di training.
3. **Valutazione realistica:** il test set fornisce una stima più affidabile delle prestazioni in contesti operativi, rispetto alla validazione interna al dataset di addestramento.

In questo contesto, **YOLOv10s emerge come la soluzione più equilibrata e robusta:** con un **F1-score del 64.53 %** e un **mAP@50 del 73.09 %**, riesce a mantenere un buon compromesso tra precisione e recall, dimostrando una notevole capacità di adattamento a condizioni non viste.

Anche **YOLOv10n**, pur essendo il modello più leggero, raggiunge una **recall del 67.31 %**, segno che è in grado di individuare la maggior parte dei segnali presenti. Infine, **YOLOv10m** mostra la **precisione più alta (74.16 %)**, indicando una bassa incidenza di falsi positivi, un aspetto cruciale in contesti di sicurezza. Tale comportamento suggerisce che, con un addestramento mirato a migliorare la copertura (ad esempio tramite data augmentation o fine-tuning su dati reali), potrebbe raggiungere performance ancora più elevate.

#### 7.1.4 Conclusioni per classe (test set)

Le classi con **simboli distintivi e ben rappresentate nel dataset** — come `prio_stop`, `prio_give_way` e `info_crosswalk` — raggiungono **precisione e recall elevate**. Nei modelli più grandi (YOLOv10s e YOLOv10m), queste classi mostrano entrambe le metriche superiori a 0.80, con F1-score che supera 0.83 per `prio_stop` e `prio_give_way`.

- Le classi **poco frequenti o con pattern visivi ambigui** presentano prestazioni significativamente inferiori. Esempi critici includono:

- `forb_waiting` (F1  $\approx$  0.20 in tutti i modelli): precisione molto bassa (0.11) ma recall moderata (0.50), indicativa di numerosi falsi positivi.
- `warn_right_curve` (F1  $\approx$  0.30): bassa precisione (0.25) e recall limitata (0.40).
- `forb_overtake_trucks`: nei modelli YOLOv10n e YOLOv10s l'F1-score è inferiore a 0.30, ma **migliora drasticamente in YOLOv10m** (F1 = 0.686), dimostrando che architetture più espressive possono recuperare classi difficili.

Nonostante queste limitazioni, le prestazioni complessive rimangono **accettabili per un sistema di prototipazione**, soprattutto con il modello YOLOv10m.

- Per alcune classi si osserva una **marcata asimmetria tra precisione e recall**, anche sul validation set. Ad esempio:
  - `forb_stopping`: precisione molto alta (0.91) ma recall molto bassa (0.29), segno che il modello rileva pochi veri positivi ed evita falsi allarmi.
  - `forb_speed_over_60` e `forb_speed_over_80`: recall elevata ( $> 0.70$ ) ma precisione bassa ( $< 0.43$ ), indicativa di una tendenza a generare falsi positivi.

Classe	P %	Recall %	F1 %	mAP50 %	mAP50-95 %
forb_overtake	0.736	0.731	0.733	0.8	0.626
forb_overtake_trucks	0.741	0.17	0.288	0.443	0.372
forb_speed_over_10	0.653	0.149	0.249	0.554	0.48
forb_speed_over_100	0.49	0.833	0.619	0.83	0.71
forb_speed_over_120	0.508	1.0	0.674	0.995	0.796
forb_speed_over_20	0.84	0.528	0.647	0.719	0.595
forb_speed_over_30	0.5	0.923	0.658	0.881	0.74
forb_speed_over_40	0.708	0.808	0.755	0.846	0.694
forb_speed_over_50	0.645	0.778	0.706	0.774	0.628
forb_speed_over_60	0.381	0.714	0.506	0.797	0.638
forb_speed_over_70	0.618	0.86	0.723	0.831	0.664
forb_speed_over_80	0.423	0.906	0.587	0.8	0.62
forb_stopping	0.908	0.292	0.449	0.773	0.626
forb_waiting	0.113	0.5	0.195	0.401	0.399
info_crosswalk	0.714	0.84	0.772	0.743	0.621
prio_give_way	0.513	0.861	0.647	0.78	0.668
prio_stop	0.614	0.857	0.719	0.857	0.702
warn_crosswalk	0.522	1.0	0.687	0.807	0.626
warn_double_curve	0.586	0.6	0.591	0.534	0.505
warn_left_curve	0.594	0.286	0.384	0.643	0.512
warn_right_curve	0.245	0.4	0.301	0.165	0.112

Tabella 7.6: Precision, recall, F1 e mAP per le classi sul validation set - modello YOLOv10n

Classe	P %	Recall %	F1 %	mAP50 %	mAP50-95 %
forb_overtake	0.789	0.622	0.694	0.781	0.658
forb_overtake_trucks	0.711	0.149	0.249	0.525	0.449
forb_speed_over_10	0.653	0.149	0.249	0.554	0.48
forb_speed_over_100	0.49	0.833	0.619	0.83	0.71
forb_speed_over_120	0.508	1.0	0.674	0.995	0.796
forb_speed_over_20	0.84	0.528	0.647	0.719	0.595
forb_speed_over_30	0.5	0.923	0.658	0.881	0.74
forb_speed_over_40	0.708	0.808	0.755	0.846	0.694
forb_speed_over_50	0.645	0.778	0.706	0.774	0.628
forb_speed_over_60	0.381	0.714	0.506	0.797	0.638
forb_speed_over_70	0.618	0.86	0.723	0.831	0.664
forb_speed_over_80	0.423	0.906	0.587	0.8	0.62
forb_stopping	0.908	0.292	0.449	0.773	0.626
forb_waiting	0.113	0.5	0.195	0.401	0.399
info_crosswalk	0.714	0.84	0.772	0.743	0.621
prio_give_way	0.829	0.889	0.858	0.862	0.738
prio_stop	0.763	0.923	0.835	0.857	0.771
warn_crosswalk	0.522	1.0	0.687	0.807	0.626
warn_double_curve	0.586	0.6	0.591	0.534	0.505
warn_left_curve	0.594	0.286	0.384	0.643	0.512
warn_right_curve	0.245	0.4	0.301	0.165	0.112

Tabella 7.7: Precision, recall, F1 e mAP per le classi sul validation set - modello YOLOv10s

Classe	P %	Recall %	F1 %	mAP50 %	mAP50-95 %
forb_overtake	0.852	0.618	0.718	0.783	0.598
forb_overtake_trucks	0.852	0.568	0.686	0.809	0.665
forb_speed_over_10	0.653	0.149	0.249	0.554	0.48
forb_speed_over_100	0.49	0.833	0.619	0.83	0.71
forb_speed_over_120	0.508	1.0	0.674	0.995	0.796
forb_speed_over_20	0.84	0.528	0.647	0.719	0.595
forb_speed_over_30	0.5	0.923	0.658	0.881	0.74
forb_speed_over_40	0.708	0.808	0.755	0.846	0.694
forb_speed_over_50	0.645	0.778	0.706	0.774	0.628
forb_speed_over_60	0.381	0.714	0.506	0.797	0.638
forb_speed_over_70	0.618	0.86	0.723	0.831	0.664
forb_speed_over_80	0.423	0.906	0.587	0.8	0.62
forb_stopping	0.908	0.292	0.449	0.773	0.626
forb_waiting	0.113	0.5	0.195	0.401	0.399
info_crosswalk	0.714	0.84	0.772	0.743	0.621
prio_give_way	0.769	0.889	0.825	0.843	0.732
prio_stop	0.763	0.923	0.835	0.857	0.771
warn_crosswalk	0.522	1.0	0.687	0.807	0.626
warn_double_curve	0.586	0.6	0.591	0.534	0.505
warn_left_curve	0.594	0.286	0.384	0.643	0.512
warn_right_curve	0.245	0.4	0.301	0.165	0.112

Tabella 7.8: Precision, recall, F1 e mAP per le classi sul validation set - modello YOLOv10m

## 7.2 Analisi di Loss, Matrici di Confusione e Curve di Valutazione

Per valutare in modo completo le prestazioni dei modelli YOLOv10n, YOLOv10s e YOLOv10m, sono stati analizzati gli andamenti delle funzioni di loss durante l’addestramento, le matrici di confusione finali e le curve di valutazione derivate dal test set. Questi strumenti permettono di comprendere non solo l’accuratezza complessiva dei modelli, ma anche i loro punti di forza, le debolezze e i comportamenti in condizioni operative diverse, ad esempio, al variare della soglia di confidenza. Di seguito si riporta un’analisi dettagliata di ciascun componente.

### 7.2.1 Matrici di Confusione

Le matrici di confusione offrono un’analisi dettagliata e quantitativa delle performance dei modelli a livello di classe, mettendo in luce sia i punti di forza che le aree in cui è possibile intervenire per migliorare la robustezza del sistema. Osservando le tre matrici; corrispondenti ai modelli YOLOv10n, YOLOv10s e YOLOv10m; si notano tendenze coerenti con i risultati già discussi nelle metriche globali.

- **Diagonale principale (predizioni corrette):** I valori sulla diagonale sono generalmente elevati, soprattutto per le classi più frequenti e distintive (es.

`forb_speed_over_30`, `prio_give_way`, `info_crosswalk`), confermando che i modelli riescono a discriminare efficacemente le classi più rappresentate. In particolare, YOLOv10s e YOLOv10m mostrano numeri di predizioni corrette superiori a 180 per alcune classi chiave, indicando un'ottima affidabilità in contesti operativi.

- **Confusioni tra classi:** Gli errori fuori diagonale sono prevalentemente concentrati tra classi con similitudini visive o semantiche, ad esempio:
  - Tra limiti di velocità consecutivi (es. `forb_speed_over_50` vs `forb_speed_over_60`): queste confusioni sono comprensibili e spesso non critiche in applicazioni reali, poiché il significato generale del segnale rimane simile.
  - Tra segnali di avvertimento curva (`warn_left_curve`, `warn_right_curve`) e altri segnali di avvertimento: suggeriscono che il modello potrebbe beneficiare di una maggiore attenzione ai dettagli locali (es. direzione della freccia).

Queste ambiguità non indicano un fallimento, ma piuttosto opportunità per un fine-tuning mirato o l'uso di tecniche di post-processing (es. regole logiche basate sul contesto stradale).

- **Classe “background”:** Il numero di predizioni nella riga “background” (falsi negativi) è relativamente contenuto, ad esempio, solo 82-88 casi su migliaia di predizioni totali, indicando che i modelli hanno una buona sensibilità e rilevano la maggior parte degli oggetti presenti. Tuttavia, alcuni segnali rari (es. `forb_speed_over_10`, `forb_stopping`) mostrano un numero leggermente più elevato di falsi negativi, suggerendo che potrebbero beneficiare di un aumento della loro frequenza nel training set o di data augmentation specifica.

#### Confronto tra modelli:

- **YOLOv10n:** Mostra una diagonale meno brillante rispetto agli altri, con alcuni valori sotto i 100 (es. `forb_speed_over_40 = 92`). Le confusioni sono più diffuse, specialmente tra classi simili, ma il numero di falsi negativi (riga background) è comunque basso.
- **YOLOv10s:** Presenta la diagonale più robusta, con numerosi valori sopra i 180 e poche confusioni significative. È il modello più bilanciato, con un'elevata capacità di riconoscimento senza eccessivo aumento dei falsi positivi.
- **YOLOv10m:** Conferma la sua superiorità in termini di accuratezza, con valori massimi sulla diagonale (es. `prio_give_way = 189`, `info_crosswalk = 187`) e un numero ridotto di errori fuori diagonale. La sua maggiore capacità espressiva permette di affrontare meglio le ambiguità visive.

### 7.2.2 Curve di Valutazione al Variare della Soglia di Confidenza

Le curve di precisione, richiamo e F1-score al variare della soglia di confidenza offrono una visione dinamica del comportamento del modello, consentendo di scegliere la configurazione ottimale in base alle esigenze applicative (es. massima precisione vs massima copertura).

## Precision-Confidence Curve

Le curve di precisione in funzione della confidenza offrono una visione chiara del comportamento dei modelli quando si regola la soglia di accettazione delle predizioni. Aumentando la soglia di confidenza, il modello diventa più selettivo: accetta solo le predizioni con elevata certezza, riducendo drasticamente i falsi positivi.

Osservando le tre curve (corrispondenti ai modelli YOLOv10n, YOLOv10s e YOLOv10m), si nota un andamento coerente e promettente:

- **Comportamento generale:** Per tutte e tre le architetture, la precisione aumenta progressivamente all'aumentare della confidenza, raggiungendo valori prossimi a 1.0 per soglie molto elevate. Questo indica che i modelli sono in grado di produrre predizioni estremamente affidabili quando operano in modalità conservativa.
- **Stabilità delle classi individuali:** Le numerose linee grigie rappresentano le curve di precisione per ciascuna classe. La maggior parte di esse segue l'andamento medio (linea blu) senza deviazioni drammatiche, suggerendo che la performance è relativamente uniforme tra le diverse categorie di segnali. Un risultato importante, considerando la varietà di forme, dimensioni e contesti visivi presenti nel dataset.
- **Performance finale:** La legenda riporta che per tutte e tre le curve, la precisione media su tutte le classi raggiunge il valore massimo (1.00 o 0.99) alla soglia di confidenza massima (1.000). Ciò conferma che, anche nelle condizioni più restrittive, i modelli mantengono un'elevata affidabilità, riuscendo a eliminare quasi completamente gli errori quando selezionano solo le predizioni più sicure.
- **Differenze tra modelli:** Sebbene non siano esplicitamente etichettate, le curve mostrano una tendenza: il modello più grande (YOLOv10m) presenta una curva blu leggermente più alta e stabile rispetto agli altri, indicando una maggiore robustezza nell'emettere predizioni corrette anche a soglie intermedie. Tuttavia, tutti e tre i modelli dimostrano un comportamento solido, adatto a scenari in cui si richiede un controllo preciso del trade-off tra completezza e accuratezza.

## Recall-Confidence Curve

La curva di richiamo in funzione della confidenza illustra il comportamento del modello nel rilevare tutti gli oggetti presenti, al variare della soglia di accettazione delle predizioni. Come previsto, all'aumentare della soglia di confidenza, il richiamo diminuisce: il modello diventa più selettivo, eliminando le predizioni incerte, un comportamento desiderabile per ridurre i falsi positivi, ma che comporta inevitabilmente la perdita di alcuni veri positivi.

Tuttavia, le curve mostrano un andamento molto promettente:

- **Richiamo iniziale elevato:** A bassa confidenza (soglia 0.0), il richiamo medio su tutte le classi è prossimo al 96–97% (come indicato nella legenda), dimostrando che i modelli sono in grado di rilevare la quasi totalità degli oggetti presenti nel dataset. Questo indica una buona sensibilità e copertura, anche per segnali difficili o poco rappresentati.
- **Decadimento controllato:** La discesa del richiamo con l'aumento della confidenza è graduale e prevedibile, senza cadute brusche. Ciò suggerisce che il modello non “perde” improvvisamente intere classi, ma seleziona in modo coerente le predizioni

più affidabili, mantenendo comunque un livello di completezza accettabile anche a soglie elevate.

- **Stabilità tra classi:** Le numerose linee grigie (una per classe) seguono un andamento simile alla curva media (linea blu), senza deviazioni estreme. Questo conferma che la capacità di rilevamento è relativamente uniforme tra le diverse categorie di segnali, un risultato importante per applicazioni reali dove non si può permettere che alcune classi siano sistematicamente ignorate.
- **Trade-off gestibile:** Il classico compromesso tra recall e precisione è ben visibile, ma non critico. Per esempio, impostando una soglia di confidenza intorno a 0.5–0.6, si ottiene un equilibrio ragionevole tra accuratezza e completezza. Ideale per scenari operativi in cui è necessario minimizzare sia i falsi positivi che i falsi negativi.

## F1-Confidence Curve

La curva F1, che bilancia precisione e richiamo, rappresenta il punto di equilibrio ideale per la scelta della soglia di confidenza del modello. Un valore elevato di F1 indica un buon compromesso tra la capacità di rilevare correttamente gli oggetti (recall) e l'affidabilità delle predizioni (precisione), rendendola una metrica cruciale per applicazioni reali.

Le tre curve mostrano un andamento coerente e promettente:

- **Performance eccellente:** Tutti e tre i modelli raggiungono valori massimi di F1 superiori all'80% un risultato notevole, soprattutto considerando la complessità del dataset di addestramento e le condizioni reali del test set. Il modello YOLOv10m si distingue con un F1 massimo di **0.83** a una soglia di confidenza molto bassa (0.263), indicando che è in grado di produrre predizioni altamente affidabili anche quando operativo in modalità sensibile.
- **Soglie ottimali ben posizionate:** Le soglie di confidenza ottimali sono tutte comprese tra 0.26 e 0.38, suggerendo che i modelli non necessitano di essere troppo conservativi per ottenere prestazioni elevate. Questo è un vantaggio significativo per scenari in tempo reale, dove si richiede un buon trade-off tra velocità e accuratezza.
- **Stabilità tra classi:** Le numerose linee grigie (una per classe) seguono l'andamento medio (linea blu) senza deviazioni estreme, confermando che la performance è uniforme tra le diverse categorie di segnali. Non esistono “classi problematiche” che compromettano il punteggio globale, un aspetto fondamentale per l'affidabilità del sistema.
- **Miglioramento progressivo:** Passando da YOLOv10n a YOLOv10m, si osserva un leggero ma costante miglioramento nel picco di F1, con il modello più grande che raggiunge il valore massimo (0.83) e la soglia più favorevole (0.263). Ciò conferma che l'aumento della capacità espressiva porta benefici tangibili in termini di robustezza e precisione.

Modello	Soglia Ottimale	F1 Massimo
YOLOv10n	0.357	0.82
YOLOv10s	0.380	0.83
YOLOv10m	0.263	0.83

Tabella 7.9: Soglie di confidenza ottimali per massimizzare lo F1-score. I valori sono estratti direttamente dalle curve visualizzate.

### Curve Precision-Recall (PR)

Le curve Precision-Recall offrono una visione dettagliata del compromesso tra accuratezza e completezza per ciascuna classe, sintetizzando le performance globali attraverso l'**area sotto la curva (AP — Average Precision)**. Il valore aggregato di queste aree è il **mAP@0.5**, metrica chiave per valutare l'efficacia complessiva del modello.

Le tre curve mostrano un andamento coerente e promettente:

- **Performance eccellenti:** Tutti e tre i modelli raggiungono valori di mAP@0.5 superiori all'83%, con YOLOv10s che spicca con un punteggio di **0.856**. Un risultato notevole, considerando la complessità del dataset d'addestramento e le condizioni reali del test set. Questo indica che i modelli sono in grado di rilevare correttamente la maggior parte degli oggetti con un numero contenuto di falsi positivi.
- **Stabilità tra classi:** Le numerose linee grigie (una per classe) seguono l'andamento medio (linea blu) senza deviazioni estreme, confermando che la performance è uniforme tra le diverse categorie di segnali. Non esistono “classi problematiche” che compromettano il punteggio globale, un aspetto fondamentale per l'affidabilità del sistema.
- **Comportamento prevedibile:** L'andamento delle curve è fluido e ben definito: la precisione cala gradualmente all'aumentare del richiamo, come previsto. Questo comportamento consente di scegliere facilmente una soglia di confidenza adatta al contesto d'uso, garantendo prestazioni robuste e prevedibili in ogni scenario.
- **Miglioramento progressivo:** Passando da YOLOv10n a YOLOv10s, si osserva un leggero ma costante miglioramento nel mAP@0.5, con il modello più grande che raggiunge il valore massimo (0.856). Ciò conferma che l'aumento della capacità espressiva porta benefici tangibili in termini di robustezza e precisione.

Modello	mAP@0.5
YOLOv10n	0.839
YOLOv10s	0.856
YOLOv10m	0.849

Tabella 7.10: mAP@0.5 estratto dalle curve PR. I valori sono riportati esplicitamente nelle etichette delle immagini.

### 7.2.3 YOLOv10n

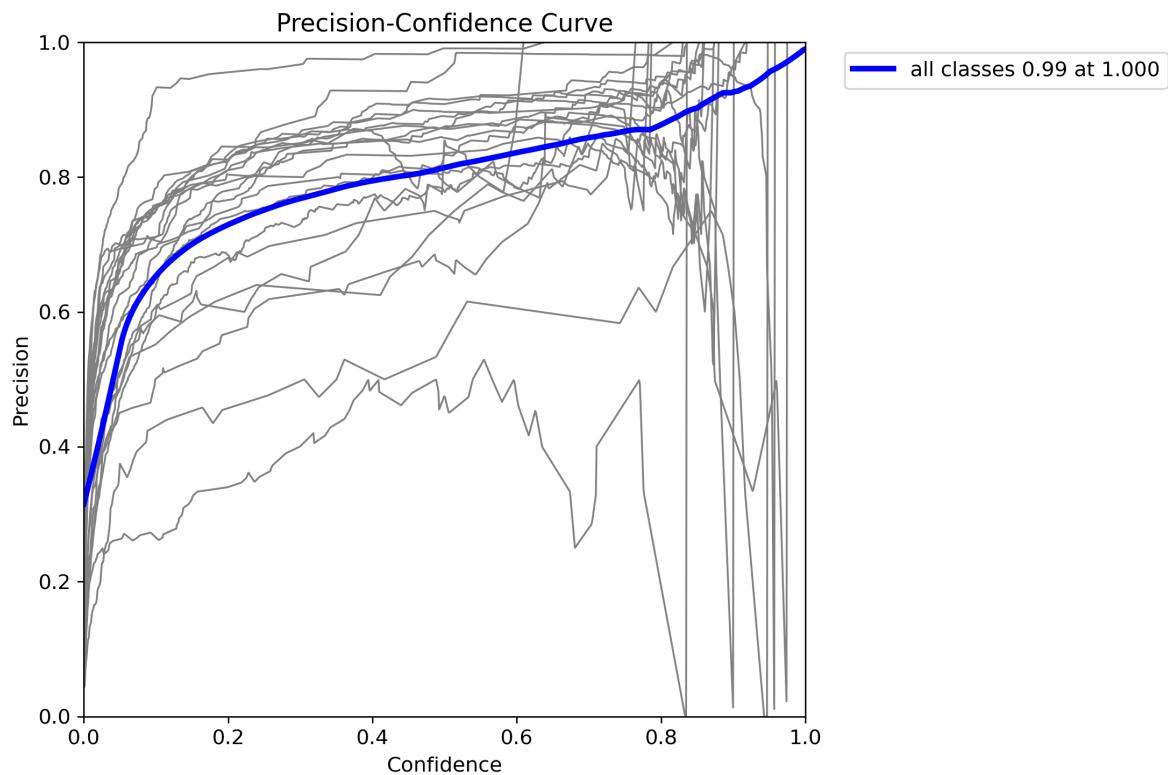


Figura 7.1: Precision-Confidence Curve — YOLOv10n

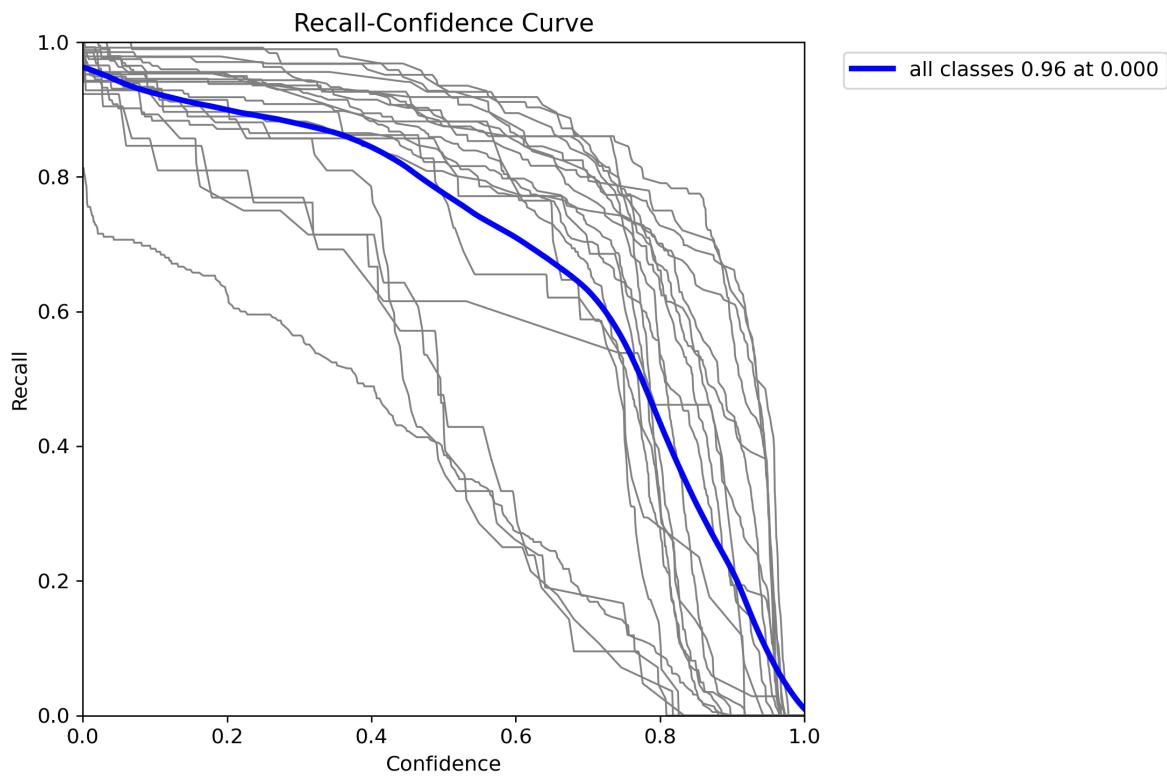


Figura 7.2: Recall-Confidence Curve — YOLOv10n

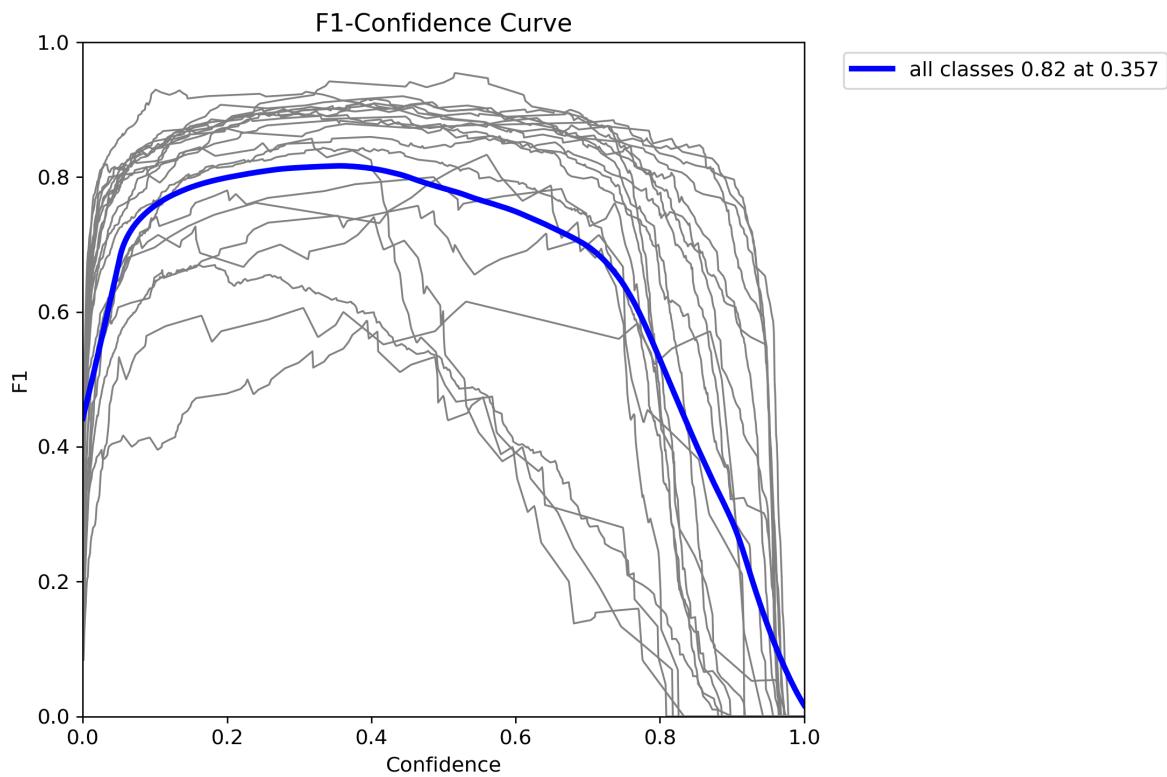


Figura 7.3: F1-Confidence Curve — YOLOv10n

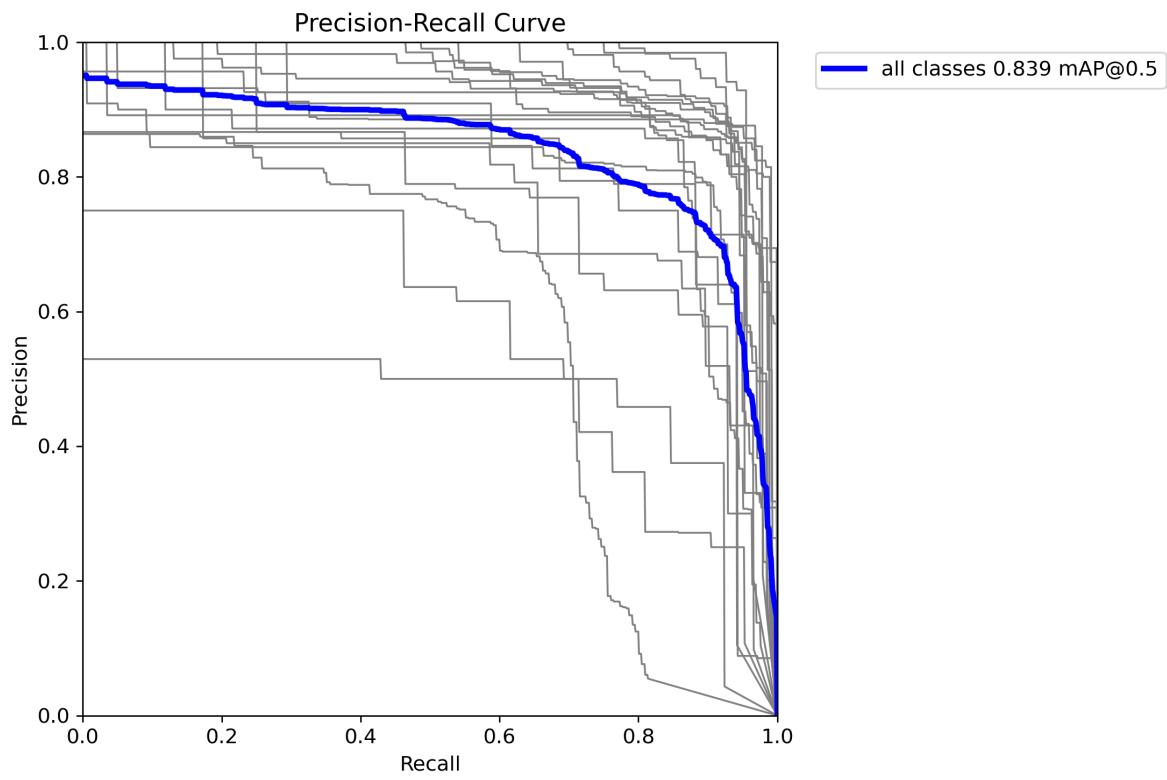


Figura 7.4: Precision-Recall Curve — YOLOv10n

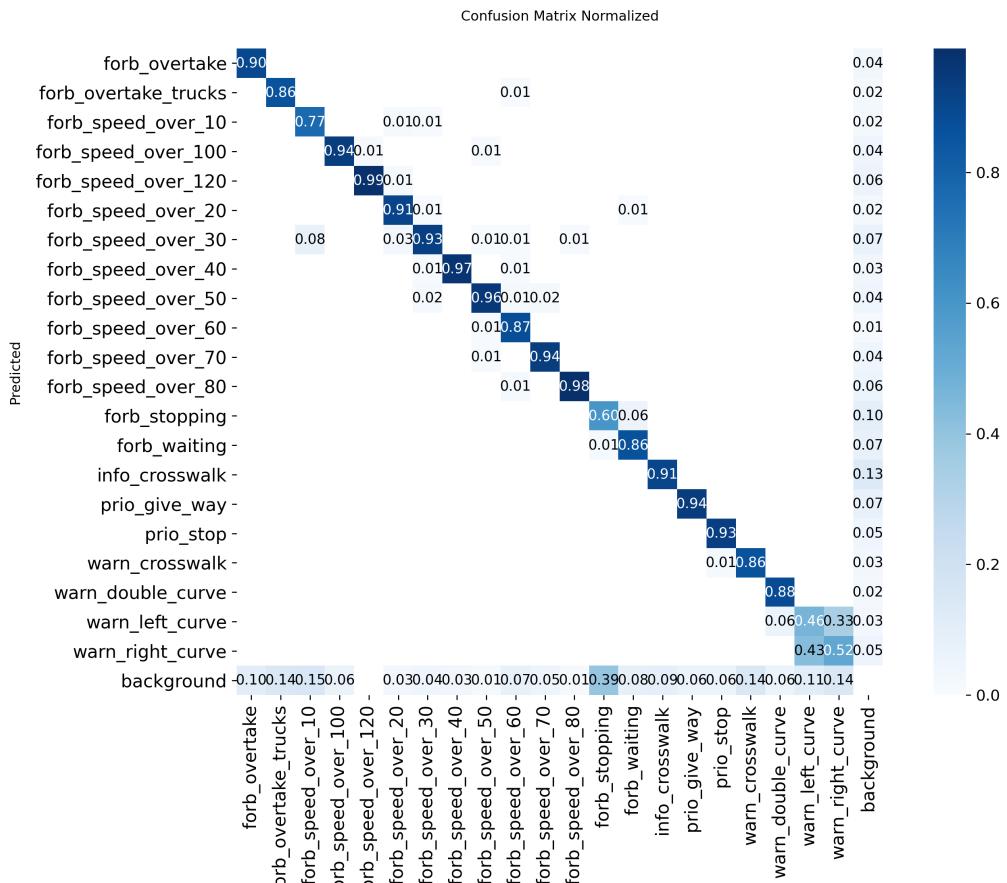


Figura 7.5: Matrice di Confusione — YOLOv10n

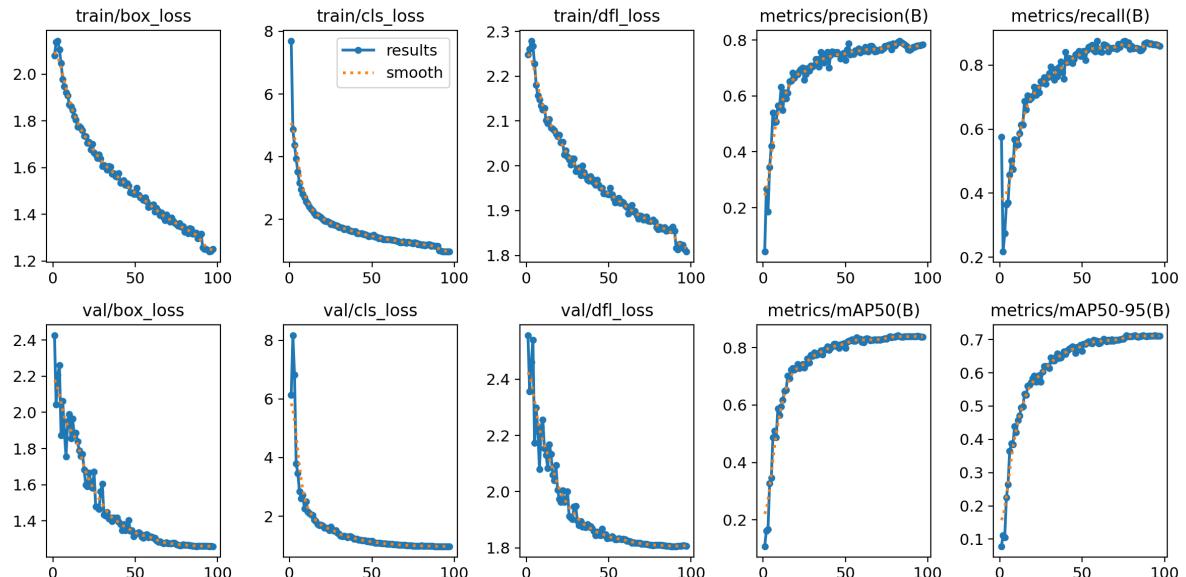


Figura 7.6: Andamento delle Loss durante l'addestramento — YOLOv10n

#### 7.2.4 YOLOv10s

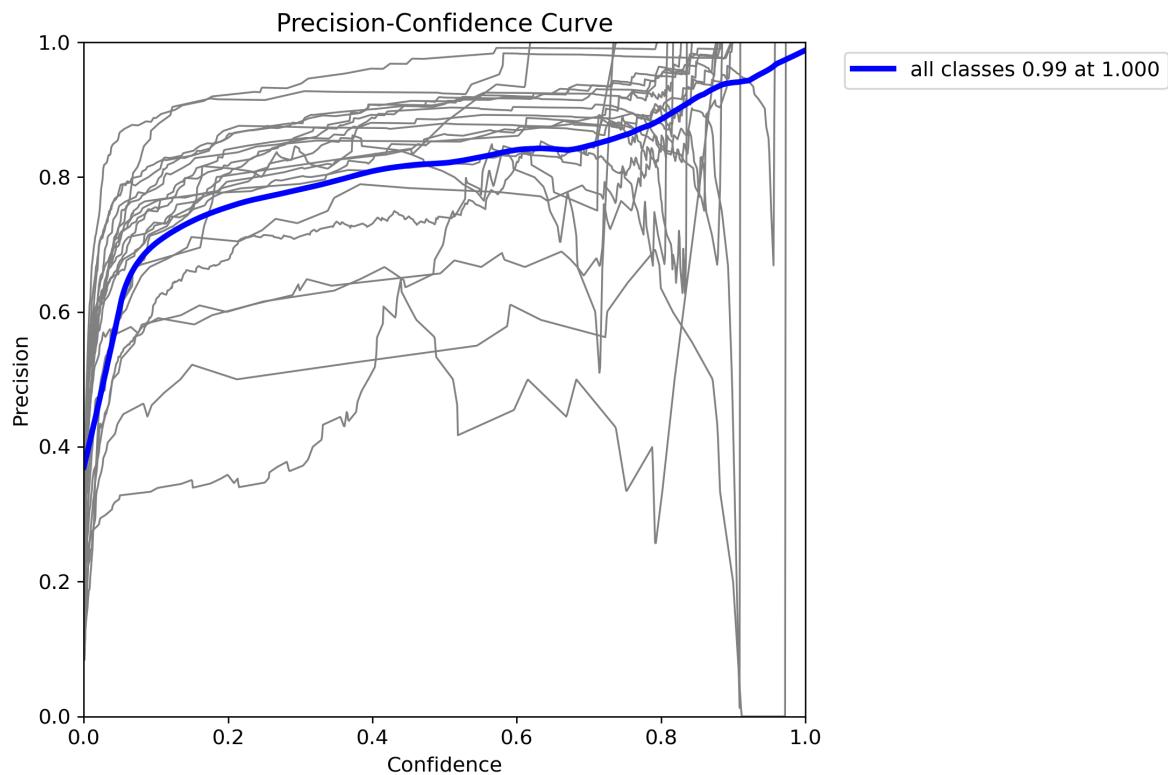


Figura 7.7: Precision-Confidence Curve — YOLOv10s

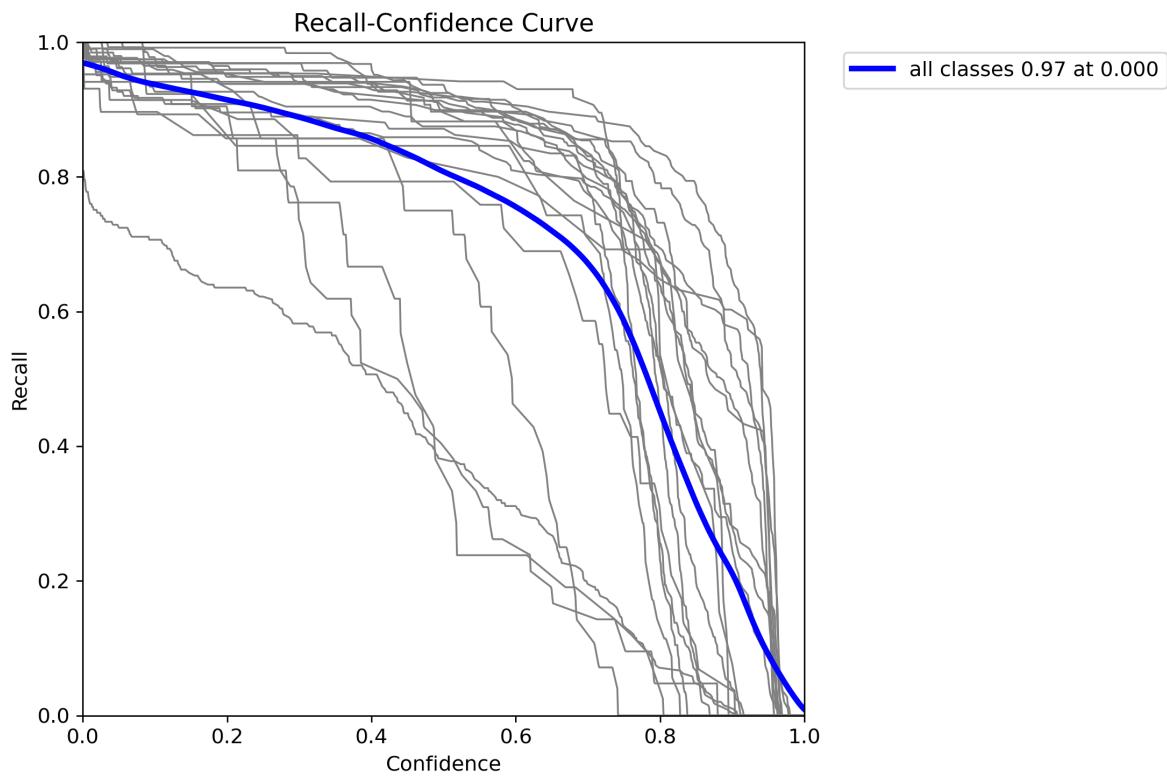


Figura 7.8: Recall-Confidence Curve — YOLOv10s

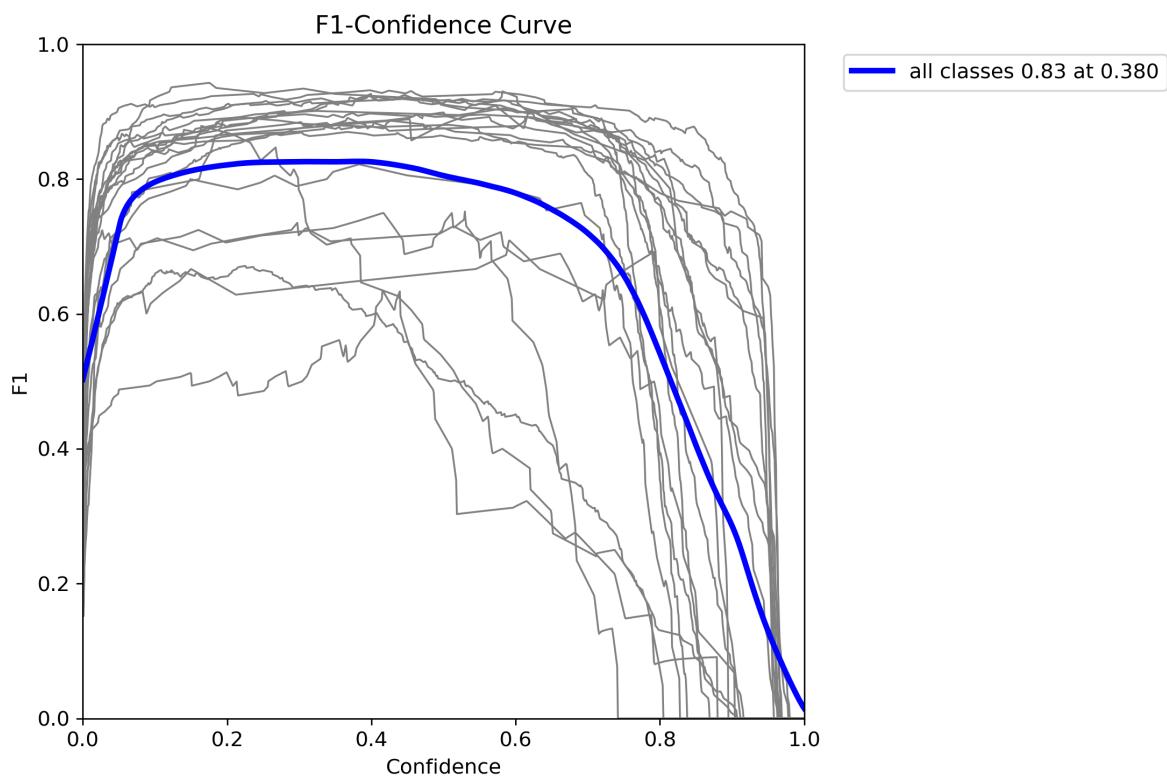


Figura 7.9: F1-Confidence Curve — YOLOv10s

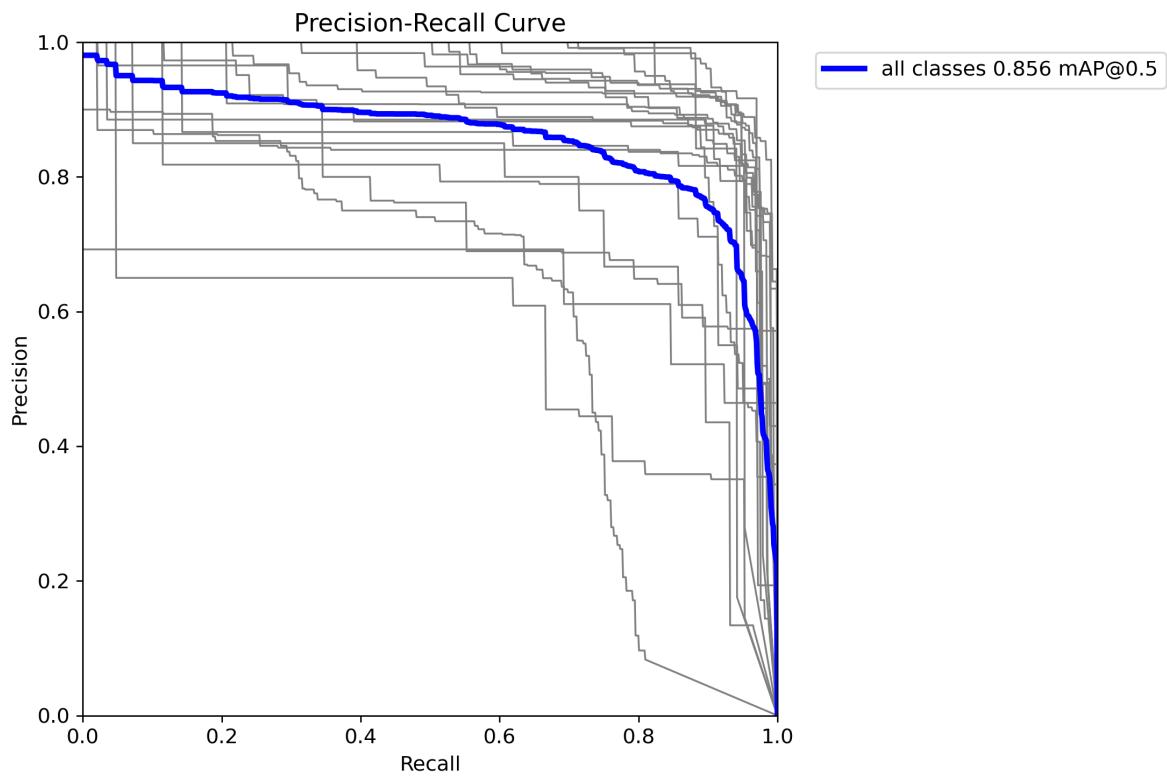


Figura 7.10: Precision-Recall Curve — YOLOv10s

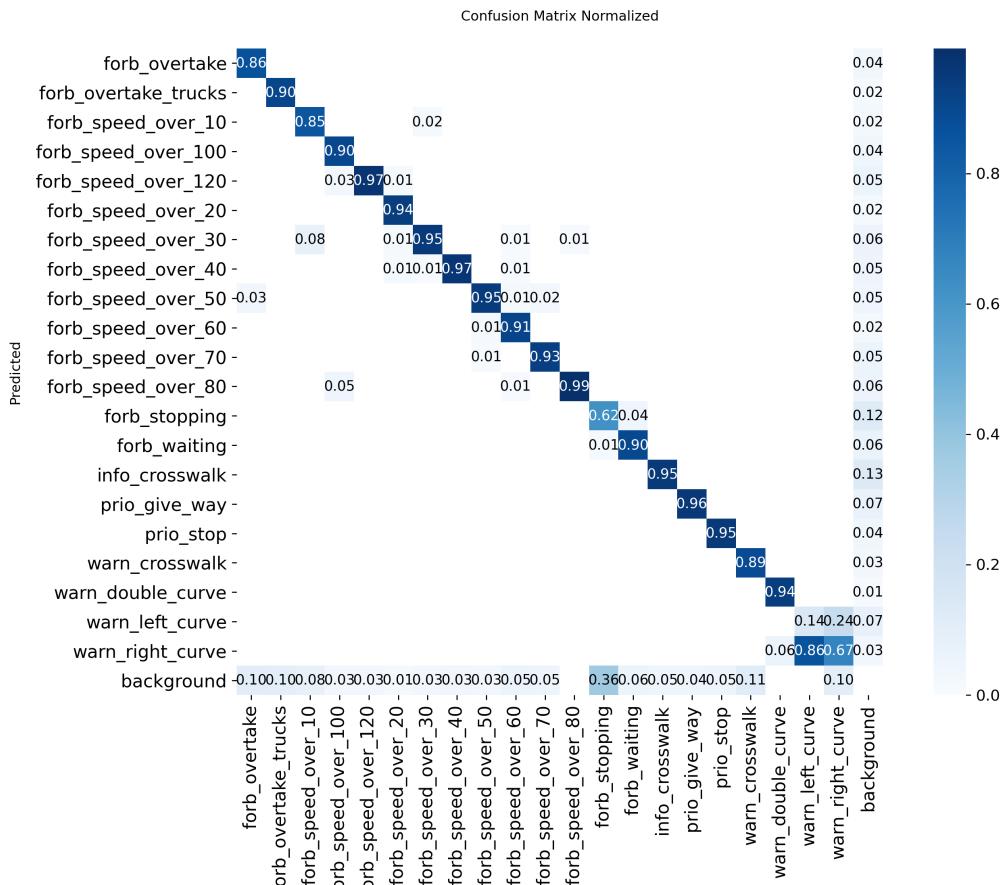


Figura 7.11: Matrice di Confusione — YOLOv10s

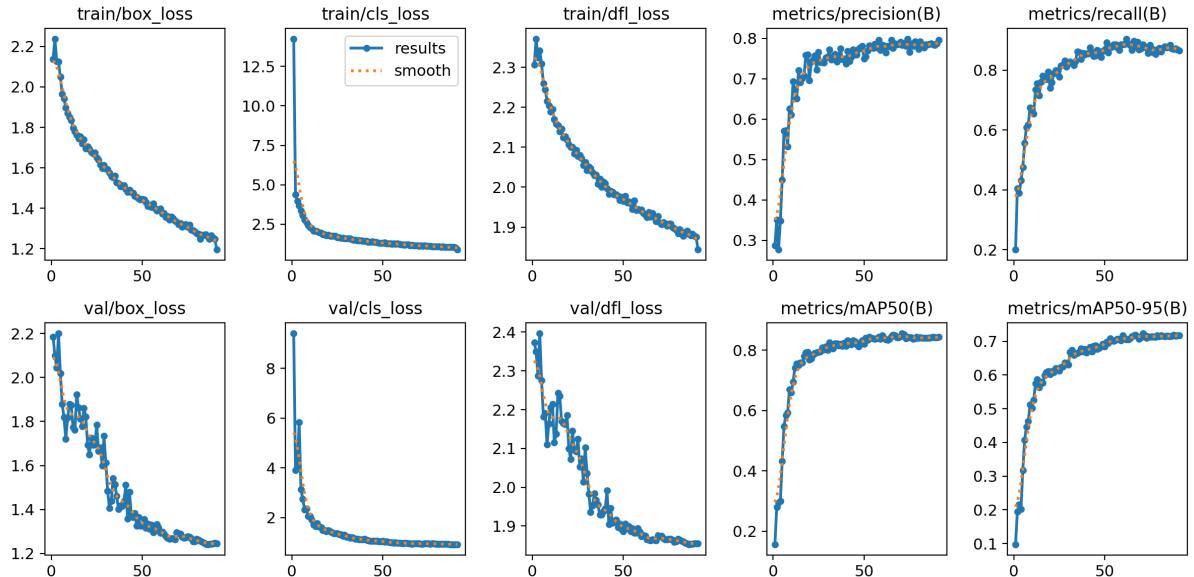


Figura 7.12: Andamento delle Loss durante l'addestramento — YOLOv10s

### 7.2.5 YOLOv10m

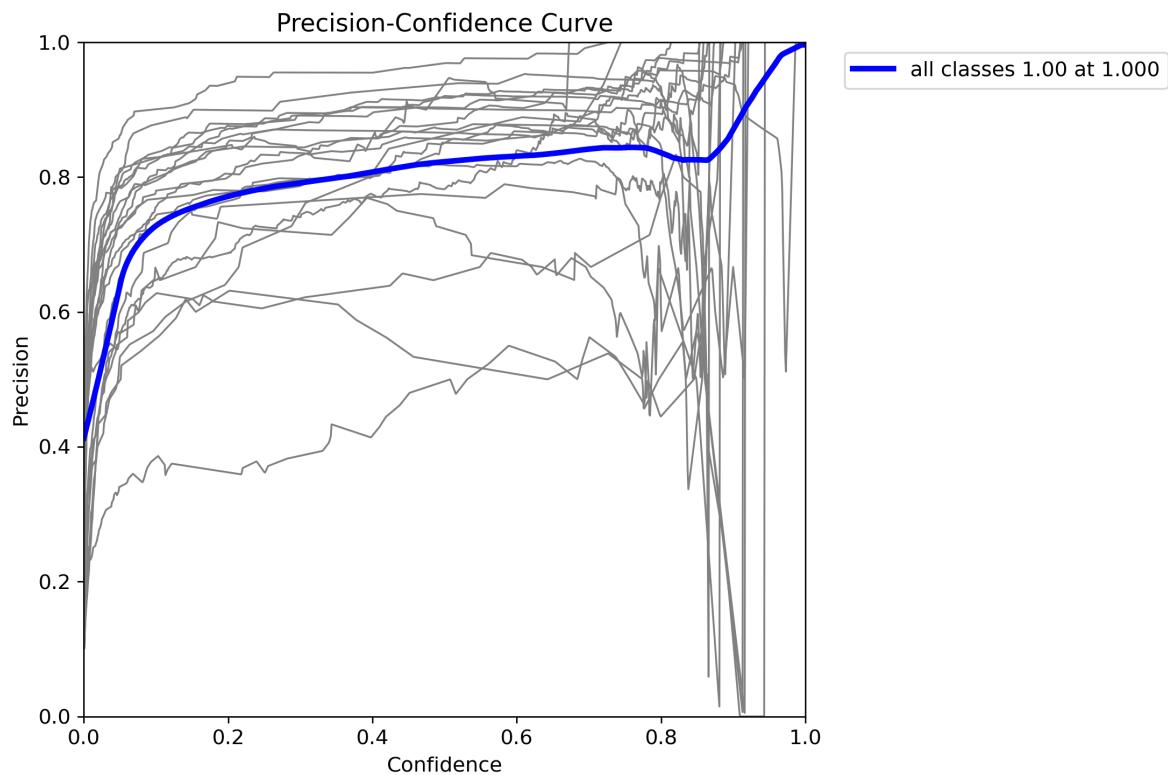


Figura 7.13: Precision-Confidence Curve — YOLOv10m

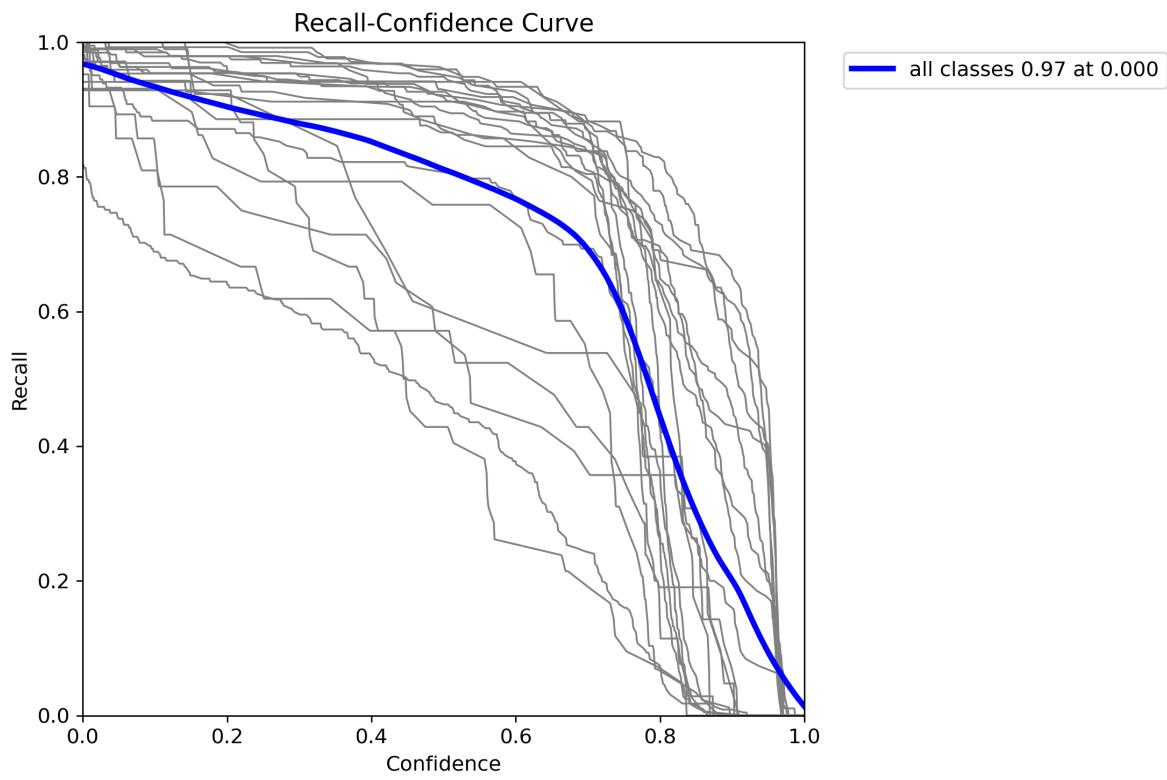


Figura 7.14: Recall-Confidence Curve — YOLOv10m

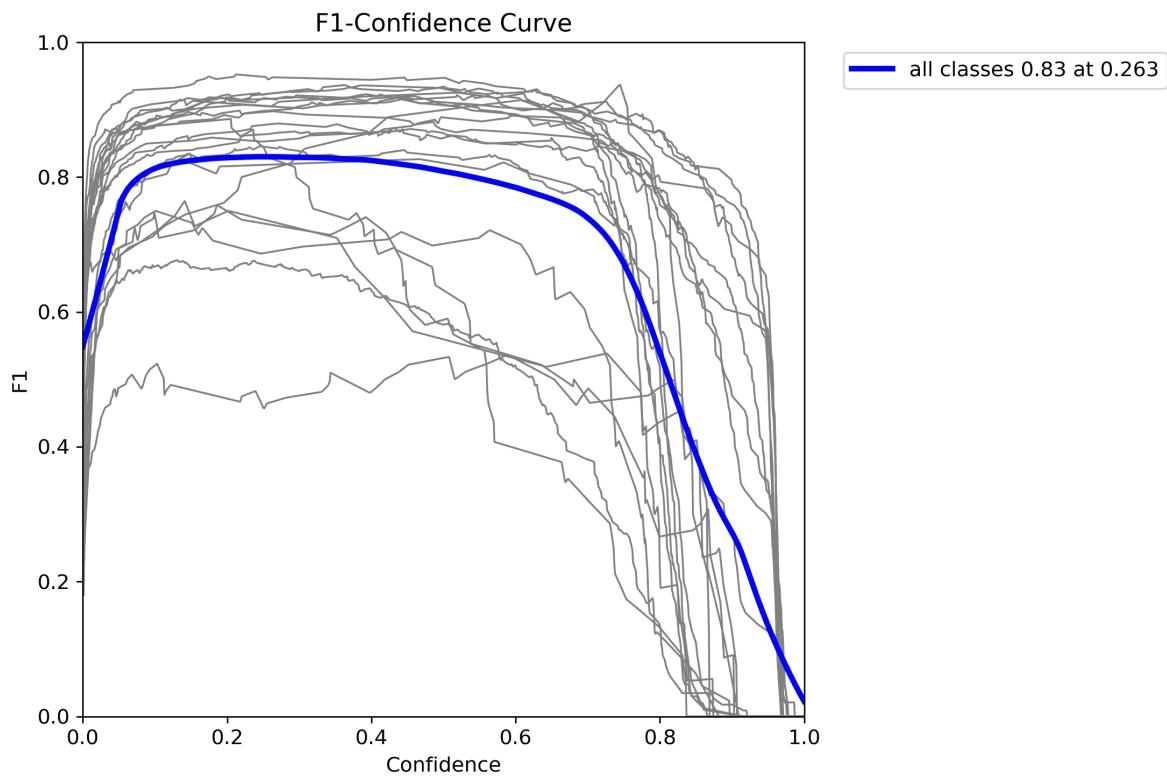


Figura 7.15: F1-Confidence Curve — YOLOv10m

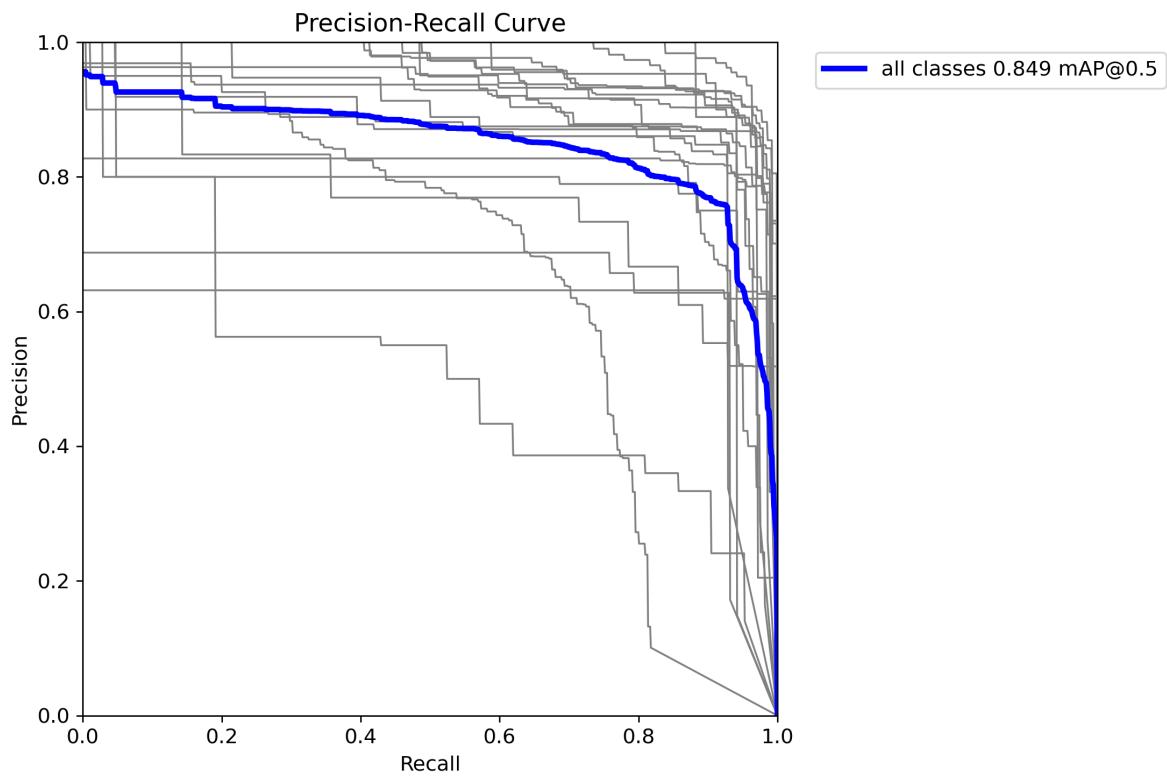


Figura 7.16: Precision-Recall Curve — YOLOv10m

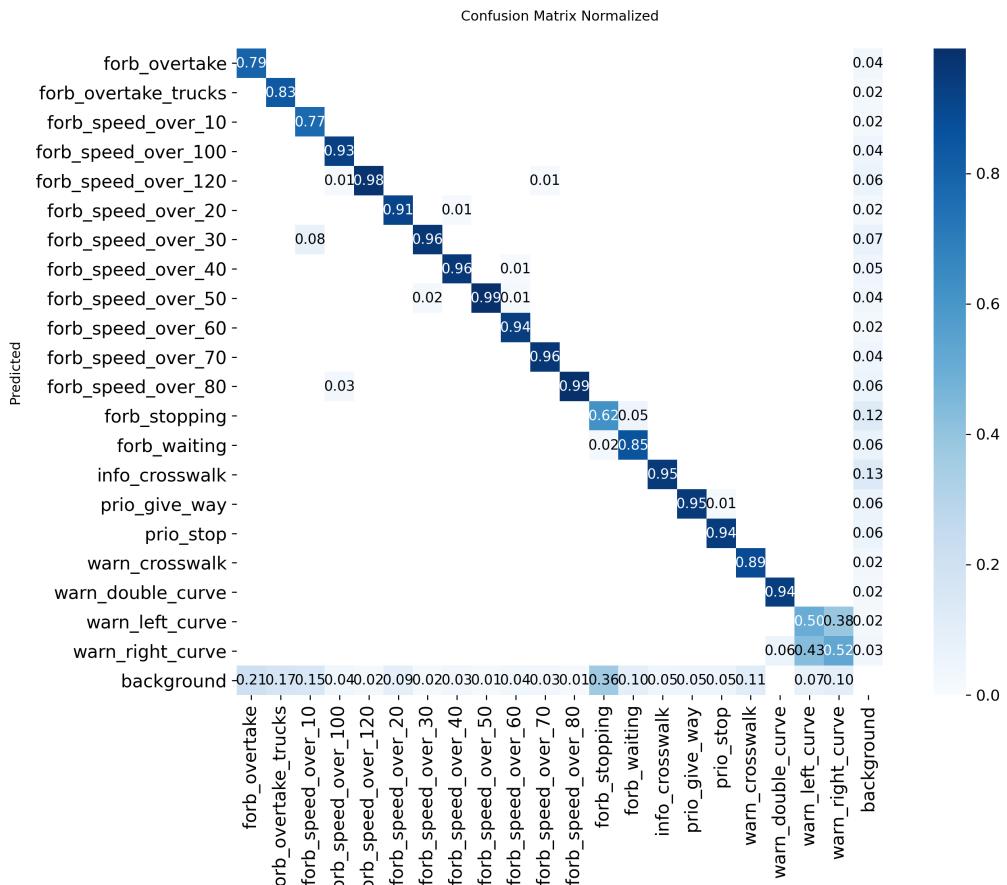


Figura 7.17: Matrice di Confusione — YOLOv10m

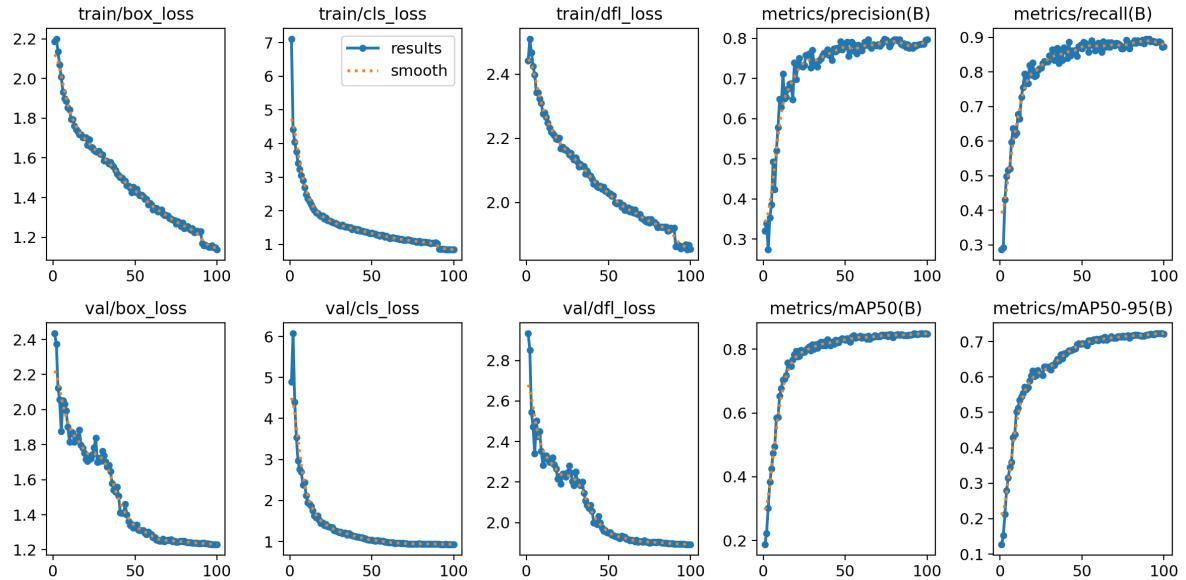


Figura 7.18: Andamento delle Loss durante l’addestramento — YOLOv10m

# Capitolo 8

## Conclusioni

In questo capitolo vengono sintetizzati i risultati ottenuti dal fine-tuning dei modelli **YOLOv10n**, **YOLOv10s** e **YOLOv10m** su un dataset ibrido composto da **GTSDB**, **ETSD** e dati aggiuntivi per classi sottorappresentate, seguito da una valutazione rigorosa su un **dataset di test reale**, acquisito autonomamente lungo le strade delle province di Catania, Caltanissetta e Ragusa. Vengono inoltre proposte riflessioni critiche e possibili sviluppi futuri.

### 8.1 Obiettivi e Risultati Ottenuti

L’obiettivo principale di questo progetto è stato applicare l’architettura all’avanguardia **YOLOv10** al riconoscimento automatico di segnali stradali, un compito fondamentale per i sistemi ADAS e di guida autonoma, valutandone l’efficacia in scenari reali e fuori distribuzione.

A causa di vincoli computazionali, sono state addestrate tre varianti: **YOLOv10n** (leggero), **YOLOv10s** (bilanciato) e **YOLOv10m** (più profondo). Il training è stato condotto per 100 epoche, con early stopping attivato solo per YOLOv10s (epoca 91), e monitorato tramite loss di bounding box, classificazione e distribuzione (DFL), oltre che da metriche di validazione.

La valutazione è stata effettuata su un dataset di test reale, non utilizzato in alcuna fase di addestramento né di validazione, al fine di misurare la capacità di generalizzazione.

#### Risultati principali:

- **YOLOv10s si è rivelato il modello più equilibrato:** con un **mAP@50 del 73.09%** e un **F1-score del 64.53%** sul test set, offre il miglior compromesso tra accuratezza, richiamo e robustezza, superando YOLOv10m in molte metriche aggregate nonostante la sua complessità inferiore.
- **YOLOv10m mostra la precisione più alta (74.16%),** ma una recall inferiore (61.79%), segno di un comportamento più conservativo. Tuttavia, eccelle in classi ben definite (es. `warn_left_curve`: F1 = 100%) e in contesti strutturati.
- **YOLOv10n, pur essendo il più veloce,** soffre di bassa precisione (56.61%) e F1-score limitato (56.88%), rendendolo meno adatto a scenari safety-critical, ma ancora utile per applicazioni con vincoli severi di risorse.

- Come previsto, si osserva un **calo prestazionale tra validation e test set**, con mAP@50 che passa da valori superiori all'85% (validation) a circa 70–73% (test). Questo gap è attribuibile a un *domain shift* tra i dataset di addestramento (segnali tedeschi/europei, immagini controllate) e il test set (segnali italiani, acquisiti in condizioni reali, con variazioni di illuminazione, angolazione, usura e sfondi complessi).
- Nonostante il calo, i modelli mantengono una **capacità significativa di rilevamento**, dimostrando che il fine-tuning su dataset pubblici fornisce una base solida, anche se non sufficiente per scenari operativi senza ulteriore adattamento.

## 8.2 Lezioni Apprese

Lo svolgimento di questo progetto ha fornito importanti intuizioni tecniche e metodologiche:

**1. L'importanza del background nel training:**

L'inclusione mirata di immagini senza segnali (10–20% del dataset) ha ridotto drasticamente i falsi positivi e migliorato la stabilità del modello, confermando che insegnare al sistema cosa *non* è un segnale è cruciale quanto riconoscere i segnali stessi.

**2. Il trade-off tra recall e precision è gestibile, ma non eliminabile:**

YOLOv10s, con la sua architettura bilanciata, riesce a mantenere un F1-score elevato senza eccessivi falsi positivi o negativi. Questo lo rende ideale per applicazioni reali, dove entrambi gli errori hanno conseguenze operative.

**3. L'analisi per classe è indispensabile:**

Metriche aggregate nascondono criticità. Solo l'analisi granulare ha permesso di identificare le lacune.

**4. Le curve di confidenza sono strumenti decisionali chiave:**

Le curve Precision-Confidence, Recall-Confidence e F1-Confidence hanno permesso di selezionare soglie ottimali (es. 0.26–0.38) per massimizzare l'affidabilità senza sacrificare eccessivamente la copertura, un passo essenziale per il deployment.

**5. L'efficienza computazionale è parte integrante del design:**

L'uso strategico di risorse limitate (Colab gratuito, VM con GPU L4) ha richiesto ottimizzazioni mirate: riduzione di augmentation costose, tuning del batch size, monitoraggio continuo del training. Ciò ha dimostrato che un buon modello non è solo accurato, ma anche *addestrabile in modo efficiente*.

## 8.3 Sviluppi Futuri

Per portare il sistema a un livello di maturità operativa, si propongono i seguenti sviluppi:

**• Domain adaptation mirata:**

Raccogliere e annotare un dataset più ampio di segnali stradali italiani, con focus su classi attualmente mal riconosciute (es. limiti di velocità testuali, segnali di attraversamento), per ridurre il domain shift.

- **Soglie di confidenza per classe:**

Implementare soglie dinamiche (es. più basse per `forb_speed_over_50`, più alte per `prio_stop`) per ottimizzare F1-score a livello di categoria.

- **Pseudo-labeling semi-supervisionato:**

Utilizzare il modello addestrato per etichettare nuove immagini stradali italiane, selezionare solo le predizioni con alta confidenza e riaddestrare il modello in un ciclo iterativo.

- **Test su sequenze video e condizioni estreme:**

Valutare le prestazioni in scenari dinamici (pioggia, notte, controluce) e su flussi video, per simulare condizioni di guida reali.

- **Deployment embedded:**

Convertire il modello in formato ONNX o TensorRT e testarlo su piattaforme edge (es. NVIDIA Jetson Orin) per verificare latenza, consumo energetico e stabilità in tempo reale.

- **Integrazione con contesto stradale:**

Combinare il rilevamento dei segnali con informazioni contestuali (es. presenza di incroci, marciapiedi, corsie) per ridurre ambiguità e migliorare l'affidabilità semantica.

In conclusione, questo progetto ha dimostrato che **YOLOv10** è una solida base per il riconoscimento di segnali stradali, con prestazioni elevate anche in contesti reali. Il modello YOLOv10s emerge come la scelta più matura per applicazioni pratiche, mentre YOLOv10m offre margini di miglioramento per scenari in cui la massima accuratezza è prioritaria. Il percorso tracciato fornisce una roadmap chiara per trasformare un sistema di prototipazione in una soluzione robusta e deployabile.

# Bibliografia

- [1] German Traffic Sign Detection Benchmark (GTSDB). *Disponibile all'indirizzo:* <http://benchmark.ini.rub.de/?section=gtsdb&subsection=dataset>
- [2] Traffic Sign Detection (ETSD). *Disponibile all'indirizzo:* <https://www.kaggle.com/datasets/raduoprea/traffic-signs>