



UNIVERSITÀ DEGLI STUDI DI CATANIA
DIPARTIMENTO DI MATEMATICA E INFORMATICA
CORSO DI LAUREA MAGISTRALE IN INFORMATICA

Salvatore Alfio Sambataro - Matricola: 1000015834

Traffic signs detection with YOLOv10

MACHINE LEARNING - PROGETTO FINALE

Docenti: Prof. Giovanni Maria Farinella, Prof. Rosario Leonardi

Anno Accademico 2023 - 2024

Indice

1	Introduzione	3
2	Task: Object Detection	5
2.1	Applicazioni pratiche	5
2.2	Applicazione: Traffic sign detection	7
2.3	YOLO - "You Only Look Once"	8
2.4	Metodo utilizzato: YOLOv10	9
2.4.1	Novità di YOLOv10	9
2.4.2	Architettura YOLOv10	10
3	Dataset	11
3.1	Dataset GTSDB	11
3.1.1	Heatmap delle annotazioni del training set	16
3.2	Dataset acquisito per la fase di test	17
3.2.1	Numero di immagini e annotazioni del test set	18
3.2.2	Heatmap delle annotazioni del test set	19
3.2.3	Iistogramma del numero di annotazioni per immagine nel test set	20
3.2.4	Iistogramma del numero di annotazioni per classe nel test set	21
3.2.5	Dati di esempio del test set	22
3.3	Pre-processing dei dataset	23
3.3.1	Pre-processing sul dataset GTSDB	23
3.3.2	Pre-processing sul dataset di test	25
3.3.3	Codici per il pre-processing	25
3.4	Organizzazione del dataset per il training	26
4	Valutazione del modello	28
4.1	Metriche di valutazione	28
4.2	Grafici	31
4.3	Loss functions	33

<i>INDICE</i>	2
5 Metodi di realizzazione del progetto	34
6 Training e validazione del modello	36
6.1 Parametri di training	36
6.2 Organizzazione della directory per il training	37
6.3 Fase di training	37
6.4 App Demo	45
6.5 Organizzazione delle immagini per utilizzare l'app	46
6.6 Avviare l'app	46
7 Risultati	48
7.1 Tabelle riassuntive	49
7.1.1 Architettura dei modelli allenati	49
7.1.2 Precion, Recall, F_1 -score medie (validation set)	50
7.1.3 mAP medio (validation set)	50
7.1.4 Precion, Recall, F_1 -score medie (test set)	51
7.1.5 mAP medio (test set)	51
7.1.6 Metriche di valutazione sulle singole classi	52
7.2 Loss, matrice di confusione e curve	56
7.2.1 YOLOv10n	57
7.2.2 YOLOv10s	60
7.2.3 YOLOv10m	63
8 Conclusione	66
8.1 Obiettivi e risultati ottenuti	66
8.2 Lezioni apprese	67
8.3 Sviluppi futuri	68
Appendice	69
Bibliografia	79

Capitolo 1

Introduzione

L'object detection è una delle principali applicazioni del machine learning nel campo della Computer Vision. Essa prevede simultaneamente l'identificazione e la localizzazione di oggetti di interesse all'interno di un'immagine o di un video. Tra i diversi algoritmi sviluppati per l'object detection, la famiglia di algoritmi YOLO (You Only Look Once) è emersa come una delle più efficienti e popolari, ed essa si distingue principalmente per la sua capacità di eseguire la rilevazione degli oggetti in tempo reale. YOLOv10 rappresenta l'ultima evoluzione di questa serie di modelli, e punta a consolidare queste caratteristiche con ulteriori ottimizzazioni architetturali e tecniche di addestramento avanzate. Esistono nello specifico sei diverse versioni di YOLOv10:

- **YOLOv10n:** versione "Nano" per ambienti con risorse estremamente limitate
- **YOLOv10s:** versione "Small" che bilancia accuratezza e velocità
- **YOLOv10m:** versione "Medium" per un uso generico
- **YOLOv10b:** versione "Balanced" con maggiore accuratezza
- **YOLOv10l:** versione "Large" con maggiori accuratezza e, tuttavia, maggior costo computazionale
- **YOLOv10x:** versione "Extra-large" con accuratezza e performance più elevate possibile

L'obiettivo del presente progetto è applicare alcuni dei modelli YOLOv10 proposti, in particolare **YOLOv10n**, **YOLOv10s** e **YOLOv10m**, al problema della **detection di segnali stradali**.

La valutazione delle performance dei modelli sarà effettuata sulla base di appropriate metriche di valutazione, tra cui la precision, la recall e la mean Average

Precision. Al fine di effettuare una valutazione esaustiva, il benchmarking verrà eseguito utilizzando il dataset **German Traffic Sign Detection Benchmark (GTSDB)** per la fase di training, mentre per la fase di testing sarà usato un **dataset di segnali stradali acquisito personalmente**, secondo le modalità illustrate nelle sezioni successive.

Model	Test Size	#Params	FLOPs	AP^{val}	Latency
YOLOv10-N	640	2.3M	6.7G	38.5%	1.84ms
YOLOv10-S	640	7.2M	21.6G	46.3%	2.49ms
YOLOv10-M	640	15.4M	59.1G	51.1%	4.74ms
YOLOv10-B	640	19.1M	92.0G	52.5%	5.74ms
YOLOv10-L	640	24.4M	120.3G	53.2%	7.28ms
YOLOv10-X	640	29.5M	160.4G	54.4%	10.70ms

Figura 1.1: Carrateristiche dei diversi modelli di YOLOv10. Metrica AP e latenza misurate sul dataset **COCO**

Capitolo 2

Task: Object Detection

La Object detection è un task di computer vision il cui obiettivo è quello di **identificare la classe di appartenenza e individuare l'esatta posizione degli oggetti presenti in un'immagine o sequenza video, definendo dei cosiddetti "bounding box"**. Un algoritmo di Object Detection può essere quindi visto come la combinazione di:

- Un algoritmo di **classificazione**, in quanto è necessario predire la classe di appartenenza degli oggetti individuati
- Un algoritmo di **regressione**, in quanto è necessario restituire dei valori che identifichino la posizione del bounding box che racchiuda al meglio gli oggetti individuati.

2.1 Applicazioni pratiche

Il task di Object Detection risulta essere di elevata importanza in molte applicazioni pratiche. Alcuni esempi sono:

- **Automotive**: i veicoli a guida autonoma hanno la necessità di "comprendere" l'ambiente circostante, identificando gli oggetti che li circondano (pedoni, segnali stradali, ostacoli, ...);
- **Sicurezza e sorveglianza**: ad esempio, al fine di individuare la presenza di soggetti in aree riservate o con accesso vietato
- **Automazione di processi industriali**: conteggio delle unità prodotte, individuazione di difetti nelle unità prodotte, ...

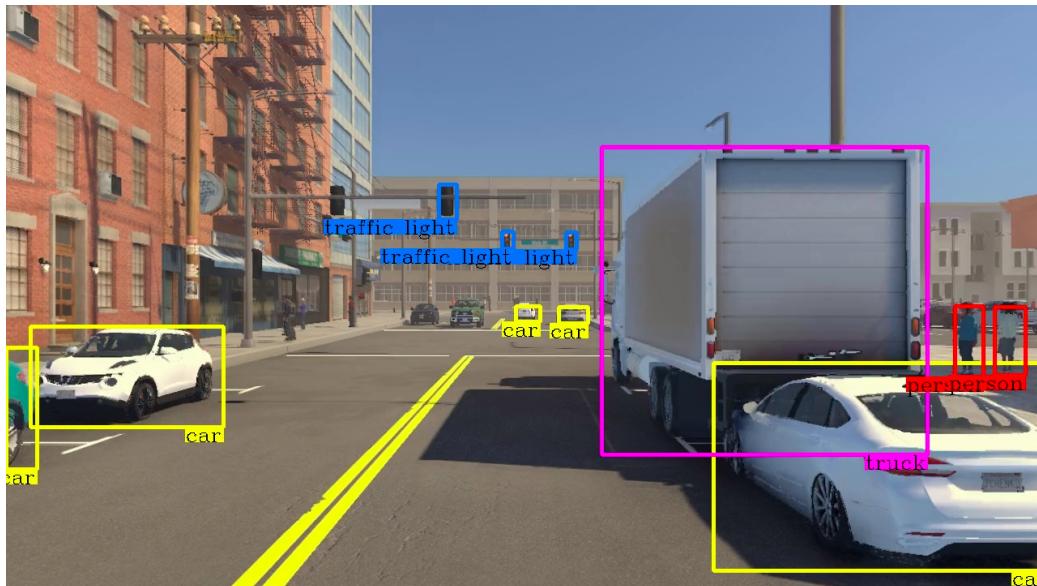


Figura 2.1: Object Detection applicata all'automotive



Figura 2.2: Object Detection applicata a sistemi di videosorveglianza

2.2 Applicazione: Traffic sign detection

Il task principale affrontato nell'ambito del presente progetto è il task di **”Traffic signs detection”**, ovvero la **rilevazione di segnali stradali**.

La Traffic signs detection è un particolare task legato all'automotive, che si focalizza sulla classificazione e la localizzazione automatica dei segnali stradali presenti nelle immagini o nei video, solitamente acquisiti da telecamere montate su veicoli o apposite infrastrutture. Tale tecnologia trova il suo principale scopo nel **fornire supporto ai sistemi di guida autonoma o assistita (ADAS)** al fine di **migliorare la sicurezza stradale**.

Nella realizzazione pratica di sistemi che eseguano tali compiti emergono però delle importanti problematiche, che rendono la rilevazione di segnali molto complessa. Tra queste abbiamo:

- grande variabilità dei segnali stradali in termini di forma, colore e dimensioni, dovuta alle normative delle varie nazioni
- fattori ambientali come condizioni atmosferiche avverse o illuminazione variabile
- ostruzioni parziali dei segnali, causate ad esempio dalla presenza di vegetazione o altri veicoli

Per superare queste difficoltà, è fondamentale effettuare il training dei modelli su dataset quanto più possibile diversificati.

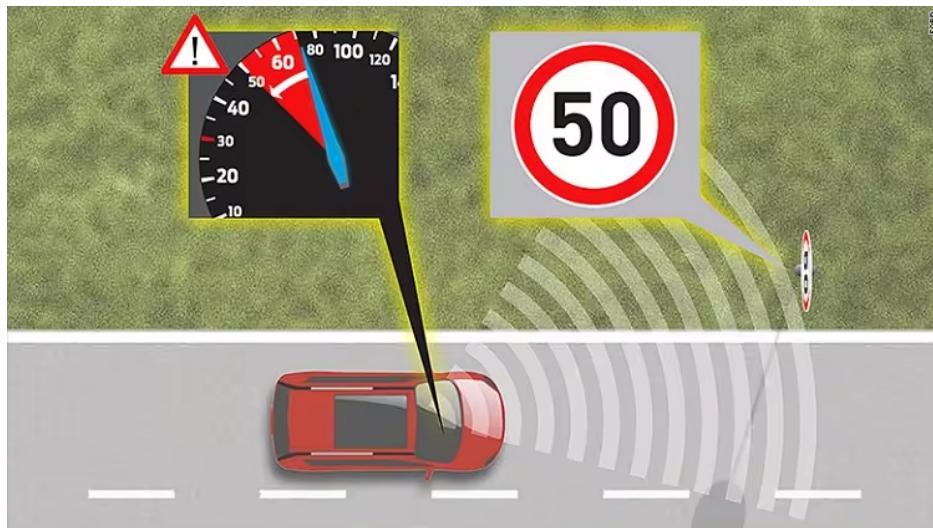


Figura 2.3: Sistema di detection su auto: una telecamera interna rileva la presenza di segnali stradali e adatta di conseguenza la velocità

2.3 YOLO - "You Only Look Once"

You Only Look Once (YOLO) è un modello di Object Detection di tipo **"Single Stage"**, ovvero **effettua la predizione dei bounding boxes e delle class probabilities in un unico passo**. L'altra caratteristica fondamentale è quella di **permettere la detection di oggetti in Real Time**. Seguendo questo approccio, YOLO ha superato gli algoritmi di Object Detection che rappresentavano lo stato dell'arte.

Diverse versioni di YOLO sono state proposte nel corso degli anni, sin dal rilascio della prima versione di YOLO, nel 2015:

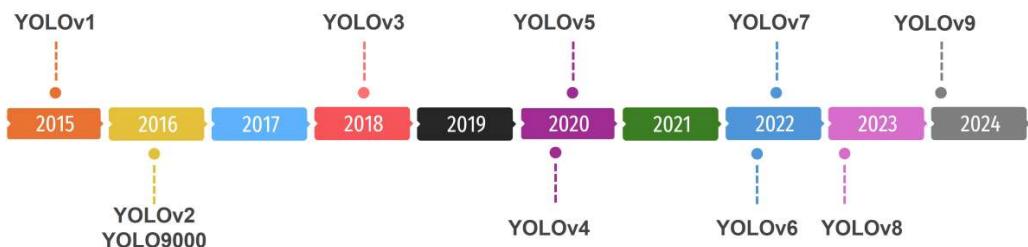


Figura 2.4: Timeline dei modelli YOLO

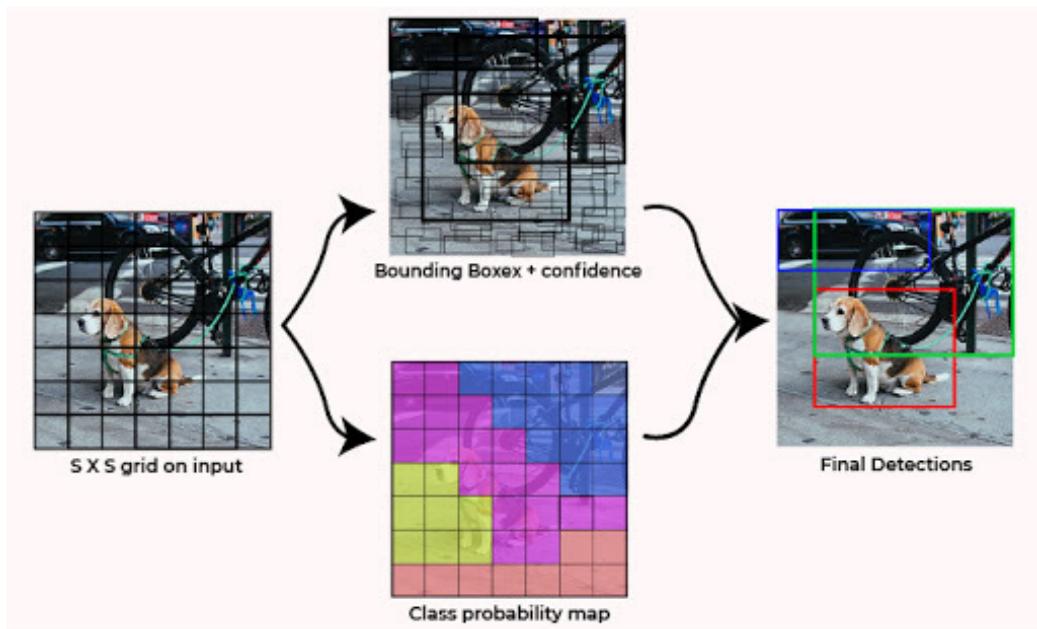


Figura 2.5: Funzionamento Single-Step

2.4 Metodo utilizzato: YOLOv10

YOLOv10, prodotto dai ricercatori della Tsinghua University, introduce un nuovo approccio alla object detection in real time, risolvendo le carenze di post-elaborazione e di architettura del modello riscontrate nelle precedenti versioni di YOLO. In particolare, i principali cambiamenti riguardano:

- Eliminazione del meccanismo di **Non Maxima Suppression (NMS)** in fase di training, necessario per eliminare predizioni ridondanti
- Ottimizzazione di componenti già presenti nell'architettura

2.4.1 Novità di YOLOv10

La strategia implementata nell'architettura di YOLOv10 prende il nome di ***Dual Label Assignment with NMS-free training***. Per ovviare all'overhead introdotto dal processo di NMS, YOLOv10 introduce un processo di **dual label assignment**, che consiste in due parti:

- **One-to-One Matching**: assegna una singola predizione ad ogni istanza/oggetto, eliminando quindi la necessità del processo di NMS. Il problema di questo approccio sono risultati con accuracy sub-ottimale e lenta convergenza.
- **One-to-Many Assignment**: permette di ottenere performance migliori, ma è necessario introdurre un processo di NMS

YOLOv10 combina queste strategie introducendo un ulteriore **one-to-one head**, mantenendo comunque la struttura uno-a-molti originale. Durante la fase di training, entrambi i "branch" vengono ottimizzati.

Una componente chiave della strategia "Dual Label Assignment" è la cosiddetta **"Consistent Matching Metric"**, utilizzata per valutare la concordanza tra previsioni e istanze ground truth. In particolare, questa metrica permette di **ottimizzare il One-to-One Head in maniera congiunta con il One-to-Many Head**. Tale metrica è basata essenzialmente sulla **similarità tra le misure IoU restituite dai due Heads, e ne "allinea"** le previsioni.

2.4.2 Architettura YOLOv10

L'architettura di YOLOv10 prevede le seguenti componenti:

- **Backbone:** responsabile dell'**estrazione di feature**
- **Neck:** aggrega feature a diverse scale e le passa ai due heads. L'aggregazione di feature su scale diverse è effettuato tramite un'architettura **PAN (Path Aggregation Network)**
- **Dual Label Assignment:** assegnamento "**One-to-Many**" per velocizzare la fase di training, assegnamento "**One-to-One**" usato in fase di inferenza
- **Consistent Matching Metric:** usata per valutare la concordanza tra predizioni e ground-truth. È definita come:

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

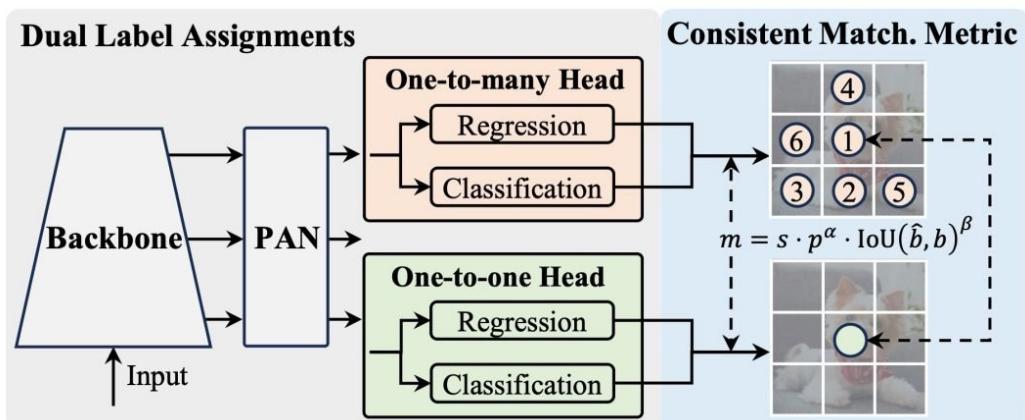


Figura 2.6: Architettura di YOLOv10

Per maggiori informazioni si rimanda al paper originale di YOLOv10 [1].

Capitolo 3

Dataset

La realizzazione del progetto ha previsto l'uso di **due diversi dataset**:

- Il dataset di segnali stradali **German Traffic Sign Detection Benchmark**, che rappresenta lo stato dell'arte dei dataset di segnali stradali, usato per la fase di **training** dei diversi modelli di YOLOv10;
- Un ulteriore dataset di segnali stradali acquisito personalmente, usato per la fase di **testing** dei diversi modelli.

3.1 Dataset GTSDB

Il dataset usato per la fase di training dei modelli è il dataset **German Traffic Sign Detection Benchmark** [2], il quale contiene **immagini** di segnali stradali scattate in mezzo al traffico, a bordo di un veicolo, **in diverse condizioni di illuminazione, posa e orientamento**. Sono in particolare fornite **800 immagini**, divise in **600 immagini di train** e **200 immagini di validation**, memorizzate in formato **PPM**. In ognuna delle immagini sono presenti **da 0 a 6 segnali** appartenenti a **43 possibili classi**, con una dimensione che varia tra 16×16 e 128×128 . Insieme alle immagini sono fornite delle **annotazioni ground truth** all'interno di appositi file di testo, i quali hanno lo **stesso nome dell'immagine a cui fanno riferimento**. Ognuno di questi file contiene **0 o più righe**, le quali forniscono informazioni sui segnali stradali eventualmente presenti nelle immagini.

In particolare, ogni riga ha il seguente formato, tipico dei modelli YOLO:

<class><x_center><y_center><width><height>

- **Class.** Class ID dello specifico segnale. Il mapping *ID - Segnale* è il seguente:

- **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))**

- **<x_center>, <y_center>**: coordinate *x* e *y* **normalizzate** del centro del bounding box ground-truth che racchiude il segnale.
- **<width>, <height>**: larghezza e altezza *x* e *y* **normalizzate** del bounding box ground-truth che racchiude il segnale.



Figura 3.1: Immagine train 115. Etichette ground-truth:
 14 0.6617647058823529 0.334375 0.07352941176470588 0.12375
 38 0.25036764705882353 0.6675 0.015441176470588236 0.0275



Figura 3.2: Immagine train 284. Etichette ground-truth:
12 0.7496323529411765 0.385 0.05808823529411765 0.0975
2 0.7514705882352941 0.469375 0.04264705882352941 0.07125



Figura 3.3: Immagine train 426. Etichette ground-truth:
25 0.8360294117647059 0.558125 0.08823529411764706 0.13875

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

Tabella 3.1: Distribuzione delle classi nel training set (fonte: **DatasetNinja**)

3.1.1 Heatmap delle annotazioni del training set

La **Annotation Heatmap** mostra le aree delle immagini ove sono localizzate le annotazioni nelle diverse immagini del training set.

E' utile conoscere tale informazione perché **se le annotazioni sono localizzate solo in specifiche parti delle immagini, il modello potrebbe erroneamente considerare tale pattern durante l'inferenza**. In questo caso, la heatmap sembra ben distribuita, con il maggior numero delle annotazioni localizzate sulla parte destra delle immagini. Ciò è ovviamente dovuto al fatto che le immagini sono acquisite dall'interno di un'automobile, e la maggior parte dei segnali si trova sulla parte destra della carreggiata.

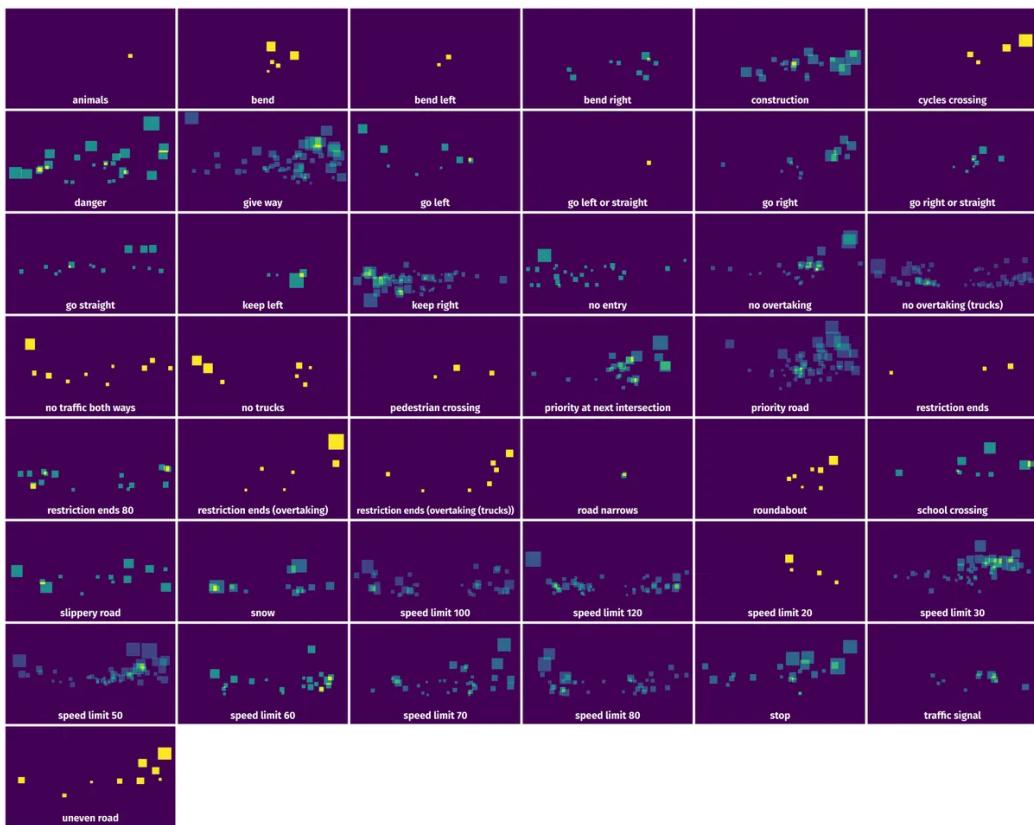


Figura 3.4: Spatial Heatmap delle etichette del training set per ognuno dei segnali stradali del dataset

3.2 Dataset acquisito per la fase di test

Al fine di effettuare la fase di testing del modello su dati mai visti prima, è stato necessario realizzare da zero un nuovo dataset di testing, formato da **immagini di segnali stradali acquisite dal sottoscritto lungo le strade della provincia di Catania**.

Il dataset contiene al suo interno **205 immagini** acquisite a bordo di un'auto, attraverso un dispositivo **Samsung Galaxy S21+**, a risoluzione originale **2268 x 4032 pixels**.

Un'altro aspetto importante è che le immagini sono state acquisite con:

- diverse condizioni di illuminazione
- diverse angolazioni
- diverse condizioni atmosferiche
- diversi "stati", es. segnali deformati, rovinati o scoloriti

L'utilizzo di immagini catturate da diverse pose, angolazioni e condizioni atmosferiche offre numerosi vantaggi, tra cui un aumento della robustezza e una maggiore generalizzazione del modello.

In fase di acquisizione del dataset si è verificata un'importante problematica: **non è stato possibile acquisire immagini per tutte le possibili classi presenti nel dataset di training GTSDB**. In particolare, non è stato possibile acquisire immagini relative alle seguenti classi:

- Speed limit 120
- Snow
- Restriction ends (overtaking)
- Restriction ends (overtaking (trucks))

Per tanto, ai fini del progetto, si è deciso di **rimuovere tali segnali dalle possibili classi**.

Le immagini sono successivamente state annotate attraverso il tool **Robo-Flow**, il quale permette di effettuare le annotazioni delle immagini grazie ad una intuitiva interfaccia grafica, e permette poi di scaricare le annotazioni in diversi formati, tra cui proprio il formato YOLO visto in precedenza.

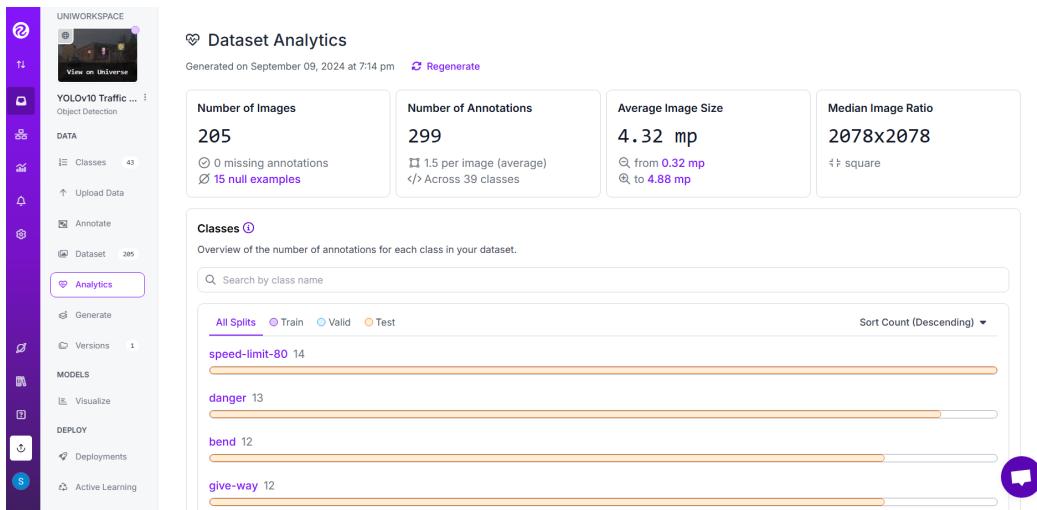


Figura 3.5: Statistiche sul dataset acquisito, mostrate su RoboFlow

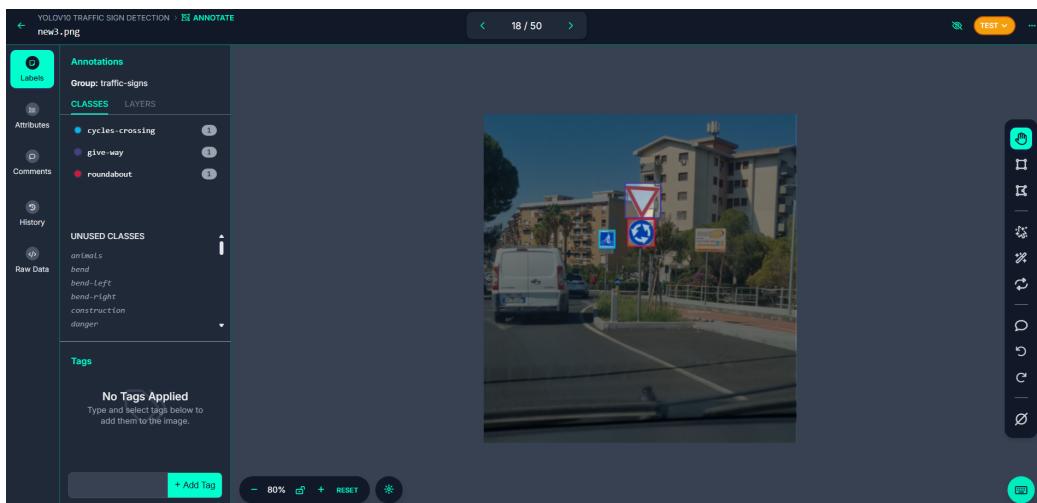


Figura 3.6: Tool per le annotazioni di RoboFlow

3.2.1 Numero di immagini e annotazioni del test set

- **Numero di immagini:** 205 (di cui 15 di background, ovvero senza alcuna annotazione);
- **Numero di annotazioni:** 299
- **Annotazioni medie per immagine:** 1.5

3.2.2 Heatmap delle annotazioni del test set

La **Annotation Heatmap** mostra le aree delle immagini ove sono localizzate le annotazioni nelle diverse immagini del dataset.

E' utile conoscere tale informazione perché **se le annotazioni sono localizzate solo in specifiche parti delle immagini, il modello potrebbe erroneamente considerare tale pattern durante l'inferenza**. Anche in questo caso, la heatmap è ben distribuita, con il maggior numero delle annotazioni localizzate sulla parte destra delle immagini. Possono essere quindi fatte le stesse considerazioni fatte per il training set.

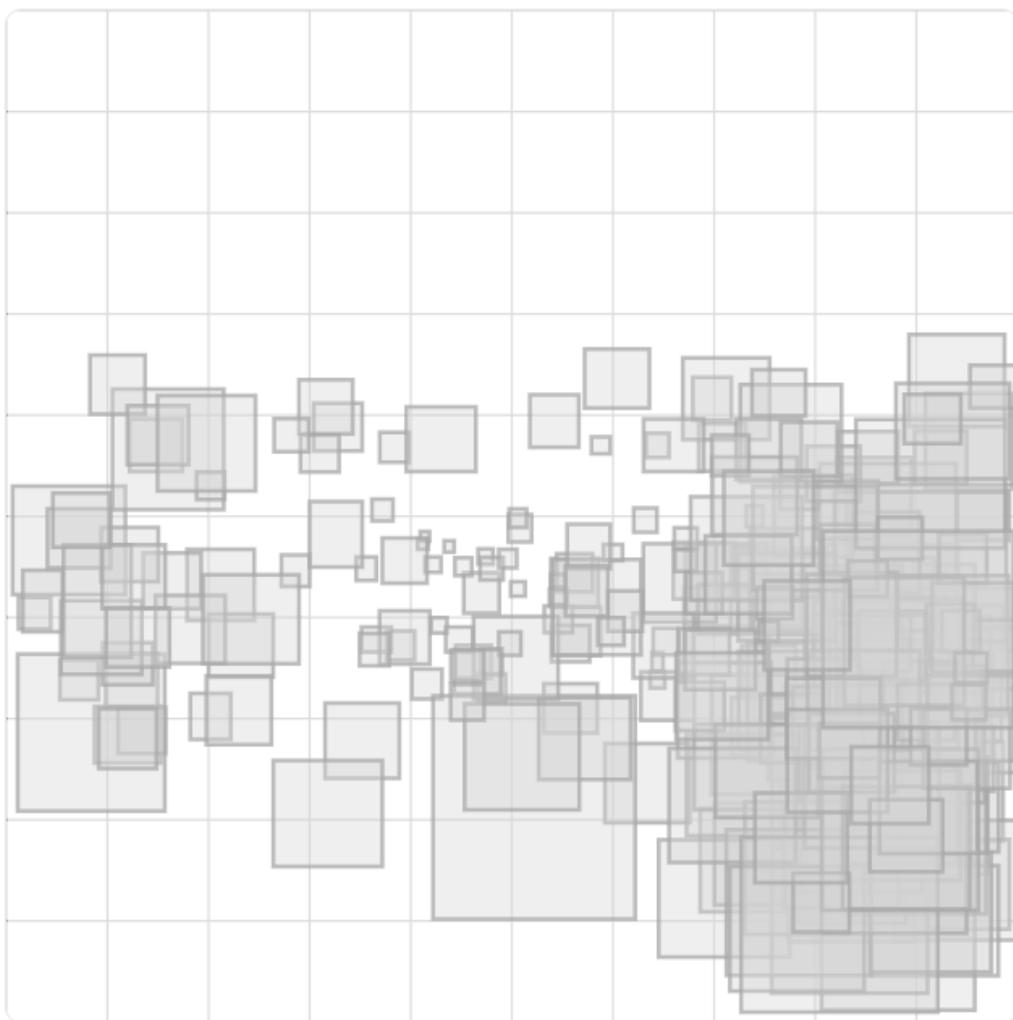
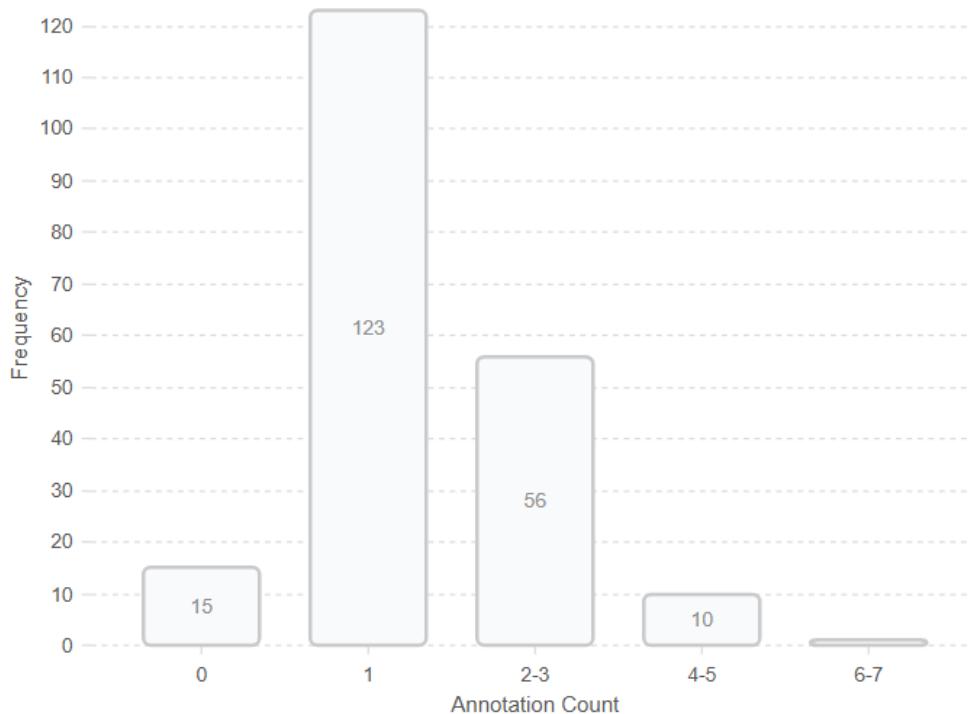


Figura 3.7: Annotation Heatmap delle annotazioni del test set

3.2.3 Istogramma del numero di annotazioni per immagine nel test set

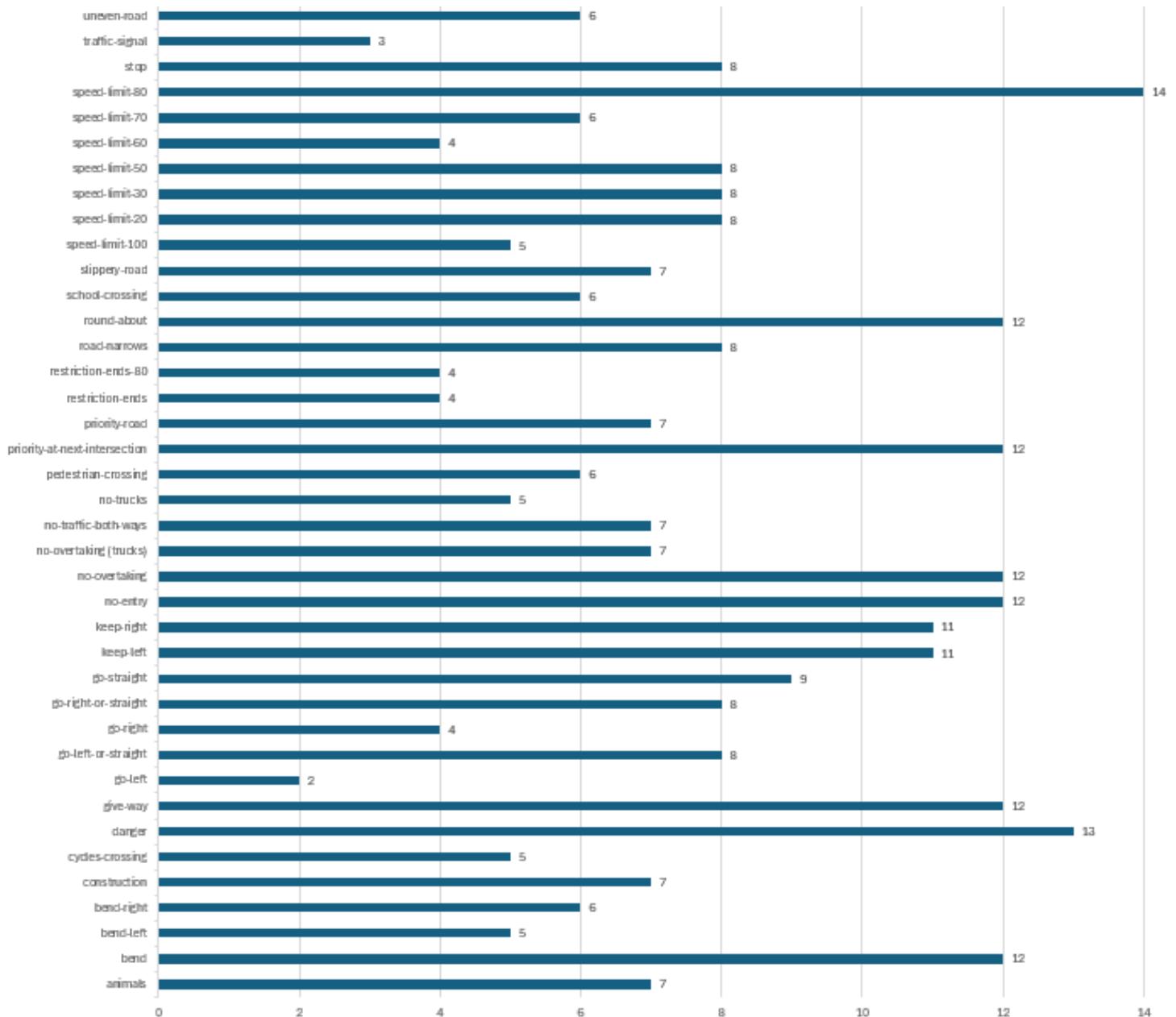
L'**istogramma del numero di annotazioni** da una panoramica su quante classi sono annotate in ognuna delle immagini.

In questo caso, si può notare che **in gran parte delle immagini è annotato un unico segnale**. Questo perchè nella maggior parte delle situazioni reali i segnali sono posti a distanza tra loro e singolarmente.



3.2.4 Istogramma del numero di annotazioni per classe nel test set

Di seguito è riportato un istogramma che mostra il numero di occorrenze di ogni singola classe nel dataset di test.



3.2.5 Dati di esempio del test set

Al fine di comprendere la tipologia di immagini che formano i dataset utilizzati, sono di seguito riportate alcune coppie di immagini di esempio: ogni coppia rappresenta **una delle immagini del test set e la stessa immagine con evidenziati i bounding box ground-truth**. Questi ultimi sono stati generati, a partire dalle annotazioni in formato .txt, usando il codice 8.4 riportato nella sezione "Appendice".



Figura 3.8: Test Image 47. Segnale di "stop" leggermente deformato, e condizioni di luce non ottimali



Figura 3.9: Test Image 124. Segnali in primo piano ben visibili, mentre i segnali più lontani sono piccoli e non facilmente riconoscibili



Figura 3.10: Test Image 147. Pioggia e illuminazione non favorevole in questo caso rendono non ben visibili i segnali

3.3 Pre-processing dei dataset

Sono state necessarie diverse operazioni di pre-processing su entrambi i dataset, al fine di rendere le immagini "utilizzabili" ai fini del training e del testing dei modelli YOLOv10.

3.3.1 Pre-processing sul dataset GTSDB

Le principali operazioni di pre-processing sul dataset GTSDB effettuate sono:

- **Modifica del formato delle immagini, da PPM a JPG**
- **Resize a dimensioni 640×640 , ovvero la dimensione consigliata in YOLOv10.**
- **Re-mapping dei Class-ID.** Come infatti esposto nella precedente sezione, non è stato possibile acquisire immagini di alcune classi di segnali presenti nel dataset originale. Inoltre, per come progettato, **il tool RoboFlow associa automaticamente alle diverse classi un Class-ID sulla base dell'ordinamento alfabetico delle classi.** Per questo motivo, è stato necessario:
 - **Rimuovere dai file delle annotazioni di GTSDB le righe relative alle classi non acquisite per la fase di test**

- Ri-mappare gli indici delle diverse classi, seguendo il mapping fornito automaticamente da Roboflow. Il remapping degli indici delle 39 classi considerate è il seguente:
 - * **0 = animals**
 - * **1 = bend**
 - * **2 = bend-left**
 - * **3 = bend-right**
 - * **4 = construction**
 - * **5 = cycles-crossing**
 - * **6 = danger**
 - * **7 = give-way**
 - * **8 = go-left**
 - * **9 = go-left-or-straight**
 - * **10 = go-right**
 - * **11 = go-right-or-straight**
 - * **12 = go-straight**
 - * **13 = keep-left**
 - * **14 = keep-right**
 - * **15 = no-entry**
 - * **16 = no-overtaking**
 - * **17 = no-overtaking-trucks**
 - * **18 = no-traffic-both-ways**
 - * **19 = no-trucks**
 - * **20 = pedestrian-crossing**
 - * **21 = priority-at-next-intersection**
 - * **22 = priority-road**
 - * **23 = restriction-ends**
 - * **24 = restriction-ends-80**
 - * **25 = road-narrows**
 - * **26 = roundabout**
 - * **27 = school-crossing**
 - * **28 = slippery-road**
 - * **29 = speed-limit-100**
 - * **30 = speed-limit-20**
 - * **31 = speed-limit-30**

- * **32 = speed-limit-50**
- * **33 = speed-limit-60**
- * **34 = speed-limit-70**
- * **35 = speed-limit-80**
- * **36 = stop**
- * **37 = traffic-signal**
- * **38 = uneven-road**

3.3.2 Pre-processing sul dataset di test

Le principali operazioni di pre-processing effettuate sul dataset di test personalmente acquisito sono:

- **Cambio in un aspect ratio 1:1**
- **Resize a dimensione 640x640**

3.3.3 Codici per il pre-processing

Per maggiori dettagli sulla fase di pre-procesing dei due dataset, si rimanda ai codici 8.1, 8.2 e 8.3 presenti nella sezione "Appendice".

3.4 Organizzazione del dataset per il training

Il training dei modelli di YOLOv10 richiede che la directory contenente il dataset abbia il seguente formato:

```
/ TrafficSigns # dataset_directory
  /images
    /train #immagini nel training set
      image1.jpg
      image2.jpg
      ...
    /val #immagini nel validation set
      image1.jpg
      image2.jpg
      ...
    /test #immagini nel test set
      image1.jpg
      image2.jpg
      ...
  /labels
    /train #label ground-truth delle immagini nel training set
      image1.txt
      image2.txt
      ...
    /val #label ground-truth delle immagini nel validation
      set
      image1.txt
      image2.txt
      ...
    /test #label ground-truth delle immagini nel test set
      image1.txt
      image2.txt
      ...
```

Una volta organizzata in tal modo la directory contenente il dataset, la fase di training prevede in input un **file di configurazione** in formato **YAML**, che riassume **i path in cui si trovano training set, validation set e test set e l'insieme delle possibili classi di appartenenza degli oggetti presenti nelle immagini del dataset**.

Il file YAML appena citato dovrà avere nello specifico la seguente struttura:

```
1 train: dataset_directory/images/train
2 val: dataset_directory/images/val
3 test: dataset_directory/images/test
4
5 nc: N      # N = numero delle classi
6 names: ['classe1', 'classe2', ..., 'classeN']
```

In questo caso, il file YAML ha la seguente struttura (la parte finale, non menzionata nella struttura mostrata in precedenza, viene automaticamente aggiunta da RoboFlow, e tra le varie informazioni riporta il link a cui è possibile visionare il dataset creato, sul loro sito):

```
1 train: ./TrafficSigns/images/train
2 val: ./TrafficSigns/images/val
3 test: ./TrafficSigns/images/test
4
5 nc: 39
6 names: ['animals', 'bend', 'bend-left', 'bend-right', 'construction', 'cycles-crossing', 'danger', 'give-way', 'go-left', 'go-left-or-straight', 'go-right', 'go-right-or-straight', 'go-straight', 'keep-left', 'keep-right', 'no-entry', 'no-overtaking', 'no-overtaking-trucks', 'no-traffic-both-ways', 'no-trucks', 'pedestrian-crossing', 'priority-at-next-intersection', 'priority-road', 'restriction-ends', 'restriction-ends-80', 'road-narrows',
7 'roundabout', 'school-crossing', 'slippery-road', 'speed-limit-100', 'speed-limit-20', 'speed-limit-30', 'speed-limit-50', 'speed-limit-60', 'speed-limit-70', 'speed-limit-80', 'stop', 'traffic-signal', 'uneven-road']
8
9 roboflow:
10   workspace: uniworkspace-otguk
11   project: yolov10-traffic-sign-detection
12   version: 8
13   license: CC BY 4.0
14   url: https://universe.roboflow.com/uniworkspace-otguk/yolov10-traffic-sign-detection/dataset/8
```

Capitolo 4

Valutazione del modello

Nel presente capitolo sono elencate le metriche di valutazione, sia in termini di veri e propri valori numerici che grafici, utilizzate ai fini della valutazione dei modelli YOLO addestrati.

4.1 Metriche di valutazione

Le metriche utilizzate per i benchmarking del presente progetto sono alcune delle metriche più utilizzate nell'ambito della valutazione di algoritmi di Object Detection. In particolare, sono state usate le seguenti metriche:

- **Precision:** misura la proporzione di predizioni positive corrette rispetto a tutte le predizioni positive effettuate dal modello. Viene definita come il rapporto tra i True Positive e la somma dei True Positive e False Positive:

$$Precision = \frac{TruePositive}{TruePositive + FalsePositive}$$

- **Recall:** detta anche "sensitivity", misura la proporzione di rilevamenti positivi corretti rispetto a tutte le istanze effettivamente positive. Viene definita come il rapporto tra i True Positive e la somma dei True Positive e False Negative:

$$Recall = \frac{TruePositive}{TruePositive + FalseNegative}$$

- **F1-score:** è una metrica che tiene conto sia della precision che della recall. E' molto utile nel caso in cui si vogliano bilanciare precision

e recall, in quanto penalizza valori fortemente negativi di precision o recall. Viene calcolata come una media armonica tra precision e recall:

$$F_1 = 2 \times \frac{Precision \times Recall}{Precision + Recall}$$

- **Valore di confidenza:** detto anche **confidence score**, è un numero compreso tra 0 e 1, associato ad ogni predizione effettuata da un modello di Machine Learning, che indica la probabilità che quest'ultima sia corretta.

Nel caso di un modello di Object Detection come YOLO, essa indica la probabilità che la classe associata dal modello all'oggetto individuato sia corretta.

- **Curva Precision-Recall:** grafico utilizzato per valutare le prestazioni di un modello. Questa curva mostra la variazione della precision in funzione della recall.
- **Average Precision:** misura l'area al di sotto della curva precision - recall. Viene calcolata come segue:

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

- **Intersection over Union (IoU):** è una metrica di valutazione usata per misurare l'accuracy di un modello di Object Detection. Al fine di calcolarne il valore, si devono considerare:
 - Bounding box ground-truth
 - Bounding box predetti

Come dice il nome stesso, è definita come il rapporto tra l'intersezione e l'unione tra l'area occupata dal bounding box reale e dal bounding box predetto. In altre parole, calcola il rapporto tra l'area di overlap tra i due bounding box e la loro unione:

$$IoU = \frac{IntersectionArea}{UnionArea}$$

Nella pratica, la metrica IoU assume valori elevati nel momento in cui il bounding box predetto è quanto più uguale possibile al bounding box reale, sia in termini di dimensione che di posizione.

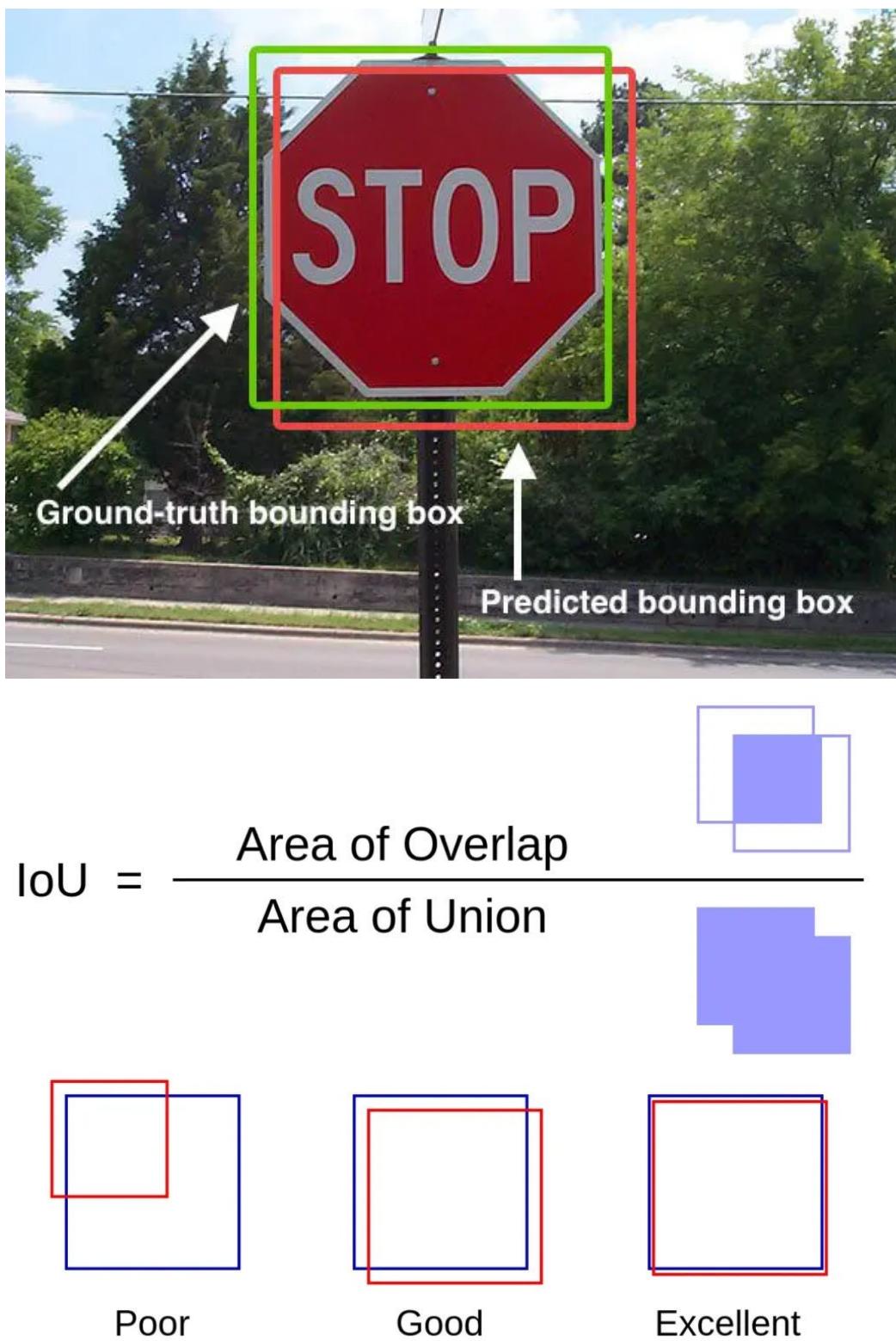


Figura 4.1: Esempio di calcolo della misura di Intersection Over Union

- **Mean Average Precision - mAP:** media delle AP calcolate su tutte le classi presenti nel dataset. Viene calcolata come segue:

$$mAP = \frac{1}{N} \sum_i^N AP_i$$

con N numero di classi. Nel presente progetto saranno usate due varianti:

- **mAP50:** misura la mAP impostando una soglia di **”Intersection over Union (IoU)”** pari a 0.5
- **mAP75:** misura la mAP impostando una soglia di **”Intersection over Union (IoU)”** pari a 0.75
- **mAP50-95:** misura la mAP impostando una soglia minima di **”Intersection over Union (IoU)”** pari a 0.5 e una soglia massima pari a 0.95

4.2 Grafici

I grafici usati ai fini della valutazione dei modelli addestrati sono i seguenti:

- **Matrice di confusione:** è una tabella che mostra il numero di previsioni corrette e errate effettuate da un modello su un set di dati. Viene in genere utilizzato per valutare le prestazioni di un modello di classificazione binaria, ma può anche essere applicato a attività di classificazione multiclasse. In particolare, l'asse delle ascisse rappresenta le classi reali, mentre l'asse delle ordinate rappresenta le classi predette. Nel caso di un problema di classificazione binario, all'interno della matrice di confusione è possibile distinguere:
 - **Veri positivi (TP):** il numero di esempi che sono stati previsti come positivi e sono effettivamente positivi.
 - **Veri negativi (TN):** il numero di esempi che sono stati previsti come negativi e sono effettivamente negativi.
 - **Falsi positivi (FP):** il numero di esempi previsti come positivi ma in realtà negativi.
 - **Falsi negativi (FN):** il numero di esempi previsti come negativi ma in realtà positivi.

		Ground Truth Value	
		True	False
Predicted Value	True	TP True Positive	FP False Positive
	False	FN False Negative	TN True Positive

Figura 4.2: Esempio di matrice di confusione

- **Grafici che mostrano relazioni tra misure di valutazione e score di confidenza:** nella fase di valutazione del modello sono stati utilizzati grafici che mostrano come variano i valori assunti dalle metriche di valutazione al variare di una threshold sul valore di confidenza delle predizioni, ovvero **vengono considerate valide esclusivamente le predizioni la cui confidenza è maggiore rispetto alla soglia impostata**. Segue che maggiore è la soglia scelta, minore sarà il numero di predizioni considerate valide.
 - **Precision-Confidence Curve:** è una rappresentazione visuale della relazione tra precision e confidenza. In particolare, rappresenta il valore di precision ottenuto per diverse threshold di valori di confidenza. In particolare, per valori elevati di soglia di confidenza, il modello risulterà molto preciso, in quanto. impostando una determinata soglia, sono considerate valide solo predizioni con valori di confidenza al di sopra di essa.
 - **Recall-Confidence Curve:** è una rappresentazione visuale della relazione tra recall e confidenza. In particolare, mostra come varia la recall in funzione della variazione della soglia di confidenza. Utilizzando tale grafico, si nota come quando la threshold di confidenza diminuisce, più predizioni sono considerate valide.
 - **F_1 -Confidence Curve:** è una rappresentazione visuale della relazione tra F_1 score e confidenza. In particolare, mostra come

varia il F_1 score in funzione della variazione della soglia di confidenza. Un valore alto di F_1 score indica performance migliori, e di conseguenza, la soglia di confidenza per cui si ottiene tale valore solitamente è la migliore per effettuare predizioni quanto più accurate possibile.

4.3 Loss functions

Sono di seguito riportate le principali funzioni di loss analizzate. Esse in particolare riguardano sia la fase di training che la validazione.

Le principali sono le seguenti:

- **train/box_loss**: indica la loss legata alla localizzazione dei box durante il training.
- **train/cls_loss**: indica la loss legata alle predizioni errate riguardanti le classi di appartenenza degli oggetti identificati, durante il training.
- **train/dfl_loss**: indica la cosiddetta "**Distribution Focal Loss**", e riguarda principalmente alla loss legata a predizioni errate del modello per istanze più "complesse" e meno rappresentate all'interno del dataset, le quali sono "pesate" maggiormente.
- **val/box_loss**: come la *train/box_loss*, ma riguarda stava il validation set
- **val/cls_loss**: come la *train/cls_loss*, ma riguarda stava il validation set
- **val/dfl_loss**: come la *train/dfl_loss*, ma riguarda stava il validation set

Capitolo 5

Metodi di realizzazione del progetto

La realizzazione pratica del progetto avverrà come segue:

La rilevazione dei segnali stradali sarà effettuata tramite apposite tecniche di machine learning, in particolare tramite i modelli della famiglia YOLOv10, descritta in precedenza.

Ai fini dell'addestramento dei 3 modelli scelti, ovvero YOLOv10n, YOLOv10s e YOLOv10m, sarà fatto uso dei due dataset precedentemente descritti:

- Dataset **GTSDB** per la fase di training e validazione
- Dataset **acquisito personalmente da zero** per la fase di testing

Come già detto, etrambi i dataset sono costruiti in modo tale da rappresentare segnali stradali in diverse condizioni atmosferiche, ambientali, di illuminazione e di posa, in maniera tale da rendere quanto più robusto possibile il modello finale.

Le immagini di training saranno poi utilizzate da YOLO per l'addestramento. Durante questo processo, YOLO analizza le immagini e apprende le caratteristiche distintive delle diverse classi di segnali stradali, come la dimensione, la forma e altri dettagli rilevanti.

Al termine della fase di training e di successiva validazione, vengono generati diversi file che contengono i risultati dell'addestramento, tra cui i pesi del modello addestrato, i valori delle metriche di valutazione e alcune batch di esempio con le predizioni del modello.

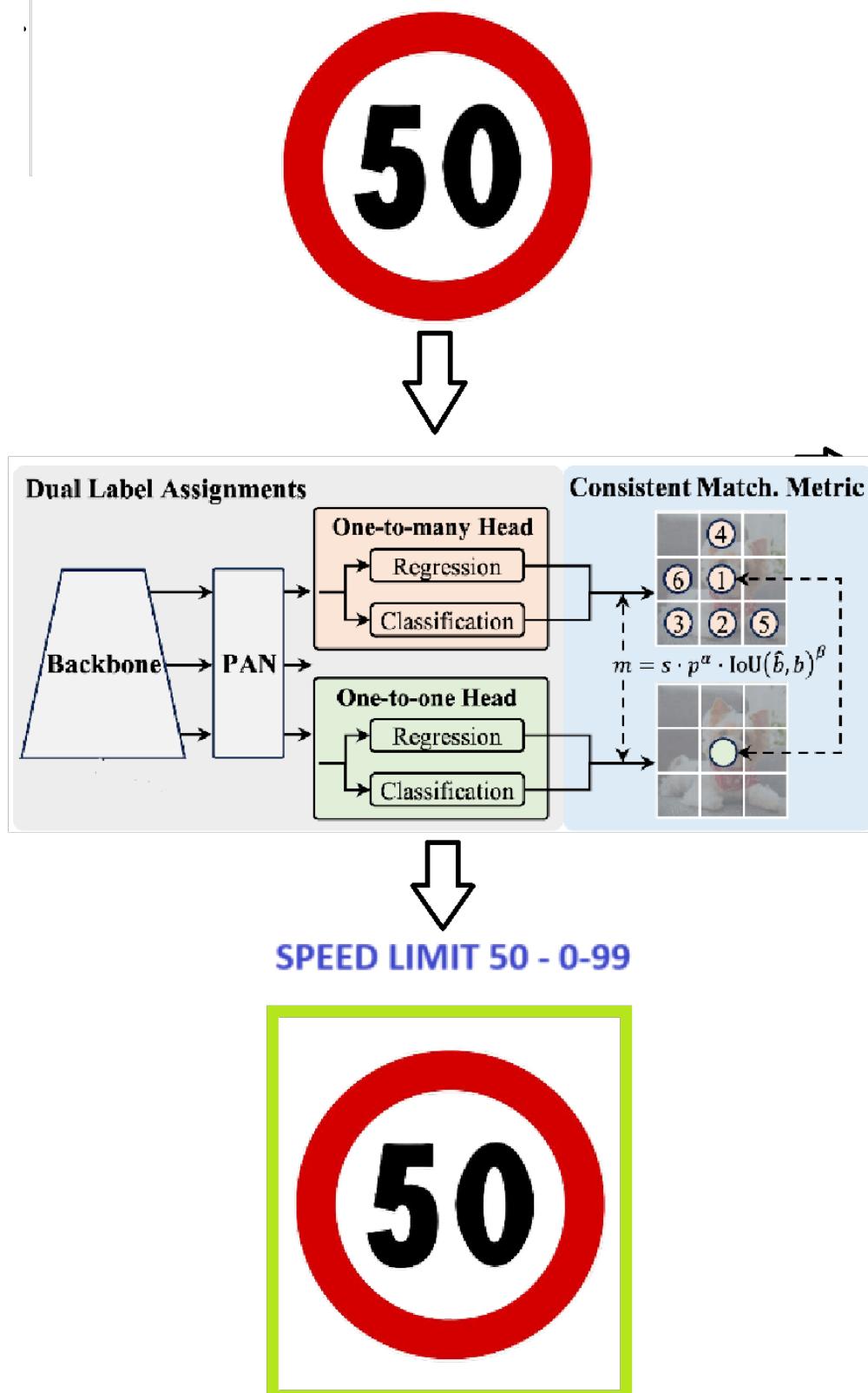


Figura 5.1: Pipeline di funzionamento del modello

Capitolo 6

Training e validazione del modello

Nel presente capitolo sono riportati gli aspetti salienti delle fasi di training e validazione dei diversi modelli di YOLOv10 utilizzando dataset e metriche di valutazione enunciati nei capitoli precedenti.

Sono in particolare riportati i parametri di training, i passi seguiti per effettuare il training e la validazione, inclusi i codici utilizzati, e verranno riportati i principali risultati per mezzo di appositi grafici e tabelle.

Si assume che sul calcolatore sia già installata una distribuzione Python.

N.B. Tutti i codici indicati nella presente relazione possono essere scaricati dal repository remoto che ospita il progetto, incluso un notebook di esempio su come effettuare training, validazione e test dei modelli.

6.1 Parametri di training

Sono di seguito riportati alcuni dei parametri più importanti, utilizzati per la fase di training dei diversi modelli di YOLOv10.

Si noti che:

- I parametri relativi al numero di epoch e Patience sono stati scelti tenendo in considerazione la disponibilità limitata di risorse computazionali per effettuare il fine-tuning dei modelli
- I parametri relativi a Learning Rate, Momentum e Weight Decay sono stati scelti sulla base del paper originale di YOLOv10

Parametro	Valore di default	A cosa serve
<i>epochs</i>	500	Numero di epoche
<i>patience</i>	100	numero di epoche prima di applicare early stopping
<i>lr</i>	0.01	Learning rate
<i>weight_decay</i>	0.0005	Valore per la regolarizzazione L2 "Weight Decay"
<i>imgsz</i>	640	Dimensione delle immagini (640×640)
<i>optimizer</i>	auto	Ottimizzatore (ADAM, SGD, etc.)

Tabella 6.1: Parametri di training principali

6.2 Organizzazione della directory per il training

La directory in cui sarà eseguito il codice che permette di effettuare il training dovrà avere la seguente struttura:

```

1 / TrafficSigns # dataset_directory
2 / weights # directory con i pesi originali dei modelli YOLO
3 / yolov10
4 / data.yaml
5 / dataVal.yaml #ulteriore file YAML, usato per la validazione
6 / requirements.txt # file con la lista di librerie da installare

```

Tutti i file e le directory indicate nel precedente schema sono scaricabili dal repository remoto che ospita i codici relativi al presente progetto.

6.3 Fase di training

Facendo uso della shell messa a disposizione dal sistema operativo, spostarsi nella directory contenente i file scaricati dal repository remoto che ospita il progetto.

Una volta fatto, è possibile seguire i passi elencati di seguito al fine di ripetere la fase di training dei diversi modelli di YOLOv10:

- **Installazione delle librerie richieste:** le librerie possono essere facilmente installate attraverso il seguente comando:

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

il quale installerà tutte le librerie richieste per l'esecuzione del progetto, appositamente indicate all'interno del file *requirements.txt*.

- **Training dei diversi modelli:** utilizzando il seguente codice è possibile effettuare il training dei modelli YOLOv10 usando i dataset menzionati nelle sezioni precedenti:

```

1 from ultralytics import YOLO
2
3 # Training del modello YOLOv10n
4 model = YOLO("weights/yolov10n.pt")
5 model.train(data="data.yaml", epochs=500, imgsz= 640)
6
7 # Training del modello YOLOv10s
8 model2 = YOLO("weights/yolov10s.pt")
9 model2.train(data="data.yaml", epochs=500, imgsz= 640)
10
11 # Training del modello YOLOv10m
12 model3 = YOLO("weights/yolov10m.pt")
13 model3.train(data="data.yaml", epochs=500, imgsz= 640)

```

I risultati del training dei diversi sono solitamente salvati in una nuova directory denominata ***"runs"***, all'interno della quale, per ognuno dei modelli allenati, sono presenti informazioni riguardanti i risultati del training, tra cui:

- **Grafici e metriche di valutazione**
- **Validation batch con esempi di predizioni effettuate dal modello allenato**
- **Checkpoint con i pesi ottimali**
- **Checkpoint con i pesi relativi all'ultima epoca**

Per comodità, all'interno del repository è possibile ritrovare le cartelle **YOLOv10nTRAINED**, **YOLOv10sTRAINED** e **YOLOv10mTRAINED**, le quali contengono i risultati del training effettuato dal sottoscritto.

- **Validazione del modello:** la fase successiva prevede la validazione del modello, sulla base del test set acquisito dal sottoscritto lungo le strade di Catania.

Come si può notare dal codice di seguito riportato, è stato scelto di usare, per comodità, un nuovo file ***"dataVal.yaml"*** che indica la directory in cui si trovano le immagini del test set. La sua struttura è la seguente:

```

1 train: ./TrafficSigns/images/train
2 val: ./TrafficSigns/images/test
3
4 nc: 39
5 names: ['animals','bend','bend-left', 'bend-right', ,
       'construction','cycles-crossing', 'danger', 'give-way', ,
       'go-left','go-left-or-straight', 'go-right', 'go-right-or-
       -straight', 'go-straight', 'keep-left', 'keep-right', ,
       'no-entry', 'no-overtaking',
6 'no-overtaking-trucks-', 'no-traffic-both-ways', 'no-trucks
     ','pedestrian-crossing', 'priority-at-next-intersection
     ','priority-road', 'restriction-ends', 'restriction-ends
     -80', 'road-narrows', 'roundabout', 'school-crossing', ,
     'slippery-road', 'speed-limit-100', 'speed-limit-20', ,
     'speed-limit-30', 'speed-limit-50', 'speed-limit-60', ,
     'speed-limit-70', 'speed-limit-80', 'stop', 'traffic-
     signal', 'uneven-road']

```

Il codice utilizzato per effettuare la validazione è il seguente:

```

1 import numpy as np
2
3 newDatasetValN = YOLO("/runs/detect/train/weights/best.pt")
4 metricsN = newDatasetValN.val(data="dataVal.yaml")
5 print("\n\nmAP50-95:",metricsN.box.map) # map50-95
6 print("mAP-50:",metricsN.box.map50) # map50
7 print("mAP-75:",metricsN.box.map75) # map75
8 print("Average Precision: ", np.mean(metricsN.box.p)) # precision
9 print("Average Recall: ",np.mean(metricsN.box.r)) # recall
10 print("Average F1: ",np.mean(metricsN.box.f1))

```

Tale codice effettua in particolare la validazione del modello "YOLOv10n", e permette di visualizzare a riga di comando:

- Numero di occorrenze per singola classe nelle immagini
- Box Precision
- Box Recall
- Box mAP50
- Box mAP75
- Box mAP50-95
- Precision media su tutte le classi
- Recall media su tutte le classi
- F_1 score medio su tutte le classi

Sono di seguito riportati anche i codici per la validazione di YOLOv10s e YOLOv10m:

```

1 # Validazione di YOLOv10s
2
3 newDatasetVals = YOLO("/runs/detect/train2/weights/best.pt")
4 metricsS = newDatasetVals.val(data="dataVal.yaml")
5 print("\n\nmAP50-95:",metricsS.box.map) # map50-95
6 print("mAP-50:",metricsS.box.map50) # map50
7 print("mAP-75:",metricsS.box.map75) # map75
8 print("Average Precision: ", np.mean(metricsS.box.p)) # precision
9 print("Average Recall: ",np.mean(metricsS.box.r)) # recall
10 print("Average F1: ",np.mean(metricsS.box.f1))
11
12 # Validazione di YOLOv10m
13 newDatasetValM = YOLO("/runs/detect/train3/weights/best.pt")
14 metricsM = newDatasetValM.val(data="dataVal.yaml")
15 print("\n\nmAP50-95:",metricsM.box.map) # map50-95
16 print("mAP-50:",metricsM.box.map50) # map50
17 print("mAP-75:",metricsM.box.map75) # map75
18 print("Average Precision: ", np.mean(metricsM.box.p)) # precision
19 print("Average Recall: ",np.mean(metricsM.box.r)) # recall
20 print("Average F1: ",np.mean(metricsM.box.f1))

```

E' inoltre possibile generare dei grafici di confronto dei diversi modelli allenati, sulla base delle diverse metriche di valutazione ottenute, attraverso il seguente codice:

```

1 # Confronto per valori di mAP
2
3 import matplotlib.pyplot as plt
4
5 metriche = ['mAP-95', 'mAP-50', 'mAP-75']
6 modelli = ['Yolov10n', 'Yolov10s', 'Yolov10m']
7 valori = np.array([
8     [metricsN.box.map, metricsS.box.map, metricsM.box.map], # Valori per mAP-95
9     [metricsN.box.map50, metricsS.box.map50, metricsM.box.map50], # Valori per mAP-50
10    [metricsN.box.map75, metricsS.box.map75, metricsM.box.map75] # Valori per mAP-75
11 ])
12
13 num_metriche = len(metriche)
14 num_modelli = len(modelli)
15 larghezza_colonna = 0.2
16 spazio = 0.1

```

```

17 posizioni = np.arange(num_metriche) * (num_modelli *
18     larghezza_colonna + spazio)
19 fig, ax = plt.subplots(figsize=(10, 6))
20
21 for i in range(num_modelli):
22     ax.bar(posizioni + i * larghezza_colonna, valori[:, i],
23             larghezza_colonna, label=modelli[i])
24
25 ax.set_xlabel('Metriche')
26 ax.set_ylabel('Valore')
27 ax.set_title('Confronto delle metriche mAP tra modelli')
28 ax.set_xticks(posizioni + (num_modelli - 1) *
29                 larghezza_colonna / 2)
30 ax.set_xticklabels(metriche)
31 ax.legend(title='Modelli')
32
33
34
35 # Confronto per valori di Precision, Recall e F1-score
36
37 metriche = ['AVG.Precision', 'Avg.Recall', 'Avg.F1']
38 modelli = ['Yolov10n', 'Yolov10s', 'Yolov10m']
39 valori = np.array([
40     [np.mean(metricsN.box.p), np.mean(metricsS.box.p), np.
41      mean(metricsM.box.p)],           # Valori per Precision
42     [np.mean(metricsN.box.r), np.mean(metricsS.box.r), np.
43      mean(metricsM.box.r)],           # Valori per Recall
44     [np.mean(metricsN.box.f1), np.mean(metricsS.box.f1), np.
45      mean(metricsM.box.f1)]          # Valori per F1-score
46 ])
47
48 num_metriche = len(metriche)
49 num_modelli = len(modelli)
50 larghezza_colonna = 0.2
51 spazio = 0.1
52 posizioni = np.arange(num_metriche) * (num_modelli *
53     larghezza_colonna + spazio)
54
55 fig, ax = plt.subplots(figsize=(10, 6))
56
57 for i in range(num_modelli):
58     ax.bar(posizioni + i * larghezza_colonna, valori[:, i],
59             larghezza_colonna, label=modelli[i])
60
61 ax.set_xlabel('Metriche')
62 ax.set_ylabel('Valore')

```

```

58 ax.set_title('Confronto delle metriche di Precision, Recall
59     e F1-score tra modelli')
60 ax.set_xticks(posizioni + (num_modelli - 1) *
61                 larghezza_colonna / 2)
62 ax.set_xticklabels(metriches)
63 ax.legend(title='Modelli')
64
65 plt.tight_layout()
66 plt.show()
67 plt.close()

```

- **Testing del modello:** una volta ultimata la fase di validazione, è possibile testare il modello su nuove immagini. Ciò può essere fatto in due modi:

- Utilizzando l'**applicazione Demo** inclusa nel progetto, il cui utilizzo è illustrato nella sezione successiva
- Usando il seguente codice, anch'esso nel notebook di esempio:

```

1 import cv2
2 from PIL import Image
3 from google.colab.patches import cv2_imshow
4 from IPython.display import display, HTML, clear_output
5 import os
6
7 # Visualizza il risultato della predizione sulla i-
8 # esima immagine della cartella specificata
9 folder = "./TrafficSigns/images/test"
10 i=0
11 modelTest = YOLO("./runs/detect/train/weights/best.pt")
12 # use YOLOv10n
13 #modelTest = YOLO("./runs/detect/train2/weights/best.pt"
14 #") # use YOLOv10s
15 #modelTest = YOLO("./runs/detect/train3/weights/best.pt"
16 #") # use YOLOv10m
17
18 image_files = sorted([f for f in os.listdir(folder) if
19 f.endswith('.jpg', '.png', '.jpeg')])
20
21 if 0 <= i < len(image_files):
22     image_path = os.path.join(folder, image_files[i])
23     # se si vuole semplicemente effettuare la
24     # prediction, senza salvare i risultati, e senza
25     # impostare soglie di IoU/confidence, eseguire la
26     # seguente istruzione
27     #results = modelTest("datasets/TrafficSigns/images/
28 #test")

```

```
21
22     # se per ognuno dei file si vuole conservare la
23     # nuova immagine con i bounding box stampati in esso,
24     # eseguire invece la seguente istruzione
25     #results = modelTest("datasets/TrafficSigns/images/
26     #test", save=True)
27
28     # se si vogliono impostare delle soglie di IoU e
29     # confidenza per scartare risultati incerti, eseguire
30     # invece il seguente codice:
31     results = modelTest(image_path, save=False, iou
32     =0.5, conf=0.5)
33
34     noPredImage = cv2.imread(results[0].path, cv2.
35     COLOR_BGR2RGB)
36     noPredImage = cv2.resize(noPredImage , (350,350))
37     image = results[0].orig_img
38     boxes = []
39     for box in results[0].boxes:
40         x1, y1, x2, y2 = box.xyxy[0].tolist()
41         confidence = box.conf[0].item()
42         class_id = int(box.cls[0].item())
43         class_name = results[0].names[class_id]
44         boxes.append((x1, y1, x2, y2, confidence,
45         class_name))
46
47     boxes = sorted(boxes, key=lambda x: x[4], reverse=
48     True)
49
50     for (x1, y1, x2, y2, confidence, class_name) in
51     boxes:
52         label = f'{class_name} - {round(confidence,2)}'
53         cv2.rectangle(image, (int(x1), int(y1)), (int(x2)
54         , int(y2)), (0, 255, 0), 2)
55
56         font = cv2.FONT_HERSHEY_DUPLEX
57         font_scale = 0.7
58         font_thickness = 1
59         (text_width, text_height), baseline = cv2.
60         getTextSize(label, font, font_scale, font_thickness)
61
62         text_x = int(x1 + (x2 - x1 - text_width) / 2)
63         text_y = int(y1)-10
64
65         cv2.putText(image, label, (text_x, text_y), font,
66         font_scale, (255, 0, 0), font_thickness)
67
68         cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
69         image = cv2.resize(image, (350,350))
```

```
57
58     groundThruthImage = cv2.imread(results[0].path.
59         replace('/images', '/groundThruthAnnotations'), cv2.
60         COLOR_BGR2RGB)
61     groundThruthImage = cv2.resize(groundThruthImage ,
62         (350,350))
63
64     (h, w) = image.shape[:2]
65
66     text = "Original"
67     (text_width, text_height), baseline = cv2.
68     getTextSize(text, font, font_scale, font_thickness)
69     text_x = (w - text_width) // 2
70     text_y = text_height + 20
71     cv2.putText(noPredImage, text, (text_x, text_y),
72         font, font_scale, (255,255,255), font_thickness)
73
74     text = "Ground-thruth"
75     (text_width, text_height), baseline = cv2.
76     getTextSize(text, font, font_scale, font_thickness)
77     text_x = (w - text_width) // 2
78     text_y = text_height + 20
79     cv2.putText(groundThruthImage, text, (text_x, text_y),
80         font, font_scale, (255,255,255), font_thickness)
81
82     text = "Predictions"
83     (text_width, text_height), baseline = cv2.
84     getTextSize(text, font, font_scale, font_thickness)
85     text_x = (w - text_width) // 2
86     text_y = text_height + 20
87     cv2.putText(image, text, (text_x, text_y), font,
88         font_scale, (255,255,255), font_thickness)
89
90     # mostra in ordine: immagine originale, box/classi
91     # ground thruth, box/classi predetti
92
93     concat = cv2.hconcat([noPredImage,
94         groundThruthImage, image])
95     cv2.imshow(concat)
96     cv2.waitKey(0)
97     cv2.destroyAllWindows()
98
99 else:
100     print(f"Index {i} out of bound. {len(image_files)}")
101     images found in folder: {folder}.\n Please insert a
102     value from 0 to {len(image_files) - 1}.")
```



Figura 6.1: Esempio di output del codice precedente

6.4 App Demo

Il progetto ha previsto lo sviluppo di un'applicazione in linguaggio Python avente le seguenti funzionalità:

- **Caricamento di un'immagine su cui effettuare la predizione**
- **Scelta del modello con cui effettuare la predizione**
- **Visualizzazione grafica dei risultati:** una volta effettuata la predizione sull'immagine in input, è possibile visualizzare una accanto all'altra:
 - Immagine originale
 - Immagine con classi e box ground-truth
 - Immagine con classi e box predetti dal modello scelto, con accanto il relativo **valore di confidenza**
- **Lista delle classi trovate e il relativo numero di occorrenze**, in basso

6.5 Organizzazione delle immagini per utilizzare l'app

Al fine di poter utilizzare l'app per la rilevazione di segnali in nuove immagini, è necessario prima di tutto organizzare la directory delle immagini come segue:

```
/ appDirectory # directory dove si trova l'app
    /images #directory delle immagini di cui effettuare la
    detection
        image1.jpg
        image2.jpg
        ...
    /groundThruthAnnotations #directory con le immagini in
    cui sono disegnati i bounding-box ground-thruth
        image1.jpg
        image2.jpg
        ...
```

E' importante che **le immagini e le rispettive annotazioni ground-truth abbiano lo stesso file name**.

A titolo di esempio, all'interno della cartella *datasets* presente nel progetto sono già state caricate le immagini con annotazioni ground-truth relative alle immagini presenti nei dataset utilizzati, all'interno della cartella **"groundThruthAnnotations"**.

Nel caso in cui si vogliano generare immagini ground-truth relative ad altre immagini, è possibile usare lo script python ***printGroundThruthAnnotation.py*** presente nella cartella **"Utils"** contenuta nel repository. L'esecuzione di questo script richiede le immagini originali e dei file .txt contenenti le annotazioni ground-truth delle diverse immagini, organizzate in maniera esattamente identica a come richiesto da YOLO.

6.6 Avviare l'app

L'app può essere avviata eseguendo nella directory del progetto il seguente comando:

```
python trafficSignsDetectionApp.py
```

Una volta avviata, i passi da effettuare per poter ottenere predizioni su immagini sono i seguenti:

1. **Caricare l'immagine su cui effettuare la predizione**, tramite il pulsante **"Load Image"**

2. **Caricare il modello da usare per effettuare la predizione, tramite la listbox "Select YOLO model"**
3. **In caso di errore, rimuovere l'immagine caricata, tramite il pulsante "Remove image"**
- 4. Effettuare la predizione, tramite il pulsante "Predict"**

All'interno della cartella "**assets**" nel repository remoto è possibile trovare un video esplicativo del funzionamento dell'applicazione.

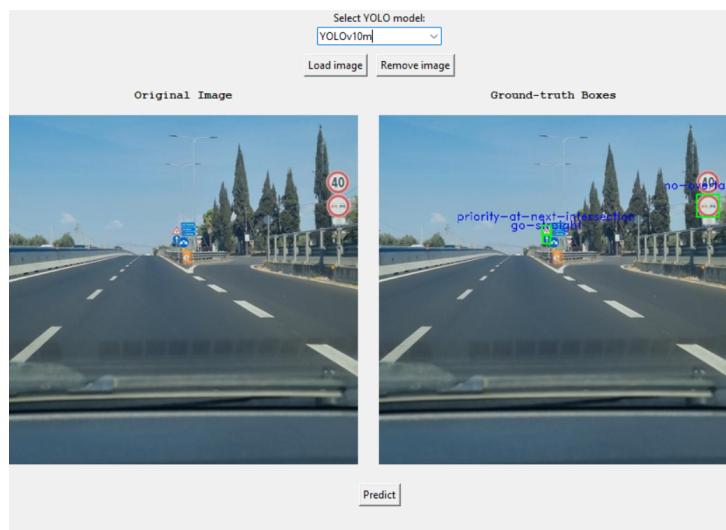


Figura 6.2: Schermata dell'app, prima di effettuare una predizione



Figura 6.3: Schermata dell'app, dopo aver effettuato una predizione

Capitolo 7

Risultati

Il seguente capitolo illustra i risultati ottenuti grazie alla fase di training e validazione dei modelli YOLOv10n, YOLOv10s e YOLOv10m, sui dataset di segnali stradali GTSDB, usato come training set e il Dataset acquisito personalmente, usato invece come test set.

Sono in particolare riportati grafici e tavole che riassumono le principali misure di valutazione calcolate sui diversi modelli, oltre a diverse informazioni riguardanti il training.

Il training è stato effettuato tramite una GPU Nvidia L4 con 24GB di VRAM GDDR6, attraverso l'ambiente di sviluppo "Google Co-laboratory".

7.1 Tabelle riassuntive

Le tabelle seguenti mostrano i risultati della fase di fine-tuning dei modelli di YOLOv10 sul dataset GTSDB, utilizzando i parametri precedentemente elencati.

7.1.1 Architettura dei modelli allenati

Modello	# layer	Training time (h)	Inference speed (ms)
YOLOv10n	285	<u>0.8</u>	3.1
YOLOv10s	293	<u>0.956</u>	5.1
YOLOv10m	369	<u>2.22</u>	11.8

Tabella 7.1: Informazioni sull’architettura dei diversi modelli di YOLOv10 dopo il fine-tuning. I valori sottolineati nella colonna ”Training time” indicano fasi di training in cui è stato applicato Early Stopping. In particolare:

- YOLOv10n: early-stopped all’epoca n.341
- YOLOv10s: early-stopped all’epoca n.239
- YOLOv10m: early-stopped all’epoca n.260

L’early stopping osservato nei tre modelli suggerisce che il training si è interrotto anticipatamente rispetto al numero massimo di epoche previsto. Questo meccanismo entra in gioco quando il modello smette di migliorare in termini di performance su un set di validazione per un certo numero di epoche, in questo caso 100.

Nella pratica, l’early stopping è una strategia che mira a prevenire l’overfitting, evitando che il modello continui ad allenarsi su pattern specifici del dataset di training che non generalizzano bene su dati nuovi. L’obiettivo è fermare l’allenamento quando il modello ha già raggiunto un buon equilibrio tra l’accuratezza sui dati di training e la capacità di generalizzazione, impedendo un calo delle prestazioni sui dati di validazione.

Il fatto che tutti e tre i modelli abbiano attivato l’early stopping prima di raggiungere le 500 epoche potrebbe essere dovuto a diversi fattori. Un possibile motivo è che il dataset di training non è abbastanza complesso o vario per richiedere ulteriori adattamenti da parte dei modelli. In altre parole, i modelli potrebbero aver raggiunto il loro limite di capacità predittiva per quel particolare insieme di dati.

Un altro fattore che potrebbe aver influenzato l’early stopping è la configurazione degli iperparametri, come il learning rate.

7.1.2 Precision, Recall, F_1 -score medie (validation set)

Modello	Precision %	Recall %	F_1 %
YOLOv10n	70.3	50.4	58.7
YOLOv10s	71.5	70.1	70.8
YOLOv10m	77.3	75	76.1

7.1.3 mAP medio (validation set)

Modello	mAP50 %	mAP50-95 %
YOLOv10n	64.9	57
YOLOv10s	78.3	69.4
YOLOv10m	85.1	74.6

Dalle precedenti tabelle, le quali riportano i valori rilevati per metriche quali precision, recall, F_1 score e mAP registrate sul validation set, è possibile giungere alle seguenti conclusioni:

- **YOLOv10n:** Questo modello, il più leggero tra tutti da un punto di vista computazionale, mostra una precision media del 70.3% e una recall media del 50.4%, dalle quali si ricava un valore F_1 score di 58.7%. I valori di mAP sono del 64.9% su mAP50 e del 57% su mAP50-95, indicando buone prestazioni generali ma non ottimali.
- **YOLOv10s:** Il modello YOLOv10s migliora sia la precision (71.5%) che la recall (70.1%) rispetto al modello precedente, con un F_1 score di 70.8%, mostrando quindi un equilibrio migliore nelle rilevazioni. Le prestazioni di mAP50 e mAP50-95 sono rispettivamente 78.3% e 69.4%, e mostra quindi una capacità più elevata di rilevare correttamente i segnali stradali rispetto al modello precedente.
- **YOLOv10m:** YOLOv10m, il modello più grande e performante tra i tre considerati, ottiene la precision più alta (77.3%) e un recall del 75%, con un F1 score di 76.1%. Anche il mAP medio è il più alto tra i tre modelli, con 85.1% su mAP50 e 74.6% su mAP50-95, confermando le prestazioni superiori in termini di accuratezza e generalizzazione rispetto agli altri due modelli.

7.1.4 Precion, Recall, F_1 -score medie (test set)

Modello	Avg. Precision %	Avg. Recall %	Avg. F_1 %
YOLOv10n	38	18	15
YOLOv10s	16.1	46	20.2
YOLOv10m	48.8	29	25.6

Tabella 7.2: Metriche di valutazione Precision, Recall e F_1 -score dei diversi modelli allenati

7.1.5 mAP medio (test set)

Modello	mAP50 %	mAP75 %	mAP50-95 %
YOLOv10n	17.8	14.6	11.9
YOLOv10s	26.7	21.6	18.3
YOLOv10m	30.1	27	21.7

Tabella 7.3: Metriche di valutazione mAP dei diversi modelli allenati

I dati riportati per i modelli YOLOv10n, YOLOv10s e YOLOv10m sul test set evidenziano una performance complessivamente non soddisfacente rispetto alle metriche di precision, recall e F1 score, con valori molto inferiori rispetto a quelli ottenuti sul validation set. Per esempio, il modello YOLOv10n mostra una precision media del 38%, ma un recall estremamente basso del 18%, con un F1 score di appena 15%. Analogamente, YOLOv10s, pur avendo un recall più alto (46%), ha una precision bassa (16.1%) e un F1 score di 20.2%. Il modello YOLOv10m, il più performante, raggiunge una precision del 48.8%, recall del 29% e un F1 score di 25.6%. Anche le metriche di mAP confermano questo trend, con risultati deludenti per mAP50 (30.1% per YOLOv10m), mAP75 (27%) e mAP50-95 (21.7%).

Le scarse prestazioni sul test set possono essere attribuite a diverse motivazioni. Una possibile causa è la differenza tra il set di validazione e il set di test, che potrebbe contenere immagini con condizioni, angolazioni o variazioni non presenti nel set di allenamento, portando a una minore capacità di generalizzazione dei modelli. In particolare, alcuni segnali acquisiti dal sottoscritto potrebbero essere "disegnati" in maniera leggermente diversa rispetto a quelli acquisiti nel dataset GTSDB.

7.1.6 Metriche di valutazione sulle singole classi

Le seguenti tabelle mostrano i dati relativi alle metriche di valutazione (precision, recall, F1 score e mAP) analizzate per le singole classi nei modelli YOLOv10n, YOLOv10s e YOLOv10m addestrati. Questi dati evidenziano le performance specifiche di ciascuna classe di segnaletica stradale considerata, dando quindi la possibilità di analizzare più nel dettaglio il comportamento dei modelli sulle singole classi.

Andando più nel dettaglio, i dati riportati per le diverse classi sui tre modelli mostrano una grande variabilità nelle performance di precision, recall, F1 score e mAP. In particolare, alcune classi come "animals", "go-left-or-straight" e "priority-road" mostrano una precision del 100% in alcuni modelli, ma con valori di recall molto bassi o addirittura pari a 0, indicando che i modelli hanno difficoltà a rilevare correttamente tutte le istanze di queste classi. Altre classi, come "danger" e "give-way", raggiungono risultati equilibrati con buoni valori di precision e recall, e quindi F_1 score elevati. Tuttavia, ci sono numerose classi, come "pedestrian-crossing", "cycles-crossing" e "go-left", dove sia precision che recall sono pari a 0, indicando che i modelli non sono in grado di riconoscere queste categorie. Anche le metriche di mAP50 e mAP50-95 mostrano che alcune classi hanno una performance molto bassa, segnalando difficoltà generali nel riconoscimento accurato delle classi su immagini del test set, con differenze significative tra le varie classi e modelli.

Classe	P %	Recall %	F1 %	mAP50 %	mAP50-95 %
animals	100.0	0.0	0.00	18.0	12.1
bend	18.2	8.33	11.61	7.5	6.03
bend-left	0.0	0.0	0.00	17.3	12.1
bend-right	0.0	0.0	0.00	0.0	0.0
construction	0.0	0.0	0.00	0.78	0.47
cycles-crossing	0.0	0.0	0.00	0.0	0.0
danger	37.1	30.8	33.71	25.8	16.5
give-way	60.1	75.0	66.91	70.7	44.2
go-left	0.0	0.0	0.00	6.1	4.9
go-left-or-straight	100.0	0.0	0.00	0.0	0.0
go-right	18.9	9.44	12.57	14.0	9.78
go-right-or-straight	100.0	0.0	0.00	0.0	0.0
go-straight	100.0	0.0	0.00	5.6	3.94
keep-left	100.0	0.0	0.00	14.0	12.9
keep-right	19.5	45.5	27.36	34.1	23.9
no-entry	51.1	58.3	54.47	70.3	36.3
no-overtaking	49.6	8.33	14.29	10.3	7.49
no-overtaking-trucks-	35.7	14.3	20.41	28.1	22.4
no-traffic-both-ways	37.1	25.8	30.52	37.9	20.0
no-trucks	21.8	40.0	28.39	18.0	9.44
pedestrian-crossing	0.0	0.0	0.00	0.0	0.0
priority-at-next-intersection	26.7	8.33	12.73	14.8	8.72
priority-road	94.1	85.7	89.71	93.0	63.3
restriction-ends	100.0	0.0	0.00	0.0	0.0
restriction-ends-80	100.0	0.0	0.00	0.0	0.0
road-narrows	0.0	0.0	0.00	6.8	5.13
roundabout	53.0	16.7	25.27	22.4	16.9
school-crossing	25.5	16.7	20.14	21.1	18.2
slippery-road	9.77	42.9	15.98	16.0	11.6
speed-limit-100	20.2	60.0	30.30	25.0	16.0
speed-limit-20	100.0	0.0	0.00	0.0	0.0
speed-limit-30	7.82	12.5	9.67	20.7	16.6
speed-limit-50	9.15	62.5	15.91	17.2	14.0
speed-limit-60	12.8	25.0	17.03	7.03	3.51
speed-limit-70	0.0	0.0	0.00	6.4	4.02
speed-limit-80	0.0	0.0	0.00	15.6	9.88
stop	66.0	24.5	35.79	39.1	27.5
traffic-signal	10.6	33.3	16.11	10.9	5.58
uneven-road	0.0	0.0	0.00	1.47	0.88

Tabella 7.4: Precision, recall, F1 e mAP per le diverse classi sul **test set** - modello YOLOv10n

Classe	P %	Recall %	F1 %	mAP50 %	mAP50-95 %
animals	0.0	0.0	0.0	0.0	0.0
bend	33.3	16.7	22.2	27.8	22.7
bend-left	12.5	40.0	18.8	10.2	7.79
bend-right	11.9	83.3	20.5	46.0	29.2
construction	1.6	14.3	2.8	1.2	0.96
cycles-crossing	0.0	0.0	0.0	0.0	0.0
danger	36.0	69.2	47.7	49.4	33.9
give-way	40.0	83.3	54.3	79.0	57.6
go-left	0.0	0.0	0.0	0.0	0.0
go-left-or-straight	0.0	0.0	0.0	0.0	0.0
go-right	7.1	75.0	12.5	9.5	5.19
go-right-or-straight	33.3	12.5	18.4	18.7	13.1
go-straight	4.5	11.1	6.2	2.9	2.02
keep-left	11.8	18.2	14.6	10.9	6.25
keep-right	9.4	90.9	16.2	39.1	27.2
no-entry	30.0	100.0	46.2	90.6	48.0
no-overtaking	20.0	33.3	25.0	16.4	10.0
no-overtaking-trucks-	20.0	57.1	28.6	24.1	20.0
no-traffic-both-ways	28.6	57.1	38.0	60.5	46.1
no-trucks	14.3	40.0	21.4	11.9	7.14
pedestrian-crossing	0.0	0.0	0.0	0.0	0.0
priority-at-next-intersection	25.0	16.7	20.0	22.2	13.9
priority-road	25.9	100.0	41.0	99.5	69.5
restriction-ends	50.0	25.0	32.0	31.2	21.9
restriction-ends-80	25.0	25.0	25.0	17.7	3.53
road-narrows	0.0	0.0	0.0	0.0	0.0
roundabout	29.0	75.0	43.1	72.6	54.1
school-crossing	22.2	33.3	27.4	28.1	25.3
slippery-road	6.6	57.1	10.8	5.74	4.78
speed-limit-100	5.6	60.0	10.7	22.6	15.3
speed-limit-20	14.8	50.0	21.0	36.6	22.1
speed-limit-30	9.4	37.5	13.1	9.3	7.61
speed-limit-50	6.9	87.5	12.4	12.5	10.2
speed-limit-60	2.2	50.0	4.2	7.3	5.12
speed-limit-70	9.1	83.3	16.1	66.0	51.3
speed-limit-80	15.6	100.0	26.7	25.6	17.1
stop	60.0	75.0	67.0	78.7	50.2
traffic-signal	4.5	66.7	8.3	3.9	2.85
uneven-road	5.3	50.0	9.2	4.4	2.92

Tabella 7.5: Precision, recall, F1 e mAP per le diverse classi sul **test set** - modello YOLOv10s

Classe	P %	Recall %	F1 %	mAP50 %	mAP50-95 %
animals	100.0	100.0	100.0	0.0	0.0
bend	100.0	100.0	100.0	0.0	0.0
bend-left	100.0	100.0	100.0	53.4	39.2
bend-right	33.6	66.7	44.4	56.0	36.7
construction	10.4	28.6	15.2	5.7	3.6
cycles-crossing	0.0	0.0	0.0	0.0	0.0
danger	43.6	41.8	42.7	45.2	32.6
give-way	73.0	75.0	74.0	80.4	54.6
go-left	0.0	0.0	0.0	0.0	0.0
go-left-or-straight	100.0	100.0	100.0	0.0	0.0
go-right	0.0	0.0	0.0	3.8	3.1
go-right-or-straight	46.2	11.6	18.8	10.2	8.2
go-straight	60.7	11.1	18.6	21.1	15.3
keep-left	100.0	100.0	100.0	0.0	0.0
keep-right	29.9	72.7	41.8	59.4	39.9
no-entry	72.9	83.3	77.8	83.7	48.5
no-overtaking	62.0	27.4	38.0	49.4	36.5
no-overtaking-trucks-	100.0	57.1	72.4	64.5	55.5
no-traffic-both-ways	36.5	14.3	20.4	14.4	10.1
no-trucks	24.1	38.3	29.9	40.8	27.8
pedestrian-crossing	100.0	0.0	0.0	0.0	0.0
priority-at-next-intersection	56.4	25.0	34.6	27.9	18.1
priority-road	58.3	85.7	69.2	86.9	68.6
restriction-ends	0.0	0.0	0.0	0.0	0.0
restriction-ends-80	0.0	0.0	0.0	0.0	0.0
road-narrows	57.6	18.2	27.4	51.6	39.1
roundabout	100.0	23.4	38.0	50.5	38.1
school-crossing	45.3	33.3	38.0	35.4	29.7
slippery-road	8.2	28.6	12.5	13.4	10.5
speed-limit-100	29.9	12.0	16.6	19.4	14.2
speed-limit-20	100.0	0.0	0.0	32.3	22.1
speed-limit-30	15.1	37.5	21.4	17.6	14.3
speed-limit-50	24.2	75.0	36.8	48.7	40.8
speed-limit-60	10.8	50.0	17.4	12.4	8.7
speed-limit-70	41.2	33.3	37.0	42.5	36.6
speed-limit-80	46.9	78.6	58.4	60.8	41.7
stop	83.9	37.5	50.0	55.7	37.8
traffic-signal	20.6	33.3	25.6	25.7	12.3
uneven-road	14.2	33.3	20.0	8.2	5.7

Tabella 7.6: Precision, recall, F1 e mAP per le diverse classi sul **test set** - modello YOLOv10m

7.2 Loss, matrice di confusione e curve

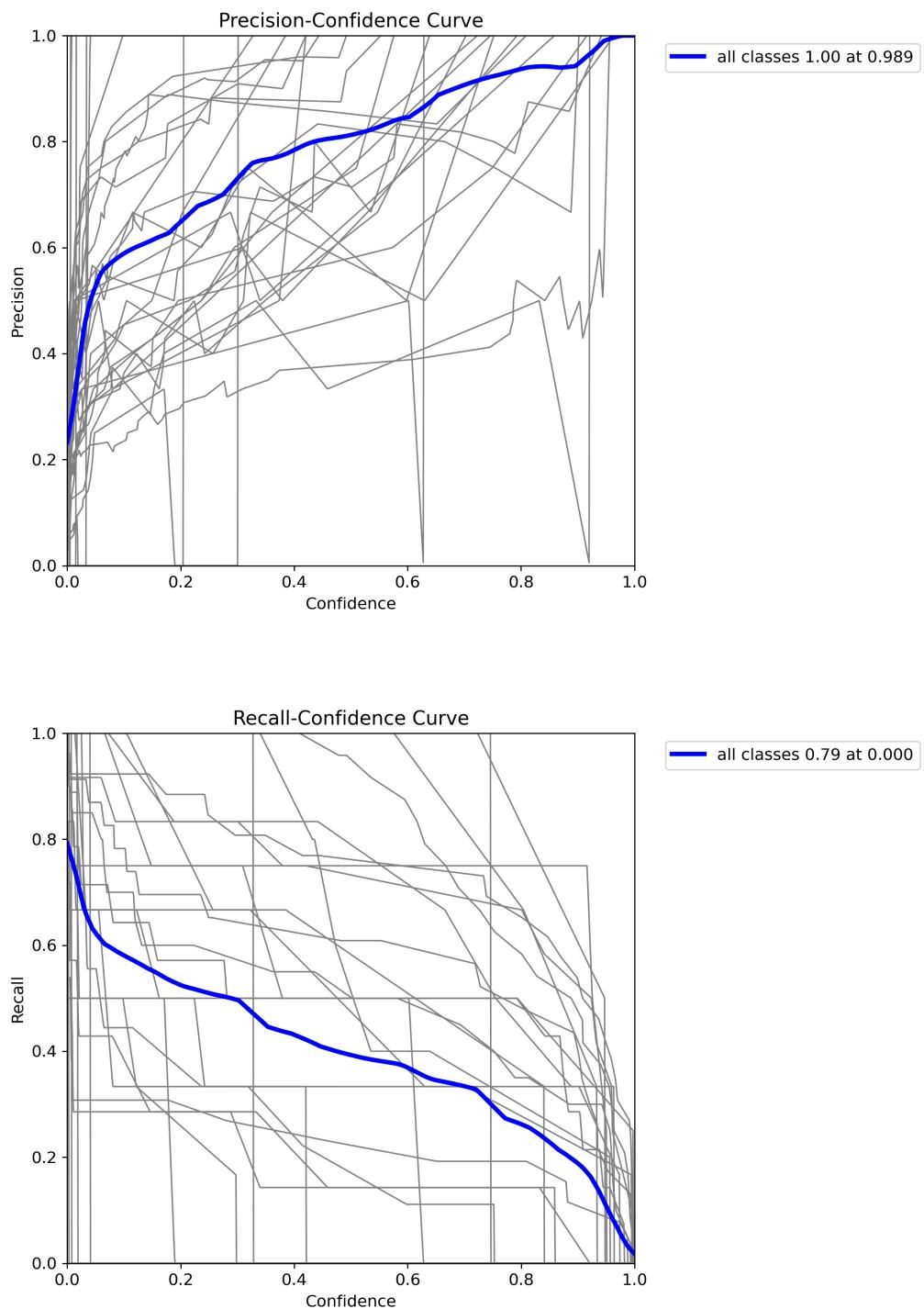
Sono di seguito riportati gli andamenti delle funzioni di loss, delle metriche di valutazione e delle curve relative ai singoli modelli allenati.

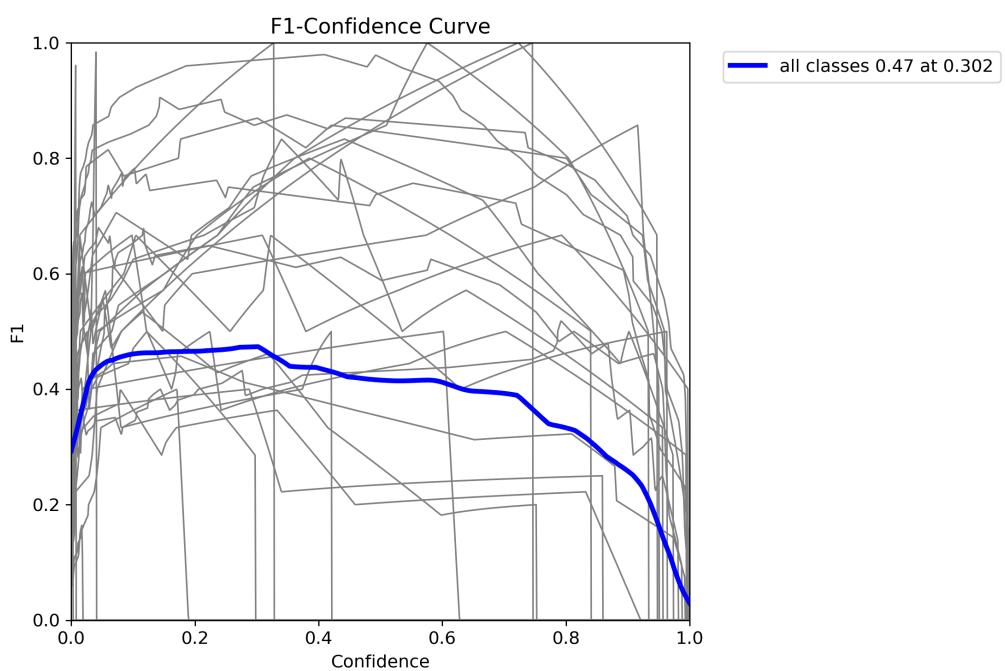
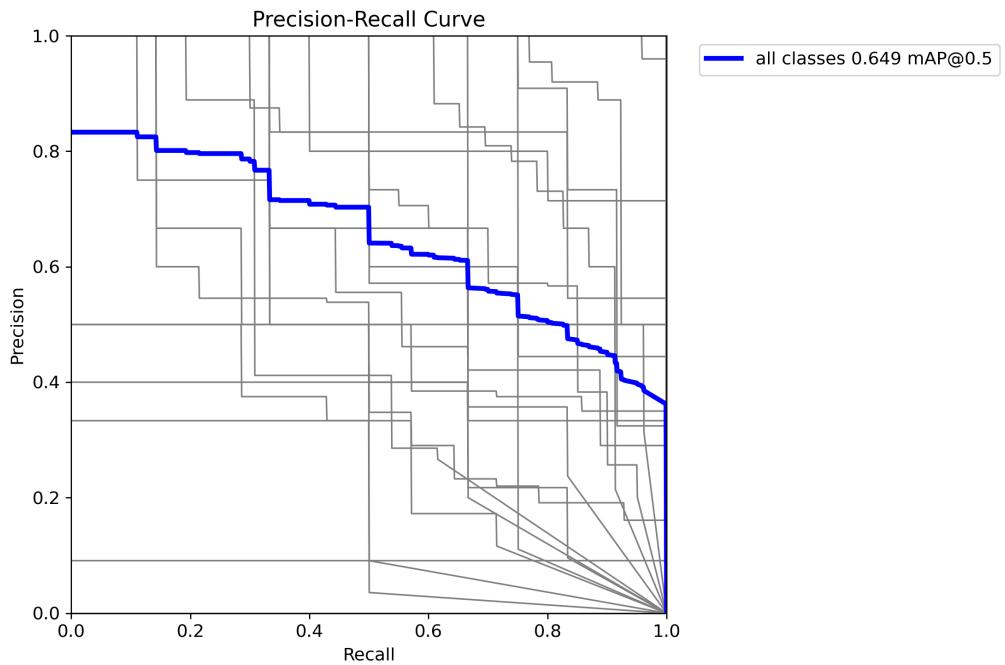
Per quanto riguarda la matrice di confusione dei diversi modelli, le celle sulla diagonale principale rappresentano le classi in cui il modello ha fatto una previsione corretta, mentre le celle al di fuori di essa rappresentano gli errori di classificazione commessi. Si può poi notare la presenza di un’ulteriore classe **”background”**: se il modello predice tale classe per un’immagine, significa che in quell’immagine non è presente alcun oggetto appartenente alle classi considerate.

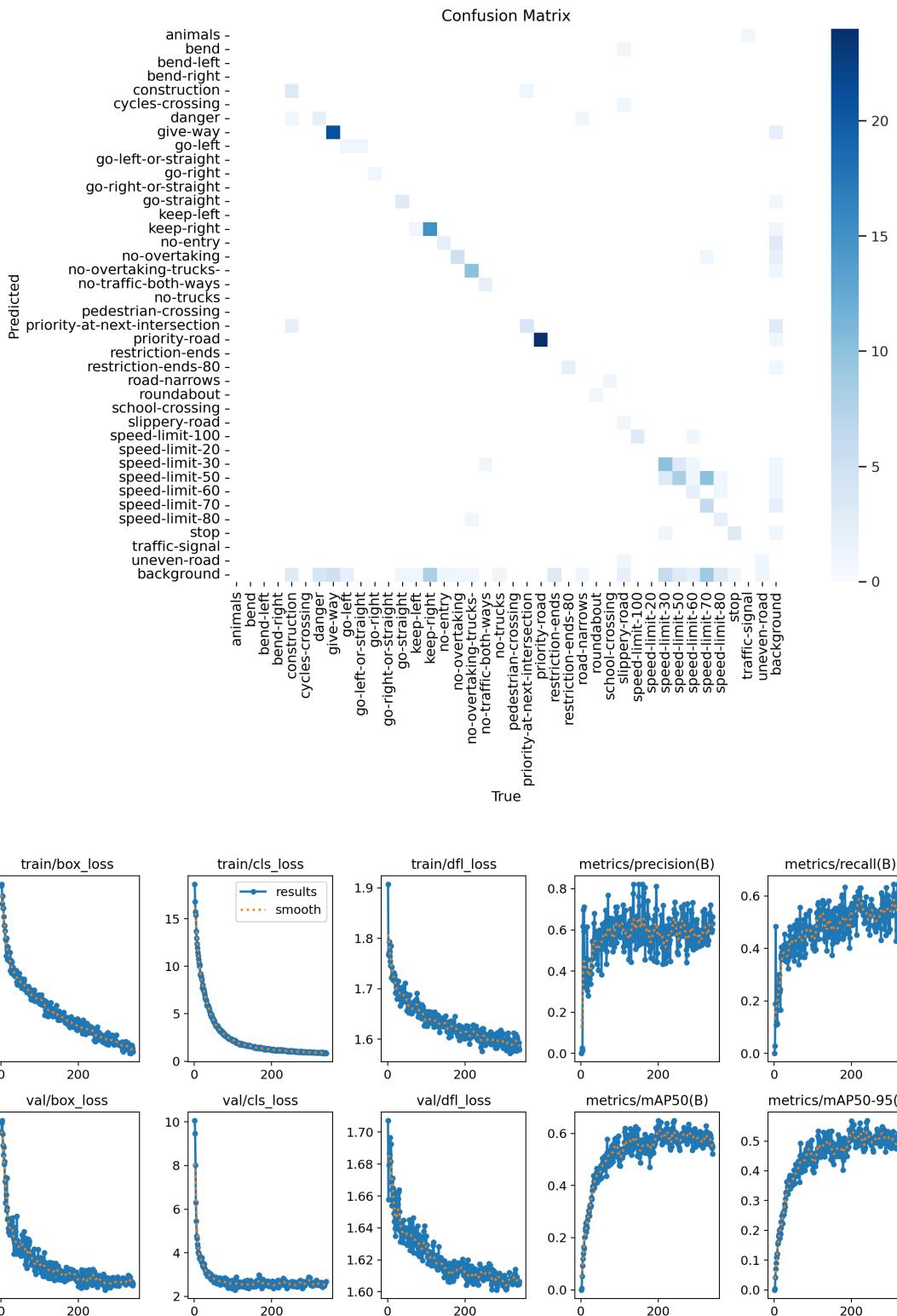
Per quanto concerne invece le curve relative ai modelli allenati, si possono fare delle considerazioni comuni a tutti e tre i modelli:

- **Precision-confidence curve:** quando la confidenza (ascisse) assume valori piccoli, i modelli rilevano molti oggetti, compresi molti falsi positivi, quindi la precisione risulta essere inferiore. Nel momento in cui la soglia di confidenza aumenta, il modello diventa più selettivo, e rileva un numero inferiore di oggetti, con una precisione più elevata.
- **Recall-confidence curve:** dai grafici dei 3 modelli si può notare che la recall diminuisce nel momento in cui aumenta la soglia di confidenza considerata, in quanto il modello diventa più selettivo, e si riducono in questo modo i falsi positivi, anche se allo stesso tempo aumentano i falsi negativi.
- **F_1 -confidence curve:** dalle curve F_1 -confidence dei tre modelli è possibile evidenziare quali siano le migliori soglie di confidenza per massimizzare la qualità delle rilevazioni, considerando tutte le possibili classi. In particolare:
 - **YOLOv10n:** il massimo mAP su tutte le classi, pari a 0.47 (47%) si ha con una soglia di confidenza pari a **0.302**.
 - **YOLOv10s:** il massimo mAP su tutte le classi, pari a 0.61 (61%) si ha con una soglia di confidenza pari a **0.214**.
 - **YOLOv10m:** il massimo mAP su tutte le classi, pari a 0.66 (66%) si ha con una soglia di confidenza pari a **0.237**.
- **Precision-Recall curve:** le curve PR relative ai 3 modelli mostrano che, come ci si aspetta di solito, la precisione diminuisce all'aumentare della recall. A partire da queste curve è anche possibile ricavare, come già detto in precedenza, i valori di AP per le diverse classi, e di conseguenza anche il valore finale di mAP, riportato nelle figure.

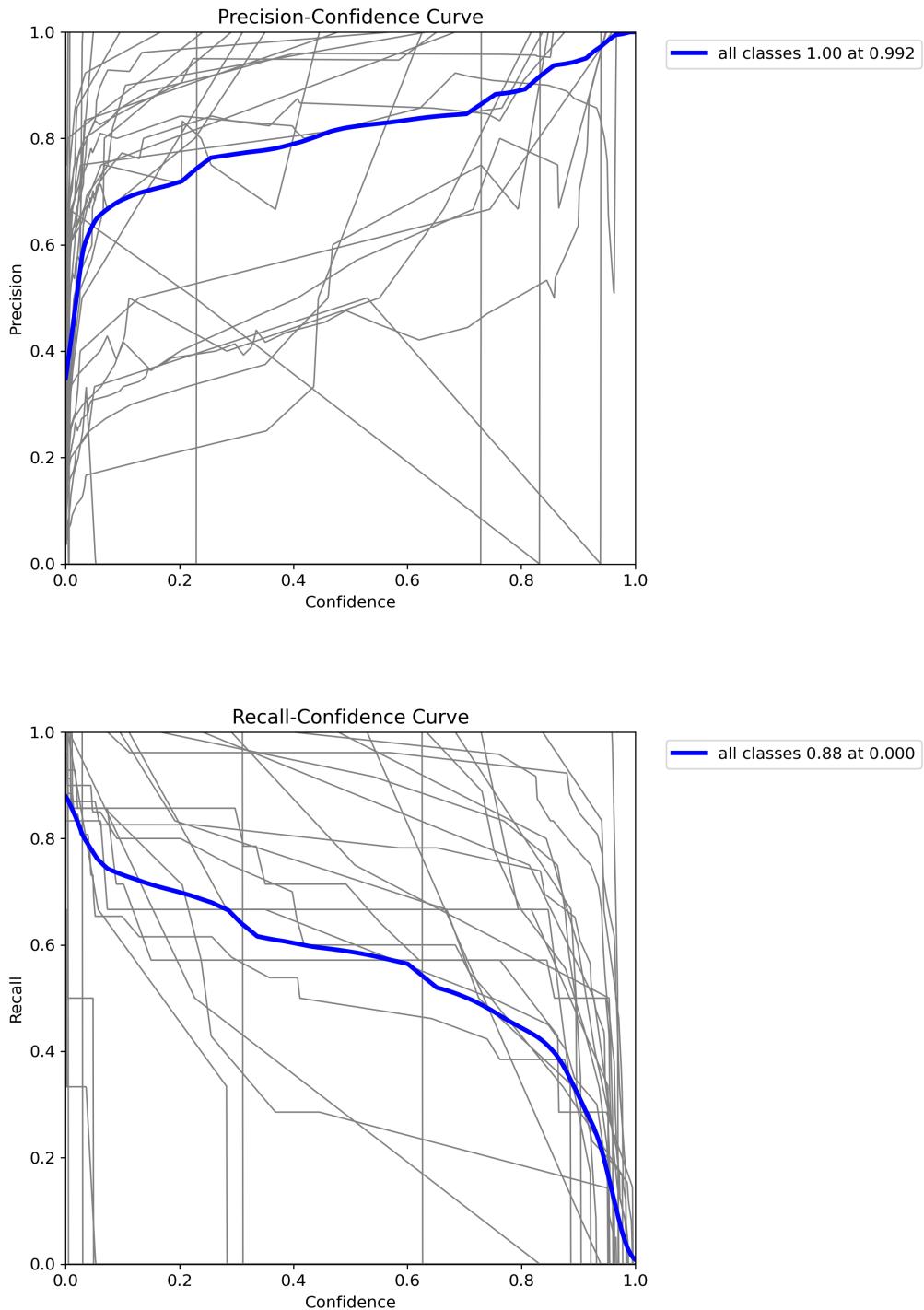
7.2.1 YOLOv10n

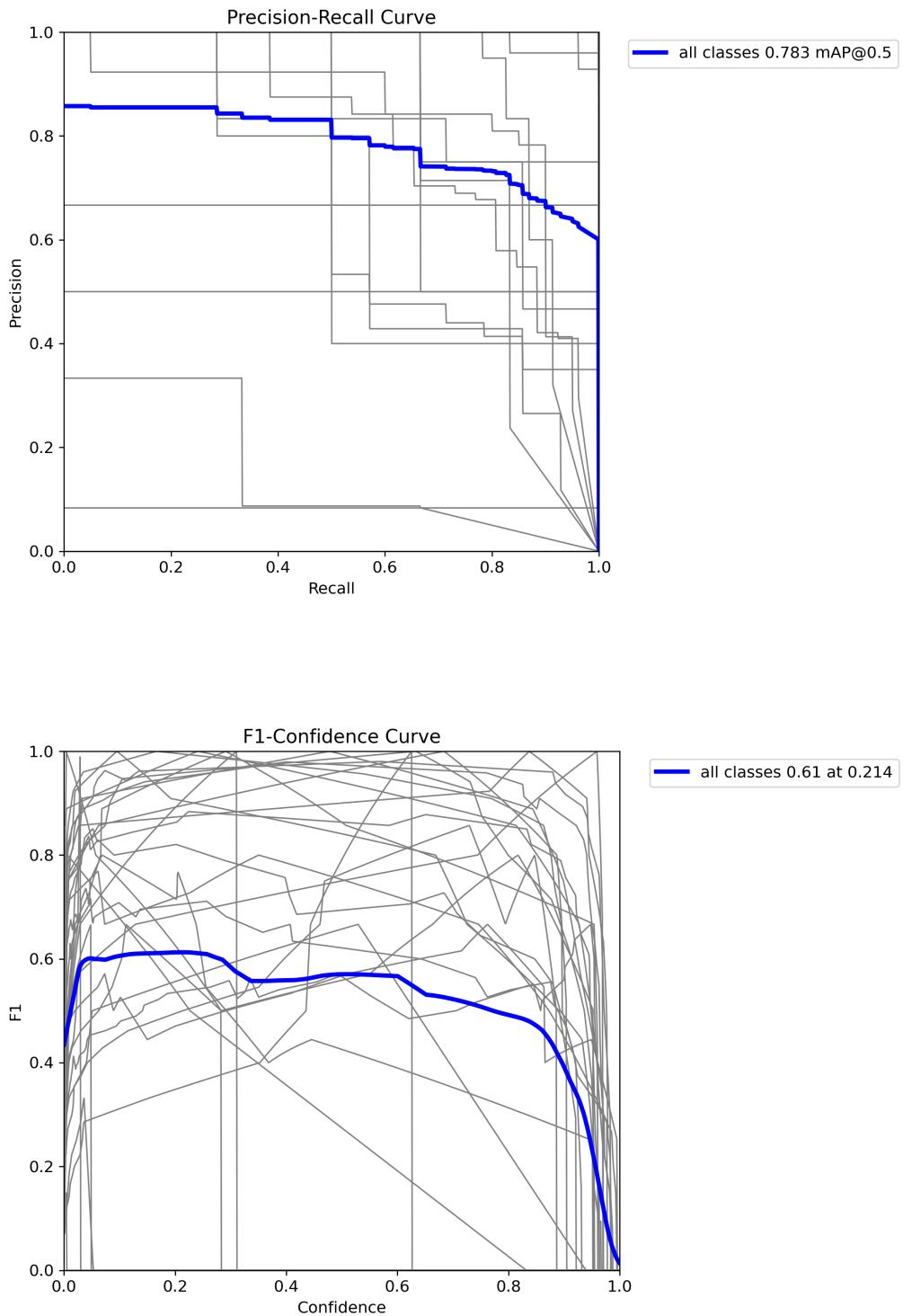


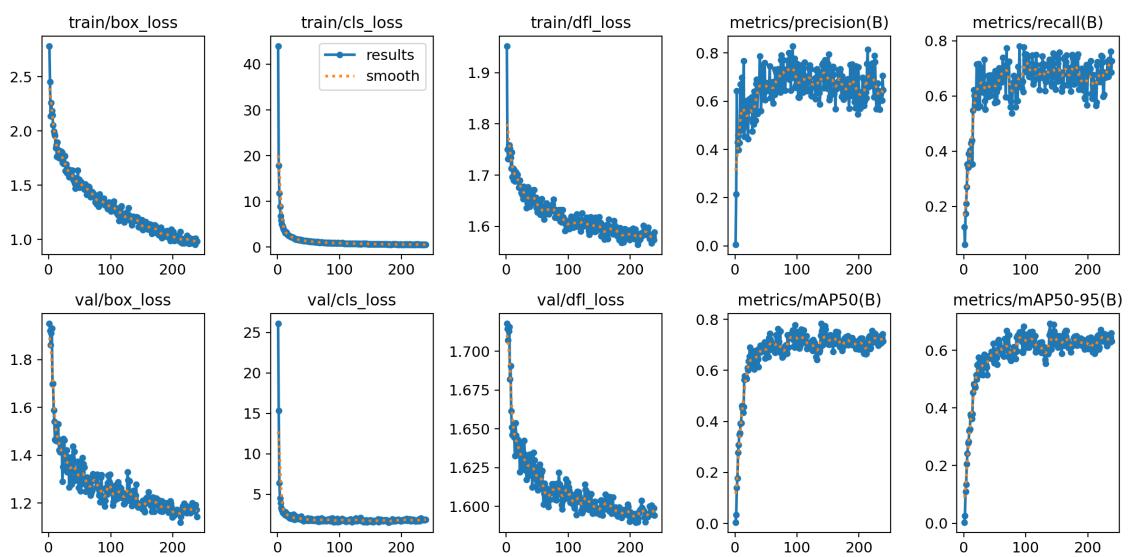
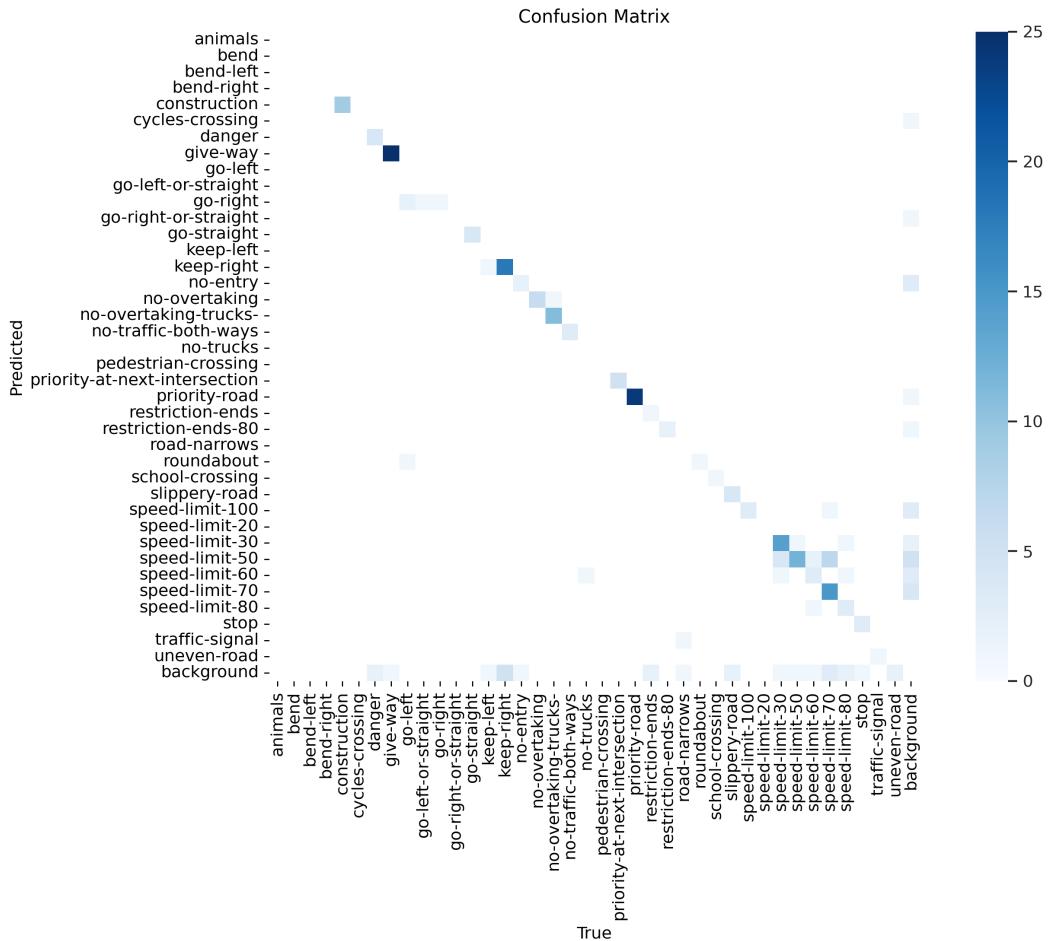




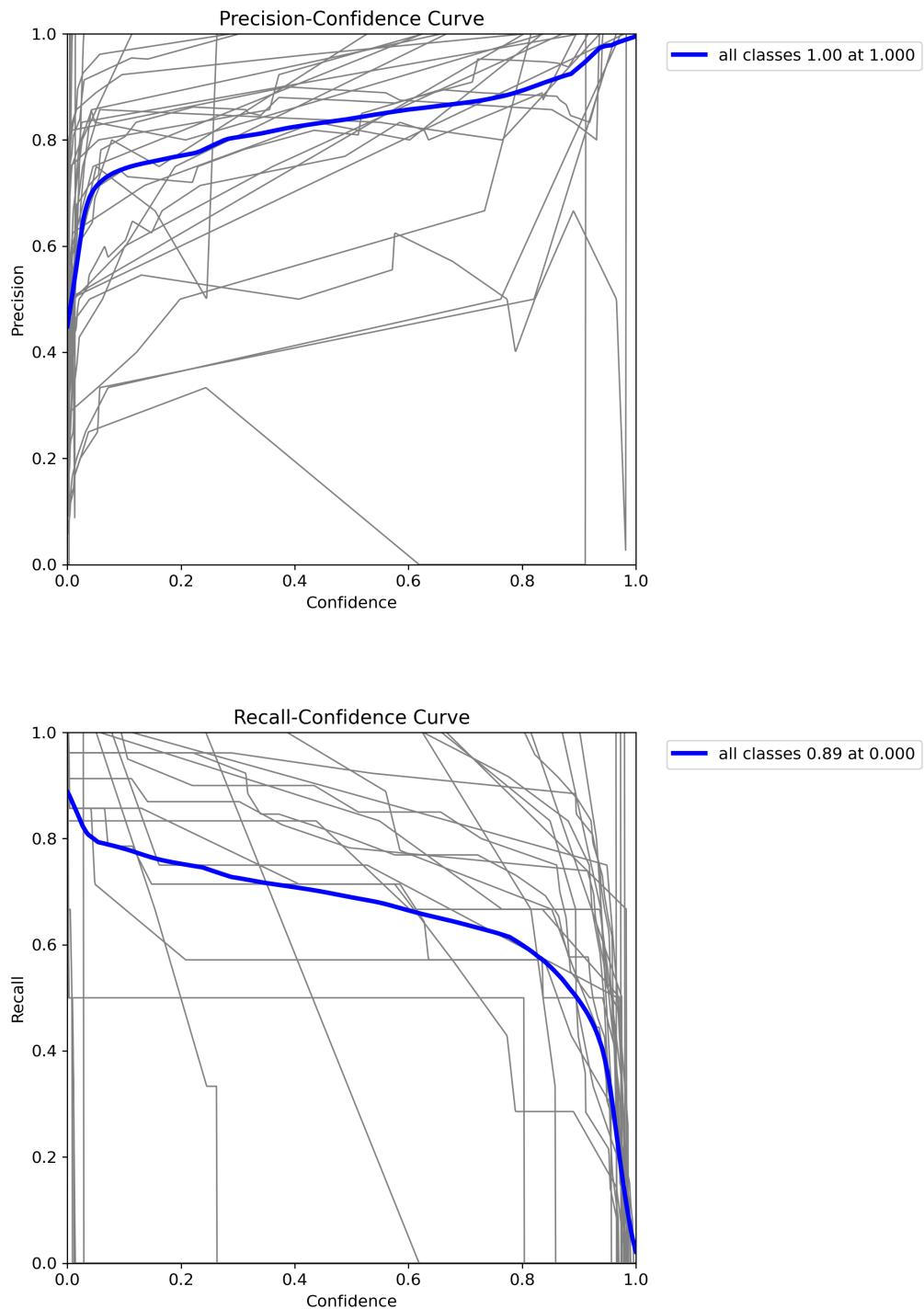
7.2.2 YOLOv10s

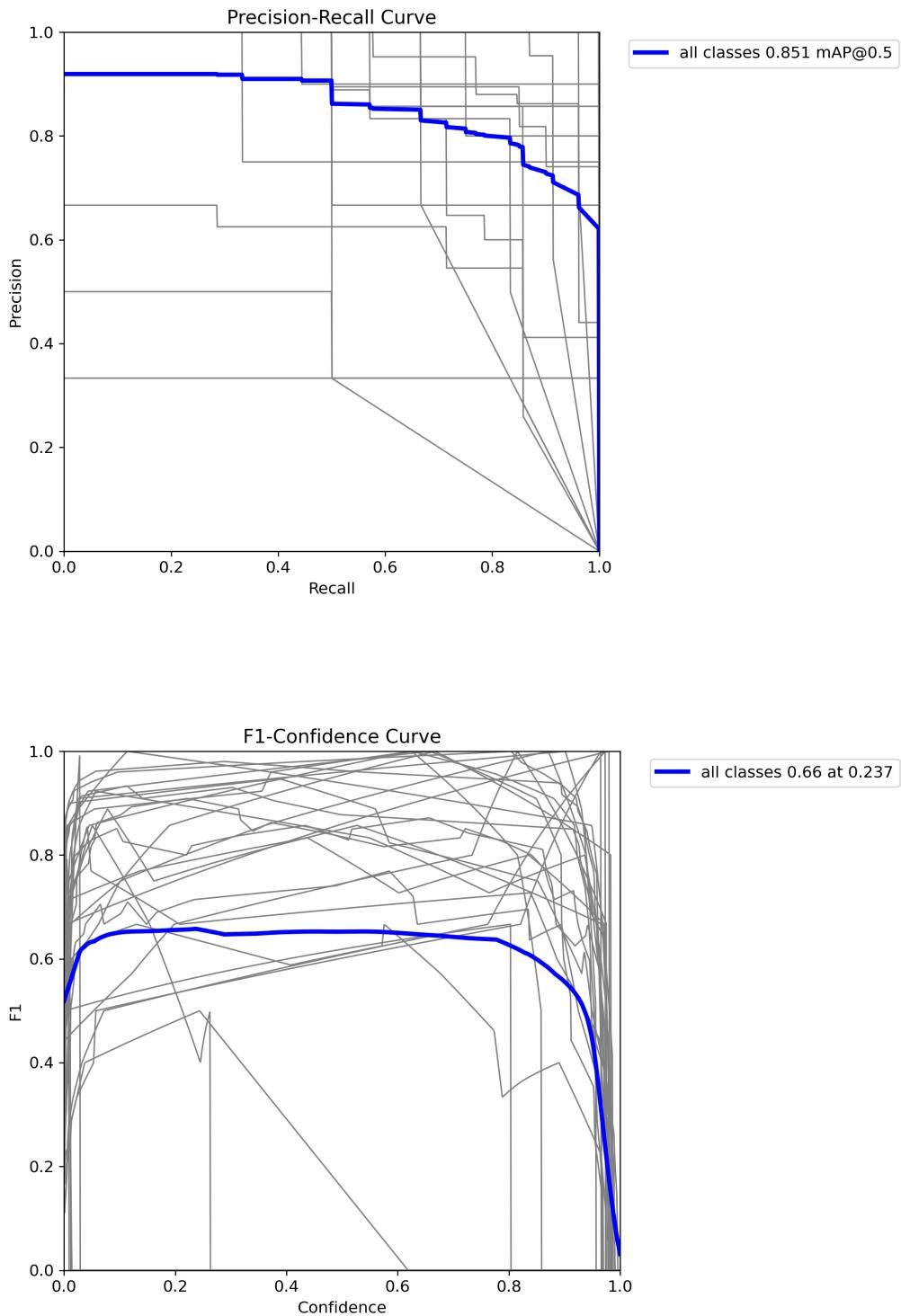


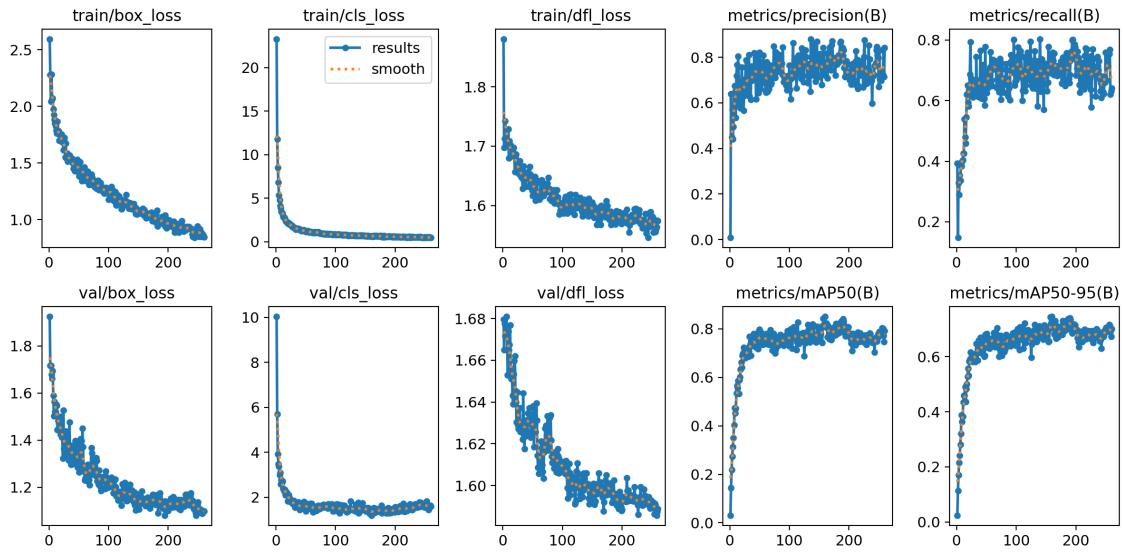
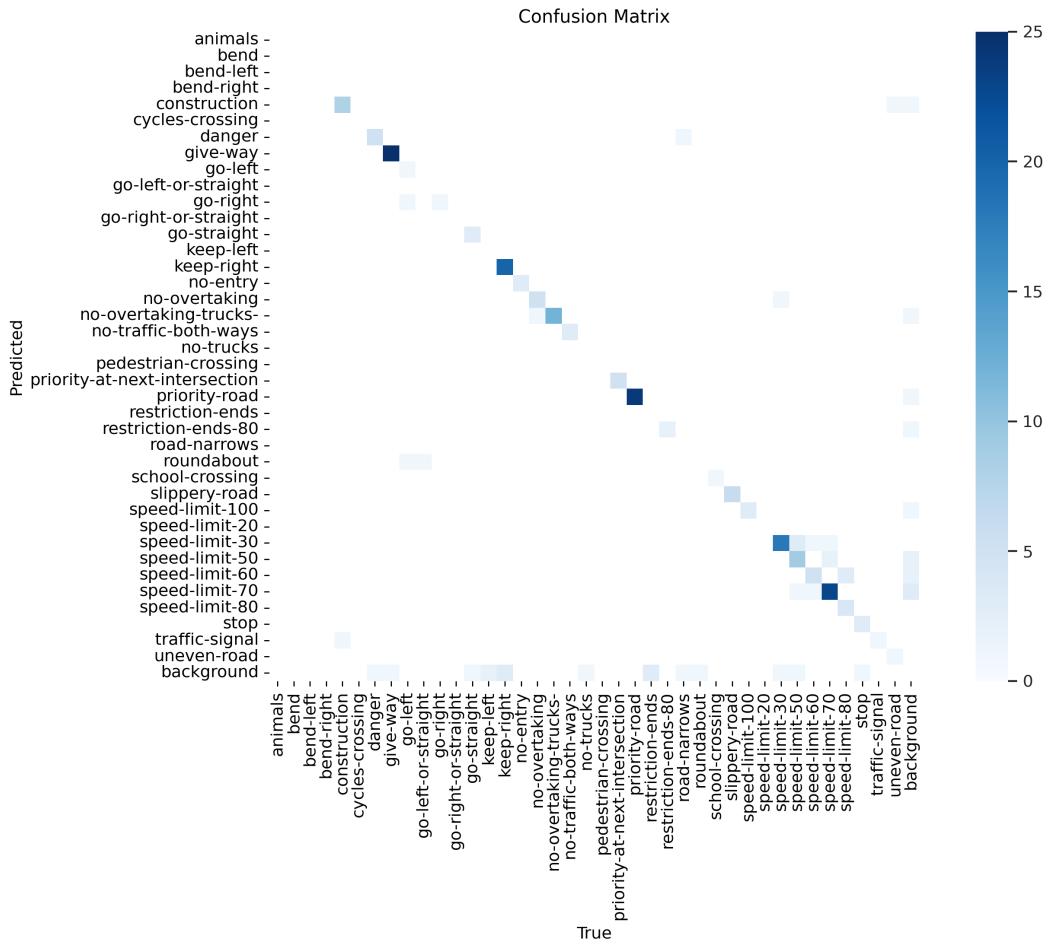




7.2.3 YOLOv10m







Capitolo 8

Conclusione

In questo capitolo saranno riassunti i risultati ottenuti dopo aver effettuato il fine-tuning dei modelli YOLO sul dataset GTSDB e la successiva fase di test di questi ultimi sul dataset da me acquisito.

Saranno poi riportate alcune considerazioni su possibili sviluppi futuri applicabili al fine di migliorare i risultati ottenuti.

8.1 Obbiettivi e risultati ottenuti

L'obbiettivo del presente progetto è stato quello di fare uso di un modello di object detection che rappresenta lo stato dell'arte, ovvero YOLOv10, al fine di risolvere il task di object detection nell'ambito dell'automotive, e in particolare il riconoscimento di segnali stradali.

A causa delle limitate risorse computazionali a disposizione, si è scelto di utilizzare solamente tre dei modelli messi a disposizione da YOLOv10, ovvero YOLOv10n, YOLOv10s e YOLOv10m.

L'addestramento di tali modelli è stato effettuato grazie ad un dataset di segnali stradali, anch'esso rappresentate lo stato dell'arte, ovvero il dataset GTSDB. Nella fase di training, al fine di valutare successivamente le performance dei modelli, si è tenuto conto di diverse funzioni di loss che riguardassero sia il problema della classificazione degli oggetti rilevati, sia il problema di regressione riguardante l'individuazione delle coordinate di un bounding-box che racchiudesse gli oggetti individuati.

Una volta effettuato il training, i diversi modelli sono stati messi alla prova in una fase di testing, la quale ha richiesto l'utilizzo di un ulteriore dataset, diverso da quello usato per il training, acquisito personalmente lungo le strade della provincia di Catania.

In particolare, il testing dei diversi modelli ha previsto l'uso di diverse metri-

che di valutazione, tra cui la Precision, la Recall, l' F_1 -score, la mAP nelle sue diverse forme, oltre a vari grafici, tra cui i principali sono la curva Precision-Confidence, la curva Recall-Confidence, la curva F_1 -Confidence e la curva Precision-Recall.

Dall'analisi dei risultati ottenuti, è emerso che, come ci si aspettava, il modello YOLOv10m ha avuto le performance migliori sia in fase di training che in fase di testing.

Tuttavia, è emerso anche che, sebbene i modelli abbiano delle prestazioni abbastanza elevate su training e validation set, non si può dire altrettanto riguardo le prestazioni dei modelli sul test set: ciò è probabilmente dovuto ad una differenza tra il dataset di test e il dataset utilizzato per il training. Il dataset GTSDB, usato per il training, è infatti costituito da immagini raccolte in Germania, mentre il dataset da me acquisito contiene segnali stradali italiani. Di conseguenza, differenze nel design, nella forma e nel colore dei segnali possono aver confuso i modelli, portandoli a fare predizioni meno accurate sul test set.

8.2 Lezioni apprese

Lo svolgimento del presente progetto si è rivelato estremamente utile dal punto di vista didattico, offrendo l'opportunità di utilizzare tecnologie avanzate di object detection, rappresentative dello stato dell'arte.

Tra le lezioni apprese, è emerso chiaramente come le differenze tra i dati utilizzati per il training e quelli impiegati per il testing possano generare significative difficoltà nella performance del modello.

E' stata inoltre molto formativa la fase di acquisizione di un nuovo dataset da zero. La raccolta di immagini ha richiesto infatti un'attenzione particolare a diversi aspetti, come angolazioni, condizioni atmosferiche e scenari vari. Poi, la successiva fase di annotazione manuale delle immagini, sebbene effettuata tramite un tool grafico come Roboflow, è risultata essere la fase più lunga e laboriosa del progetto. Questa fase ha richiesto infatti precisione e cura nelle annotazioni per avere un dataset di qualità sufficientemente alta.

Infine, è stato molto utile analizzare le diverse metriche di valutazione di un modello di machine learning, in quanto in questo modo è stato possibile valutare le prestazioni dei modelli allenati e determinare se essi funzionino adeguatamente o meno.

8.3 Sviluppi futuri

Come già espresso, i risultati del training dei modelli YOLOv10n, YOLOv10s e YOLOv10m ha mostrato risultati accettabili in fase di training per la maggior parte delle possibili classi, mentre i risultati per il test set non sono stati ottimali.

E' possibile ipotizzare alcuni sviluppi futuri che portino al miglioramento dei risultati:

- **Uso di un dataset di training più vasto**
- **Estensione del dataset di test acquisito personalmente**
- **Considerare un maggior numero di classi**, in quanto un alto numero di segnali stradali molto comuni non sono presi in considerazione dal modello.
- **Utilizzo di modelli più performanti**, ad esempio YOLOv10l, YOLOv10b e YOLOv10x

Appendice: codici

Nella presente appendice sono mostrati i codici usati per la realizzazione del progetto, ad eccezione del codice per il training, il quale si trova all'interno del notebook di esempio ***”Training_Demo_Traffic_Signs_Detection.ipynb”***, contenuto nel repository del progetto.

Tutti i codici mostrati di seguito sono presenti nella **cartella del progetto “Utils”** (fatta eccezione per l'app demo, che si trova nella directory principale), e possono essere avviati semplicemente usando il comando della shell:

```
python nomeFile.py
```

Si assume che sul calcolatore sia installata una distribuzione Python, e che siano state installate tutte le librerie necessarie, presenti nel file ***requirements.txt*** contenuto nel progetto.

```

1 import os
2 from PIL import Image
3
4 def convert_ppm_to_jpg(input_folder, output_folder):
5     if not os.path.exists(output_folder):
6         os.makedirs(output_folder)
7     for filename in os.listdir(input_folder):
8         if filename.endswith(".ppm"):
9             ppm_path = os.path.join(input_folder, filename)
10            with Image.open(ppm_path) as im:
11                jpg_filename = os.path.splitext(filename)[0] + ".jpg"
12                jpg_path = os.path.join(output_folder,
13                jpg_filename)
14                im.convert('RGB').save(jpg_path, "JPEG")
15                print(f"Convertito {ppm_path} a {jpg_path}")
16
17 if __name__ == "__main__":
18     input_folder = "dataset/input/directory"
19     output_folder = "dataset/output/directory"
20     convert_ppm_to_jpg(input_folder, output_folder)

```

Codice 8.1: Pre-processing sul dataset: conversione del formato delle immagini da .ppm a .jpg - file "Utils/ppm2jpg.py"

```

1 import os
2 from PIL import Image
3
4 if __name__ == "__main__":
5     folder = "dataset/input/directory"
6     for filename in os.listdir(folder):
7         if filename.lower().endswith(('png', 'jpg', 'jpeg', 'bmp')):
8             img_path = os.path.join(folder, filename)
9             img = Image.open(img_path)
10            img_resized = img.resize(size)
11            img_resized.save(img_path)

```

Codice 8.2: Pre-processing sul dataset: resize delle immagini a dimensione 640x640 - file "Utils/resize.py"

```

1 import os
2 mapping = {31: 0, 21: 1, 19: 2, 20: 3, 25: 4, 29: 5, 18: 6, 13:
3     7, 34: 8, 37: 9, 33: 10, 36: 11, 35: 12, 39: 13, 38: 14, 17:
4     15, 9: 16, 10: 17, 15: 18, 16: 19, 27: 20, 11: 21, 12: 22,
5     32: 23, 6: 24, 24: 25, 40: 26, 28: 27, 23: 28, 7: 29, 0: 30,
6     1: 31, 2: 32, 3: 33, 4: 34, 5: 35, 14: 36, 26: 37, 22: 38}
7
8 def process_files(input_folder, output_folder):
9     if not os.path.exists(output_folder):
10         os.makedirs(output_folder)
11     for filename in os.listdir(input_folder):
12         if filename.endswith('.txt'):
13             input_file = os.path.join(input_folder, filename)
14             output_file = os.path.join(output_folder, filename)
15             with open(input_file, 'r') as f:
16                 lines = f.readlines()
17                 new_lines = []
18                 for line in lines:
19                     values = line.strip().split()
20                     x = int(values[0])
21                     if x in mapping:
22                         mapped_x = mapping[x]
23                         new_line = f"{mapped_x} " + " ".join(values
24 [1:])
25                         new_lines.append(new_line)
26                 with open(output_file, 'w') as f:
27                     f.write("\n".join(new_lines))
28
29 input_folder = "your/input/folder"
30 output_folder = "your/output/folder"
31 process_files(input_folder, output_folder)

```

Codice 8.3: Pre-processing sul dataset: re-mapping degli indici - file "Utils/modificaIndiciClassi.py"


```
32         x1 = int(x_center - bbox_width / 2)
33         y1 = int(y_center - bbox_height / 2)
34         x2 = int(x_center + bbox_width / 2)
35         y2 = int(y_center + bbox_height / 2)
36         cv2.rectangle(image, (x1, y1), (x2, y2), (0,
37             255, 0), 2)
38         label = class_names[class_id]
39         font = cv2.FONT_HERSHEY_DUPLEX
40         font_scale = 0.7
41         font_thickness = 1
42         (text_width, text_height), baseline = cv2.
43         getTextSize(label, font, font_scale, font_thickness)
44         text_x = int(x1 + (x2 - x1 - text_width) / 2)
45         text_y = int(y1)-10
46         cv2.putText(image, label, (text_x, text_y), font
47             , font_scale, (255, 0, 0), font_thickness)
48         output_image_path = os.path.join(output_folder,
49             image_file)
50         cv2.imwrite(output_image_path, image)
51 cv2.destroyAllWindows()
```

Codice

8.4:

Codice

per

la generazione delle immagini con bounding boxes ground-truth a partire dalle
label in formato .txt - file "Utils/printGroundTruthAnnotation.py"

```
1 import tkinter as tk
2 from tkinter import filedialog, messagebox
3 from tkinter import ttk
4 import cv2
5 from PIL import Image, ImageTk
6 from collections import Counter
7 from ultralytics import YOLO
8
9 MAX_IMAGE_SIZE = (400, 400)
10
11 def resize_image(image, max_size):
12     image.thumbnail(max_size, Image.ANTIALIAS)
13     return image
14
15 def load_ground_truth():
16     global gt_img
17     if img_path:
18         gt_img_path = img_path.replace("/images", "/groundTruthAnnotations")
19
20     try:
21         gt_img = cv2.imread(gt_img_path)
22         if gt_img is not None:
23             gt_img_rgb = cv2.cvtColor(gt_img, cv2.COLOR_BGR2RGB)
24             gt_img_pil = Image.fromarray(gt_img_rgb)
25             gt_img_pil = resize_image(gt_img_pil,
26             MAX_IMAGE_SIZE)
27             gt_img_tk = ImageTk.PhotoImage(gt_img_pil)
28             img_panel_middle.config(image=gt_img_tk)
29             img_panel_middle.image = gt_img_tk
30             title_middle.grid()
31     else:
32         messagebox.showerror("Error", "Ground truth
33 image not found!")
34     except Exception as e:
35         messagebox.showerror("Error", f"Ground truth image
36 loading error: {str(e)}")
37
38 def load_image():
39     global img, img_path
40     remove_prediction()
41     img_path = filedialog.askopenfilename()
42     if img_path:
43         img = cv2.imread(img_path)
44         img_rgb = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
45         img_pil = Image.fromarray(img_rgb)
46         img_pil = resize_image(img_pil, MAX_IMAGE_SIZE)
47         img_tk = ImageTk.PhotoImage(img_pil)
```

```
45         img_panel_left.config(image=img_tk)
46         img_panel_left.image = img_tk
47         title_left.grid()
48         load_ground_truth()
49
50     def remove_images():
51         global img, gt_img
52         img_panel_left.config(image='')
53         img_panel_middle.config(image='')
54         img_panel_right.config(image='')
55         title_left.grid_remove()
56         title_middle.grid_remove()
57         title_right.grid_remove()
58         prediction_listbox.pack_forget()
59         prediction_listbox.delete(0, tk.END)
60         img = None
61         gt_img = None
62
63     def remove_prediction():
64         img_panel_right.config(image='')
65         title_right.grid_remove()
66         prediction_listbox.pack_forget()
67
68     def load_model(event):
69         global model
70         model_name = selected_model.get()
71         try:
72             if model_name == "YOLOv10n":
73                 model = YOLO("./YOLOv10nTRAINED/weights/best.pt")
74             elif model_name == "YOLOv10s":
75                 model = YOLO("./YOLOv10sTRAINED/weights/best.pt")
76             elif model_name == "YOLOv10m":
77                 model = YOLO("./YOLOv10mTRAINED/weights/best.pt")
78             messagebox.showinfo("Model loaded", f"{model_name} model loaded!")
79         except Exception as e:
80             messagebox.showerror("Error", f"Error in model loading: {str(e)}")
81
82     def predict():
83         global img
84         if img is None:
85             messagebox.showerror("Error", "Load image first!")
86             return
87         if model is None:
88             messagebox.showerror("Error", "Load a model first!")
89             return
90
91     results = model(img_path)
```

```

92     class_count = Counter()
93
94     for result in results:
95         image = result.orig_img
96         boxes = []
97         for result in results:
98             for box in result.bboxes:
99                 x1, y1, x2, y2 = box.xyxy[0].tolist()
100                confidence = box.conf[0].item()
101                class_id = int(box.cls[0].item())
102                class_name = results[0].names[class_id]
103                boxes.append((x1, y1, x2, y2, confidence,
104                             class_name))
105                class_count[class_name] += 1
106
107                boxes = sorted(boxes, key=lambda x: x[4], reverse=True)
108
109                for (x1, y1, x2, y2, confidence, class_name) in boxes:
110                    label = f'{class_name}-{confidence:.2f}'
111                    cv2.rectangle(image, (int(x1), int(y1)), (int(x2),
112                                                 int(y2)), (0, 255, 0), 2)
113
114                    font = cv2.FONT_HERSHEY_DUPLEX
115                    font_scale = 0.7
116                    font_thickness = 1
117                    (text_width, text_height), baseline = cv2.
118                    getTextSize(label, font, font_scale, font_thickness)
119
120                    text_x = int(x1 + (x2 - x1 - text_width) / 2)
121                    text_y = int(y1) - 10
122                    cv2.putText(image, label, (text_x, text_y), font,
123                               font_scale, (255, 0, 0), font_thickness)
124
125                    img_pil = Image.fromarray(cv2.cvtColor(image, cv2.
126 COLOR_BGR2RGB))
127                    img_pil = resize_image(img_pil, MAX_IMAGE_SIZE)
128                    img_tk = ImageTk.PhotoImage(img_pil)
129                    img_panel_right.config(image=img_tk)
130                    img_panel_right.image = img_tk
131                    title_right.grid()
132
133                    prediction_listbox.delete(0, tk.END)
134                    if class_count:
135                        prediction_listbox.insert(tk.END, "PREDICTIONS")
136                        for class_name, count in class_count.items():
137                            prediction_listbox.insert(tk.END, f"{class_name}
138                                         x{count}")
139                        prediction_listbox.pack(side=tk.BOTTOM, fill=tk.X)
140                    else:

```

```
135     prediction_listbox.insert(tk.END, "No traffic sign  
136 detected!")  
137     prediction_listbox.pack(side=tk.BOTTOM, fill=tk.X)  
138  
139 root = tk.Tk()  
140 root.title("Traffic Sign Detection App")  
141 root.update_idletasks()  
142 width = root.winfo_screenwidth()  
143 height = root.winfo_screenheight()  
144 root.geometry(f"{width}x{height}+0+0")  
145  
146 model_label = tk.Label(root, text="Select YOLO model:")  
147 model_label.pack()  
148  
149 selected_model = tk.StringVar()  
150 model_menu = ttk.Combobox(root, textvariable=selected_model)  
151 model_menu['values'] = ('YOLOv10n', 'YOLOv10s', 'YOLOv10m')  
152 model_menu.pack()  
153 model_menu.bind("<<ComboboxSelected>>", load_model)  
154  
155 button_frame = tk.Frame(root)  
156 button_frame.pack(pady=10)  
157  
158 load_img_btn = tk.Button(button_frame, text="Load image",  
    command=load_image)  
159 load_img_btn.pack(side=tk.LEFT, padx=5)  
160  
161 remove_img_btn = tk.Button(button_frame, text="Remove image",  
    command=remove_images)  
162 remove_img_btn.pack(side=tk.LEFT, padx=5)  
163  
164 frame = tk.Frame(root)  
165 frame.pack()  
166  
167 title_left = tk.Label(frame, text="Original Image", font=(  
    "Courier", 10, "bold"))  
168 title_left.grid(row=0, column=0)  
169 title_left.grid_remove()  
170  
171 title_middle = tk.Label(frame, text="Ground-truth Boxes", font=(  
    "Courier", 10, "bold"))  
172 title_middle.grid(row=0, column=1)  
173 title_middle.grid_remove()  
174  
175 title_right = tk.Label(frame, text="Predictions", font=( "Courier  
    ", 10, "bold"))  
176 title_right.grid(row=0, column=2)  
177 title_right.grid_remove()
```

```
178 img_panel_left = tk.Label(frame)
179 img_panel_left.grid(row=1, column=0, padx=10, pady=10)
180
181 img_panel_middle = tk.Label(frame)
182 img_panel_middle.grid(row=1, column=1, padx=10, pady=10)
183
184 img_panel_right = tk.Label(frame)
185 img_panel_right.grid(row=1, column=2, padx=10, pady=10)
186
187 prediction_listbox = tk.Listbox(root, height=15, width=40, font
188     =("Courier", 10), justify="center")
189 prediction_listbox.pack_forget()
190
191 predict_btn = tk.Button(root, text="Predict", command=predict)
192 predict_btn.pack(pady=10)
193
194 img = None
195 img_path = None
196 gt_img = None
197 model = None
198 root.mainloop()
```

Codice 8.5: Codice dell'app demo - file "trafficSignsDetectionApp.py"

Bibliografia

- [1] Ao Wang, Hui Chen, Lihao Liu, Kai Chen, Zijia Lin, Jungong Han, and Guiguang Ding. Yolov10: Real-time end-to-end object detection, 2024.
- [2] Sebastian Houben, Johannes Stallkamp, Jan Salmen, Marc Schlipsing, and Christian Igel. Detection of traffic signs in real-world images: The German Traffic Sign Detection Benchmark. In *International Joint Conference on Neural Networks*, number 1288, 2013.
- [3] Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi. You only look once: Unified, real-time object detection, 2016.