



Università  
di Catania

## UNIVERSITY OF CATANIA

DEPARTMENT OF MATHEMATICS AND COMPUTER SCIENCE  
MASTER'S DEGREE IN COMPUTER SCIENCE

---

# Traffic Sign Detection and Recognition

---

FINAL MACHINE LEARNING PROJECT  
REPORT

---

**Nominativo:** Giulio Pedicone  
**Matricola:** 1000084718

**Prof.** Giovanni Maria Farinella

**Nominativo:** Vincenzo Villanova  
**Matricola:** 1000084722

**Prof.** Rosario Leonardi

**Nominativo:** Francesco P. A. Virzi'  
**Matricola:** 1000084723

# Indice

<b>1 Problema</b>	<b>4</b>
1.1 Motivazioni . . . . .	4
1.2 Contesto applicativo . . . . .	5
1.3 Descrizione visiva del problema . . . . .	6
<b>2 Dataset</b>	<b>7</b>
2.1 Descrizione generale . . . . .	7
2.2 Fase di etichettatura . . . . .	7
2.3 Statistiche del dataset . . . . .	8
2.4 Organizzazione delle cartelle e nomenclatura . . . . .	9
2.5 Esempi visuali . . . . .	9
2.6 Download e Struttura del Dataset . . . . .	9
2.6.1 Struttura del Dataset . . . . .	10
2.6.2 Contenuto del File <code>data.yaml</code> . . . . .	11
2.6.3 Formato dei File di Etichetta . . . . .	12
2.7 Data Augmentation . . . . .	12
2.7.1 Trasformazioni applicate . . . . .	13
2.7.2 Implementazione del codice . . . . .	14
<b>3 Metodi</b>	<b>15</b>
3.1 YOLO: You Only Look Once - Approccio One-Stage per l'Object Detection	15
3.1.1 Architettura e Principi Fondamentali . . . . .	15
3.1.2 Vantaggi e Limitazioni del Paradigma One-Stage . . . . .	16
3.1.3 Confronto con Metodi Two-Stage . . . . .	17
3.2 Architettura del Sistema . . . . .	17
3.3 YOLOv8: Architettura Dettagliata . . . . .	18
3.3.1 Backbone: Estrazione Gerarchica delle Feature . . . . .	18
3.3.2 Neck: Fusione Multi-scala con FPN e PAN . . . . .	19
3.3.3 Detection Head: Predizione Disaccoppiata . . . . .	19
3.3.4 Anchor-Free Prediction . . . . .	19
3.3.5 YOLO Loss: Ottimizzazione End-to-End . . . . .	20
3.4 Innovazioni Architettoniche . . . . .	20
3.5 Motivazioni per la Scelta di YOLOv8 . . . . .	21
<b>4 Valutazione</b>	<b>22</b>
4.1 Intersection over Union (IoU) . . . . .	22
4.2 Confusion Matrix e valutazione delle prestazioni . . . . .	24

4.3	Average Precision (AP) . . . . .	25
4.4	Mean Average Precision (mAP) . . . . .	25
4.5	Recall . . . . .	26
4.6	F1-Score . . . . .	26
4.7	Precision-Recall Curve . . . . .	26
4.8	Recall Curve . . . . .	28
4.9	Metriche Aggiuntive per l'Analisi Sperimentale . . . . .	28
<b>5</b>	<b>Esperimenti</b>	<b>29</b>
5.1	Fase I . . . . .	29
5.1.1	Risultati Preliminari . . . . .	30
5.1.1.1	Matrice di Confusione Normalizzata . . . . .	30
5.1.1.2	Precision Curve . . . . .	31
5.1.1.3	Recall Curve . . . . .	31
5.1.1.4	F1 Score Curve . . . . .	32
5.1.1.5	Precision-Recall Curve . . . . .	32
5.1.1.6	Ulteriori Metriche . . . . .	33
5.1.2	Discussione . . . . .	33
5.2	Fase II . . . . .	34
5.2.1	Risultati Preliminari . . . . .	35
5.2.1.1	Matrice di Confusione Normalizzata . . . . .	35
5.2.1.2	Precision Curve . . . . .	36
5.2.1.3	Recall Curve . . . . .	36
5.2.1.4	F1 Score Curve . . . . .	37
5.2.1.5	Precision-Recall Curve . . . . .	37
5.2.1.6	Ulteriori Metriche . . . . .	38
5.2.2	Discussione . . . . .	38
5.3	Fase III . . . . .	39
5.3.1	Risultati Preliminari . . . . .	40
5.3.1.1	Matrice di Confusione Normalizzata . . . . .	40
5.3.1.2	Precision Curve . . . . .	41
5.3.1.3	Recall Curve . . . . .	41
5.3.1.4	F1 Score Curve . . . . .	42
5.3.1.5	Precision-Recall Curve . . . . .	42
5.3.1.6	Ulteriori Metriche . . . . .	43
5.3.1.7	Discussione . . . . .	43
5.4	Fase IV: Confronto tra Varianti del Modello YOLO . . . . .	44
5.4.1	Metriche di addestramento di YOLOv11 . . . . .	45
<b>6</b>	<b>Demo</b>	<b>47</b>
6.1	Fase 1: Visualizzazione della Web Application . . . . .	47
6.2	Fase 2: Caricamento dell'Immagine . . . . .	48
6.3	Fase 3: Esecuzione della Predizione . . . . .	49
6.4	Fase 4: Visualizzazione della Predizione . . . . .	50
<b>7</b>	<b>Codice</b>	<b>51</b>
7.1	Installazione della Libreria . . . . .	51
7.2	Importazione del Modulo YOLO . . . . .	51
7.3	Caricamento del Modello . . . . .	51

7.4	Addestramento del Modello . . . . .	52
7.5	Web Application per la Predizione . . . . .	53
7.6	Funzionalità dell'Applicazione . . . . .	54
7.7	Installazione e Configurazione del Sistema . . . . .	54
7.7.1	Requisiti di Sistema . . . . .	54
7.7.2	Procedura di Installazione . . . . .	54
7.7.3	Configurazione del Sistema . . . . .	55
7.7.4	Avvio dell'Applicazione Web . . . . .	55

# Capitolo 1

## Problema

Il problema affrontato in questo progetto è la **rilevazione automatica dei segnali stradali** all'interno di immagini, un compito noto nel campo della computer vision come *object detection*. In particolare, il task si suddivide in due obiettivi principali:

1. **Individuazione della posizione** di ciascun segnale presente nell'immagine tramite tecniche di *regressione*, al fine di stimare le coordinate del *bounding box* (ovvero il rettangolo che racchiude il segnale).
2. **Classificazione del tipo di segnale** contenuto in ciascun bounding box, scegliendo una tra le classi predefinite, attraverso tecniche di *classificazione*.

### 1.1 Motivazioni

Il riconoscimento automatico dei cartelli stradali rappresenta un problema fondamentale nei campi dell'elaborazione delle immagini e del riconoscimento di pattern. Esso consiste nel rilevare e classificare correttamente segnali stradali a partire da immagini o video acquisiti da dispositivi di bordo.

Tali applicazioni rivestono un ruolo centrale nei moderni *sistemi avanzati di assistenza alla guida* (ADAS), che utilizzano telecamere e sensori per interpretare la segnaletica stradale con l'obiettivo di migliorare la sicurezza sulle strade. In particolare, con la diffusione delle auto a guida autonoma, il riconoscimento dei segnali stradali diventa indispensabile per consentire al veicolo di prendere decisioni critiche in tempi brevissimi, come rallentare, fermarsi o deviare.

Una decisione inaccurata o un errore nel riconoscimento dei cartelli possono infatti portare a gravi conseguenze, potenzialmente letali per le persone coinvolte. Per questo motivo, è essenziale che gli algoritmi impiegati siano in grado di operare in tempo reale con elevata precisione e affidabilità.

Le principali difficoltà nel riconoscimento automatico dei segnali stradali sono dovute a numerosi fattori ambientali e fisici. Tra questi, si annoverano le condizioni atmosferiche avverse, come pioggia intensa o nebbia, che compromettono la visibilità dei cartelli. Un'altra problematica significativa è rappresentata dallo stato di usura, sporcizia, deformazioni o danneggiamenti presenti su molti segnali, frequenti anche nel dataset utilizzato in questo progetto. Le condizioni di illuminazione variabili costituiscono un ulteriore ostacolo, in quanto possono essere eccessive o carenti, alterando il contrasto e la percezione cromatica dei cartelli. Infine, la presenza di ostacoli o inclinazioni dei segnali, ad esempio cartelli parzialmente nascosti da pali, veicoli o altri elementi, complica ulteriormente il processo di riconoscimento.

Molte di queste problematiche sono state effettivamente riscontrate nel dataset raccolto per questo studio, come mostrano le immagini riportate nelle sezioni successive. Sebbene il problema possa essere affrontato con metodi classici basati sull'analisi di istogrammi o forme geometriche, gli algoritmi moderni di machine learning, e in particolare le reti neurali convoluzionali, hanno dimostrato una superiore capacità di adattamento e accuratezza grazie all'apprendimento diretto dai dati.

## 1.2 Contesto applicativo

Il progetto si inserisce nell'ambito dell'apprendimento automatico applicato all'elaborazione delle immagini. In particolare, si utilizzano *reti neurali convoluzionali* (CNN) per apprendere simultaneamente:

- la posizione dei segnali (regressione);
- la classe del segnale rilevato (classificazione).

Le classi di segnali stradali oggetto di classificazione sono le seguenti:

- **Limite di velocità 30 km/h**
- **Limite di velocità 50 km/h**
- **Limite di velocità 80 km/h**
- Segnale di **STOP**
- Segnale di **precedenza**
- **Direzione obbligatoria a destra**
- **Direzione obbligatoria a sinistra**
- **Rotatoria**

**Figura 1.1:** Segnali stradali

### 1.3 Descrizione visiva del problema

Ogni immagine del dataset può contenere uno o più segnali stradali. Per ciascun segnale, il sistema deve fornire in output:

- Le coordinate del bounding box:  $(x_{\min}, y_{\min}, x_{\max}, y_{\max})$ ;
- La classe corretta del segnale.

**Figura 1.2:** Esempio di immagine con segnali stradali e relative bounding box annotate.

L’obiettivo finale è la costruzione di un sistema in grado di apprendere da un dataset annotato e, successivamente, rilevare automaticamente i segnali stradali in immagini non viste, localizzandoli e classificandoli correttamente.

# Capitolo 2

## Dataset

### 2.1 Descrizione generale

Il dataset utilizzato in questo progetto è composto da un totale di **3049 immagini** di segnali stradali, raccolte con lo scopo di addestrare e testare il sistema di riconoscimento automatico sviluppato.

Le immagini sono state acquisite tramite diverse modalità complementari. Una parte significativa del dataset è costituita da foto personali scattate in vari punti della città di **Catania**, al fine di raccogliere dati rappresentativi delle condizioni reali del territorio locale. Questa raccolta diretta ha garantito la presenza di segnali stradali tipici del contesto urbano catanese, con le sue specifiche caratteristiche architettoniche e ambientali.

Per integrare ulteriormente la varietà di scenari e condizioni di visibilità, sono stati utilizzati screenshot da **Google Maps**, che hanno permesso di includere nel dataset segnali provenienti da diverse aree geografiche e situazioni di illuminazione. Infine, il dataset è stato arricchito con immagini prelevate dal dataset pubblico disponibile su Kaggle<sup>1</sup>, che contiene ulteriori segnali stradali già classificati e validati dalla comunità scientifica.

Questa combinazione metodologica ha permesso di costruire un dataset eterogeneo e realistico, contenente segnali in diverse condizioni ambientali e di visibilità, garantendo così una maggiore robustezza e generalizzabilità del sistema di riconoscimento sviluppato.

### 2.2 Fase di etichettatura

Le immagini raccolte sono state annotate manualmente mediante l'utilizzo di Roboflow<sup>2</sup>. Per ciascuna immagine è stato definito un *bounding box* intorno a ogni segnale stradale, associandogli una delle classi previste dal progetto.

L'annotazione ha riguardato esclusivamente i segnali appartenenti alle classi specificate, garantendo una copertura completa e accurata.

---

<sup>1</sup><https://www.kaggle.com/datasets/pkdarabi/cardetection/data>

<sup>2</sup><https://universe.roboflow.com/progettoml/progetto-ml/dataset/7>

## 2.3 Statistiche del dataset

Il dataset è stato suddiviso in tre sottoinsiemi per la fase di addestramento, validazione e test, rispettivamente:

- **Training set:** 2150 immagini (circa il 70%);
- **Validation set:** 630 immagini (circa il 20%);
- **Test set:** 269 immagini (circa il 10%).

La distribuzione delle immagini per ciascuna classe è la seguente:

Classe	Numero di immagini
destra	490
limite 30	502
limite 50	474
limite 80	427
precedenza	377
rotatoria	271
sinistra	472
stop	584

Tabella 2.1: Distribuzione delle immagini per classe nel dataset.

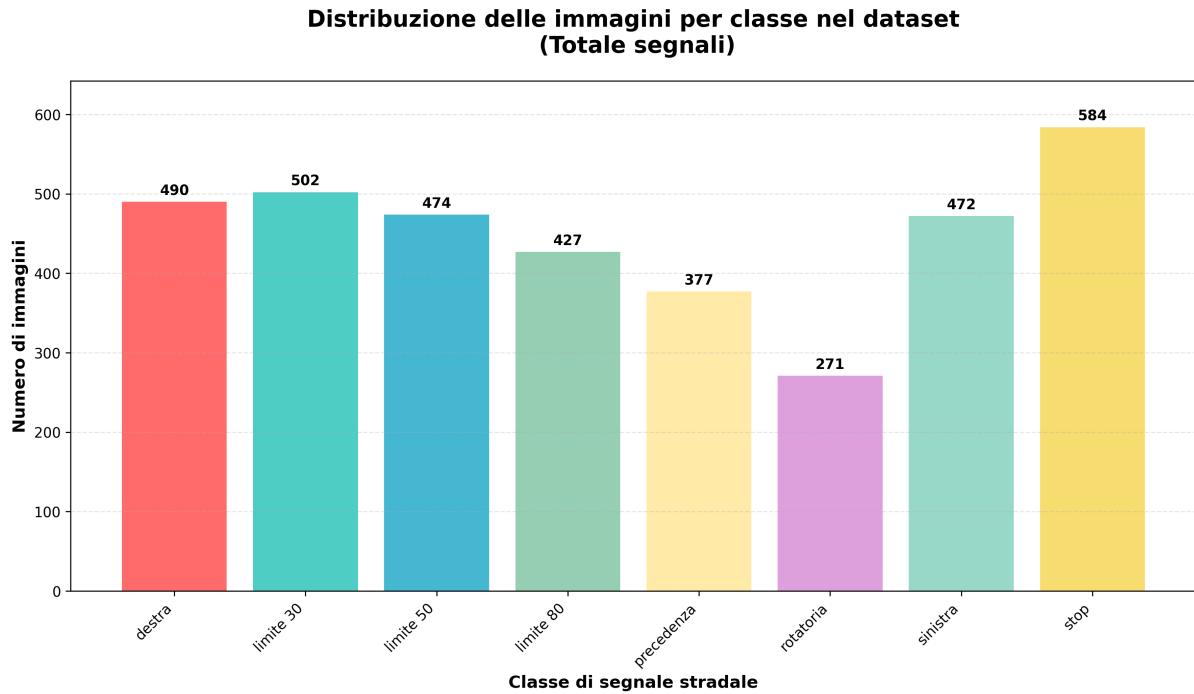


Figura 2.1: Distribuzione delle immagini per classe nel dataset

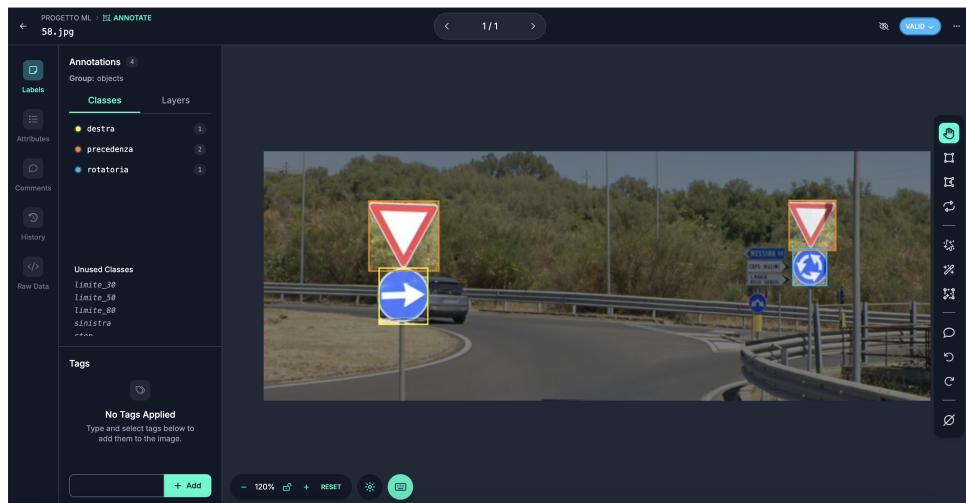
## 2.4 Organizzazione delle cartelle e nomenclatura

Il dataset è organizzato in cartelle separate per le tre suddivisioni:

- `train/` — contiene le immagini destinate all’addestramento;
- `val/` — contiene le immagini destinate alla validazione;
- `test/` — contiene le immagini destinate al test finale.

## 2.5 Esempi visuali

Di seguito sono riportati alcuni esempi rappresentativi delle immagini contenute nel dataset, con i relativi bounding box e classi annotate:



**Figura 2.2:** Esempi di immagini del dataset con bounding box sui segnali stradali.

## 2.6 Download e Struttura del Dataset

Il dataset può essere scaricato automaticamente attraverso un semplice script Python che utilizza la libreria `roboflow`. Di seguito è riportato il codice necessario per effettuare il download. Si noti che, per ragioni di sicurezza, la chiave API personale (`api_key`) non è inclusa nel codice ma verrà fornita separatamente via email:

```

1 !pip install roboflow
2
3 from roboflow import Roboflow
4 rf = Roboflow(api_key="INSERIRE_API_KEY")
5 project = rf.workspace("projettoml").project("progetto-ml")
6 version = project.version(7)
7 dataset = version.download("yolov8")
  
```

Il parametro passato alla funzione `download`, ovvero "`yolov8`", indica che il dataset verrà esportato in un formato compatibile con l’architettura YOLOv8, utilizzata in questo progetto per l’object detection.

### 2.6.1 Struttura del Dataset

Una volta completato il download, il dataset sarà organizzato secondo la seguente struttura gerarchica:

```
1 dataset/
2 +-+ train/
3 |   +-+ images/
4 |   +-+ labels/
5 +-+ valid/
6 |   +-+ images/
7 |   +-+ labels/
8 +-+ test/
9 |   +-+ images/
10 |   +-+ labels/
11 +-+ data.yaml
12 +-+ README.dataset.txt
13 +-+ README.roboflow.txt
```

- Le cartelle `images/` contengono le immagini dei rispettivi set: addestramento (`train`), validazione (`valid`) e test finale (`test`).
- Le cartelle `labels/` contengono i file di annotazione corrispondenti, nel formato YOLO: ogni file di testo ha lo stesso nome dell'immagine e descrive le bounding box associate agli oggetti presenti.
- Il file `data.yaml` è il file di configurazione necessario per l'addestramento del modello YOLO; contiene i percorsi relativi alle directory dei dati e la lista delle classi oggetto.
- Il file `README.dataset.txt` fornisce informazioni generali sul dataset esportato, mentre `README.roboflow.txt` documenta le impostazioni adottate su Roboflow durante la creazione e suddivisione dei dati.

Tale organizzazione è perfettamente compatibile con le librerie di training come Ultralytics YOLOv8, semplificando notevolmente il processo di importazione dei dati e l'avvio dell'addestramento.

### 2.6.2 Contenuto del File `data.yaml`

Il file `data.yaml` è necessario per l'addestramento con YOLOv8. Di seguito si riporta un esempio del suo contenuto:

```
1 names:
2 - destra
3 - limite_30
4 - limite_50
5 - limite_80
6 - precedenza
7 - rotatoria
8 - sinistra
9 - stop
10 nc: 8
11 roboflow:
12   license: CC BY 4.0
13   project: progetto-ml
14   url: https://universe.roboflow.com/progettoml/progetto-ml/
15     dataset/7
16   version: 7
17   workspace: progettoml
18 test: ../test/images
19 train: ../train/images
20 val: ../valid/images
```

- `names` è la lista delle classi presenti nel dataset.
- `nc` rappresenta il numero totale di classi (8).
- I campi `train`, `val` e `test` indicano i percorsi relativi alle immagini nei rispettivi set.
- La sezione `roboflow` contiene metadati utili per la tracciabilità del dataset.

### 2.6.3 Formato dei File di Etichetta

Ogni immagine ha un corrispondente file di etichetta (con estensione `.txt`) che contiene una o più righe nel formato YOLOv8. Ogni riga rappresenta un oggetto individuato e ha la seguente struttura:

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

Tutti i valori sono **normalizzati** rispetto alle dimensioni dell'immagine (cioè compresi tra 0 e 1).

**Esempio:**

```
1 1 0.53359375 0.3171875 0.1703125 0.3171875
```

Questa riga può essere interpretata come segue:

- **1** — classe con ID 1, corrispondente a `limite_30`
- **0.5336** — coordinata orizzontale del centro della bounding box
- **0.3172** — coordinata verticale del centro della bounding box
- **0.1703** — larghezza della bounding box
- **0.3172** — altezza della bounding box

Questo formato consente al modello YOLO di interpretare correttamente la posizione e la dimensione degli oggetti all'interno delle immagini, in maniera scalabile su qualsiasi risoluzione.

## 2.7 Data Augmentation

Per migliorare le prestazioni del modello e garantire una distribuzione più equilibrata tra le classi, è stata applicata una strategia di data augmentation mirata alle categorie **"destra"** e **"sinistra"**, che inizialmente presentavano un numero di campioni inferiore rispetto ad altre classi del dataset.

La data augmentation è stata implementata utilizzando la libreria **Albumentations**, che offre un'ampia gamma di trasformazioni ottimizzate per l'elaborazione delle immagini. L'obiettivo principale di questa fase è stato quello di aumentare artificialmente il numero di esempi disponibili per l'addestramento, mantenendo al contempo la variabilità e la qualità dei dati originali.

Le trasformazioni applicate sono state selezionate con particolare attenzione per preservare le caratteristiche distintive dei segnali stradali, evitando modifiche che potessero compromettere la leggibilità o l'interpretazione corretta dei simboli. Il processo di augmentation ha previsto la generazione di 5 varianti per ogni immagine originale, moltiplicando efficacemente il dataset disponibile per le classi target.

### 2.7.1 Trasformazioni applicate

Le trasformazioni implementate comprendono diverse categorie di modifiche, ciascuna progettata per simulare condizioni reali che il sistema potrebbe incontrare in ambiente operativo:

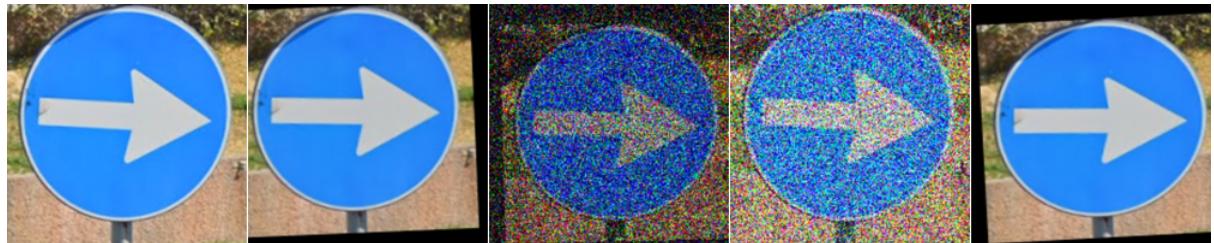
**Variazioni di luminosità e contrasto:** È stata applicata una trasformazione di tipo RandomBrightnessContrast con probabilità del 80%, che simula diverse condizioni di illuminazione naturale e artificiale. Questa trasformazione è particolarmente importante per rendere il sistema robusto alle variazioni di luce che si verificano durante le diverse ore del giorno e in condizioni meteorologiche variabili.

**Effetti di movimento:** L'applicazione di MotionBlur con limite di sfocatura di 3 pixel e probabilità del 30% replica le condizioni di acquisizione da veicoli in movimento, un scenario tipico per i sistemi di riconoscimento automatico dei segnali stradali. Questa trasformazione aiuta il modello a gestire immagini leggermente sfocate dovute al movimento relativo tra camera e segnale.

**Trasformazioni geometriche:** La trasformazione ShiftScaleRotate è stata configurata per applicare leggere modifiche geometriche alle immagini, includendo spostamenti fino al 5% delle dimensioni dell'immagine, variazioni di scala fino al 10% e rotazioni fino a 5 gradi. Questi parametri sono stati scelti per mantenere l'integrità visiva del segnale pur introducendo variabilità nella posizione e nell'orientamento, simulando diverse angolazioni di ripresa.

**Rumore gaussiano:** L'aggiunta di GaussNoise con varianza compresa tra 10 e 50 e probabilità del 40% simula le imperfezioni tipiche dei sensori fotografici e le condizioni di acquisizione non ideali. Questo tipo di rumore aiuta il modello a sviluppare robustezza verso le variazioni nella qualità dell'immagine.

**Normalizzazione dimensionale:** Tutte le immagini sono state ridimensionate a 640x640 pixel per garantire uniformità nell'input del modello, mantenendo un compromesso ottimale tra risoluzione e efficienza computazionale.



**Figura 2.3:** Trasformazioni Immagini

## 2.7.2 Implementazione del codice

L'implementazione pratica del processo di data augmentation è stata realizzata attraverso il seguente codice Python:

**Listing 2.1:** Implementazione della data augmentation per le classi "destra" e "sinistra"

```

1 import os
2 import cv2
3 import albumentations as A
4 from tqdm import tqdm
5
6 # Cartelle
7 input_dir = "input_dir"
8 output_dir = "output_dir"
9 os.makedirs(output_dir, exist_ok=True)
10
11 # Augmentazioni sicure
12 augment = A.Compose([
13     A.RandomBrightnessContrast(p=0.8),
14     A.MotionBlur(blur_limit=3, p=0.3),
15     A.ShiftScaleRotate(shift_limit=0.05, scale_limit=0.1,
16                         rotate_limit=5, p=0.7),
17     A.GaussNoise(var_limit=(10.0, 50.0), p=0.4),
18     A.Resize(640, 640),
19 ])
20
21 copies_per_img = 5
22
23 # Applica augmentation
24 for file in tqdm(sorted(os.listdir(input_dir))):
25     if not file.lower().endswith((".jpg", ".png", ".jpeg")):
26         continue
27     img_path = os.path.join(input_dir, file)
28     img = cv2.imread(img_path)
29     if img is None:
30         print(f"Immagine non valida: {img_path}")
31         continue
32     for i in range(copies_per_img):
33         augmented = augment(image=img) ["image"]
34         name = f"{os.path.splitext(file)[0]}_aug_{i+1}.jpg"
35         cv2.imwrite(os.path.join(output_dir, name), augmented)

```

Il codice implementa un processo automatizzato che itera attraverso tutte le immagini presenti nella cartella di input, applica le trasformazioni definite nella pipeline di augmentation e salva le versioni modificate nella cartella di output. L'utilizzo della libreria tqdm fornisce un feedback visivo sul progresso dell'operazione, mentre la gestione degli errori assicura che immagini corrotte o non valide vengano identificate e saltate durante il processo.

# Capitolo 3

## Metodi

Questo capitolo presenta la metodologia adottata per il riconoscimento automatico dei segnali stradali, con particolare attenzione all’evoluzione degli approcci di object detection, culminando nella scelta di **YOLOv8** come architettura di riferimento. Dopo una panoramica dei principi alla base del paradigma **one-stage** introdotto da YOLO, verranno analizzate in dettaglio l’architettura, le innovazioni tecniche e le motivazioni che giustificano la selezione di YOLOv8 nel contesto applicativo dei sistemi di guida assistita.

### 3.1 YOLO: You Only Look Once - Approccio One-Stage per l’Object Detection

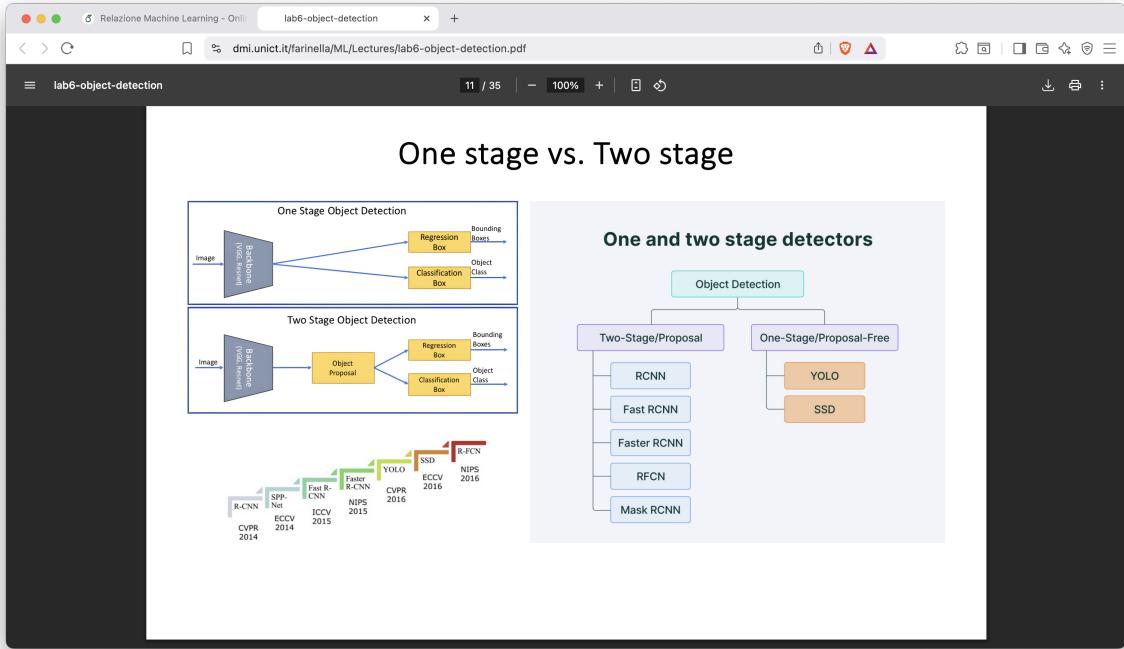
L’evoluzione dei metodi di object detection ha portato alla definizione di due paradigmi principali: gli approcci **two-stage** e **one-stage**. Mentre i primi, rappresentati dalla famiglia R-CNN, adottano una strategia sequenziale che separa la generazione di region proposals dalla classificazione degli oggetti, i secondi propongono un framework unificato che risolve simultaneamente i task di localizzazione e classificazione.

In questo contesto, YOLO (You Only Look Once) rappresenta un punto di svolta nell’ambito dell’object detection, introducendo per la prima volta un approccio one-stage che formula il problema come una singola regressione diretta dalle pixel dell’immagine alle coordinate dei bounding box e alle probabilità di classe. Questa formulazione innovativa consente di ottenere prestazioni in real-time, raggiungendo velocità di processing fino a 45 frames per second, performance che risultano significativamente superiori rispetto ai metodi two-stage tradizionali.

#### 3.1.1 Architettura e Principi Fondamentali

L’architettura YOLO si distingue per la sua elegante semplicità concettuale: l’intera immagine viene processata da una singola rete neurale convoluzionale che produce direttamente le predizioni finali. Questo approccio elimina la necessità di una fase separata di region proposal generation, caratteristica invece fondamentale nei metodi two-stage come R-CNN, Fast R-CNN e Faster R-CNN.

Il framework YOLO divide l’immagine di input in una griglia di dimensione  $S \times S$  celle, dove ogni cella è responsabile della predizione degli oggetti i cui centri cadono all’interno



**Figura 3.1:** One Stage vs Two Stage

della cella stessa. Ogni cella predice  $B$  bounding box, ciascuno caratterizzato da cinque parametri: le coordinate  $(x, y)$  del centro del box relativo alla cella, le dimensioni  $(w, h)$  del box relative all'intera immagine, e un confidence score che rappresenta la probabilità che il box contenga un oggetto moltiplicata per l'Intersection over Union (IoU) tra il box predetto e il ground truth.

Parallelamente, ogni cella predice  $C$  probabilità di classe condizionali, rappresentanti  $P(\text{Class}_i|\text{Object})$ . Durante la fase di inference, queste probabilità vengono moltiplicate per i confidence score individuali dei box, ottenendo così confidence score specifici per ogni classe:  $P(\text{Class}_i|\text{Object}) \times P(\text{Object}) \times \text{IoU}_{\text{truth}}^{\text{pred}}$ .

### 3.1.2 Vantaggi e Limitazioni del Paradigma One-Stage

L'approccio one-stage di YOLO presenta numerosi vantaggi rispetto ai metodi two-stage. Innanzitutto, la velocità di processing rappresenta il beneficio più evidente: mentre Faster R-CNN richiede approssimativamente 0.2 secondi per processare una singola immagine, YOLO raggiunge performance di real-time con tempi di elaborazione dell'ordine dei millesimi. Questa caratteristica rende YOLO particolarmente adatto per applicazioni che richiedono processing in tempo reale, come la videosorveglianza, la guida autonoma e l'analisi di flussi video.

Un ulteriore vantaggio risiede nella capacità di YOLO di ragionare globalmente sull'immagine durante la fase di training e testing. Diversamente dai metodi basati su sliding window o region proposal, YOLO osserva l'intera immagine simultaneamente, consentendo una migliore comprensione del contesto e riducendo significativamente i falsi positivi dovuti a background patches.

Tuttavia, l'approccio YOLO presenta anche alcune limitazioni intrinseche. La principale criticità riguarda la detection di oggetti di piccole dimensioni, particolarmente quando questi appaiono in gruppi. Questa limitazione deriva dalle forti vincoli spaziali imposte dall'architettura: ogni cella della griglia può predire solamente  $B$  bounding box e appartiene a una sola classe, rendendo difficile la detection di oggetti piccoli e ravvicinati che possono cadere nella stessa cella.

### 3.1.3 Confronto con Metodi Two-Stage

Il confronto tra YOLO e i metodi two-stage rivela trade-off significativi tra velocità e accuratezza. Mentre Faster R-CNN raggiunge performance superiori in termini di mean Average Precision (mAP) su dataset standard come PASCAL VOC e COCO, YOLO offre un compromesso interessante per applicazioni dove la velocità di processing è un requisito critico.

L'analisi delle performance temporali mostra l'evoluzione dei metodi two-stage: dall'originale R-CNN che richiedeva 47 secondi per immagine, a Fast R-CNN con 2.3 secondi, fino a Faster R-CNN con 0.2 secondi. Tuttavia, anche la versione più efficiente dei metodi two-stage risulta significativamente più lenta rispetto a YOLO, che mantiene performance di 45 fps.

Questa differenza prestazionale deriva fondamentalmente dall'architettura: i metodi two-stage devono processare migliaia di region proposals (tipicamente 2000 in R-CNN), mentre YOLO effettua predizioni dirette su un numero fisso di locations, determinato dalle dimensioni della griglia di output.

## 3.2 Architettura del Sistema

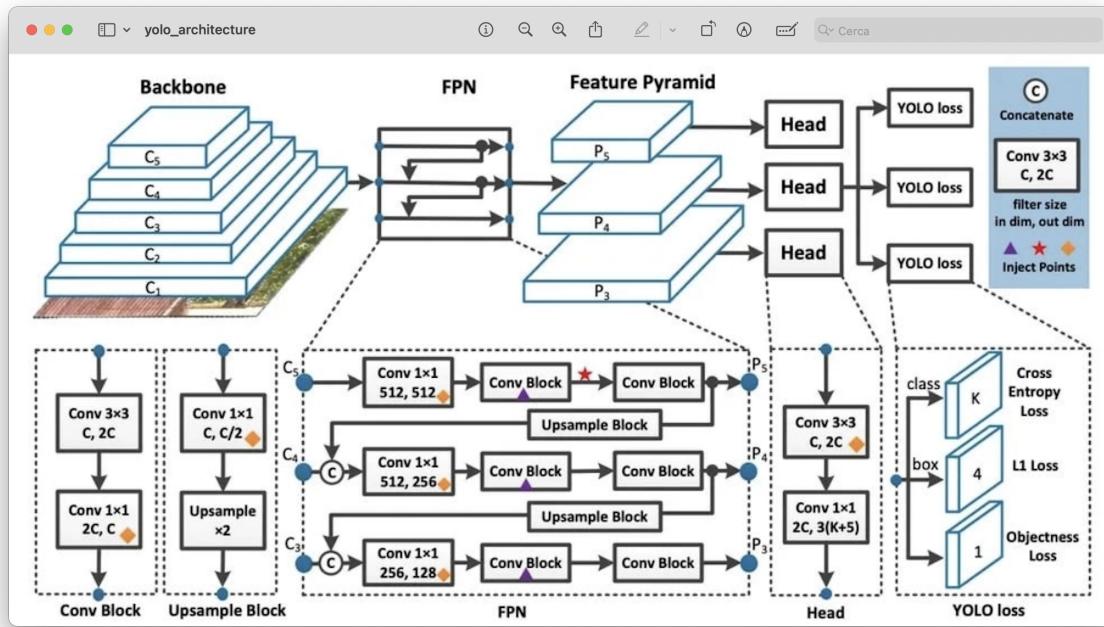
La pipeline di riconoscimento automatico si articola attraverso una sequenza di operazioni coordinate che trasformano l'input grezzo in predizioni strutturate. Il processo inizia con l'acquisizione e preprocessing dei dati, dove le immagini vengono ridimensionate a una risoluzione standard di  $640 \times 640$  pixel, normalizzate secondo i parametri di training del modello e convertite in tensori compatibili con l'architettura neurale. Questa fase di preprocessing è fondamentale per garantire la coerenza dei dati in ingresso e ottimizzare le prestazioni del modello durante l'inferenza.

Il rilevamento vero e proprio viene effettuato tramite YOLOv8, che elabora le immagini attraverso un'architettura neurale convoluzionale specificatamente progettata per task di object detection. L'architettura, descritta dettagliatamente nella sezione successiva, combina efficienza computazionale e capacità rappresentativa per gestire la varianabilità intrinseca dei segnali stradali in termini di dimensioni, orientamento e condizioni ambientali.

La fase di post-processing include l'applicazione di algoritmi di Non-Maximum Suppression (NMS) per rimuovere le predizioni ridondanti e sovrapposti, garantendo output puliti e privi di duplicazioni. Infine, il sistema restituisce le coordinate delle bounding box in formato normalizzato, la classe del segnale rilevato e il punteggio di confidenza associato a ciascuna predizione.

### 3.3 YOLOv8: Architettura Dettagliata

YOLOv8 rappresenta l'evoluzione più recente della famiglia YOLO, introducendo significative innovazioni architettoniche che migliorano sia l'accuratezza che l'efficienza computazionale rispetto alle versioni precedenti. L'architettura implementa una pipeline completamente convoluzionale composta da tre componenti principali: il backbone per l'estrazione delle feature, il neck per la fusione multi-scala delle informazioni, e la detection head per la generazione delle predizioni finali.



**Figura 3.2:** Architettura di YOLOv8

#### 3.3.1 Backbone: Estrazione Gerarchica delle Feature

Il backbone di YOLOv8 costituisce la componente responsabile dell'estrazione delle rappresentazioni visive dai dati grezzi. L'architettura utilizza blocchi C2f (Cross Stage Partial con Fusione), un'innovazione che massimizza il riutilizzo delle feature mantenendo efficiente il flusso informativo attraverso la rete. Questa soluzione architettonica rappresenta un miglioramento significativo rispetto ai precedenti blocchi C3 utilizzati in YOLOv5, introducendo connessioni residuali più efficaci e riducendo la ridondanza computazionale.

I livelli convoluzionali del backbone, denominati  $C_1$  attraverso  $C_5$ , apprendono progressivamente feature di complessità crescente seguendo il paradigma della rappresentazione gerarchica tipico delle reti neurali profonde. I primi livelli ( $C_1$  e  $C_2$ ) si specializzano nella cattura di pattern locali quali bordi, texture e gradienti di intensità, fornendo una base robusta per l'elaborazione successiva. I livelli superiori ( $C_3$ ,  $C_4$  e  $C_5$ ) sviluppano invece rappresentazioni semantiche più astratte, capaci di codificare informazioni relative a forme complesse e strutture oggetto-specifiche.

### 3.3.2 Neck: Fusione Multi-scala con FPN e PAN

La gestione efficace di oggetti di dimensioni diverse rappresenta una sfida centrale nell'object detection, particolarmente rilevante nel contesto dei segnali stradali che possono variare significativamente in scala a seconda della distanza di osservazione. YOLOv8 affronta questa problematica attraverso un'architettura di neck che combina Feature Pyramid Network (FPN) e Path Aggregation Network (PAN).

Il meccanismo di fusione multi-scala opera concatenando informazioni provenienti da diversi livelli del backbone, creando una piramide di feature ( $P_3, P_4, P_5$ ) che mantiene sia la risoluzione spaziale necessaria per rilevare oggetti piccoli sia la profondità semantica richiesta per oggetti di dimensioni maggiori. Questa strategia consente al modello di mantenere un'elevata sensibilità per segnali di piccole dimensioni, spesso critici nelle applicazioni di guida autonoma, senza sacrificare la capacità di rilevamento per oggetti di scala maggiore.

### 3.3.3 Detection Head: Predizione Disaccoppiata

Una delle innovazioni più significative introdotte in YOLOv8 è la detection head disaccoppiata, che rappresenta un'evoluzione rispetto alle architetture precedenti dove classificazione e localizzazione condividevano parametri comuni. La head disaccoppiata suddivide il processo predittivo in tre rami specializzati che operano in parallelo sulle feature multi-scala estratte dal neck.

Il regression branch si concentra esclusivamente sulla predizione delle coordinate delle bounding box, ottimizzando i parametri per la precisione geometrica della localizzazione. L'objectness branch stima la probabilità che una regione specifica contenga un oggetto di interesse, fornendo una misura di confidenza sulla presenza effettiva di un target. Il classification branch assegna una classe semantica all'oggetto rilevato, specializzandosi nell'identificazione delle diverse tipologie di segnali stradali presenti nel dataset.

Ciascun ramo è implementato attraverso una sequenza di convoluzioni  $3 \times 3$  e  $1 \times 1$ , dove le prime catturano dipendenze spaziali locali mentre le seconde fungono da trasformazioni puntuali per la riduzione dimensionale e l'adattamento delle feature. Questa architettura parallela consente una specializzazione ottimale dei parametri per ciascun task, migliorando significativamente le prestazioni complessive del modello.

### 3.3.4 Anchor-Free Prediction

YOLOv8 adotta un approccio anchor-free che rappresenta una semplificazione significativa rispetto alle versioni precedenti della famiglia YOLO. Tradizionalmente, i modelli YOLO utilizzavano anchor box predefinite che fungevano da template per la generazione delle predizioni, richiedendo un'accurata configurazione manuale e limitando la capacità di generalizzazione del modello.

L'approccio anchor-free elimina questa dipendenza permettendo a ogni cella della feature map di predire direttamente le distanze dal centro dell'oggetto ai suoi bordi (left, top, right, bottom). Questa metodologia semplifica notevolmente la configurazione del modello, eliminando la necessità di definire anchor box specifiche per il dominio applicativo e migliora la capacità di generalizzazione su oggetti di forme e dimensioni diverse. Inoltre,

tre, la riduzione della complessità computazionale associata alla gestione degli anchor contribuisce a migliorare l'efficienza durante l'inferenza.

### 3.3.5 YOLO Loss: Ottimizzazione End-to-End

L'ottimizzazione del modello YOLOv8 avviene attraverso una funzione di loss composta che bilancia i diversi obiettivi del task di object detection. La classification loss, implementata tramite cross-entropy, valuta l'accuratezza delle predizioni di classe, penalizzando le classificazioni errate e incoraggiando predizioni confident per le classi corrette.

La box loss combina multiple metriche geometriche per valutare la precisione della localizzazione. Mentre la loss L1 misura la distanza euclidea tra coordinate predette e ground truth, metriche più sofisticate come GIoU (Generalized Intersection over Union) e DIoU (Distance-IoU) incorporano informazioni sulla forma e posizione relativa delle bounding box, fornendo un segnale di ottimizzazione più informativo.

L'objectness loss valuta la capacità del modello di distinguere tra regioni contenenti oggetti e background, contribuendo alla robustezza delle predizioni in scenari con alta densità di oggetti o presenza di distrattori visivi.

## 3.4 Innovazioni Architetturali

Le principali innovazioni introdotte in YOLOv8 rispetto alle versioni precedenti includono l'implementazione dei blocchi C2f, che migliorano la profondità informativa riducendo la ridondanza computazionale attraverso connessioni residuali più efficaci. Il design anchor-free semplifica significativamente la gestione delle predizioni, eliminando la necessità di configurazione manuale degli anchor e migliorando la generalizzazione.

La head disaccoppiata rappresenta un'evoluzione fondamentale che consente la specializzazione ottimale dei parametri per ciascun task, mentre l'integrazione nativa della Non-Maximum Suppression nel processo di inferenza elimina le predizioni ridondanti mantenendo l'efficienza computazionale.

Questi miglioramenti architetturali permettono a YOLOv8 di raggiungere un equilibrio ottimale tra velocità e accuratezza, mantenendo i vantaggi del paradigma one-stage mentre mitigando le limitazioni storiche della famiglia YOLO, particolarmente nella detection di oggetti di piccole dimensioni.

### 3.5 Motivazioni per la Scelta di YOLOv8

La selezione di YOLOv8 come architettura di riferimento per questo progetto è motivata da diverse considerazioni tecniche e applicative. Le prestazioni del modello offrono un equilibrio eccellente tra precisione e velocità di inferenza, caratteristica fondamentale per applicazioni in tempo reale come i sistemi ADAS (Advanced Driver Assistance Systems).

La flessibilità architetturale del modello, disponibile in varianti Nano, Small, Medium e Large, consente l'adattamento a diversi contesti hardware, dalla prototipazione su workstation ad alte prestazioni fino al deployment su dispositivi edge con risorse limitate. La semplicità di training, garantita dall'architettura anchor-free e dall'auto-ottimizzazione dei parametri, riduce significativamente il tempo di sviluppo e la complessità di configurazione.

L'adattabilità del modello alle caratteristiche specifiche dei segnali stradali è garantita dalle reti multi-scala e dalla head disaccoppiata, che gestiscono efficacemente l'eterogeneità in termini di dimensioni, forme e condizioni di illuminazione tipiche del dominio applicativo. La capacità di mantenere performance elevate sia per oggetti di piccole dimensioni (segnali distanti) sia per oggetti di grandi dimensioni (segnali vicini) rappresenta un requisito critico per l'affidabilità del sistema.

Infine, il supporto maturo fornito dalla piattaforma Ultralytics garantisce strumenti moderni e ben documentati per training, valutazione e deployment, accelerando lo sviluppo e facilitando la manutenzione del sistema. L'ecosistema integrato comprende utilities per la gestione dei dataset, monitoring del training, ottimizzazione automatica degli iperparametri e esportazione in formati ottimizzati per diversi target hardware.

L'architettura multi-scala, la head specializzata e l'ottimizzazione end-to-end rendono YOLOv8 ideale per applicazioni ADAS, dove l'affidabilità delle predizioni e la velocità di risposta sono requisiti essenziali per garantire la sicurezza operativa del sistema.

# Capitolo 4

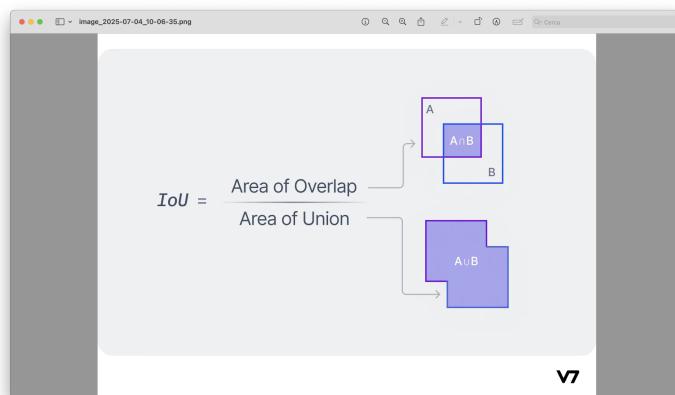
## Valutazione

Il presente capitolo descrive le metriche di valutazione che verranno impiegate nella successiva sezione sperimentale per quantificare le prestazioni del modello YOLO sviluppato. La selezione delle misure di evaluation è stata effettuata in base agli standard consolidati nella comunità scientifica dell'Object Detection, al fine di garantire una valutazione rigorosa e comparabile con i risultati presenti in letteratura.

Le metriche descritte in questo capitolo costituiscono il framework metodologico che guiderà l'analisi sperimentale, fornendo gli strumenti necessari per valutare sia l'accuratezza della localizzazione degli oggetti che l'efficacia complessiva del sistema di detection implementato. La comprensione di questi indicatori di performance risulta fondamentale per l'interpretazione dei risultati che verranno presentati nella sezione sperimentale successiva.

### 4.1 Intersection over Union (IoU)

La metrica di Intersection over Union rappresenta la misura primaria che verrà utilizzata negli esperimenti per valutare la qualità della localizzazione degli oggetti predetti dal modello YOLO. Questa metrica quantifica il grado di sovrapposizione tra le bounding box predette dal sistema e quelle definite come ground truth nel dataset di riferimento.



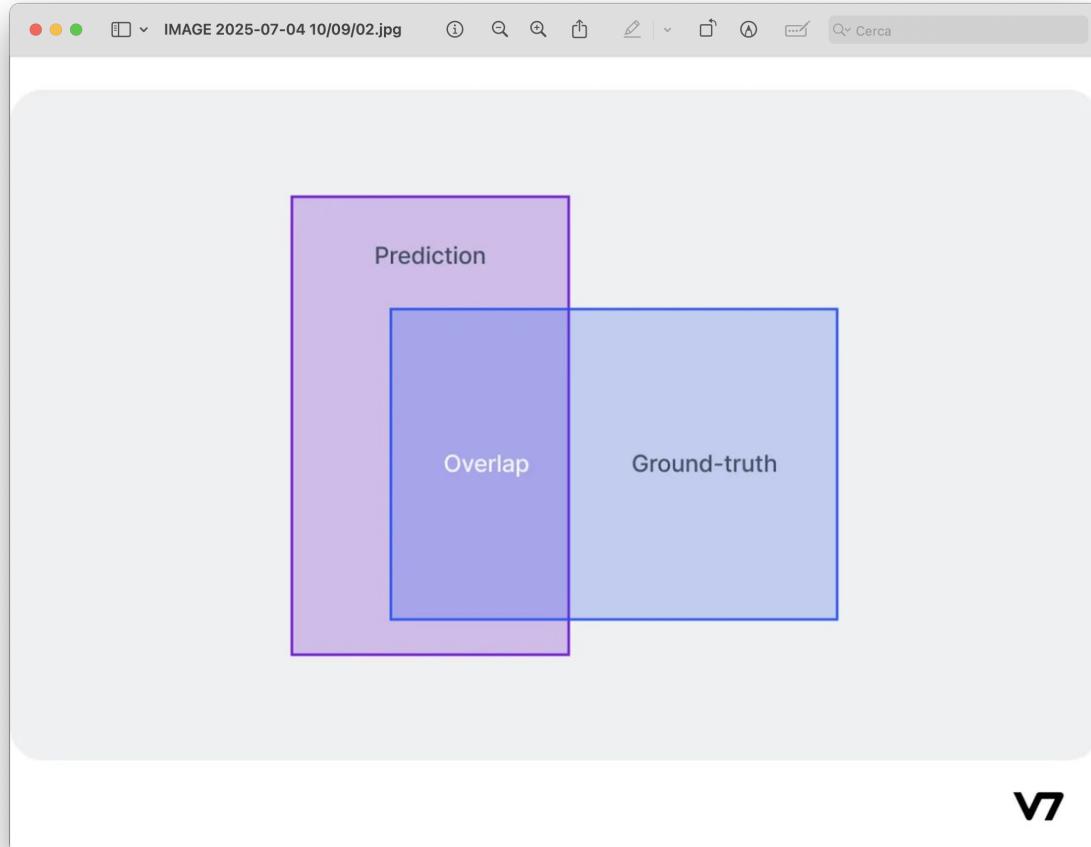
**Figura 4.1:** IoU

La formulazione matematica dell'IoU che verrà applicata durante la fase sperimentale è definita come il rapporto tra l'area di intersezione e l'area di unione delle due bounding box considerate. Formalmente, dato un oggetto con bounding box ground truth  $B_{gt}$  e bounding box predetta  $B_{pred}$ , l'IoU sarà calcolato secondo la relazione:

$$IoU = \frac{|B_{gt} \cap B_{pred}|}{|B_{gt} \cup B_{pred}|} \quad (4.1)$$

dove  $|B_{gt} \cap B_{pred}|$  rappresenta l'area di sovrapposizione tra le due bounding box e  $|B_{gt} \cup B_{pred}|$  denota l'area totale coperta dall'unione delle stesse.

Negli esperimenti successivi, una predizione verrà considerata corretta quando il valore di IoU supererà la soglia di 0.5, in accordo con gli standard adottati nella maggior parte dei benchmark di Object Detection Saranno inoltre condotte analisi comparative utilizzando soglie più restrittive (0.7 e 0.9) per valutare la robustezza del modello in condizioni di maggiore precisione richiesta.



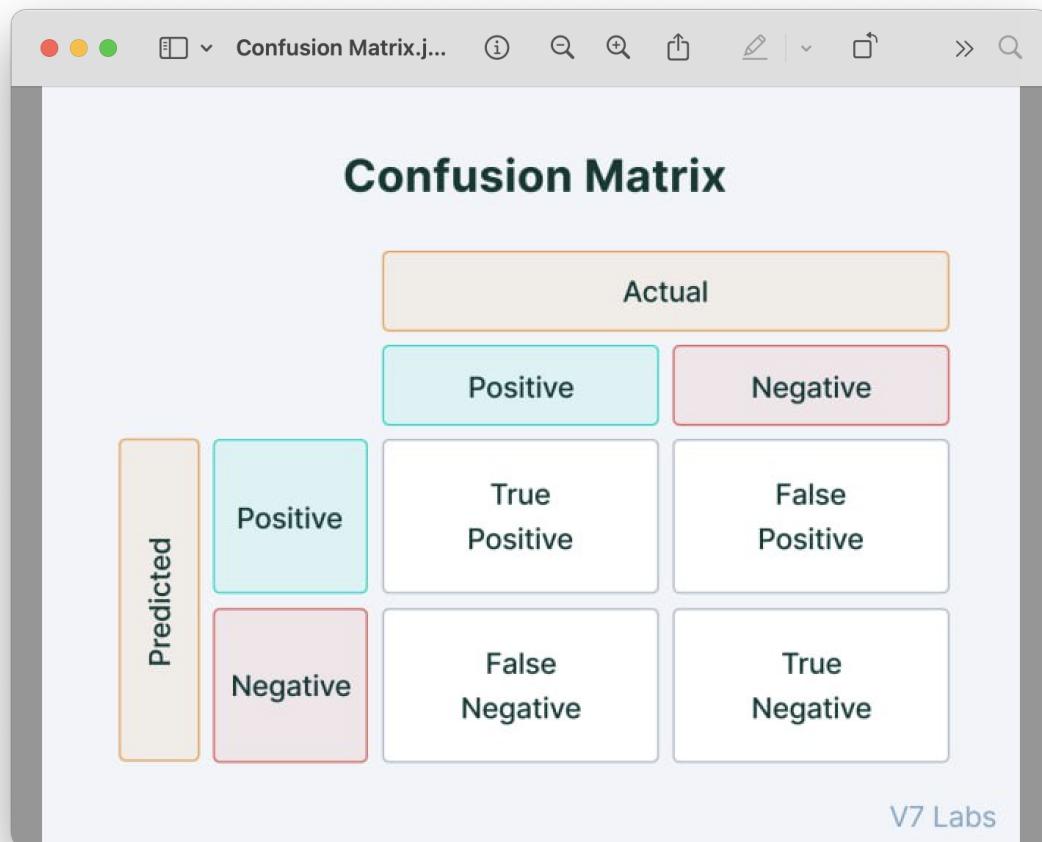
**Figura 4.2:** Prediction Overlap Ground-Truth

## 4.2 Confusion Matrix e valutazione delle prestazioni

Nel contesto dell'object detection, la *confusion matrix* rappresenta uno strumento essenziale per analizzare le prestazioni di un modello, permettendo di distinguere tra *True Positives* (TP), *False Positives* (FP) e *False Negatives* (FN). Un'istanza predetta viene considerata *positiva* se la classe è corretta e la *Intersection over Union* (IoU) tra il bounding box predetto e quello reale supera una soglia prefissata, solitamente pari a 0.5.

Da questi valori si ricavano metriche fondamentali: la *precision*, quantifica la percentuale di predizioni corrette tra tutte le istanze rilevate; la *recall*, misura invece la capacità del modello di individuare gli oggetti presenti.

Nel nostro progetto, basato sull'architettura YOLO, tali metriche sono state calcolate confrontando le predizioni con le annotazioni del ground truth sul test set. Questo ha permesso una valutazione quantitativa delle performance e ha guidato eventuali strategie di ottimizzazione.



**Figura 4.3:** Confusion Matrix

### 4.3 Average Precision (AP)

L'Average Precision costituisce la metrica principale che verrà impiegata nella valutazione sperimentale per quantificare le prestazioni complessive del sistema di Object Detection. Questa misura integra le informazioni relative sia alla capacità di detection (recall) che alla precisione delle predizioni, fornendo un indicatore sintetico delle performance del modello.

Il calcolo dell'AP negli esperimenti si baserà sulla costruzione della curva Precision-Recall, ottenuta variando sistematicamente la soglia di confidenza applicata alle predizioni del modello YOLO. Per ogni valore di soglia, verranno determinati i corrispondenti valori di precision e recall attraverso le seguenti definizioni:

$$Precision = \frac{TP}{TP + FP} \quad (4.2)$$

$$Recall = \frac{TP}{TP + FN} \quad (4.3)$$

dove  $TP$  (True Positives) rappresenta il numero di detection corrette,  $FP$  (False Positives) il numero di detection errate e  $FN$  (False Negatives) il numero di oggetti non rilevati dal modello.

L'Average Precision che verrà calcolata negli esperimenti è matematicamente definita come l'area sotto la curva Precision-Recall (AUC):

$$AP = \int_0^1 p(r) dr \quad (4.4)$$

dove  $p(r)$  rappresenta la precision in funzione del recall  $r$ .

### 4.4 Mean Average Precision (mAP)

La Mean Average Precision rappresenta la metrica complessiva che verrà utilizzata negli esperimenti per valutare le prestazioni del modello YOLO su tutte le classi di oggetti presenti nel dataset. Questa misura è calcolata come la media aritmetica delle Average Precision di tutte le classi considerate:

$$mAP = \frac{1}{N} \sum_{i=1}^N AP_i \quad (4.5)$$

dove  $N$  rappresenta il numero totale di classi e  $AP_i$  è l'Average Precision per la classe  $i$ -esima.

Negli esperimenti successivi, la mAP verrà calcolata utilizzando diverse soglie di IoU, in particolare mAP@0.5 (soglia IoU = 0.5) e mAP@0.5:0.95 (media delle mAP calcolate per soglie IoU da 0.5 a 0.95 con step di 0.05). Questa duplice valutazione consentirà di ottenere una visione completa delle prestazioni del modello, considerando sia scenari di valutazione più permissivi che più rigorosi.

## 4.5 Recall

La metrica di Recall rappresenta un indicatore fondamentale che verrà utilizzato negli esperimenti per valutare la capacità del modello di identificare correttamente tutti gli oggetti presenti nelle immagini del dataset di test. Questa misura quantifica la frazione di oggetti effettivamente presenti che vengono rilevati dal sistema di detection.

Il Recall verrà calcolato durante la fase sperimentale secondo la definizione:

$$\text{Recall} = \frac{TP}{TP + FN} \quad (4.6)$$

dove  $TP$  rappresenta il numero di true positives (oggetti correttamente rilevati) e  $FN$  il numero di false negatives (oggetti presenti ma non rilevati dal modello).

Negli esperimenti successivi, il Recall verrà valutato sia globalmente per l'intero dataset che individualmente per ciascuna classe di oggetti. Questa analisi permetterà di identificare eventuali categorie di oggetti per le quali il modello mostra particolare difficoltà nel rilevamento, fornendo indicazioni preziose per eventuali ottimizzazioni successive del sistema.

## 4.6 F1-Score

La metrica F1-Score costituisce un indicatore sintetico che verrà impiegato negli esperimenti per valutare l'equilibrio complessivo tra Precision e Recall del modello YOLO. Questa misura rappresenta la media armonica delle due metriche fondamentali, fornendo un valore unico che riflette le prestazioni bilanciate del sistema di detection.

Il calcolo dell'F1-Score che verrà implementato durante la fase sperimentale seguirà la formulazione:

$$F1 = 2 \cdot \frac{\text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}} \quad (4.7)$$

L'utilizzo della media armonica anziché di quella aritmetica garantisce che valori bassi di una delle due metriche componenti influenzino significativamente il risultato finale, penalizzando modelli che privilegiano eccessivamente una metrica a scapito dell'altra.

Negli esperimenti verrà calcolato l'F1-Score per diverse soglie di confidenza, permettendo di identificare il punto operativo ottimale che massimizza questo indicatore di performance bilanciata. Inoltre, sarà condotta un'analisi per classe dell'F1-Score, fornendo una valutazione dettagliata delle prestazioni del modello su ciascuna categoria di oggetti.

## 4.7 Precision-Recall Curve

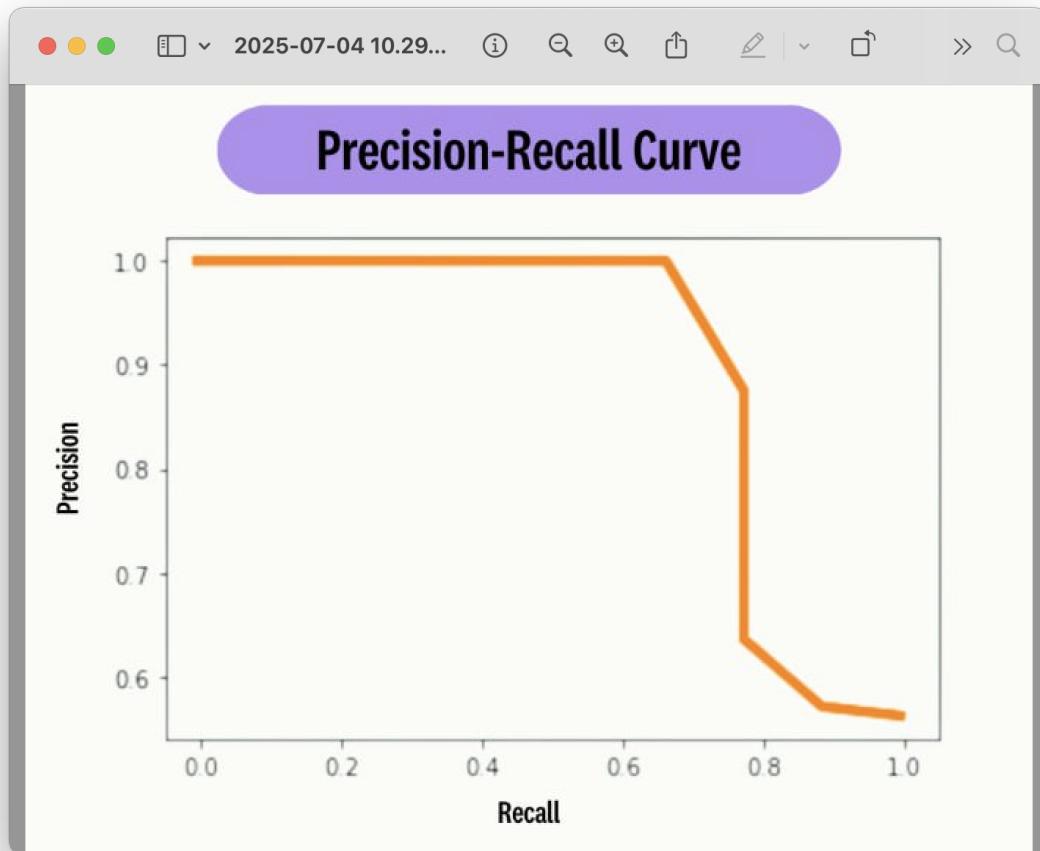
La Precision-Recall Curve rappresenta uno strumento di analisi visuale che verrà utilizzato negli esperimenti per caratterizzare in modo dettagliato il comportamento del modello YOLO al variare della soglia di confidenza applicata alle predizioni. Questa curva fornisce

una rappresentazione grafica completa del trade-off tra precision e recall, permettendo di identificare il punto operativo ottimale per l'applicazione specifica.

La costruzione della Precision-Recall Curve negli esperimenti avverrà attraverso la variazione sistematica della soglia di confidenza da 0 a 1.

Il risultato sarà una curva che tipicamente mostra un andamento decrescente della precision all'aumentare del recall, riflettendo la natura intrinseca del trade-off tra queste due metriche.

L'analisi della Precision-Recall Curve permetterà di valutare diversi aspetti delle prestazioni del modello. La forma della curva fornirà indicazioni sulla stabilità delle prestazioni al variare della soglia di confidenza, mentre l'area sottesa dalla curva corrisponderà direttamente al valore di Average Precision precedentemente descritto.



**Figura 4.4:** Precision-Recall Curve

## 4.8 Recall Curve

La Recall Curve costituisce un’analisi complementare alla Precision-Recall Curve, focalizzandosi specificatamente sull’andamento del recall in funzione della soglia di confidenza. Questa rappresentazione grafica permette di visualizzare come varia la capacità del modello di identificare tutti gli oggetti presenti nelle immagini al modificarsi del livello di confidenza richiesto per considerare valida una predizione.

La costruzione della Recall Curve avviene attraverso il calcolo del recall per valori decrescenti della soglia di confidenza, partendo da soglie elevate (prossime a 1) fino a soglie molto basse (prossime a 0).

L’analisi della Recall Curve fornisce informazioni cruciali per la calibrazione del modello. La pendenza della curva indica la sensibilità del modello alle variazioni della soglia di confidenza: una curva con pendenza ripida suggerisce che piccole variazioni della soglia producono significative variazioni nel recall, mentre una curva con pendenza graduale indica una maggiore stabilità.

Negli esperimenti, la Recall Curve verrà utilizzata in combinazione con la Precision-Recall Curve per identificare il range ottimale di soglie di confidenza. Questa analisi congiunta permetterà di determinare il punto operativo che massimizza il recall mantenendo un livello accettabile di precision, o viceversa, in funzione dei requisiti specifici dell’applicazione.

## 4.9 Metriche Aggiuntive per l’Analisi Sperimentale

Oltre alle metriche principali descritte nelle sezioni precedenti, l’analisi sperimentale includerà ulteriori indicatori di performance che forniranno una caratterizzazione dettagliata del comportamento del modello YOLO implementato. Verranno calcolate le metriche di precision e recall individuali per ogni classe di oggetti, permettendo di identificare eventuali criticità specifiche del modello su particolari categorie di detection. Inoltre, sarà condotta un’analisi delle prestazioni in funzione delle dimensioni degli oggetti, distinguendo tra oggetti piccoli, medi e grandi secondo le convenzioni del dataset COCO.

L’efficienza computazionale del modello verrà valutata attraverso la misurazione del tempo di inferenza per singola immagine, espressa in millisecondi, e del throughput complessivo, misurato in frames per second (FPS). Queste metriche risultano fondamentali per valutare l’applicabilità del modello in scenari real-time.

Infine, verrà condotta un’analisi delle curve di learning durante la fase di training, monitorando l’evoluzione della loss function e delle metriche di validation per identificare eventuali fenomeni di overfitting o underfitting che potrebbero influenzare le prestazioni finali del modello. Le metriche descritte in questo capitolo costituiscono il framework completo che guiderà l’analisi sperimentale, fornendo tutti gli strumenti necessari per una valutazione rigorosa e completa delle prestazioni del sistema di Object Detection sviluppato.

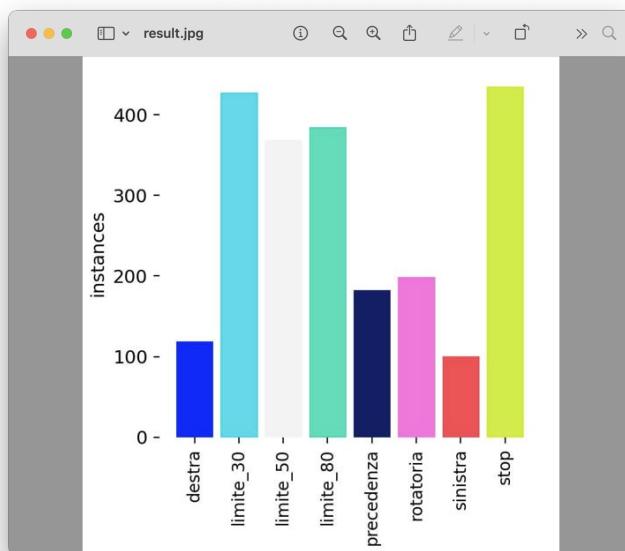
# Capitolo 5

## Esperimenti

In questo capitolo viene delineato il percorso empirico che ha guidato lo sviluppo del nostro sistema di rilevamento oggetti basato sulla famiglia di modelli *You Only Look Once* (YOLO), noti per la loro efficienza nei rilevatori a singolo stadio. L'intera indagine è stata articolata in **quattro fasi sequenziali**, finalizzate al progressivo affinamento della qualità del dataset e all'ottimizzazione degli iperparametri.

### 5.1 Fase I

Un'analisi esplorativa iniziale del dataset ha evidenziato uno squilibrio marcato nella distribuzione spaziale e semantica delle annotazioni. Il fenomeno più critico risiedeva nella scarsità di campioni relativi alle classi '**Freccia Destra**' e '**Freccia Sinistra**'. Per eseguire una prima valutazione quantitativa, è stato addestrato il modello **YOLOv8n** (la variante più leggera della famiglia YOLOv8).



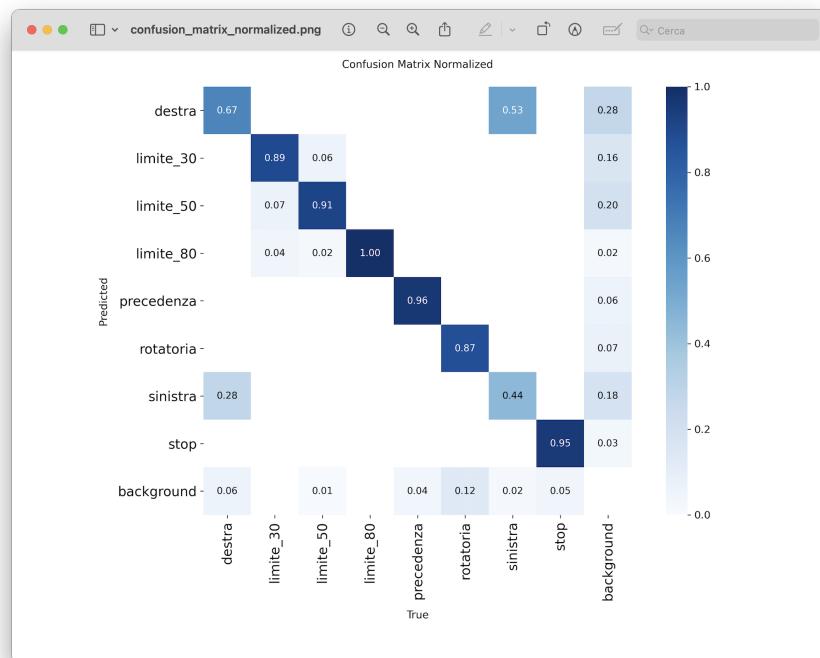
**Figura 5.1:** Distribuzione dei dati per classe

### 5.1.1 Risultati Preliminari

Nei paragrafi seguenti sono riportati i principali risultati ottenuti durante la valutazione del modello YOLOv8n sulla versione iniziale del dataset. Le metriche visuali sono fondamentali per diagnosticare problemi di *underfitting*, *class imbalance* o errata annotazione.

#### 5.1.1.1 Matrice di Confusione Normalizzata

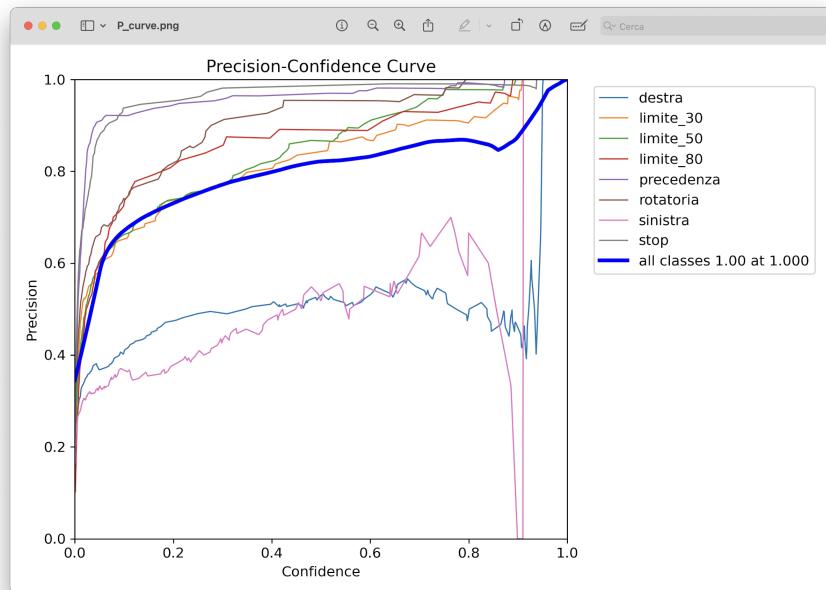
La matrice di confusione normalizzata mostra la frequenza con cui ciascuna classe viene correttamente o erroneamente classificata. In presenza di class imbalance, ci si attende una forte dominanza delle classi più rappresentate.



**Figura 5.2:** Confusion Matrix

### 5.1.1.2 Precision Curve

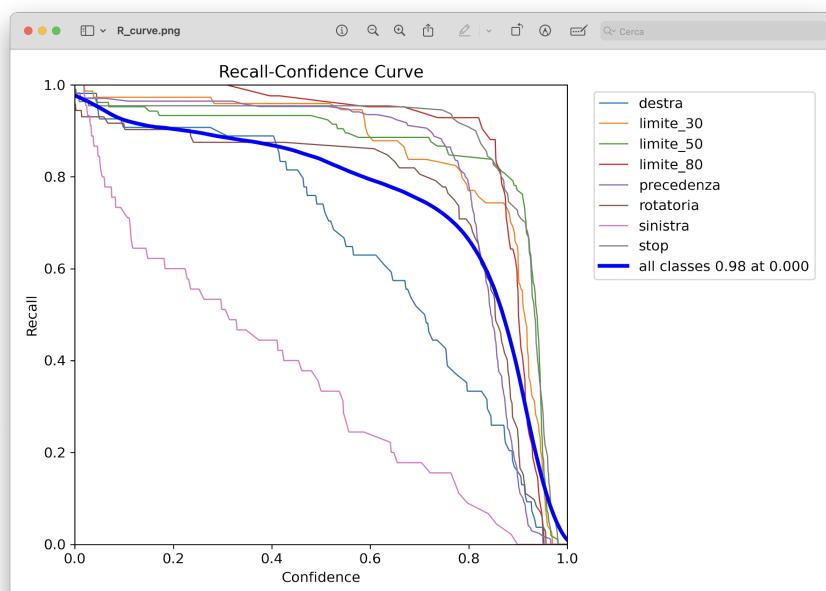
La curva di precisione mostra la capacità del modello di evitare falsi positivi su ciascuna classe, in funzione della soglia di confidenza.



**Figura 5.3:** Precision Curve

### 5.1.1.3 Recall Curve

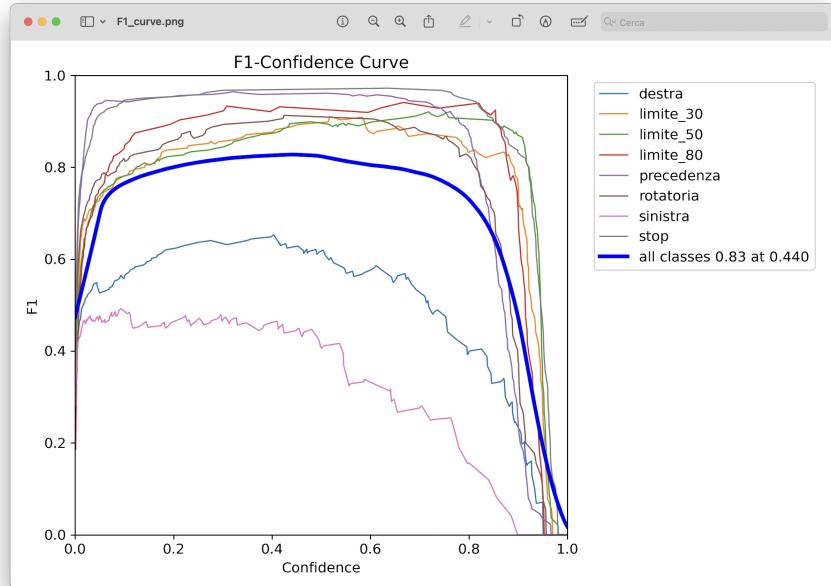
La curva di richiamo misura la capacità del modello di identificare correttamente tutti i veri positivi, indipendentemente dalla soglia.



**Figura 5.4:** Recall Curve

### 5.1.1.4 F1 Score Curve

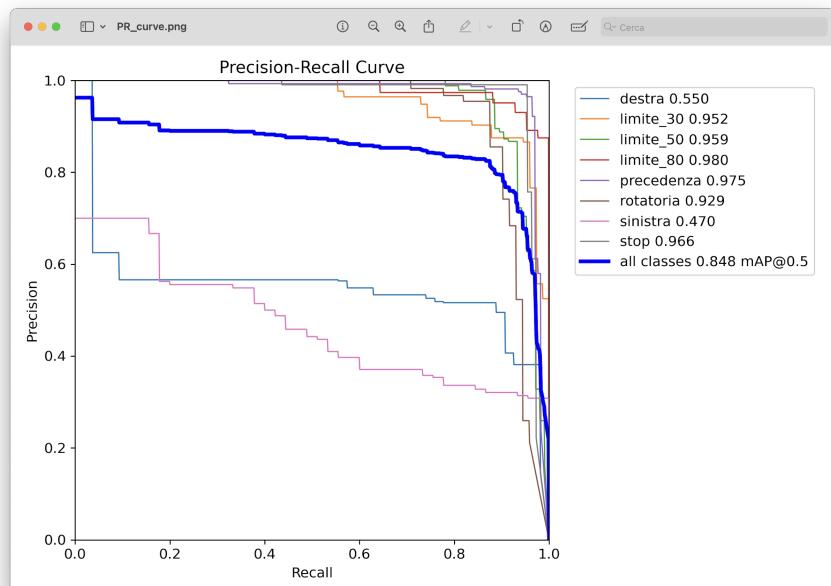
La curva dell'F1-score rappresenta l'armonica media tra precision e recall, ed è utile per valutare un compromesso tra le due metriche.



**Figura 5.5:** F1 Score Curve

### 5.1.1.5 Precision-Recall Curve

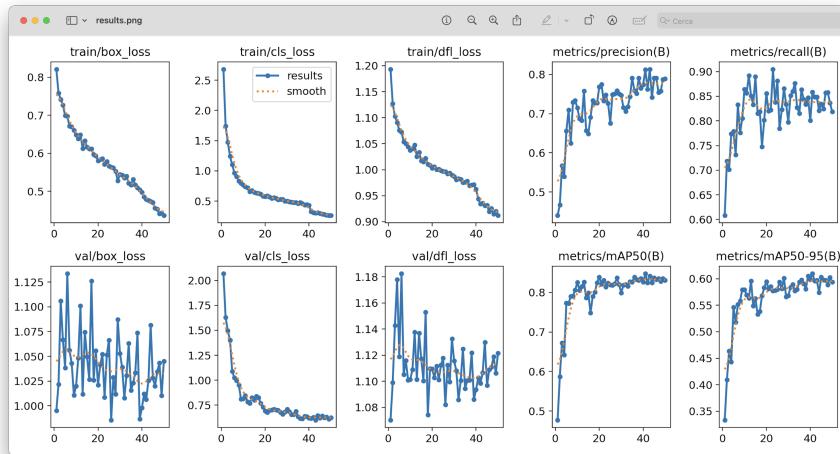
La curva precision-recall per ciascuna classe evidenzia il comportamento del classificatore in scenari sbilanciati, risultando spesso più informativa rispetto alla curva ROC.



**Figura 5.6:** PR Curve

### 5.1.1.6 Ulteriori Metriche

La figura seguente riassume l'andamento delle principali metriche durante l'addestramento e la validazione del modello. Sono inclusi i valori di `box_loss`, `cls_loss`, `dfl_loss` sia in fase di training che di validazione, oltre alle metriche di `precision`, `recall`, `mAP@0.5` e `mAP@0.5:0.95`.



**Figura 5.7:** Risultati

### 5.1.2 Discussione

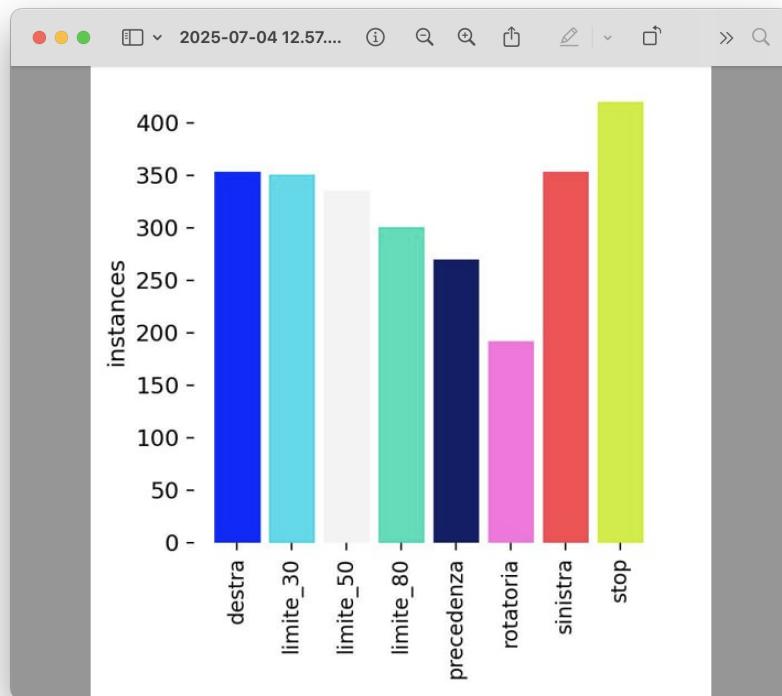
L'analisi delle metriche mostra un comportamento subottimale del modello, in particolare sulle classi “**Freccia Destra**” e “**Freccia Sinistra**”, fortemente penalizzate dal class imbalance. Le prestazioni su queste classi risultano trascurabili, con valori di `recall` e `F1-score` molto bassi.

Alla luce di ciò, nella **Fase II** si è deciso di applicare tecniche di *data augmentation* mirate alle classi minoritarie, per aumentarne la rappresentatività e valutare l'impatto sul modello. Le trasformazioni includono rotazioni, traslazioni e variazioni di luminosità.

## 5.2 Fase II

I risultati ottenuti nella Fase I hanno evidenziato chiaramente le difficoltà del modello nel riconoscere le classi “**Freccia Destra**” e “**Freccia Sinistra**”, dovute al forte squilibrio nella distribuzione del dataset. Per affrontare questo problema, si è deciso di intervenire direttamente sui dati, applicando una strategia di **data augmentation mirata**.

L’obiettivo era aumentare significativamente il numero di campioni relativi alle classi meno rappresentate, senza alterare la struttura complessiva del dataset. Sono state quindi generate numerose nuove immagini delle frecce laterali, utilizzando trasformazioni geometriche e modifiche di luminosità , preservando sempre la coerenza semantica delle annotazioni.



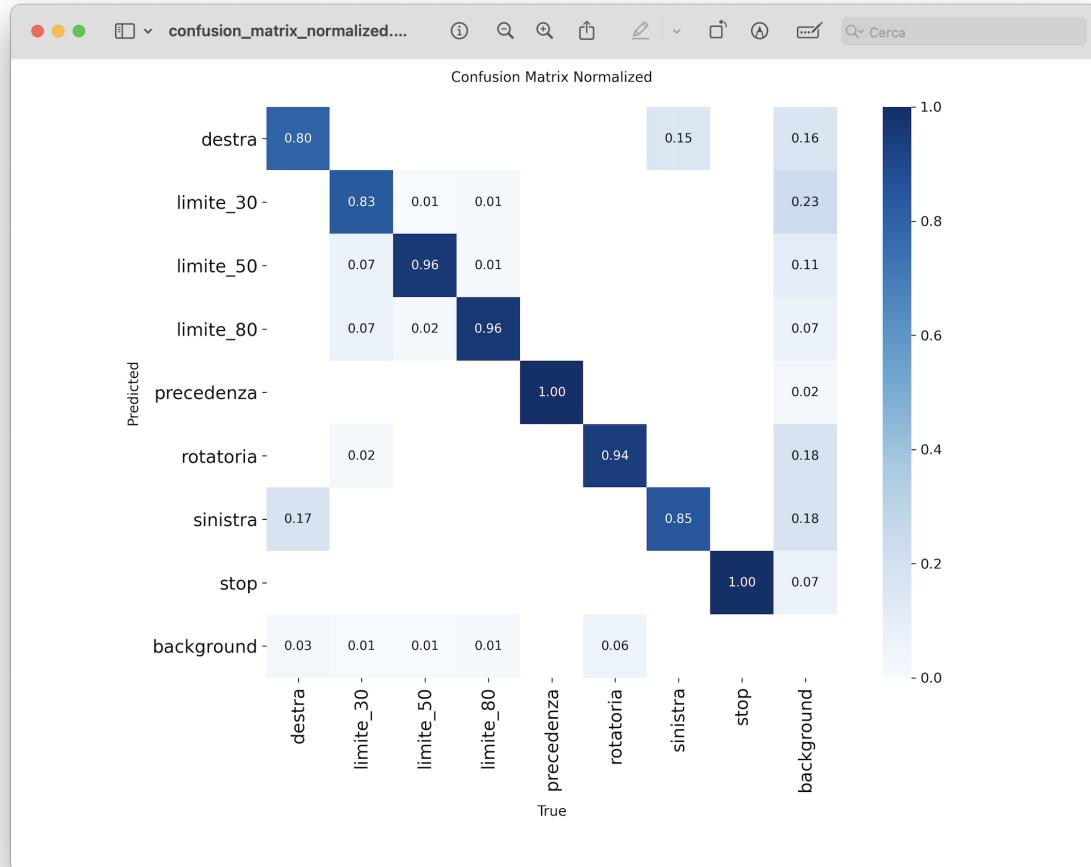
**Figura 5.8:** Distribuzione dei dati per classe

Dopo questa operazione di riequilibrio, il modello **YOLOv8n** è stato nuovamente addestrato con le stesse impostazioni iperparametriche della fase precedente. In questa fase l’obiettivo era osservare se il solo aumento dei dati delle classi minoritarie potesse produrre un miglioramento delle prestazioni, in particolare in termini di *recall* e *mAP* per le classi “Freccia Destra” e “Freccia Sinistra”.

## 5.2.1 Risultati Preliminari

### 5.2.1.1 Matrice di Confusione Normalizzata

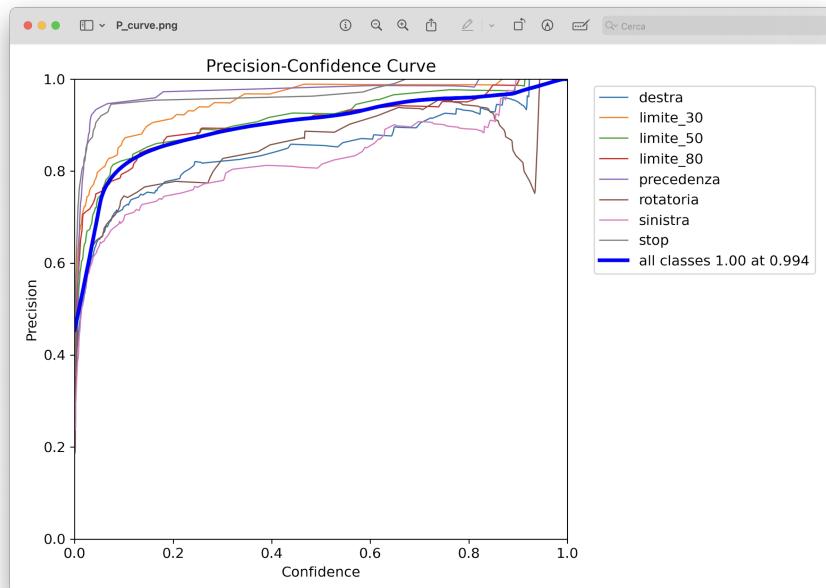
La matrice di confusione normalizzata mostra la frequenza con cui ciascuna classe viene correttamente o erroneamente classificata. In presenza di class imbalance, ci si attende una forte dominanza delle classi più rappresentate.



**Figura 5.9:** Confusion Matrix

### 5.2.1.2 Precision Curve

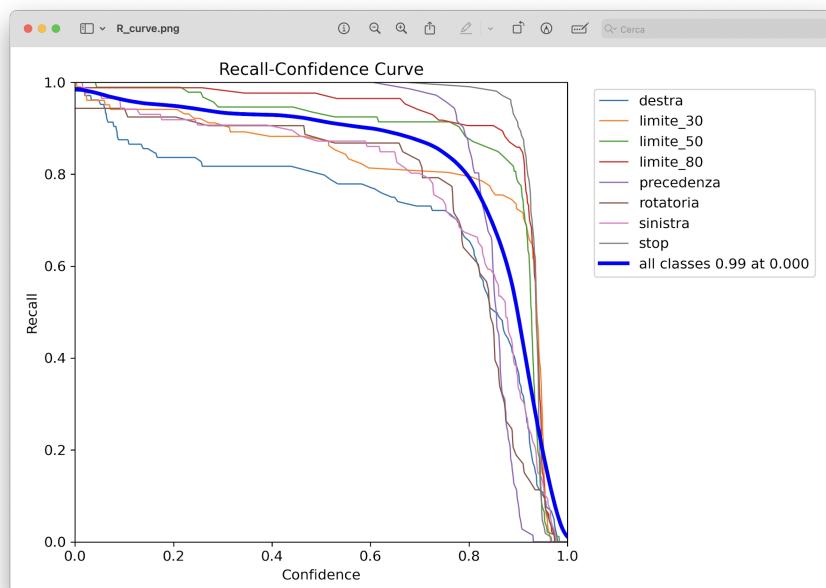
La curva di precisione mostra la capacità del modello di evitare falsi positivi su ciascuna classe, in funzione della soglia di confidenza.



**Figura 5.10:** Precision Curve

### 5.2.1.3 Recall Curve

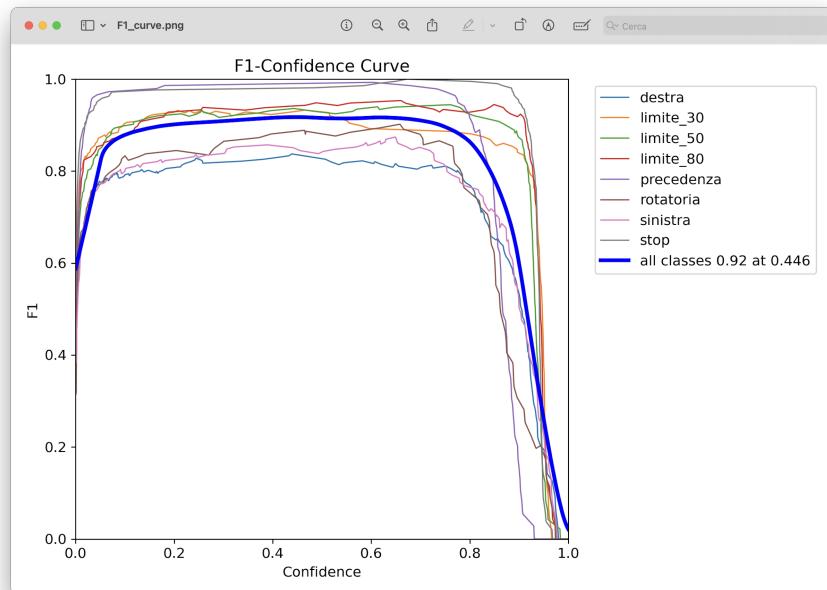
La curva di richiamo misura la capacità del modello di identificare correttamente tutti i veri positivi, indipendentemente dalla soglia.



**Figura 5.11:** Recall Curve

### 5.2.1.4 F1 Score Curve

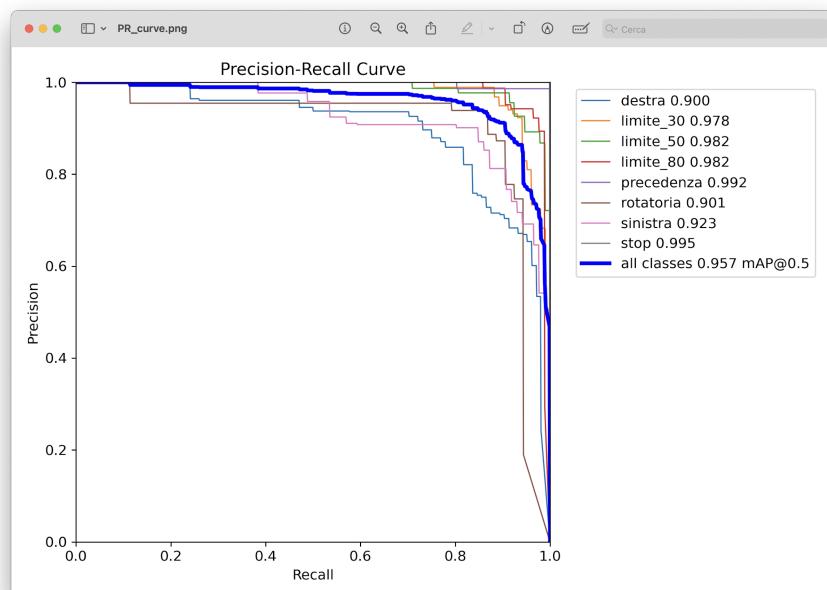
La curva dell'F1-score rappresenta l'armonica media tra precision e recall, ed è utile per valutare un compromesso tra le due metriche.



**Figura 5.12:** F1 Score Curve

### 5.2.1.5 Precision-Recall Curve

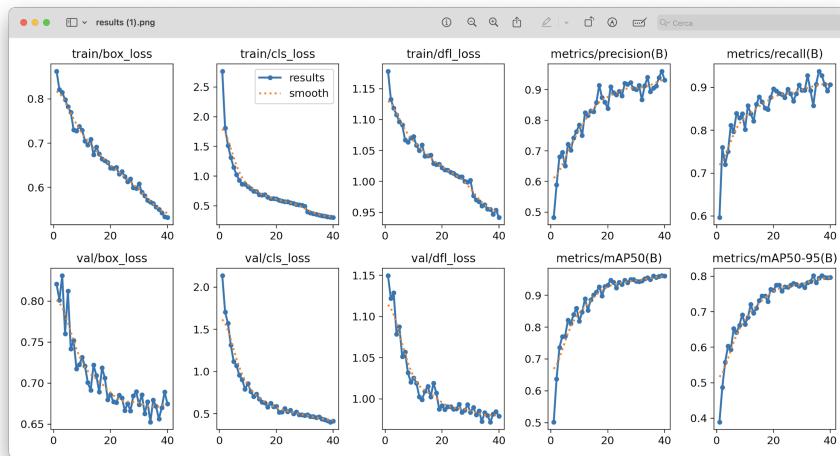
La curva precision-recall per ciascuna classe evidenzia il comportamento del classificatore in scenari sbilanciati, risultando spesso più informativa rispetto alla curva ROC.



**Figura 5.13:** PR Curve

### 5.2.1.6 Ulteriori Metriche

La figura seguente riassume l'andamento delle principali metriche durante l'addestramento e la validazione del modello. Sono inclusi i valori di `box_loss`, `cls_loss`, `dfl_loss` sia in fase di training che di validazione, oltre alle metriche di `precision`, `recall`, `mAP@0.5` e `mAP@0.5:0.95`.



**Figura 5.14:** Risultati

### 5.2.2 Discussione

A seguito dei miglioramenti ottenuti con il dataset riequilibrato nella Fase II, si è deciso di avviare una **Fase III** sperimentale finalizzata a ottimizzare il processo di addestramento del modello. In particolare, l'attenzione si è concentrata sull'introduzione della tecnica di **early stopping**, con l'obiettivo di evitare fenomeni di overfitting e rendere l'apprendimento più efficiente.

È stato quindi configurato un criterio di arresto anticipato, in modo che l'addestramento si interrompesse automaticamente se, per un certo numero di epoche consecutive, non si osservava alcun miglioramento della metrica mAP. In questo caso, la soglia di pazienza è stata fissata a 10 epoche.

Durante questa fase, sono state disattivate tutte le tecniche di data augmentation precedentemente applicate, in particolare la **riflessione orizzontale**, che poteva introdurre ambiguità tra le classi “**Freccia Destra**” e “**Freccia Sinistra**”. Tale trasformazione, infatti, rendeva le due classi visivamente intercambiabili, compromettendo l'apprendimento del modello. La rimozione di queste modifiche ha permesso di isolare l'effetto dell'ottimizzazione sul solo processo di training. I risultati ottenuti sono stati successivamente confrontati con quelli delle fasi precedenti per valutarne l'efficacia.

### 5.3 Fase III

Dopo aver migliorato la distribuzione del dataset tramite data augmentation nella Fase II, la Fase III è stata incentrata sull'ottimizzazione del processo di addestramento, con l'obiettivo di aumentare la stabilità del modello e ridurre il rischio di overfitting. A tal fine, è stata introdotta la tecnica di **early stopping**, che consente di interrompere automaticamente l'addestramento qualora non si osservino miglioramenti significativi in un intervallo definito di epoche.

In particolare, è stata impostata una soglia di **pazienza pari a 10 epoche**, ovvero il training si arresta se la metrica mAP non migliora per 10 epoche consecutive. Questo approccio si è rivelato utile per evitare l'addestramento eccessivo su un dataset già parzialmente ottimizzato.

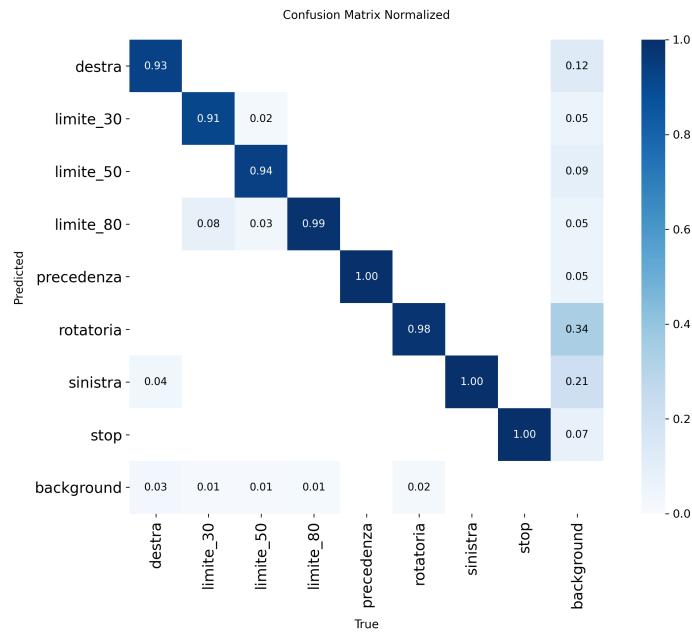
Un altro intervento rilevante ha riguardato la disattivazione di tutte le trasformazioni applicate in precedenza, con particolare attenzione alla **riflessione orizzontale (`fliplr`)**. Questa trasformazione, infatti, pur utile in scenari generici, si è rivelata controproducente nel nostro contesto: invertendo orizzontalmente un'immagine contenente una freccia, una “**Freccia Destra**” può facilmente essere scambiata per una “**Freccia Sinistra**”, generando ambiguità semantiche e compromettendo l'apprendimento del modello.

Sono state inoltre disattivate altre trasformazioni geometriche come rotazioni (`degrees`), traslazioni (`shear`) e variazioni prospettiche (`perspective`), per garantire che il modello si concentrasse unicamente sull'apprendimento dei pattern visivi reali, senza ulteriori distorsioni artificiali. Con queste modifiche, il modello è stato riaddestrato ed i risultati ottenuti sono i seguenti.

### 5.3.1 Risultati Preliminari

#### 5.3.1.1 Matrice di Confusione Normalizzata

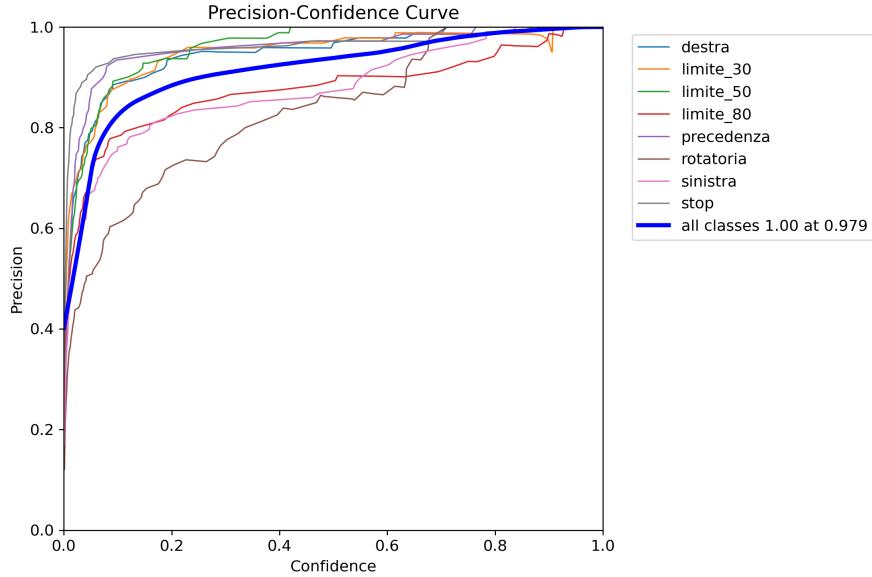
La matrice di confusione normalizzata mostra la frequenza con cui ciascuna classe viene correttamente o erroneamente classificata. In presenza di class imbalance, ci si attende una forte dominanza delle classi più rappresentate.



**Figura 5.15:** Confusion Matrix

### 5.3.1.2 Precision Curve

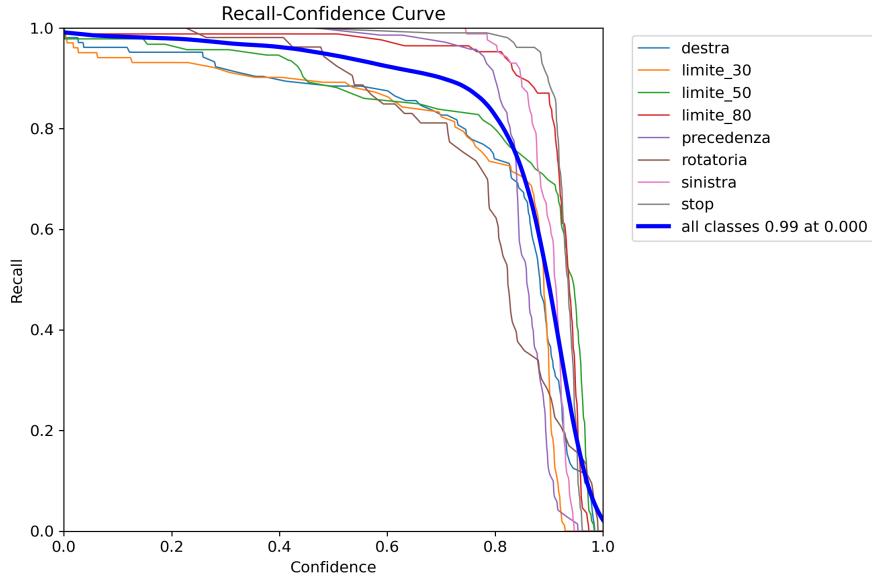
La curva di precisione mostra la capacità del modello di evitare falsi positivi su ciascuna classe, in funzione della soglia di confidenza.



**Figura 5.16:** Precision Curve

### 5.3.1.3 Recall Curve

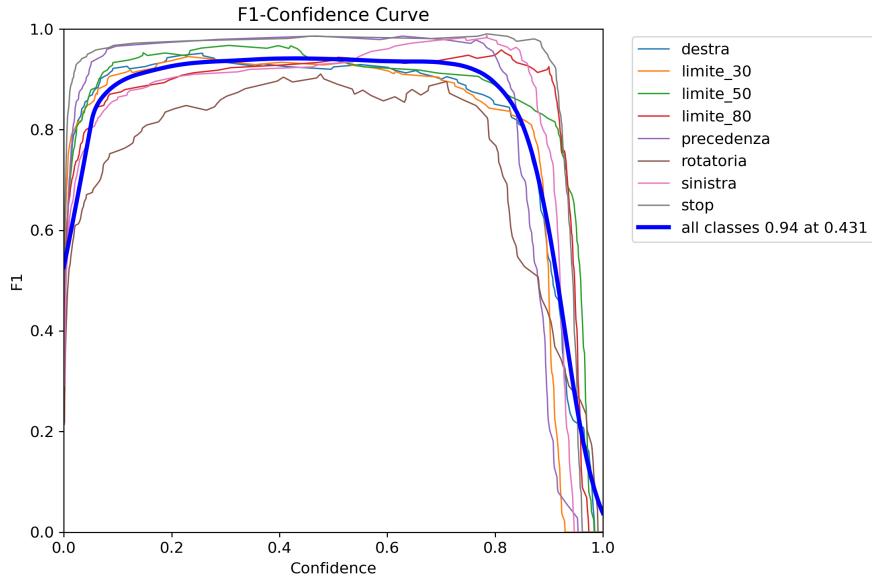
La curva di richiamo misura la capacità del modello di identificare correttamente tutti i veri positivi, indipendentemente dalla soglia.



**Figura 5.17:** Recall Curve

### 5.3.1.4 F1 Score Curve

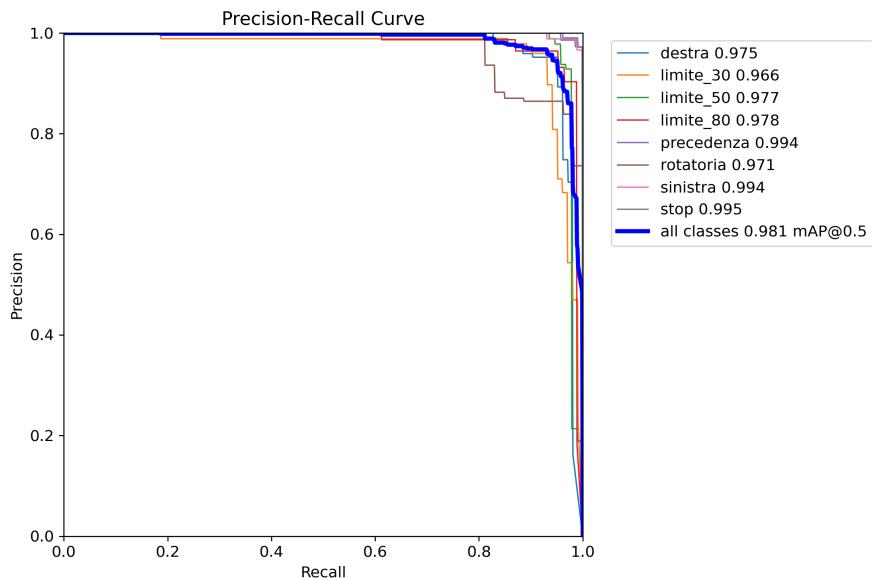
La curva dell'F1-score rappresenta l'armonica media tra precision e recall, ed è utile per valutare un compromesso tra le due metriche.



**Figura 5.18:** F1 Score Curve

### 5.3.1.5 Precision-Recall Curve

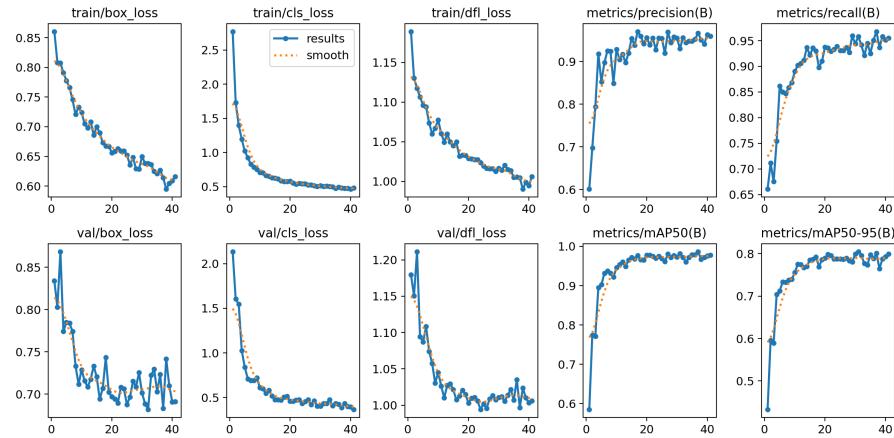
La curva precision-recall per ciascuna classe evidenzia il comportamento del classificatore in scenari sbilanciati, risultando spesso più informativa rispetto alla curva ROC.



**Figura 5.19:** PR Curve

### 5.3.1.6 Ulteriori Metriche

La figura seguente riassume l'andamento delle principali metriche durante l'addestramento e la validazione del modello. Sono inclusi i valori di `box_loss`, `cls_loss`, `dfl_loss` sia in fase di training che di validazione, oltre alle metriche di `precision`, `recall`, `mAP@0.5` e `mAP@0.5:0.95`.



**Figura 5.20:** Risultati

### 5.3.1.7 Discussione

Una volta apportate le modifiche descritte — in particolare l'introduzione dell'early stopping e la disattivazione delle trasformazioni che generavano ambiguità tra le classi “**Frecchia Destra**” e “**Frecchia Sinistra**” — il modello ha mostrato un netto miglioramento in termini di stabilità e accuratezza. Le metriche ottenute sono risultate decisamente più soddisfacenti rispetto alle fasi precedenti, con una migliore capacità di distinguere anche le classi precedentemente problematiche.

Alla luce di questi risultati, si è deciso di ampliare ulteriormente l'indagine empirica, confrontando le prestazioni ottenute con YOLOv8n con quelle di altre varianti della stessa famiglia di modelli. L'obiettivo è stato quello di valutare l'impatto della dimensione del modello sulla qualità del rilevamento, mantenendo costante il dataset e le impostazioni di training.

## 5.4 Fase IV: Confronto tra Varianti del Modello YOLO

Con l’obiettivo di valutare l’impatto della dimensione del modello sulle prestazioni del rilevatore, la Fase IV ha previsto un confronto diretto tra tre versioni della famiglia YOLO: **YOLOv8n**, **YOLOv8s** e **YOLOv11n**. Tutti i modelli sono stati addestrati sullo stesso dataset, già riequilibrato e privo di trasformazioni potenzialmente ambigue, utilizzando le stesse impostazioni iperparametriche adottate nella Fase III (incluso l’early stopping con pazienza pari a 10 epoches).

Le diverse versioni si distinguono principalmente per la loro complessità architetturale e il numero di parametri, influenzando direttamente il compromesso tra accuratezza e velocità di inferenza. La variante “n” (nano) è la più leggera e veloce, ma meno accurata, mentre la “s” (small) offre un buon bilanciamento. La variante “YOLOv11n”, infine, è una versione successiva e potenzialmente ottimizzata rispetto a YOLOv8n, mantenendo però una taglia simile.

I risultati delle tre configurazioni sono sintetizzati nella Tabella 5.1.

**Tabella 5.1:** Confronto delle prestazioni tra YOLOv8n, YOLOv8s e YOLOv11n

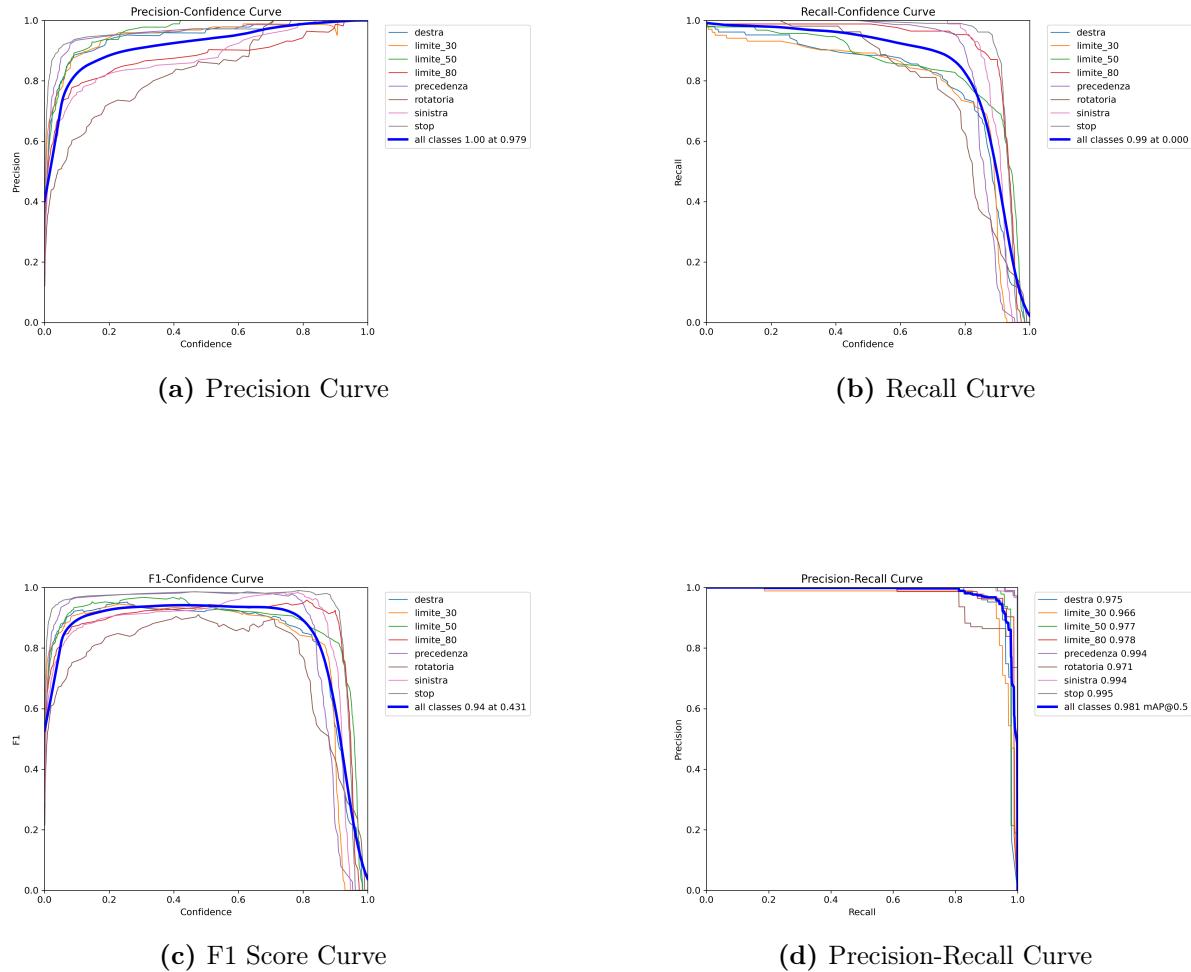
Modello	mAP@0.5	mAP@0.5:0.95	Recall	Precision
YOLOv8n	0.98644	0.80476	0.968	0.9706
YOLOv8s	0.98456	0.81039	0.95408	0.96963
YOLOv11n	0.98038	0.81223	0.94988	0.96941

Dall’analisi emerge che il modello **YOLOv8n** ottiene le migliori prestazioni in termini di *mAP@0.5* e *Recall*, rendendolo particolarmente adatto a scenari in cui è importante massimizzare la rilevazione dei target. Tuttavia, la versione **YOLOv11n** ottiene il punteggio più alto su *mAP@0.5:0.95*, indicativo di una maggiore precisione globale su diversi livelli di soglia IoU.

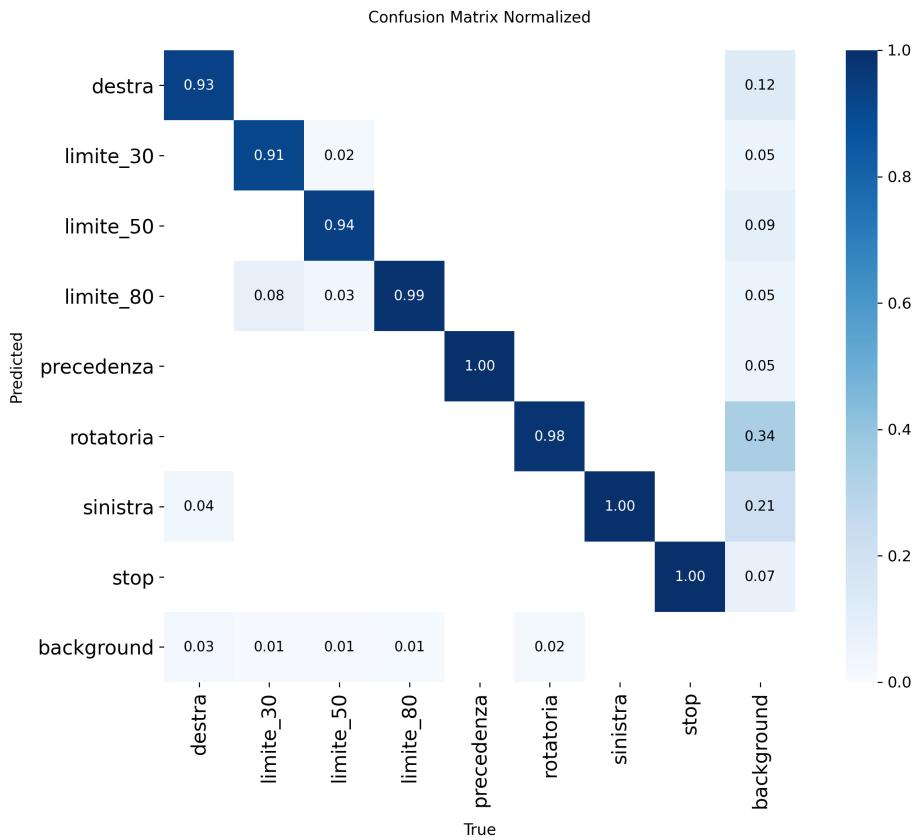
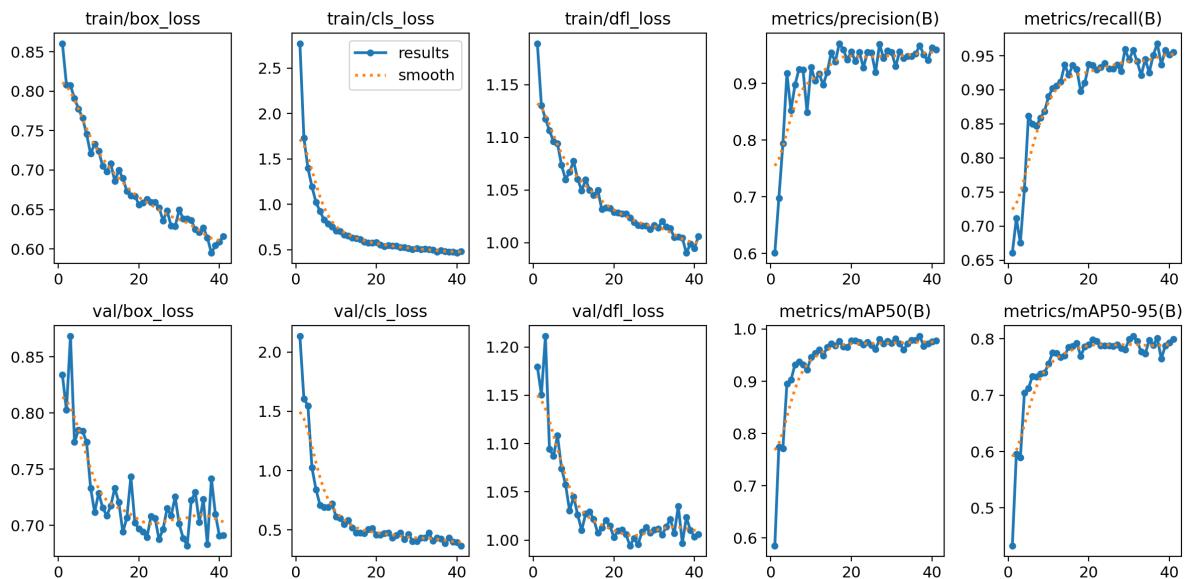
Il modello **YOLOv8s**, pur non eccellendo in nessuna singola metrica, mostra un bilanciamento complessivo molto solido, con valori elevati su tutte le metriche. Questo lo rende una scelta valida per applicazioni in cui è richiesta un’elevata affidabilità generale, a fronte di un maggiore costo computazionale.

In conclusione, la scelta del modello più adatto dipende fortemente dallo scenario applicativo: **YOLOv8n** è indicato per applicazioni real-time su dispositivi a risorse limitate, mentre **YOLOv11n** può offrire vantaggi nei contesti in cui è cruciale massimizzare la precisione media su varie soglie. **YOLOv8s** rappresenta un buon compromesso per contesti meno vincolati computazionalmente ma comunque sensibili alla qualità dei risultati.

### 5.4.1 Metriche di addestramento di YOLOv11



**Figura 5.21:** Metriche principali ottenute durante l’addestramento di YOLOv11.

**Figura 5.22:** Confusion Matrix per YOLOv11.**Figura 5.23:** Risultati finali di YOLOv11 (mAP, Precision, Recall, ecc.).

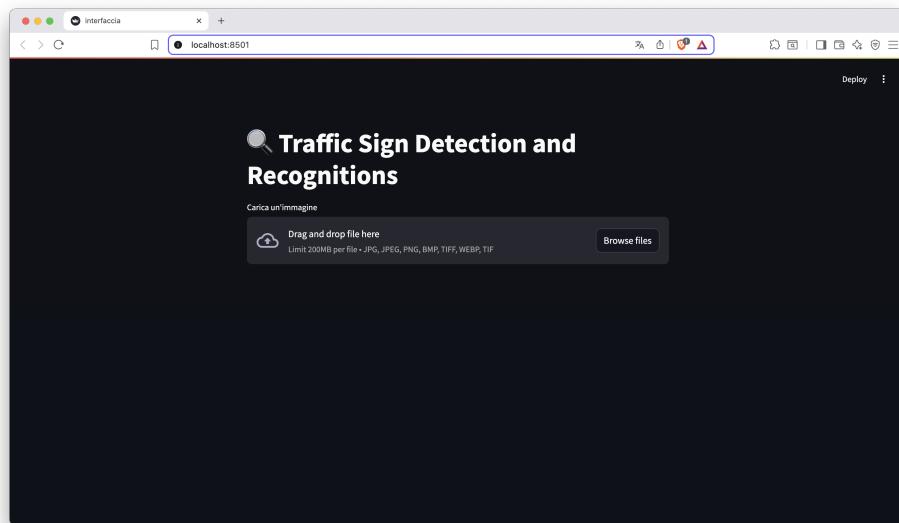
# Capitolo 6

## Demo

Questo capitolo descrive la fase dimostrativa del progetto, che consiste nello sviluppo di una web application interattiva progettata per mostrare le capacità del modello di *Object Detection* basato sull'architettura YOLO (You Only Look Once). La demo consente all'utente di caricare immagini e visualizzare in tempo reale i risultati della predizione eseguita dal modello.

### 6.1 Fase 1: Visualizzazione della Web Application

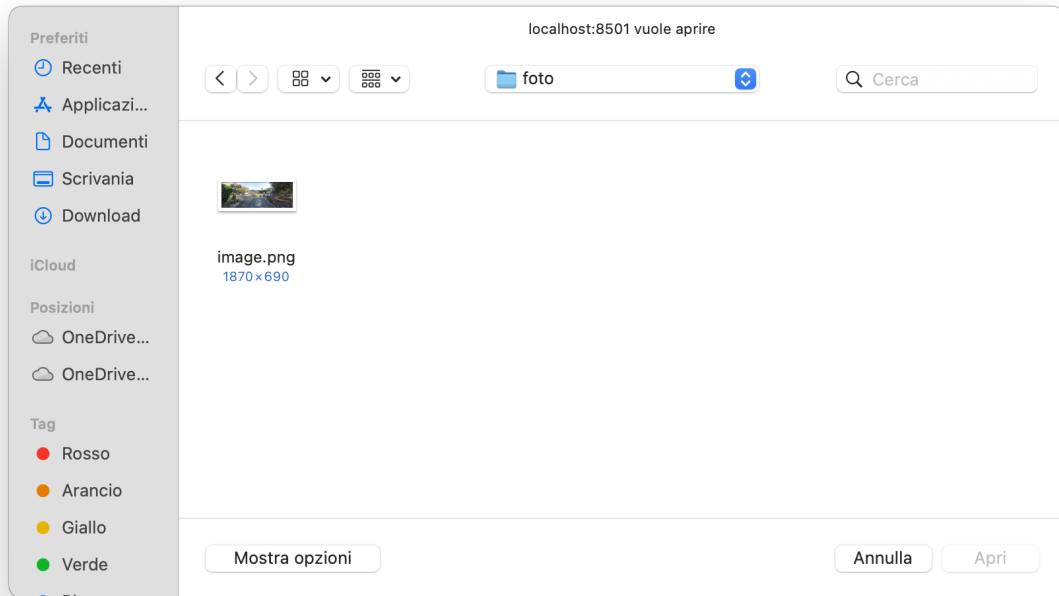
All'apertura della web application, sviluppata utilizzando il framework Streamlit, viene presentata un'interfaccia chiara e sintetica che introduce il contesto applicativo. L'interfaccia utente è progettata per massimizzare la semplicità d'uso, pur mantenendo un accesso diretto alle funzionalità principali. In questa fase iniziale, viene effettuato il caricamento in memoria del modello YOLO precedentemente addestrato.



**Figura 6.1:** Interfaccia iniziale della Web App

## 6.2 Fase 2: Caricamento dell'Immagine

L'utente ha la possibilità di caricare un'immagine locale tramite l'apposita funzione di upload. Il sistema supporta i principali formati grafici, come JPEG, PNG e TIFF. Una volta selezionato il file, l'immagine viene visualizzata a schermo, permettendo un'immediata conferma visiva. In parallelo, viene effettuata la conversione automatica dell'immagine nel formato RGB, assicurando la compatibilità con l'input richiesto dal modello di inferenza.

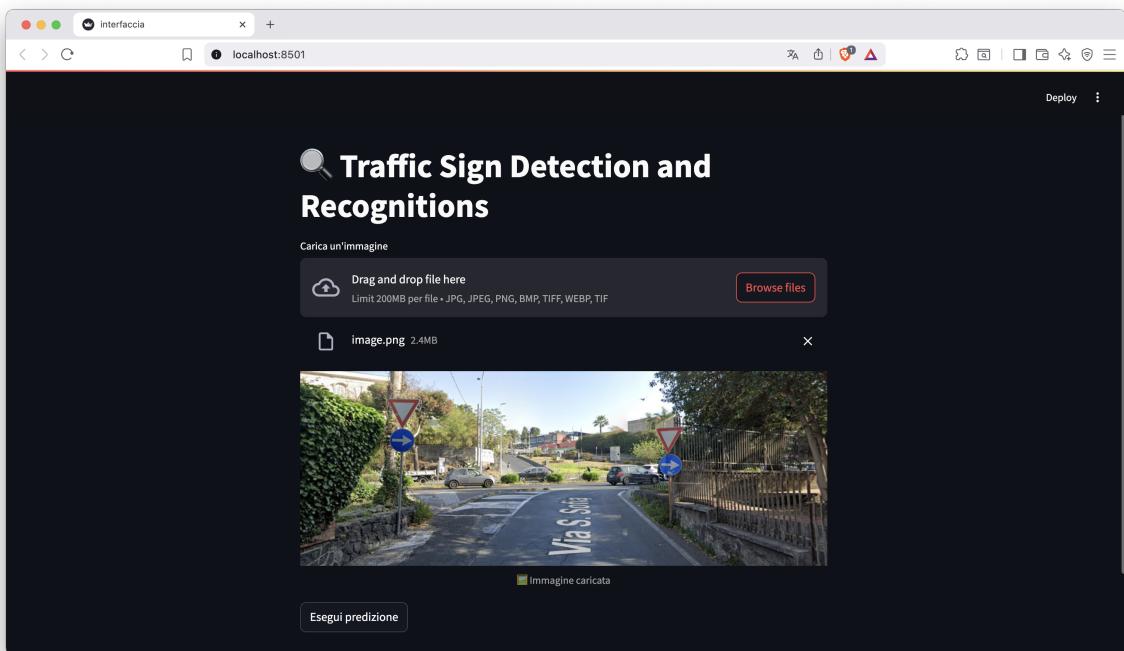


**Figura 6.2:** Caricamento dell'immagine da parte dell'utente

## 6.3 Fase 3: Esecuzione della Predizione

Una volta caricata l'immagine, l'utente può avviare manualmente il processo di predizione tramite un'interazione con l'interfaccia. Il modello YOLO, caricato in precedenza, esegue l'inferenza sull'immagine fornita, rilevando gli oggetti presenti e associando a ciascuno una classe predefinita. L'elaborazione avviene in tempo reale ed è accompagnata da un indicatore grafico che segnala lo stato del processo.

La pipeline di inferenza è progettata per mantenere un equilibrio ottimale tra velocità e precisione, sfruttando i punti di forza dell'architettura single-shot di YOLO, che consente di effettuare rilevamenti con un solo passaggio attraverso la rete neurale.

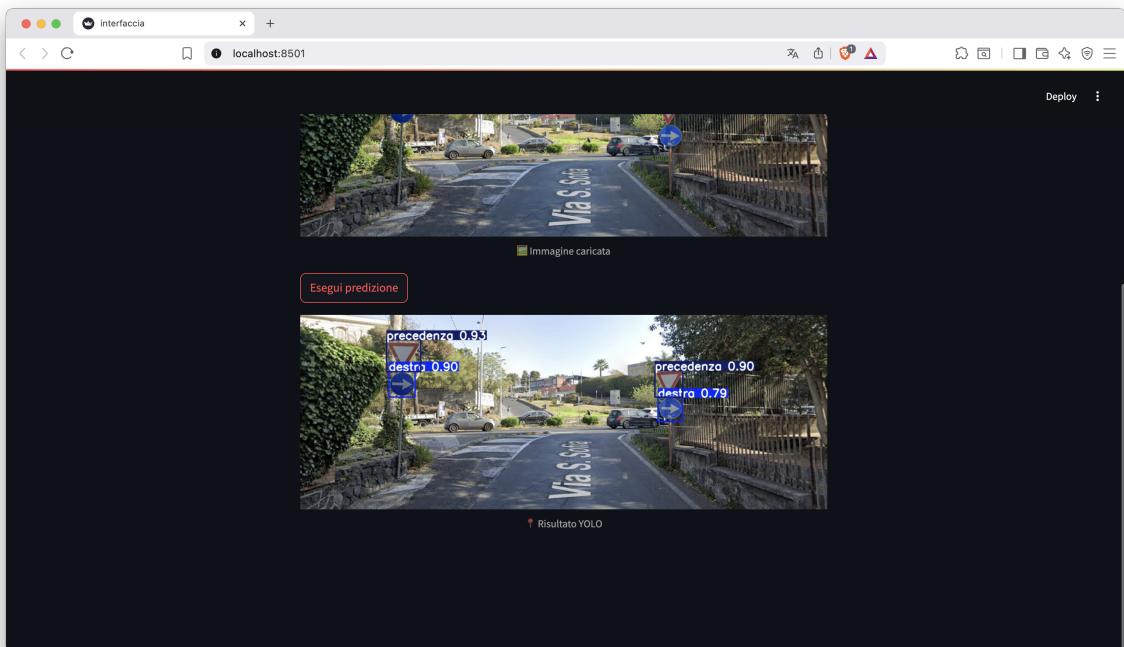


**Figura 6.3:** Esecuzione della predizione tramite il modello YOLO

## 6.4 Fase 4: Visualizzazione della Predizione

Al termine del processo di analisi, viene mostrata a video l'immagine originale arricchita con le annotazioni generate automaticamente dal modello. Gli oggetti rilevati sono evidenziati tramite *bounding boxes* colorate, accompagnate dalla rispettiva etichetta di classe e dal livello di confidenza stimato. L'immagine finale è convertita nello spazio colore RGB per garantire una corretta resa cromatica nella visualizzazione web.

Questa fase rappresenta il momento culminante della demo, in cui l'utente può osservare visivamente il risultato del processo di *Object Detection* applicato a un'immagine reale. L'intero flusso si conclude con una rappresentazione chiara, interpretabile e conforme ai requisiti funzionali definiti in fase progettuale.



**Figura 6.4:** Risultato finale della predizione

# Capitolo 7

## Codice

In questo capitolo viene illustrato e spiegato il codice Python utilizzato per addestrare un modello YOLO (You Only Look Once) utilizzando la libreria `ultralytics`. Questo framework semplifica notevolmente l'intero processo di training e inferenza, offrendo un'interfaccia ad alto livello.

### 7.1 Installazione della Libreria

**Listing 7.1:** Installazione della libreria Ultralytics

```
1 !pip install ultralytics
```

Il comando `!pip install ultralytics` consente di installare la libreria `ultralytics`, che contiene le implementazioni ufficiali dei modelli YOLO (in particolare le versioni YOLOv8 e YOLOv11) con un'interfaccia semplice e modulare per l'addestramento, la validazione e la predizione.

### 7.2 Importazione del Modulo YOLO

**Listing 7.2:** Importazione del modulo YOLO

```
1 from ultralytics import YOLO
```

Questa istruzione importa la classe YOLO dalla libreria `ultralytics`, necessaria per caricare, addestrare e utilizzare i modelli predefiniti.

### 7.3 Caricamento del Modello

**Listing 7.3:** Caricamento del modello YOLOv11n pre-addestrato

```
1 model = YOLO("yolov11n.pt")
```

Viene caricato il modello YOLOv11n, una versione "nano" del modello YOLOv11, che è più leggera e ottimizzata per ambienti con risorse limitate. Il file .pt rappresenta un checkpoint pre-addestrato.

## 7.4 Addestramento del Modello

**Listing 7.4:** Avvio dell'addestramento del modello

```

1 model.train(
2     data="Progetto-ML-7/data.yaml",
3     epochs=200,
4     imgsz=640,
5     patience=10,
6     fliplr=0.0,
7     flipud=0.0,
8     degrees=0.0,
9     shear=0.0,
10    perspective=0.0
11 )

```

Viene avviato il processo di addestramento del modello tramite il metodo `train`, con i seguenti parametri:

- **data**: percorso al file `data.yaml`, che definisce la struttura del dataset (classi, path delle immagini di training/valutazione, ecc.).
- **epochs**: numero massimo di epoche (200), ovvero cicli completi attraverso il dataset.
- **imgsz**: dimensione delle immagini di input, in questo caso 640x640 pixel.
- **patience**: meccanismo di *early stopping*; il training si interrompe se non si osservano miglioramenti per 10 epoche consecutive.
- **fliplr, flipud**: disabilitazione delle trasformazioni di flip orizzontale e verticale (valori impostati a 0.0).
- **degrees, shear, perspective**: tutti impostati a zero per escludere trasformazioni geometriche dei dati durante il training. Questo consente di avere un addestramento "pulito", senza data augmentation.

## 7.5 Web Application per la Predizione

Una volta addestrato il modello, è possibile metterlo a disposizione degli utenti tramite una semplice interfaccia web, realizzata con il framework Streamlit. Di seguito viene presentato il codice completo dell'applicazione.

**Listing 7.5:** Web App per il rilevamento dei segnali stradali

```

1 import streamlit as st
2 from PIL import Image
3 import tempfile
4 import cv2
5 from ultralytics import YOLO
6 import os
7 from dotenv import load_dotenv
8
9 load_dotenv()
10 YOLO_MODEL_PATH = os.getenv("YOLO_MODEL_PATH")
11
12 # Caricamento modello
13 model = YOLO(YOLO_MODEL_PATH)
14
15 st.title("Traffic Sign Detection and Recognition")
16
17 supported_types = ["jpg", "jpeg", "png", "bmp", "tiff", "webp"]
18 uploaded_file = st.file_uploader("Carica un'immagine", type=
19     supported_types)
20
21 if uploaded_file is not None:
22     try:
23         image = Image.open(uploaded_file)
24         st.image(image, caption="Immagine caricata",
25             use_container_width=True)
26         with tempfile.NamedTemporaryFile(suffix=".jpg", delete=
27             False) as temp:
28             image.convert("RGB").save(temp.name)
29             temp_path = temp.name
30             if st.button("Esegui predizione"):
31                 with st.spinner("In esecuzione YOLO..."):
32                     results = model.predict(
33                         source=temp_path, save=False, save_txt=False
34                     )
35                     annotated_bgr = results[0].plot()
36                     annotated_rgb = cv2.cvtColor(annotated_bgr, cv2.
37                         COLOR_BGR2RGB)
38                     st.image(annotated_rgb, caption="Risultato YOLO",
39                         use_container_width=True,)
40             if os.path.exists(temp_path):
41                 os.remove(temp_path)
42     except Exception as e:
43         st.error(f"Errore nell'apertura dell'immagine: {e}")

```

## 7.6 Funzionalità dell'Applicazione

- **Caricamento del modello:** il file `best.pt` è il modello addestrato precedentemente.
- **Upload dell'immagine:** l'utente può caricare un file immagine nei principali formati (JPEG, PNG, BMP, ecc.).
- **Visualizzazione dell'immagine:** dopo il caricamento, l'immagine viene mostrata per conferma visiva.
- **Predizione:** cliccando su "*Esegui predizione*", il modello YOLO esegue l'inferenza sull'immagine caricata.
- **Annotazione:** l'immagine risultante viene visualizzata con *bounding boxes* e etichette generate dal modello.

## 7.7 Installazione e Configurazione del Sistema

La presente sezione illustra nel dettaglio la procedura di installazione e configurazione del sistema di riconoscimento dei segnali stradali sviluppato. Il framework implementato si basa su una architettura modulare che richiede specifiche dipendenze software e una configurazione accurata per garantire il corretto funzionamento del modello YOLO per la object detection.

### 7.7.1 Requisiti di Sistema

L'ambiente di sviluppo richiede una configurazione software specifica per supportare le operazioni di machine learning e computer vision. Il sistema è stato progettato per funzionare su Python versione 3.8 o superiore, garantendo così la compatibilità con le librerie di deep learning più recenti. La gestione delle dipendenze viene effettuata tramite il package manager pip, che consente l'installazione automatica di tutte le librerie necessarie. È inoltre indispensabile disporre di una connessione internet stabile durante la fase di setup iniziale per il download dei modelli pre-addestrati e delle dipendenze esterne.

### 7.7.2 Procedura di Installazione

Il processo di installazione segue una metodologia standardizzata che garantisce la riproducibilità dell'ambiente di sviluppo. Inizialmente, è necessario clonare il repository del progetto dal sistema di version control Git utilizzando il comando `git clone`. Successivamente, è fondamentale posizionarsi nella directory principale del progetto attraverso il comando `cd TrafficSignDetectionAndRecognition` per eseguire correttamente i passaggi successivi.

La creazione di un ambiente virtuale rappresenta una best practice essenziale per isolare le dipendenze del progetto dal sistema operativo host. L'ambiente virtuale viene generato mediante il comando `python -m venv .venv`, creando una directory dedicata che conterrà tutte le librerie specifiche del progetto. L'attivazione dell'ambiente virtuale varia in base al sistema operativo: sui sistemi Unix-like (Linux e macOS) si utilizza il

comando `source .venv/bin/activate`, mentre su Windows è necessario eseguire `.venv\Scripts\activate`.

Una volta attivato l'ambiente virtuale, l'installazione delle dipendenze viene completa attraverso il comando `pip install -r requirements.txt`, che procede automaticamente al download e all'installazione di tutte le librerie specificate nel file di configurazione.

### 7.7.3 Configurazione del Sistema

La configurazione del sistema richiede la definizione di parametri specifici attraverso la creazione di un file di environment variables denominato `.env`. Questo file, posizionato nella directory root del progetto, contiene le variabili di configurazione essenziali per il funzionamento del sistema.

La variabile `YOLO_MODEL_PATH` specifica il percorso del modello YOLO addestrato, tipicamente denominato `best.pt`, che rappresenta il checkpoint del modello con le migliori performance ottenute durante la fase di training. La variabile `ROBOFLOW_API_KEY` contiene la chiave di autenticazione per l'accesso ai servizi Roboflow, piattaforma utilizzata per la gestione avanzata dei dataset e per alcune funzionalità di preprocessing delle immagini.

**È importante sottolineare che la chiave API Roboflow deve essere recuperata all'interno della mail inviata ai docenti.**

### 7.7.4 Avvio dell'Applicazione Web

Il sistema è progettato per essere utilizzato attraverso una interfaccia web sviluppata con il framework Streamlit, che fornisce un'interfaccia utente intuitiva per l'interazione con il modello di object detection. L'avvio dell'applicazione web viene effettuato attraverso il comando `streamlit run main.py`, che inizializza il server di sviluppo e rende disponibile l'interfaccia all'indirizzo `http://localhost:8501`.

Questa architettura web-based consente un accesso semplificato alle funzionalità del sistema, permettendo agli utenti di caricare immagini, visualizzare i risultati della detection e analizzare le performance del modello attraverso un'interfaccia grafica moderna e responsive. La scelta di Streamlit come framework di sviluppo garantisce una rapida prototipazione e un deployment efficiente dell'applicazione, mantenendo al contempo la flessibilità necessaria per future estensioni e miglioramenti del sistema.

# Conclusioni

Nel contesto del presente progetto di *Object Detection* è stato costruito un dataset proprietario costituito da un totale di 3049 immagini. Le immagini sono state acquisite mediante screenshot provenienti da Google Maps, nonché fotografie scattate utilizzando tre diversi dispositivi: un Samsung Galaxy S25 e due iPhone 13. I file acquisiti sono stati salvati nei formati .png, .jpeg e .jpg, al fine di garantire una certa varietà in termini di qualità, compressione e condizioni di acquisizione, simulando così scenari realistici.

Le immagini presentano una risoluzione variabile compresa tra 720p e 4K, con una dimensione media per file compresa tra 250 KB e 2.5 MB, in funzione del dispositivo e del formato utilizzato. Tale diversificazione ha consentito di creare un dataset che, pur limitato nelle dimensioni complessive, si è rivelato adeguato per validare il workflow tecnico e sperimentale del progetto.

Il sistema di rilevamento oggetti è stato sviluppato adottando l'architettura YOLO (*You Only Look Once*), una delle soluzioni più consolidate e performanti nell'ambito della *Real-Time Object Detection*. Durante le fasi di *Training* e *Validation* del modello, sono state monitorate le principali metriche di apprendimento e prestazione.

L'analisi dei risultati evidenzia un progressivo miglioramento delle metriche. In particolare:

Il *box loss* si è ridotto da un valore iniziale di circa 0.85 fino a stabilizzarsi intorno a 0.60 al termine delle epoche di training, mentre in validazione è stato osservato un valore medio compreso tra 0.70 e 0.75, segno di una buona capacità di regressione dei bounding box da parte del modello.

Il *classification loss* è diminuito sensibilmente, passando da circa 2.5 a valori prossimi a 0.5 nel training e attestandosi su valori simili nel validation set, evidenziando un miglioramento nella capacità di distinguere correttamente le classi target.

La metrica *precision* ha raggiunto valori superiori al 90% già dopo le prime 15 epoche, mentre il *recall* si è stabilizzato intorno al 95%, confermando un'elevata capacità del modello di rilevare oggetti presenti nelle immagini senza un'eccessiva incidenza di falsi negativi.

In termini di accuratezza globale, l'indice  $mAP@0.5$  ha raggiunto valori prossimi al 97%, mentre il più rigoroso  $mAP@0.5:0.95$  ha superato stabilmente il 79%, dimostrando un buon livello di precisione anche su soglie di Intersection over Union più restrittive.

Nonostante questi risultati incoraggianti, va sottolineato che la principale limitazione del progetto è stata la dimensione relativamente ridotta del dataset. Come confermato dalla letteratura, un numero maggiore di immagini e una maggiore diversità dei dati di addestramento avrebbero presumibilmente consentito al modello di ottenere prestazioni ancora superiori, migliorando la sua capacità di generalizzazione su scenari complessi e non controllati.

Il progetto ha comunque dimostrato la validità tecnica e metodologica dell'approccio, fornendo una base solida su cui poter proseguire con ulteriori sviluppi futuri, quali l'espansione del dataset, il *fine-tuning* su nuove classi o l'integrazione in sistemi real-time.

Tutto il codice sorgente, i modelli addestrati e la documentazione tecnica del progetto sono disponibili pubblicamente su GitHub ai seguenti indirizzi:

- [Repository Github di Giulio Pedicone](#)
- [Repository Github di Vincenzo Villanova](#)
- [Repository Github di Francesco Prospero Antonio Virzì](#)