Dipartimento di Matematica e Informatica
Corso di Laurea Triennale in Informatica

Machine Learning Project

Street Sign Sense

A Real-Time Detection System based on YOLO12

Autore
Alessandro Ferrante

Docenti
Prof. Giovanni Farinella
Prof. Rosario Leonardi

Anno Accademico 2024-2025
Settembre 2025

Sommario

Il progetto Street Sign Sense ha come obiettivo lo sviluppo di modelli YOLO12 per il rilevamento automatico dei segnali stradali. L'iniziativa combina applicazioni pratiche in sistemi avanzati di assistenza alla guida e visione artificiale urbana con finalità accademiche e di ricerca. Il modello è stato addestrato sul dataset personalizzato Street Sign Set, composto da circa 7000 immagini distribuite su 63 classi, con annotazioni in formato YOLO. Sono stati testati e confrontati tre modelli della famiglia YOLO12, nello specifico le varianti nano, small e medium, valutandoli attraverso metriche consolidate quali Precision, Recall, mAP@50, mAP@50-95, F1-Score e tempo medio di inferenza. Il training è stato condotto su due GPU T4 in ambiente cloud Kaggle Notebook, estendendo il processo a 600 epoche con meccanismo di early stopping. I risultati sperimentali mostrano prestazioni superiori del modello medium, che raggiunge una mAP@50 di 0.916 e una mAP@50-95 di 0.811, offrendo un'ottima accuratezza.

Indice

Capitolo 1

Introduzione

1.1 Contesto

Rilevamento automatico dei segnali stradali

Il rilevamento automatico dei segnali stradali è un'area di crescente rilevanza tra la visione artificiale e il progresso dei sistemi di trasporto, in quanto permette di analizzare e interpretare informazioni visive provenienti dall'ambiente circostante. La capacità di un sistema di analizzare e interpretare il flusso video in ingresso è il fondamento su cui si costruiscono i sistemi avanzati di assistenza alla guida (ADAS) e le funzionalità dei veicoli a guida autonoma.

La sfida non si limita alla semplice identificazione (classificazione), ma richiede la localizzazione geometrica precisa del segnale nel piano immagine, un processo noto come *object detection*. Tali applicazioni sono intrinsecamente *safety-critical*, l'identificazione tempestiva e affidabile di un segnale di Stop o di un limite di velocità è un requisito imprescindibile per la sicurezza e la corretta esecuzione delle manovre.

Per questi scenari operativi, i modelli devono risolvere un duplice problema con estrema efficienza: devono essere stabili in presenza di grande variabilità visiva (cambiamenti di illuminazione, occlusioni e angolazioni) e devono garantire inferenza in tempo reale (tipicamente a frame rate superiori ai 30 FPS). Il progetto si inserisce in questo contesto, cercando di ottimizzare l'equilibrio tra accuratezza e velocità di esecuzione.

1.2 Obiettivo Tecnico

Finalità tecnica

L'obiettivo tecnico principale consiste nella realizzazione di un processo completo di addestramento, tuning e valutazione di modelli della famiglia YOLO12 per il rilevamento dei segnali stradali. L'obiettivo è individuare la configurazione più bilanciata in termini di accuratezza e velocità di inferenza, al fine di ottimizzare le prestazioni del modello in contesti che richiedono elaborazione in tempo reale.

Architetture analizzate e dataset

Il lavoro prevede l'addestramento e l'analisi comparativa di tre versioni del modello appartenenti alla famiglia YOLO12: le varianti YOLO12n (nano), YOLO12s (small) e YOLO12m (medium). L'intero processo si basa su un dataset personalizzato,

realizzato appositamente per questo progetto e costruito in modo da garantire una distribuzione rappresentativa delle diverse classi di segnali.

Per confrontare le prestazioni dei modelli sono state utilizzate un insieme di metriche ampiamente adottate nel campo dell'object detection, quali la Precision, il Recall, la mean Average Precision calcolata a differenti soglie di Intersection over Union (IoU), l'F1-Score e il tempo medio di inferenza per frame, parametri fondamentali per stimare l'equilibrio tra accuratezza e velocità di elaborazione.

Strategia di training

Il training viene effettuato attraverso tecniche di transfer learning, partendo da pesi pre-addestrati su dataset di larga scala e adattandoli al dominio specifico dei segnali stradali attraverso fine-tuning esteso. Sono state usate inoltre tecniche di data augmentation per migliorare la capacità di generalizzazione dei modelli.

Analisi qualitativa delle predizioni

Oltre alle metriche quantitative, è stata condotta un'analisi qualitativa dei risultati ottenuti dai modelli YOLO, con particolare attenzione alle differenze tra le varianti utilizzate. Questa fase ha previsto l'osservazione diretta delle predizioni generate su campioni rappresentativi del dataset, al fine di individuare comportamenti ricorrenti, errori tipici e punti di forza di ciascun modello. Attraverso il confronto visivo delle bounding box, dei falsi positivi e delle mancate rilevazioni, è stato possibile comprendere meglio le metriche numeriche e valutare l'effettiva capacità dei modelli di generalizzare in condizioni reali. L'obiettivo di questa analisi è stato quindi quello di integrare i risultati quantitativi con un riscontro qualitativo, fornendo una visione più completa delle prestazioni e del comportamento pratico delle diverse architetture di YOLO.

Indicazioni per scenari applicativi

I risultati ottenuti consentono di trarre alcune indicazioni utili per l'applicazione dei modelli in diversi contesti operativi. Nei sistemi embedded o real-time, in cui la rapidità di elaborazione rappresenta un requisito prioritario, le varianti più leggere come YOLO12n e YOLO12s costituiscono la soluzione più adeguata, offrendo un buon compromesso tra accuratezza e velocità anche su dispositivi con risorse limitate. In contesti con maggiore potenza di calcolo, modelli come YOLO12m risultano invece più indicati, grazie a una precisione di rilevamento superiore, vantaggiosa in applicazioni dove è richiesta una predizione più accurata.

Capitolo 2

Stato dell'Arte

2.1 Panoramica sull'Object Detection

L'*object detection* è una tecnica della computer vision che impiega le *convolutional neural networks* (CNN) per individuare e classificare oggetti presenti in immagini o sequenze video. In quanto tale mira a riprodurre le capacità umane di riconoscere e distinguere gli elementi visivi, assegnandoli a specifiche categorie semantiche. La localizzazione consente di determinare con precisione la posizione di ciascun oggetto, generalmente rappresentata tramite un bounding box rettangolare definito da coordinate nell'immagine, mentre la classificazione stabilisce la categoria di appartenenza dell'oggetto rilevato. L'object detection unisce dunque entrambi gli aspetti, consentendo di stimare simultaneamente la posizione e la tipologia delle istanze di oggetti presenti nella scena.

Categorie di approcci

L'evoluzione delle tecniche di object detection nel contesto del deep learning può essere suddivisa in due macro-categorie caratterizzate da approcci architetturali fondamentalmente diversi. I metodi two-stage operano attraverso un processo sequenziale in cui viene prima generato un insieme di regioni candidate e successivamente ciascuna regione viene classificata e raffinata. I metodi one-stage effettuano detection e classificazione in un'unica passata attraverso la rete neurale, eliminando la fase esplicita di generazione delle proposte.

Metodi two-stage

I metodi two-stage hanno rappresentato per diversi anni lo standard di riferimento nell'object detection, grazie all'elevata accuratezza ottenuta. La famiglia R-CNN, introdotta da Girshick nel 2014, ha segnato un punto di svolta nell'impiego delle reti neurali convoluzionali profonde per questo compito. Nei modelli two-stage, l'intero processo di rilevamento avviene in due passaggi principali: una prima rete identifica regioni candidate che potrebbero contenere oggetti, mentre una seconda fase si occupa della classificazione e del raffinamento dei bounding box. Nonostante l'elevata precisione, la complessità e i tempi di elaborazione elevati li rendono poco adatti a contesti real-time, favorendo la diffusione dei metodi one-stage.

Metodi one-stage

Gli approcci one-stage sono stati concepiti per ridurre la complessità dei modelli two-stage e migliorare la velocità di inferenza. Invece di generare e analizzare esplicitamente regioni candidate, questi metodi suddividono l'immagine in una griglia e predicono simultaneamente le coordinate dei bounding box e le relative probabilità di classe. Soluzioni come SSD e YOLO hanno dimostrato che è possibile mantenere buoni livelli di accuratezza riducendo drasticamente i tempi di elaborazione, rendendo tali architetture ideali per applicazioni in tempo reale.

Miglioramenti nei metodi one-stage

Numerosi contributi hanno progressivamente ridotto il divario prestazionale tra i due approcci. Architetture come RetinaNet hanno introdotto funzioni di perdita avanzate, come la focal loss, per gestire lo sbilanciamento tra esempi positivi e negativi, migliorando la stabilità e la precisione del training. Le evoluzioni successive della famiglia YOLO hanno a loro volta integrato varianti di queste strategie, adottando funzioni di perdita focalizzate o adattive per affinare la qualità delle predizioni. Questi progressi hanno consolidato la rilevanza degli approcci one-stage, che oggi costituiscono la base dei modelli moderni ad alte prestazioni.

Applicazioni in contesti real-time

Nel contesto dei sistemi avanzati di assistenza alla guida (ADAS) e dei veicoli autonomi, i metodi one-stage rappresentano la soluzione più adatta grazie alla loro elevata efficienza computazionale. Queste applicazioni richiedono l'elaborazione di flussi video in tempo reale, con frame rate tipicamente pari o superiori a trenta fotogrammi al secondo, e latenze di inferenza nell'ordine di poche decine di millisecondi per frame. Solo approcci altamente ottimizzati, come la famiglia YOLO, riescono a garantire tali prestazioni mantenendo livelli di accuratezza sufficientemente elevati per contesti safety-critical, dove la tempestività e l'affidabilità delle decisioni sono fondamentali.

Figura 2.1: Esempio di rilevamento con YOLO12

2.2 La famiglia YOLO

YOLO è una famiglia di modelli di computer vision in real time, che utilizzano l'algoritmo di rilevamento degli oggetti "You Only Look Once" sviluppato da Ultralytics, questa famiglia di modelli ha rappresentato una svolta nel campo dell'object detection, introducendo un'architettura one-stage che ha ridefinito gli standard di efficienza per le applicazioni in tempo reale. Questo paradigma ha permesso di raggiungere un equilibrio ottimale tra accuratezza e velocità di elaborazione, rendendo YOLO una delle architetture più utilizzate sia in ambito accademico che industriale.

Architettura One-Stage

Dal momento che YOLO utilizza l'approccio one-stage, unifica l'intero processo di generazione delle regioni, classificazione e raffinamento in un'unica rete end-to-end. Questa filosofia architetturale consente di ridurre drasticamente la complessità computazionale, ottenendo tempi di inferenza nell'ordine di pochi millisecondi e rendendo l'object detection applicabile in scenari real-time.

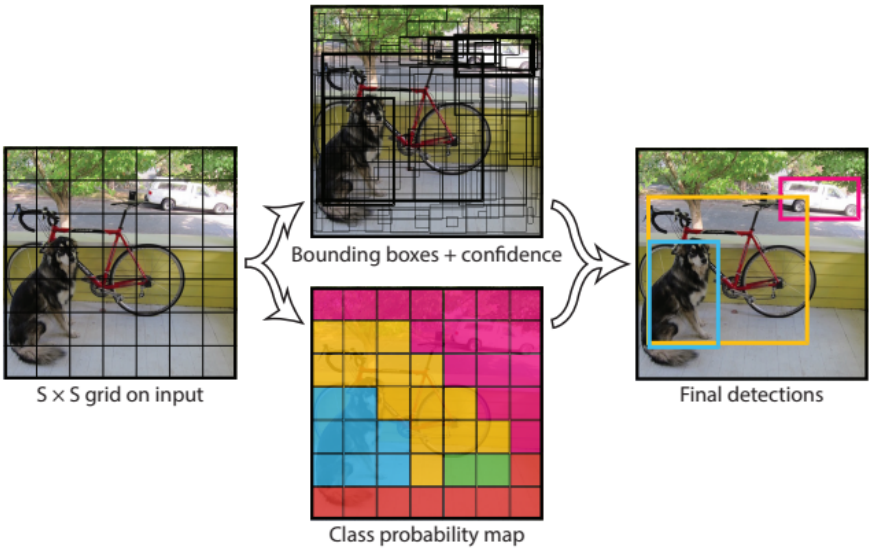


Figura 2.2: Rappresentazione concettuale del paradigma You Only Look Once (YOLO). Il processo unificato suddivide l'input in una griglia, genera bounding box e mappe di probabilità di classe simultaneamente in un unico passaggio.

Il principio fondamentale di YOLO consiste nel riformulare il rilevamento degli oggetti come un singolo problema di regressione. L'immagine in input viene suddivisa in una griglia di celle. Ciascuna cella è responsabile di predire un numero predefinito di bounding box e le probabilità di appartenenza a una classe per gli oggetti il cui centro si trova al suo interno. Questo permette di generare tutte le predizioni simultaneamente con un'unica valutazione della rete.

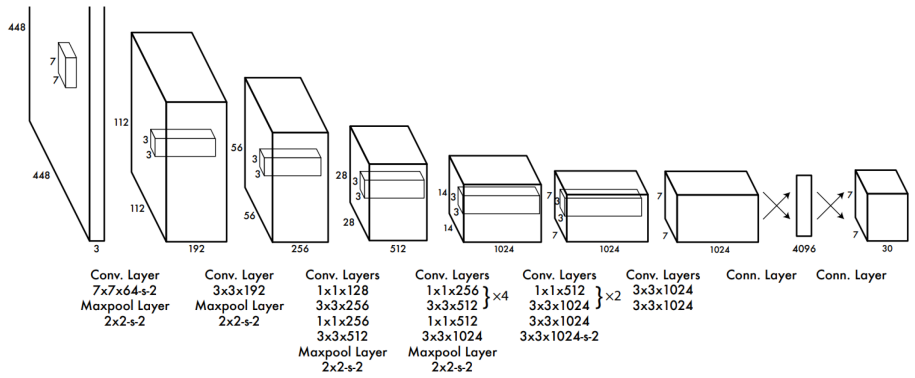


Figura 2.3: **Architettura YOLOv1.** La prima architettura YOLO ha un totale di 24 livelli convoluzionali con 2 livelli completamente connessi alla fine.

Evoluzione e Struttura Moderna

Dalla sua concezione, la filosofia YOLO si è basata su un'intuizione fondamentale: riformulare il rilevamento di oggetti come un singolo problema di regressione, eliminando la necessità di una fase separata per la generazione di proposte di regioni. Le versioni precedenti a YOLO12 hanno perfezionato questo approccio attraverso architetture interamente basate su Reti Neurali Convoluzionali (CNN) e hanno consolidato una struttura canonica articolata in tre componenti principali:

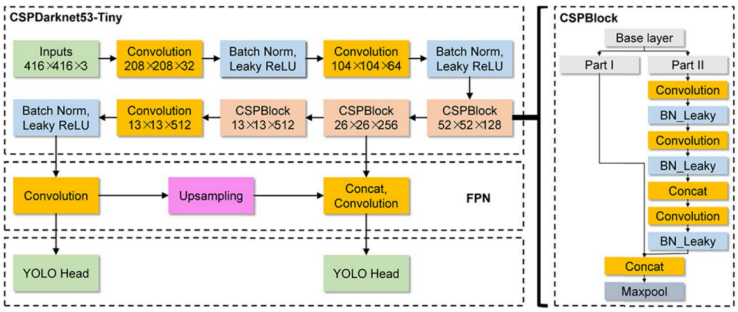


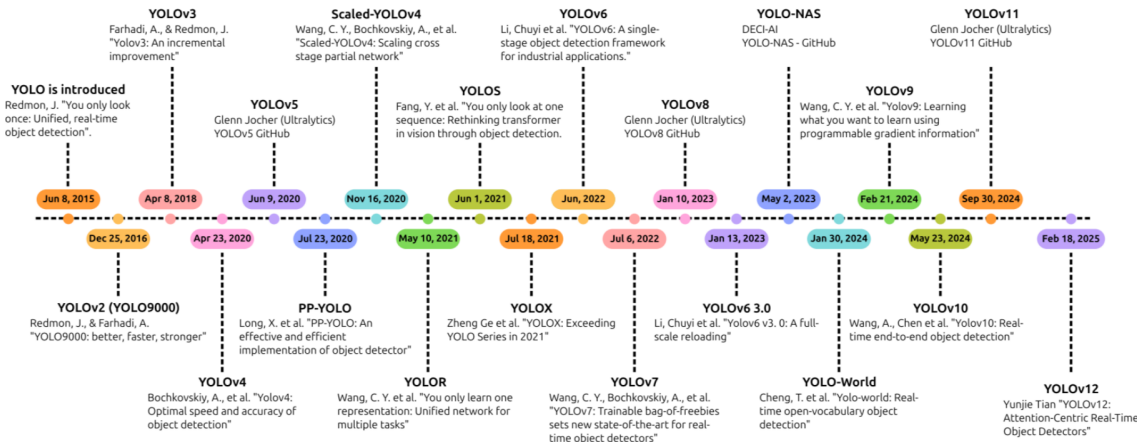
Figura 2.4: **Architettura YOLOv4.** L'achitettura contenente i tre componenti principali è stata consolidata a partire dalla versione YOLOv4 e mantenuto nelle successive

- **Backbone**: Rappresenta la spina dorsale della rete e agisce come un potente estrattore di feature gerarchiche. Costituito da una profonda rete convoluzionale pre-addestrata su dataset di vasta scala, il suo compito è analizzare l'immagine in input e apprendere a riconoscere concetti visivi a livelli di astrazione crescenti. Gli strati iniziali imparano a identificare feature di basso livello come bordi e gradienti di colore, mentre gli strati più profondi combinano queste informazioni per riconoscere texture, forme complesse e, infine, parti di oggetti.

- **Neck**: Questa componente si comporta come un ponte tra il backbone e l'head di rilevamento. La sua funzione è quella di aggregare e combinare le feature map estratte a diversi livelli di profondità. Architetture come le Path Aggregation Networks (PAN) vengono impiegate per fondere informazioni semantiche ad alto livello con dettagli spaziali a risoluzione più fine, creando rappresentazioni multi-scala essenziali per rilevare oggetti di dimensioni molto diverse.

- **Head**: È la parte finale e decisionale della rete, responsabile della generazione delle predizioni. Operando sulle feature map aggregate, l'head stima, per ogni cella della griglia, le coordinate dei bounding box, un punteggio di confidenza (objectness) e le probabilità di classe. Gli head moderni sono decoupled, ovvero utilizzano rami distinti per la regressione dei box e la classificazione, e generano predizioni su più scale per ottimizzare il rilevamento di oggetti piccoli, medi e grandi. Tuttavia, le architetture più recenti hanno superato il concetto di una "decoupled head" statica. YOLOv11 ha introdotto un "Dynamic Head Design", capace di adattare le proprie funzionalità in base alle caratteristiche dell'input per migliorare l'accuratezza. YOLO12 evolve ulteriormente questo concetto: la sua "Detection Head" è potenziata da tecnologie all'avanguardia come FlashAttention e meccanismi di adattamento dinamico.

Questo approccio, pur essendo estremamente efficiente, si affida implicitamente alla capacità degli strati convoluzionali di apprendere le relazioni spaziali e contestuali più rilevanti, un paradigma che YOLO12 ha evoluto attraverso l'integrazione di meccanismi di attenzione.



Evoluzione della famiglia YOLO

Scalabilità

Uno dei punti di forza della famiglia YOLO è la sua intrinseca scalabilità. Per rispondere alle diverse esigenze applicative, ogni versione dell'architettura viene distribuita in più varianti, identificate da suffissi come nano (n), small (s), medium (m), large (l) ed extra-large (x). Queste varianti offrono un trade-off controllato tra accuratezza e velocità, ottenuto modulando la profondità e la larghezza del modello, e di conseguenza il numero di parametri. Questa flessibilità permette di selezionare il modello più adatto al contesto, da sistemi embedded con risorse limitate che privilegiano la rapidità (es. YOLO12n) a implementazioni cloud dove la priorità è massimizzare l'accuratezza (es. YOLO12x).

Funzioni di perdita e strategie di ottimizzazione

Le versioni più recenti della famiglia YOLO hanno introdotto importanti miglioramenti anche nelle strategie di addestramento. L'obiettivo è ottimizzare la convergenza, stabilizzare il training e migliorare la precisione complessiva del modello.

Dal punto di vista delle funzioni di perdita, le architetture moderne adottano versioni adattive della *IoU Loss*, come la *CIoU* e la *SIoU*, che incorporano informazioni geometriche aggiuntive (ad esempio la distanza tra i centri e l'allineamento

angolare dei bounding box) per una regressione più accurata. Per la classificazione, sono spesso impiegate funzioni di perdita bilanciate come la *Focal Loss* o sue varianti, utili per mitigare lo sbilanciamento tra esempi facili e difficili.

Sul piano dell'ottimizzazione, le implementazioni più recenti fanno uso dell'algoritmo *Stochastic Gradient Descent* (SGD) con *momentum* e strategie come il *cosine learning rate decay*, che favoriscono una discesa più stabile e una migliore generalizzazione. In alcuni casi vengono introdotte tecniche di *warm-up* iniziale o *gradient clipping* per migliorare la stabilità numerica nelle prime fasi di training.

Questi perfezionamenti, uniti alle innovazioni architetturali, hanno contribuito a rendere YOLO una delle famiglie di modelli più versatili ed efficienti nell'ambito dell'object detection moderna.

2.3 Modello YOLO12

La scelta di adottare YOLO12 per il progetto Street Sign Sense è motivata da una convergenza di fattori tecnici, pratici e strategici che lo rendono la soluzione ottimale per gli obiettivi prefissati.

Evoluzione Architetturale

Le versioni precedenti di YOLO si basavano quasi esclusivamente su architetture convoluzionali (CNN) per l'estrazione e l'elaborazione delle feature, YOLO12 segna un'evoluzione significativa introducendo un **design "attention-centric"**. Come evidenziato dalla documentazione [3], questo modello *"si discosta dagli approcci tradizionali basati su CNN"*, integrando meccanismi di attenzione per migliorare l'accuratezza senza compromettere la velocità di inferenza.

Un'architettura *attention-centric* mira a simulare la capacità umana di concentrarsi selettivamente sulle parti più importanti di una scena visiva. Invece di trattare tutte le regioni e le feature estratte con la stessa importanza, il meccanismo di attenzione apprende dinamicamente a pesare le informazioni, assegnando maggiore rilevanza alle aree e alle caratteristiche più indicative per il compito di rilevamento.

Componenti Architetturali

L'architettura di YOLO12 mantiene la struttura canonica Backbone-Neck-Head, ma ogni componente è stato riprogettato integrando elementi avanzati:

- **Backbone Ottimizzato (Basato su R-ELAN):** Il backbone, responsabile dell'estrazione iniziale delle feature, impiega blocchi **R-ELAN (Residual Enhanced Layer Aggregation Network)**. Questi moduli rappresentano l'evoluzione diretta delle architetture precedenti, superando i limiti dei moduli CSPNet (YOLOv4/v5) e ELAN/GELAN (YOLOv7/v9). Migliorano la precedente architettura ELAN combinando l'efficacia delle connessioni residue, note per facilitare il training di reti molto profonde, con meccanismi avanzati di aggregazione delle feature, migliorando l'apprendimento delle dipendenze spaziali e contestuali. Un'ulteriore ottimizzazione in YOLO12 consiste nella riduzione della profondità dei blocchi impilati (Stacked Blocks Reduction): rispetto ai tre blocchi tipici delle versioni recenti, YOLO12 ne utilizza uno solo nell'ultima fase, semplificando l'ottimizzazione e aumentando la velocità di inferenza, specialmente nei modelli più profondi.

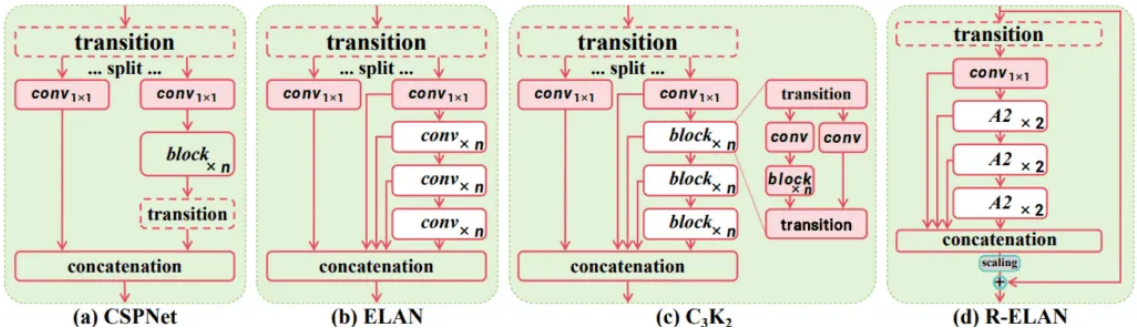


Figura 2.5: Confronto tra i moduli di backbone (a): **CSPNet**, (b) **ELAN** , (c) **C3K2** (un caso di GELAN) e (d) il nuovo modulo **R-ELAN** (Residual Efficient Layer Aggregation Networks) presente nell'architettura di YOLO12.

- **Neck Avanzato (Area Attention):** Per ottimizzare l'efficienza computazionale, YOLO12 introduce il meccanismo di **Area Attention Module**. A differenza di altri approcci di attenzione locale (come Shift Window o Axial Attention) che possono introdurre overhead, l'Area Attention divide semplicemente la feature map in segmenti più ampi (tipicamente $l = 4$, orizzontalmente o verticalmente). Questo metodo trasforma l'attenzione globale in locale senza complesse partizioni, preservando un ampio campo recettivo. Grazie a questa suddivisione, il costo computazionale viene ridotto drasticamente: passando da $2n^2hd$ a $\frac{1}{l}2n^2hd$. Con un valore di default $l = 4$, il costo diventa $(n^2hd)/2$, ottimizzando il meccanismo di Multi-Head Attention del 75%. Questo risparmio è reso possibile non riducendo i canali o le head, ma calcolando l'attenzione su segmenti spaziali ridotti, e mantenendo alta la velocità di esecuzione.

Figura 2.6: Confronto tra i meccanismi alternativi di attenzione locale e l'Area Attention

- **Head Dinamico con FlashAttention:** L'head di rilevamento rappresenta forse l'innovazione più distintiva. Superando il concetto di "decoupled head" statica, YOLO12 implementa un **Head Dinamico** con l'integrazione di **FlashAttention**, una tecnologia che ottimizza l'accesso alla memoria durante le operazioni di attenzione, risolvendo uno dei principali colli di bottiglia e colmando il divario di velocità con le CNN tradizionali. Inoltre, la **rimozione della codifica posizionale** nei livelli di attenzione semplifica ulteriormente l'architettura, rendendola più "pulita" ed efficiente senza sacrificare le prestazioni nel rilevamento. Questo permette alla rete di focalizzarsi sulle regioni e sui canali più informativi in modo adattivo, migliorando significativamente la precisione nella localizzazione (regressione dei bounding box) e nella classificazione, specialmente per oggetti piccoli, parzialmente occlusi o in condizioni di illuminazione difficili.

- **Ottimizzazione Computazionale (MLP Ratio e Operatori):** Per bilanciare il carico computazionale tra i meccanismi di attenzione e i blocchi feed-

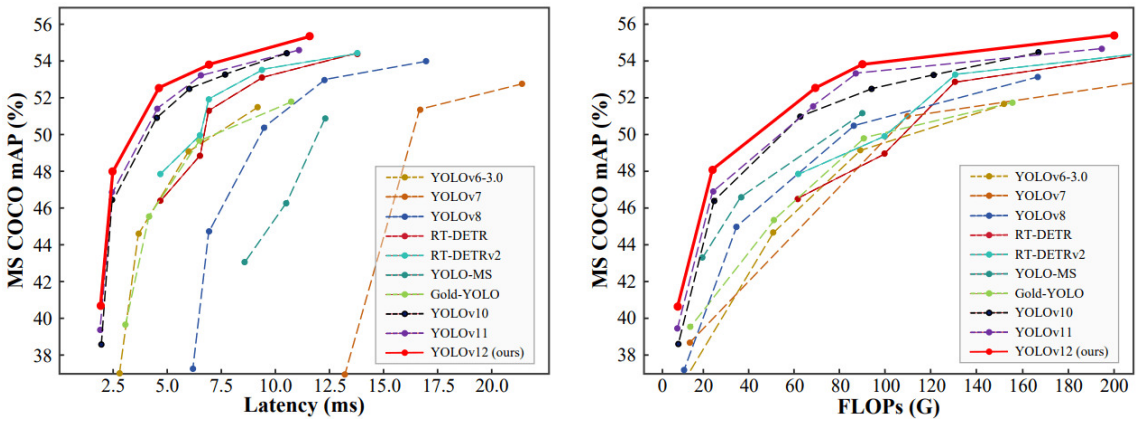
forward, il **rapporto di espansione del Multi-Layer Perceptron** (MLP Ratio Adjustment) è stato ridotto da 4 (valore comune nei Transformer) a circa 1.2. Questo evita che l'MLP domini il tempo di esecuzione. Infine, l'architettura fa ampio uso di operatori di convoluzione con batch normalization anziché layer lineari con layer normalization, sfruttando la maggiore efficienza computazionale degli operatori convoluzionali ottimizzati per l'hardware moderno.

Sinergia tra CNN e Area Attention

L'introduzione dei **meccanismi di area attention** (A2) non sostituisce le reti convoluzionali, ma le integra in un'architettura ibrida più efficace. Gli **strati convoluzionali** (CNN) restano fondamentali per l'estrazione delle feature visive, producendo feature map che codificano bordi, texture e forme, sulle quali i moduli di attenzione operano per identificare le informazioni più rilevanti, sia a livello spaziale (aree dell'immagine che contengono elementi cruciali), sia a livello di canali (tipi di feature che risultano più significative). Questa capacità consente al modello di creare **sinergia architetturale** tra estrazione e interpretazione delle feature, permettendogli di comprendere meglio il contesto, gestire occlusioni complesse e distinguere oggetti con caratteristiche simili con maggiore precisione, rappresentando un vantaggio decisivo rispetto alle architetture precedenti.

Vantaggi Applicativi e Pratici

Oltre all'innovazione architetturale, la scelta di YOLO12 è stata consolidata da una serie di considerazioni pratiche che ne hanno garantito la fattibilità e la riproducibilità. YOLO12 si posiziona come una soluzione allo stato dell'arte, offrendo un eccellente compromesso tra accuratezza di rilevamento (mAP su MS COCO) ed efficienza computazionale (Latenza e FLOPs) rispetto ai precedenti modelli Ultralytics.



Confronto delle prestazioni di YOLO12 con altri modelli Ultralytics

Questa superiorità nel bilanciamento di prestazioni e efficienza si traduce in benefici concreti:

- **Accuratezza Migliorata:** I meccanismi di attenzione e l'head dinamico affrontano specificamente le sfide del rilevamento nel mondo reale. Le diverse varianti del modello permettono di navigare efficacemente il trade-off tra velocità di inferenza e accuratezza di rilevamento.

Modello	dimensione (pixel)	mAP^{val} 50-95	Velocità CPU ONNX (ms)	Velocità T4 TensorRT (ms)	parametri (M)	FLOPs (B)	Confronto (mAP/Velocità)
YOLO12n	640	40.6	-	1.64	2.6	6.5	+2.1%/-9% (vs. YOLOv10n)
YOLO12s	640	48.0	-	2.61	9.3	21.4	+0.1%/+42% (vs. RT-DETRv2)
YOLO12m	640	52.5	-	4.86	20.2	67.5	+1.0%/-3% (vs. YOLO11m)
YOLO12l	640	53.7	-	6.77	26.4	88.9	+0.4%/-8% (vs. YOLO11l)
YOLO12x	640	55.2	-	11.79	59.1	199.0	+0.6%/-4% (vs. YOLO11x)

Confronto tra Accuratezza e Velocità

- **Efficienza Real-Time:** Nei sistemi avanzati di assistenza alla guida (ADAS), l'elaborazione in tempo reale rappresenta un requisito fondamentale. Ciò implica la necessità di processare flussi video a frame rate di almeno 30 fps per garantire una reattività adeguata. Nonostante la sua complessità architetturale, YOLO12 mantiene il vantaggio principale della famiglia YOLO: la velocità di inferenza, è in grado di raggiungere throughput superiori ai 60 fps anche su hardware non specializzato, offrendo un margine di performance essenziale per l'integrazione in sistemi con vincoli temporali.

- **Transfer Learning e Fine-Tuning Mirato:** La disponibilità di utilizzare un modello pre-addestrato su un dataset di vasta scala (in questo caso, COCO) ha permesso di sfruttare efficacemente il **transfer learning**, per trasferirne la conoscenza a un compito specifico. Nello specifico, è stata adottata una strategia di **fine-tuning**,in cui i pesi pre-addestrati non sono stati utilizzati come un estrattore di feature statico, ma sono stati raffinati continuando il processo di backpropagation sull'intero modello. Questo approccio ha due vantaggi strategici: primo, accelera drasticamente la convergenza, riducendo il tempo di training complessivo; secondo, permette di raggiungere un'accuratezza superiore anche con un dataset di dimensioni moderate, rendendo il progetto fattibile con le risorse computazionali a disposizione.

- **Ecosistema di Sviluppo Ultralytics:** L'ecosistema di sviluppo fornito da Ultralytics, astrae gran parte della complessità implementativa, offrendo un ambiente di lavoro completo e intuitivo, come la possibilità di configurazione tramite file YAML. Inoltre, il framework integra nativamente il monitoraggio automatico delle metriche e la visualizzazione dei risultati, fornendo un feedback immediato sull'andamento del training. Questo ha permesso di concentrare l'attenzione sugli aspetti metodologici del progetto.

- **Compatibilità e Accessibilità:** L'efficienza di YOLO12 e la sua compatibilità con diversi ambienti di esecuzione, dallo sviluppo su GPU fino ai sistemi embedded o server di produzione, evidenziano la flessibilità e l'adattabilità a contesti operativi realistici.

  Inoltre, l'integrazione con un ambiente di sviluppo preconfigurato ha favorito un processo di prototipazione rapido ed efficiente, permettendo di concentrare l'attenzione sull'analisi del modello e sull'ottimizzazione dei parametri, anziché sulla gestione delle dipendenze e del sistema.

Capitolo 3

Metodologia

3.1 Dataset - Street Sign Set

Per questo progetto, è stato realizzato un dataset personalizzato, denominato Street Sign Set, attraverso un processo iterativo di raccolta, integrazione e annotazione mirato a rappresentare la variabilità dei segnali stradali in contesti realistici.

Origine e Costruzione

Il dataset **Street Sign Set** è stato assemblato attraverso un processo mirato di selezione e integrazione di dati. Come punto di partenza è stato utilizzato un dataset pubblico disponibile sulla piattaforma Kaggle costituito da circa 4000 immagini. Questa base iniziale è stata ampliata in modo considerevole, con più di 3000 nuove immagini, sia integrando una selezione di immagini di altri dataset presenti su Kaggle, sia attingendo da altre fonti, per reperire esempi specifici, ottenendo così un totale di oltre 7000 immagini.

Un elemento cruciale e distintivo di questo lavoro è stato l'arricchimento del dataset con il materiale raccolto, per compensare la scarsità di esempi delle classi sottorappresentate. In questa fase è stata svolta l'acquisizione manuale di immagini da servizi di mappatura stradale e altre fonti pubbliche, attraverso una selezione qualitativa.

La decisione di creare un dataset dedicato nasce dalla necessità di rappresentare classi meno comuni e da una analisi delle risorse disponibili. Nonostante su piattaforme come Kaggle esistano diversi dataset dedicati alla segnaletica, questi presentano diverse limitazioni: molti sono di dimensioni ridotte, non coprono alcune delle classi di interesse, oppure sono composti prevalentemente da immagini sintetiche o vettoriali, prive della complessità visiva dell'ambiente stradale. Lo Street Sign Set, con 63 classi e la focalizzazione su contesti reali, è stato concepito proprio per colmare questo vuoto, offrendo una rappresentazione più completa rispetto alle alternative già presenti.

Il processo di costruzione è stato iterativo, cioè le immagini sono state aggiunte progressivamente in base alle necessità rilevate dalle analisi delle performance dei modelli durante fasi di training e di test, mirando a rafforzare le classi o le condizioni in cui si osservavano maggiori difficoltà nel rilevamento.

Formato e Annotazioni

Il dataset adotta lo standard di annotazione YOLO. A ogni immagine sorgente è associato un file di testo (`.txt`) con lo stesso filename. Ogni riga del file di testo contiene l'identificatore (indice) della classe seguito dalle coordinate normalizzate del bounding box $(x_{centro}, y_{centro}, larghezza, altezza)$. La fase di etichettatura manuale è stata condotta sulla piattaforma Roboflow, che ha permesso di definire bounding box precisi. Tuttavia, data la diversità delle fonti originali, per garantire l'integrità strutturale del dataset, è stato necessario sviluppare ed eseguire una serie di script Python dedicati al pre-processing e alla normalizzazione dei dati. Questi script hanno svolto funzioni di data curation:

- *Mapping delle classi* per rimappare etichette provenienti da fonti diverse e ottenere uno schema unificato con le 63 classi.
- *Filtraggio delle classi* per rimuovere specifiche annotazioni.
- *Gestione dei file* per gestire i file correttamente in modo da garantire struttura e file consistenti e conformi ai requisiti del framework.
- *Rinomina dei file* per ottenere filename con uno schema logico definito utilizzando il nome della classe e il progressivo (`nome_classe-n.jpg`).
- *Analisi del dataset* per analizzare le occorrenze di ogni singola classe e la loro distribuzione nei vari split (training, validation, test).

Dimensioni e Distribuzione

La versione finale del dataset **Street Sign Set** è costituita da un totale di oltre 7300 immagini. La ripartizione dei dati è stata pianificata strategicamente: il training set conta 5467 immagini e il validation set 1746, con un rapporto di 74% e 24%. . Il test set è stato intenzionalmente mantenuto di dimensioni contenute, comprendendo 160 immagini; questa scelta deriva dalla volontà di concentrare la quasi totalità delle risorse disponibili nella costituzione di set di addestramento e validazione particolarmente robusti e variegati, massimizzando così la capacità di apprendimento del modello.

Analizzando la distribuzione a livello di singole annotazioni (*bounding boxes*), su un totale complessivo di **12.170** istanze etichettate, il dataset risulta così strutturato:

- **Training Set:** 8996 annotazioni ($\approx 73.92\%$);
- **Validation Set:** 2886 annotazioni ($\approx 23.71\%$);
- **Test Set:** 288 annotazioni ($\approx 2.37\%$).

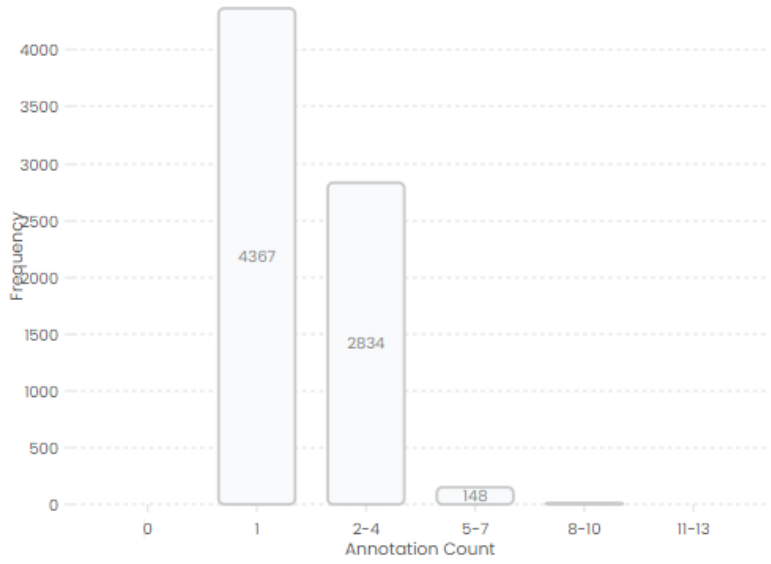


Figura 3.1: Istogramma del numero di classi annotate in ogni immagine

Classi del Dataset

Il dataset comprende 63 classi distinte di segnali stradali, tra queste ci sono 23 classi che sono state identificate come target primari per l'analisi delle performance. Le 23 classi principali riguardano la segnaletica di:

- **Limiti di velocità** (14 classi, es. 5–130 km/h).
- **Segnali di divieto** (4 classi, es. divieto di sosta/fermata, divieto di sorpasso auto/camion).
- **Segnali di precedenza** (2 classi , es. dare precedenza e stop).
- **Curve e attraversamenti**.(3 classi, es, curva pericolosa a destra/sinistra, attraversamento pedonale)

Data Augmentation

Per mitigare lo sbilanciamento nella frequenza delle diverse classi, è stata implementata data augmentation selettiva per classi sottorappresentate. Le trasformazioni includevano:

- Grayscale: Apply to 23% of images;
- Hue: Between -102° and +102;
- Saturation: Between -40% and +40%;
- Brightness: Between -25% and +25%;
- Blur: Up to 4.5px;
- Noise: Up to 2.7% of pixels;

Analisi della Distribuzione Spaziale

Un aspetto importante nella validazione del dataset riguarda la distribuzione spaziale degli oggetti di interesse. La Annotation Heat Map (Figura 3.2) offre una rappresentazione visiva della densità delle annotazioni, dove le sfumature di colore indicano la frequenza con cui i bounding box appaiono in specifiche celle della griglia dell'immagine. In un contesto di guida reale, si ha una concentrazione naturale dei segnali in determinate aree (tipicamente ai bordi della carreggiata o in alto), proprio come dimostrato dalla heatmap che nonostante è ben distribuita, indica che il maggior numero delle annotazioni è localizzato nel nella parte destra e nella fascia superiore dell'immagine. Una distribuzione eccessivamente polarizzata potrebbe introdurre un *bias posizionale* nel modello, portandolo a ignorare a priori le zone dell'immagine dove i segnali appaiono più raramente. L'analisi di questa mappa permette di verificare se occorre adottare una strategia di Data Augmentation. Tecniche come il *Mosaic* e le trasformazioni prospettiche casuali(Ritaglio casuale, Rotazione, Capovolgimento, ecc.) sono utili proprio per contrastare questo fenomeno, rendendo le feature invarianti rispetto alla posizione.

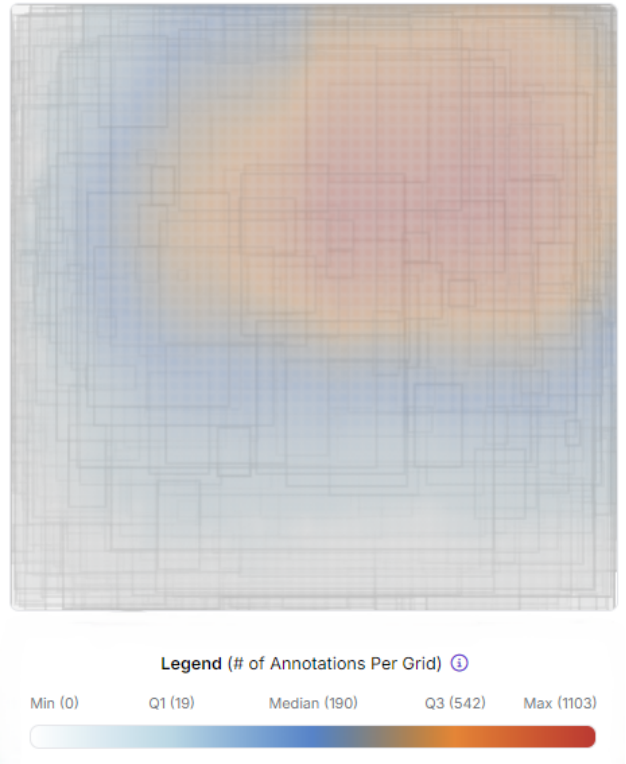


Figura 3.2: Annotation Heat Map del dataset: visualizzazione della densità posizionale dei bounding box.

3.2 Ambiente di Sviluppo

L'intero processo di sviluppo, addestramento e validazione dei modelli è stato eseguito sull'ambiente computazionale **Kaggle Notebooks**. Scelta motivata dalla combinazione di accessibilità, potenza computazionale e riproducibilità che offre.

Hardware e Cloud

Kaggle offre accesso completamente gratuito a sessioni di calcolo con acceleratori **GPU NVIDIA Tesla T4**. In ogni sessione sono state sfruttate due GPU T4 in

parallelo, permettendo di accelerare il processo di training e distribuire il carico computazionale tra le due unità.

Le GPU T4, offrono 16GB di memoria ciascuna e sono ottimizzate per inferenza e training di reti neurali. Queste risorse sono state sfruttate automaticamente dal framework Ultralytics per ridurre i tempi di training.

La possibilità di eseguire **sessioni di commit** in cui notebooks vengono eseguiti in background nel **cloud**, è stata fondamentale, per gestire il training dei modelli senza la necessità di mantenere attive le sessioni interattive.

La configurazione hardware include anche 30GB di RAM di sistema e 4 core CPU Intel Xeon, risorse sufficienti per le operazioni di preprocessing dei dati e di input output necessarie per alimentare continuamente le GPU con batch di dati.

Stack Software e Framework

L'interfaccia di sviluppo utilizzata da Kaggle è Jupyter Notebook, che ha permesso l'esecuzione delle celle di codice Python e celle di markdown per la documentazione, semplificando le fasi del progetto, dalla configurazione alla valutazione. . L'ambiente preconfigurato di Kaggle, basato su Linux Ubuntu con CUDA 11.8 e cuDNN, ha eliminato le complessità della gestione delle dipendenze, ma affidarsi esclusivamente alle versioni preinstallate nelle piattaforme cloud comporta rischi per la riproducibilità a lungo termine, a causa dei frequenti aggiornamenti automatici delle librerie. Infatti per garantire la stabilità e assicurare che i risultati siano replicabili indipendentemente dal momento dell'esecuzione, le librerie critiche per il progetto sono state definite in un file `requirements.txt`, richiedendo in questo modo l'installazione di versioni specifiche.

L'installazione delle dipendenze viene eseguita all'avvio nel seguente modo:

```
!pip install -r /kaggle/input/sssrequirements/requirements.txt
```

Il file di configurazione, fissa le versioni del framework **Ultralytics** (per l'architettura YOLO), di **OpenCV** (per la manipolazione delle immagini) e delle librerie di supporto, eliminando potenziali incompatibilità future.

requirements.txt

```
ipython==7.34.0
opencv_contrib_python==4.10.0.84
opencv_python==4.10.0.84
opencv_python_headless==4.10.0.84
Pillow==11.3.0
PyYAML==6.0.3
ultralytics==8.3.229
```

Librerie standard come OpenCV, NumPy e Matplotlib sono state ivece utilizzate per il preprocessing delle immagini, le manipolazioni numeriche e la visualizzazione dei risultati.

3.3 Fase di Addestramento

Strategia di Transfer Learning e Fine-Tuning

Come discusso nel Capitolo 2, l'addestramento si è basato sul transfer learning. Sono stati utilizzati i pesi pre-addestrati sul dataset COCO forniti da Ultralytics come punto di partenza. L'intero modello (backbone, neck e head) è stato poi sottoposto

a fine-tuning sul dataset *Street Sign Set*, permettendo alla rete di adattare le feature apprese sul dataset generico al dominio specifico dei segnali stradali.

Configurazione, Ottimizzazione ed Early Stopping

Il processo di addestramento è stato configurato tramite Python e il framework Ultralytics, definendo parametri specifici del training e passandoli direttamente alla funzione di avvio. Mentre la definizione della struttura del dataset (percorsi e classi) è stata delegata al file di configurazione `data.yaml`.

I parametri principali sono :

- **Epoche**: L'addestramento è stato eseguito con un numero fissato a 600 epoche massimo.
- **Early stopping**: È stato attivato un meccanismo di **early stopping** con una **patience** di 100 epoche. Questo significa che il training si interrompe automaticamente se le metriche di performance, in particolare la mAP@50-95 (o la fitness complessiva), non mostrano miglioramenti per 100 epoche consecutive.
- **Optimizer**: È stato utilizzato l'ottimizzatore Stochastic Gradient Descent (SGD) con momentum (valore di default, 0.9).
- **Learning rate**: Il learning rate iniziale è stato impostato a 0.01 con una fase di **warmup lineare** seguito da **cosine annealing schedule** che riduce il learning rate durante il resto del training per favorire la convergenza verso un minimo ottimale.
- **Dimensioni Immagine**: Tutte le immagini sono ridimensionate a 640×640 pixel prima di essere fornite alla rete, una dimensione standard che offre un buon compromesso tra dettaglio e carico computazionale.
- **Batch Size Differenziato:** La dimensione del batch è stata adattata per ogni variante modello addestrato. Le varianti, quando vengono caricate sulle GPU occupano differenti quantità di memoria, dunque questo influisce sullo spazio che resta disponibile per il caricamento delle immagini del batch. Per cui al fine di massimizzare l'utilizzo delle GPU disponibili (2x NVIDIA T4) i batch size per ogni variante sono stati impostati in questo modo:
  - YOLO12n: 140 immagini per step (70 per GPU).
  - YOLO12s: 80 immagini per step (40 per GPU).
  - YOLO12m: 40 immagini per step (20 per GPU).

  Questi settaggi hanno permesso di sfruttare al massimo le risorse computazionali delle GPU per ciascuna variante del modello, ottimizzando notevolmente i tempi di training evitando sprechi di risorse o errori di Out-of-Memory.
- **Gestione Augmentation: Parametro `fliplr`** Il parametro di data augmentation relativo al ribaltamento orizzontale (`fliplr`), nella configurazione dell'addestramento, è stato fissato a `0.0`. Questa decisione è motivata dalla natura asimmetrica delle classi target (es. segnaletica direzionale): il ribaltamento orizzontale dell'immagine avrebbe alterato il significato semantico della classe (trasformando, ad esempio, una indicazione "sinistra" in "destra"), compromettendo la capacità del modello di distinguere correttamente l'orientamento degli oggetti.

L'implementazione pratica di questa configurazione è riportata nei codici a seguire, dove si evidenzia anche l'impostazione esplicita per l'utilizzo parallelo delle GPU.

```
model = YOLO('yolo12n.pt')

results = model.train(
    data='data.yaml',
    imgsz=640,
    epochs=600,
    patience=100,
    save_period=50,
    device='0,1',
    batch=140,
    fliplr=0.0,
    name='yolo12n_run',
    project='streetsignsense'
)
```

```
model = YOLO('yolo12s.pt')

results = model.train(
    data='data.yaml',
    imgsz=640,
    epochs=600,
    patience=100,
    save_period=50,
    name='yolo12s_run',
    project='streetsignsense',
    device='0,1',
    batch=80,
    fliplr=0.0
)
```

```
model = YOLO('yolo12m.pt')

results = model.train(
    data='data.yaml',
    imgsz=640,
    epochs=600,
    patience=100,
    save_period=50,
    name='yolo12m_run',
    project='streetsignsense',
    device='0,1',
    batch=40,
    fliplr=0.0
)
```

Tempi di Addestramento Effettivi e Gestione delle Risorse

L'addestramento è stato configurato per estendersi fino a un massimo di 600 epoche, integrando il meccanismo di *early stopping* nativo del framework Ultralytics con una *patience* impostata a 100 epoche. L'obiettivo dell'early stopping è interrompere il processo qualora non si osservassero miglioramenti tangibili sulle metriche di validazione, in questo caso la soglia di epoche senza miglioramenti è stata impostata per 100 epoche consecutive, ottimizzando così l'uso delle risorse.

L'esecuzione sull'ambiente Kaggle (2x NVIDIA T4) ha permesso di misurare i tempi di esecuzione; le dinamiche di arresto specifiche per ciascun modello sono riportate di seguito:

- **YOLOv12n:** Il modello nano ha impiegato un runtime di 5h 38m 40s. Il meccanismo di early stopping è intervenuto correttamente, arrestando il training all'epoca 361. Il checkpoint finale con le prestazioni ottimali è stato dunque registrato all'epoca *261*.
- **YOLOv12s:** La variante small ha richiesto un runtime di 7h 4m 25s, con l'early stopping che ha determinato la convergenza ottimale all'epoca *170*, arrestando infatti l'addestramento all'epoca 270, considerata la mancanza di miglioramenti nelle ultime 100 epoche.
- **YOLOv12m:** La variante medium, caratterizzata da una maggiore complessità computazionale, ha richiesto una strategia di training suddivisa in due sessioni, per rispettare i vincoli di runtime della piattaforma Kaggle (limite di 12 ore consecutive). Il processo complessivo ha impiegato un tempo totale di 13h 23m 53s.

  Nella prima sessione, l'addestramento si è interrotto forzatamente allo scadere della quota oraria disponibile, raggiungendo l'epoca 262. Un'analisi del file di log (`results.csv`) ha permesso di identificare il picco di performance all'epoca 187, con una mAP@50-95 pari a 0.81.

  Per verificare l'eventuale presenza di margini di miglioramento, è stata avviata una seconda sessione impostando una *patience* ridotta a 25 epoche. Tuttavia, il meccanismo di *early stopping* non è intervenuto alla soglia prevista, portando il training a proseguire fino all'epoca 291. Il monitoraggio in tempo reale delle metriche ha evidenziato una stagnazione delle prestazioni, con valori di mAP@50-95 oscillanti intorno a 0.80, inferiori al picco precedentemente registrato. Di conseguenza, l'addestramento è stato interrotto manualmente per evitare un inutile dispendio di risorse. Il sistema di *checkpointing* ha comunque garantito il salvataggio dei pesi ottimali corrispondenti all'epoca *187* (`best.pt`), scartando le iterazioni successive.

Data Augmentation Online

La Data Augmentation Online viene effettuata ad ogni batch durante il training a differenza dell'augmentation offline, che viene applicata selettivamente sulle classi sottorappresentate per mitigarne lo sbilanciamento. A gestire questo tipo di augmentation è il framework Ultralytics che ha applicato queste trasformazioni direttamente in memoria GPU prima del forward pass. Questo permette di migliorare la capacità di generalizzazione del modello a nuovi contesti visivi.

La strategia di augmentation online è stata strutturata su tre livelli complementari: trasformazioni geometriche, variazioni cromatiche e filtri di image processing tramite librerie esterne.

1. Augmentation Geometrica e Strutturale (Nativa)

Queste trasformazioni modificano la disposizione spaziale e la struttura delle immagini per rendere il modello invariante alla posizione e alla scala:

- **Mosaic:** Il Mosaic è applicato con probabilità 1.0, ed è una tecnica importante di YOLO. Consiste nel fondere quattro immagini di training in un unico mosaico, costringendo il modello a riconoscere oggetti in contesti non usuali e migliorando le performance su oggetti più piccoli.
- **Trasformazioni Affini:** Sono state applicate variazioni di scala (±50%, `scale=0.5`).
- **Translation**: ±10% di traslazione, (`translate=0.1`).
- **Flip Orizzontale:** Questa trasformazione è stata **intenzionalmente disabilitata** (`fliplr=0.0`). A differenza di altri domini, nella segnaletica stradale la direzione è una caratteristica semantica distintiva; applicare un ribaltamento orizzontale altererebbe il significato del segnale (es. trasformando un "Obbligo di svolta a sinistra" in "Obbligo di svolta a destra") causando possibile confusione tra le classi.
- **Random Erasing:** Con una probabilità del 40% (`erasing=0.4`), vengono cancellate casualmente porzioni dell'immagine. Questa tecnica simula le occlusioni (es. segnali coperti da vegetazione o pali), aumentando le capacità del modello a riconoscere segnali coperti.

2. Augmentation Cromatica (Spazio HSV)

Per garantire robustezza contro le diverse condizioni di illuminazione (sole, ombra, crepuscolo), sono state introdotte modifiche nello spazio colore HSV (Hue, Saturation, Value):

- **Saturazione (`hsv_s=0.7`):** Una variazione del ±70% sulla saturazione permette al modello di generalizzare su segnali sbiaditi o estremamente vividi.
- **Luminosità (`hsv_v=0.4`):** Una variazione del ±40% di luminosità, simula diverse condizioni di esposizione solare.
- **Tonalità (`hsv_h=0.015`):** Una variazione dell'1.5% sulla tonalità preserva la semantica del colore (es. il rosso deve restare rosso) ma introduce variabilità realistica.

3. Integrazione con Albumentations

Il framework è configurato per utilizzare anche la libreria *Albumentations* per applicare filtri di image processing specifici con una probabilità dell'1% ($p = 0.01$) ciascuno. Sebbene rari, questi filtri agiscono come regolarizzatori fini:

- **Blur e MedianBlur:** Introducono sfocature (raggio 3-7px) per simulare motion blur o difetti di messa a fuoco.
- **CLAHE:** L'equalizzazione adattiva dell'istogramma migliora il contrasto locale, utile per segnali in ombra.
- **ToGray:** Converte l'immagine in scala di grigi, forzando il modello a riconoscere le forme indipendentemente dal colore.

Monitoraggio Durante il Training

Il framework Ultralytics ha fornito un monitoraggio delle prestazioni durante l'addestramento. Per ogni epoca, sono state registrate le metriche calcolate sul validation set: **Precision**, **Recall**, **mean Average Precision mAP@50 e mAP@50-95**, le tre componenti della **loss function**: *box_loss*, errore nella regressione dei bounding box; *cls_loss*, errore di classificazione; *dfl_loss*, errore nella localizzazione dei bounding box intorno alla loro posizione reale, (Distribution Focal Loss). Per ogni variante addestrata, sono stati generati i grafici con tali metriche (si vedano le Figure 3.3, 3.4 e 3.5). I dati relativi a queste metriche sono stati archiviati nei rispettivi file `results.csv`, consentendo successive analisi. In particolare, attraverso una comparazione delle curve di loss, è possibile osservare che le tre varianti hanno un comportamento di convergenza simile durante il training, pur con velocità e valori finali differenti (si vedano le Figure C.1, C.2 e C.3 nell'Appendice C).

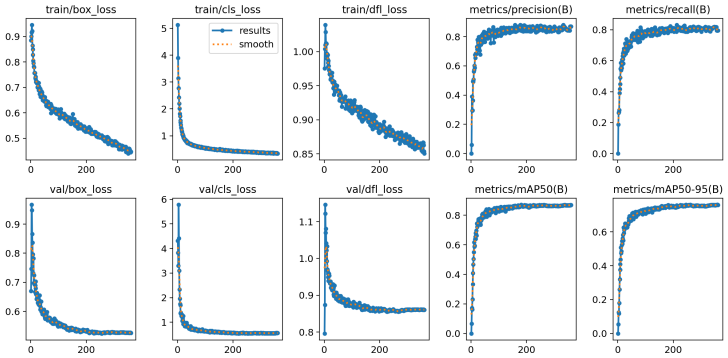


Figura 3.3: Risultati della variante nano

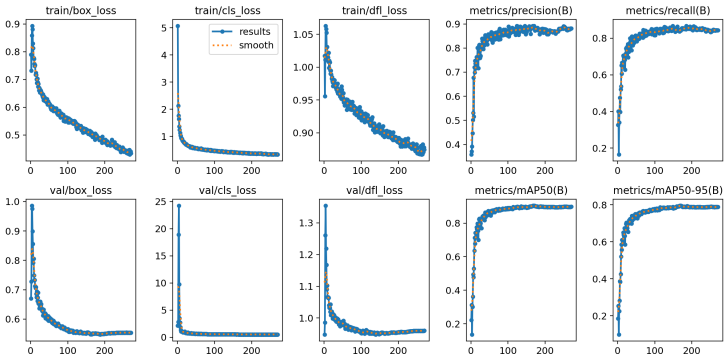


Figura 3.4: Risultati della variante small

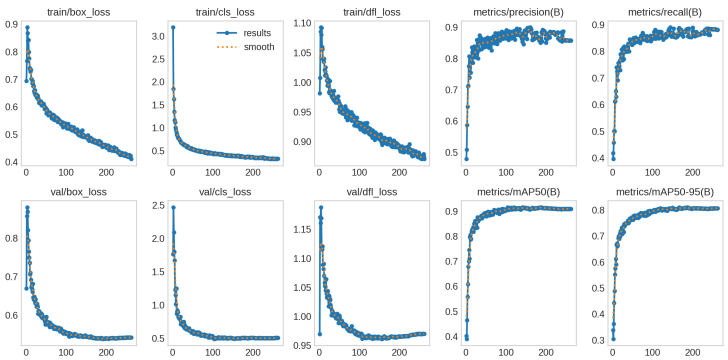


Figura 3.5: Risultati della variante medium

Capitolo 4

Risultati e Analisi

In questo capitolo vengono presentati e discussi i risultati sperimentali ottenuti dall'addestramento dei modelli YOLOv12 sul dataset *Street Sign Sense*. L'analisi si articola in una valutazione quantitativa, basata sulle metriche standard di object detection, e in un'analisi qualitativa volta a esaminare il comportamento dei modelli in scenari reali.

4.1 Metriche di Valutazione

Per garantire una valutazione rigorosa e comparabile delle prestazioni, sono state adottate le metriche standard consolidate nel campo dell'object detection. Ciascuna di queste metriche fornisce informazioni specifiche su aspetti diversi delle capacità del sistema, e la loro analisi congiunta permette di sviluppare una comprensione completa delle prestazioni effettive dei modelli

- **Precision (P):** Indica l'affidabilità delle predizioni positive. Rappresenta la frazione di rilevamenti corretti rispetto al totale dei rilevamenti effettuati, $Precision = \frac{TP}{TP+FP}$.

  In termini operativi, questa metrica risponde alla domanda: tra tutti gli oggetti che il modello ha classificato come positivi, quanti lo erano davvero? Una precision elevata implica un basso numero di falsi positivi, caratteristica estremamente desiderabile in applicazioni di assistenza alla guida dove detection errate potrebbero causare comportamenti pericolosi del sistema e ridurre la fiducia dell'utente.

- **Recall (R):** Misura la capacità del modello di trovare tutti gli oggetti rilevanti. Rappresenta la frazione di oggetti reali correttamente identificati rispetto al totale degli oggetti presenti nel dataset, $Recall = \frac{TP}{TP+FN}$.

  Dal punto di vista applicativo, questa metrica risponde alla domanda: tra tutti i casi realmente positivi, quanti ne ha trovati il modello? Un Recall elevato indica che il modello non perde oggetti di interesse, aspetto assolutamente cruciale per applicazioni safety-critical dove la mancata detection di un segnale di stop o di un limite di velocità potrebbe avere conseguenze estremamente gravi per la sicurezza.

La metrica più completa e informativa per valutare sistemi di object detection è la *mean Average Precision* che integra informazioni su Precision e Recall, su tutto lo spettro di soglie di confidenza possibili.

Il calcolo della mean Average Precision inizia con la costruzione della *curva Precision-Recall* per ciascuna classe, ottenuta variando la soglia di confidenza e calcolando Precision e Recall per ciascun valore di soglia. L'area sotto la curva Precision-Recall è definita come Average Precision. La mean Average Precision è poi calcolata come media aritmetica delle Average Precision su tutte le classi presenti nel dataset.

Nel contesto specifico di YOLO e più in generale dell'object detection, è prassi consolidata considerare due varianti della *mean Average Precision* basate su criteri diversi per determinare quando una detection debba essere considerata corretta. Una detection viene considerata True Positive se l'intersezione (IoU) tra il bounding box predetto e il ground truth supera una certa soglia prestabilita. L'Intersection over Union è definita come il rapporto tra l'area di intersezione dei due boxes e l'area della loro unione, fornendo una misura naturale e intuitiva di quanto accuratamente il box predetto copre l'oggetto reale.

- **mAP@50 (mAP a IoU 0.5):** È la media delle precisioni medie (AP) calcolata per tutte le classi, considerando una predizione corretta se l'intersezione sull'unione (IoU) con il ground truth è almeno del 50%. Questa metrica è indicativa della capacità del modello di localizzare correttamente gli oggetti in modo "generale".

- **mAP@50-95:** Considerata la metrica più robusta e completa (standard COCO), calcola la media delle mAP su diverse soglie di IoU (da 0.50 a 0.95 con incrementi di 0.05). Premia i modelli che non solo individuano l'oggetto, ma lo localizzano con estrema precisione geometrica.

- **F1-Score:** Fornisce una misura armonica che bilancia Precision e Recall, un singolo valore che sintetizza l'equilibrio tra la capacità di non generare falsi positivi e quella di non perdere oggetti (falsi negativi). L'F1-Score è calcolato nel seguente modo: $F_1 = 2 \cdot \frac{\text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}}$. Questa metrica risulta utile quando si cerca una configurazione bilanciata che non privilegi nessuna delle due metriche, ma piuttosto ottimizzi il compromesso tra le due. L'F1-Score raggiunge il suo valore massimo quando entrambe le metriche sono elevate e perfettamente bilanciate.

- **Tempo di Inferenza:** Il tempo medio necessario al modello per processare una singola immagine, misurato in millisecondi (ms). È un indicatore per valutare l'applicabilità pratica del sistema in scenari real-time. Da questa metrica è possibile ottenere il *frame rate* teorico (FPS) raggiungibile dal modello, $FPS = \frac{1000}{\text{Tempo di Inferenza (ms)}}$. Per applicazioni che operano tipicamente a trenta frame per secondo, il tempo di inferenza deve essere inferiore a 33 millisecondi per frame per garantire elaborazione in tempo reale senza introdurre latenze inaccettabili.

4.2 Risultati Quantitativi

La Tabella sottostante riassume le prestazioni finali ottenute dalle tre varianti di YOLO12 (Nano, Small, Medium) sul validation set. I valori riportati si riferiscono al miglior checkpoint (`best.pt`) salvato durante il training.

Modello	Precision	Recall	F1-Score	mAP@50	mAP@50-95	Inferenza (ms)*	Dimensione
YOLO12n	0.849	0.819	0.825	0.870	0.760	5.4	5.32 MB
YOLO12s	0.887	0.846	0.853	0.904	0.796	12.6	18.1 MB
YOLO12m	0.882	0.859	0.856	0.916	0.811	31.7	77.6 MB

Confronto delle prestazioni dei modelli YOLO12.

*Tempi di inferenza misurati su GPU NVIDIA T4

Analisi Comparativa

I dati evidenziano un chiaro trade-off tra complessità computazionale e accuratezza, tipico delle architetture di deep learning, ma con alcune specificità interessanti legate all'architettura YOLO12.

YOLO12n: Efficienza

La variante **Nano** si distingue per la sua straordinaria leggerezza. Con un peso di soli **5.32 MB** e un tempo di inferenza di appena 5.4 ms (corrispondenti a un throughput teorico di circa 185 FPS), è la scelta ideale per dispositivi edge con risorse limitate. La velocità non compromette la qualità: una mAP@50-95 di 0.760 e un F1-Score di 0.825 indicano che il modello è pienamente capace di gestire la complessità del task, e sebbene le metriche di accuratezza siano le più basse del gruppo, il modello mantiene comunque una precisione media superiore all'84% (Average Precision di 0.849) e una Recall di 0.819 superando ampiamente l'80%, dimostrandosi abbastanza capace per la sua taglia.

YOLO12s: Il Compromesso Bilanciato

Il modello **Small** emerge come la configurazione più equilibrata, spicca in particolare per la Precision di 0.887, che risulta essere la più alta tra i tre modelli testati. Questo suggerisce che è la scelta migliore per minimizzare i falsi positivi. Rispetto alla variante nano, offre un incremento del 3.6% nella mAP@50-95 salendo così a 0.796, e ad un F1-Score che supera la soglia di 0.85. Questo guadagno di accuratezza comporta però un raddoppio del tempo di inferenza a 12.6 ms e ad una crescita della dimensione del modello a 18.1 MB. Tuttavia, con circa 79 FPS teorici, rimane ampiamente nel range del real-time, rappresentando il miglior equilibrio per applicazioni generali.

YOLO12m: Massima Accuratezza

La variante **Medium** raggiunge, come previsto, le vette più alte di accuratezza, con una mAP@50 di 0.916 e una mAP@50-95 di 0.811. È interessante notare che, mentre la precisione pura è leggermente inferiore al modello Small (0.838 vs 0.846), l'elevato Recall a 0.859 indica che è il modello che "perde" meno segnali in assoluto. Questo indica che il modello Medium riesce a trovare più segnali difficili, questo infatti spiega la mAP complessiva più alta. Tuttavia, il costo computazionale è significativo: il modello pesa 77.6 MB (quattro volte la versione Small) e il tempo di inferenza sale a 31.7 ms (circa 31 FPS). Nonostante sia in grado di operare in real-time, il margine è molto più ridotto, rendendolo idoneo solo per hardware dotato di accelerazione GPU dedicata per garantire la fluidità necessaria.

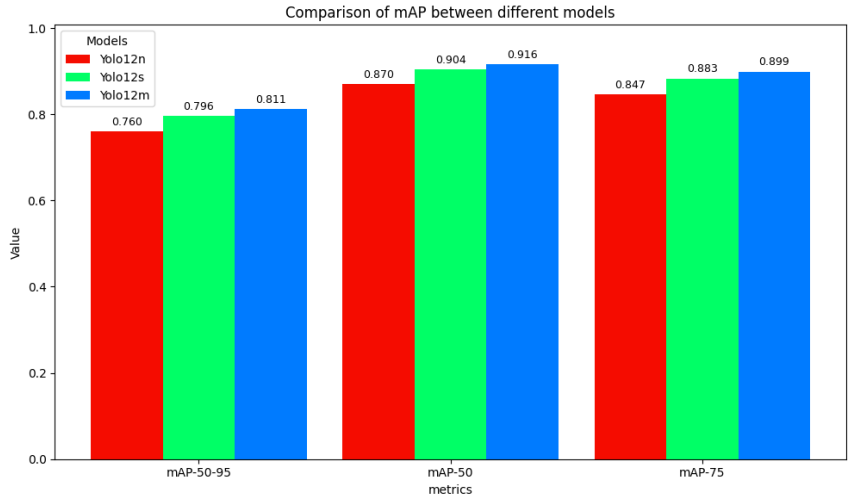


Figura 4.1: Confronto grafico delle metriche mAP tra le varianti.

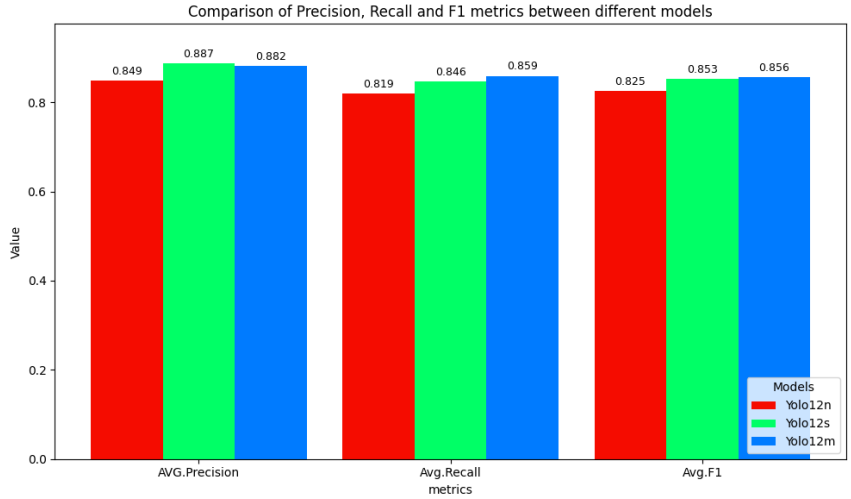


Figura 4.2: Confronto grafico delle metriche Avg-Precision, Avg-Recall e F1-Score tra le varianti.

Dettaglio per Classe: Matrici di Confusione

Le matrici di confusione normalizzate di ciascuna variante del modello, sono utili per approfondire la distribuzione degli errori tra le 63 classi. Come osservabile nelle Figure 4.3, 4.4 e 4.5, l'asse delle ordinate rappresenta le classi reali (*True Label*), mentre l'asse delle ascisse indica le predizioni del modello (*Predicted Label*).
Una diagonale principale marcata e continua indica un'elevata accuratezza di classificazione. I valori al di fuori della diagonale evidenziano invece le *misclassifications*.

- **YOLOv12n:** Sebbene la diagonale sia ben definita, si nota una maggiore dispersione (rumore) nelle aree fuori diagonale, in particolare tra classi visivamente simili (es. varianti dei limiti di velocità) e verso la classe background (falsi negativi), confermando i limiti di capacità del modello più piccolo.
- **YOLOv12s e YOLOv12m:** All'aumentare della complessità del modello, la matrice si pulisce progressivamente. Nel modello Medium, la dispersione fuori diagonale è minima, indicando una capacità superiore nel discriminare tra feature fini che distinguono segnali della stessa categoria.

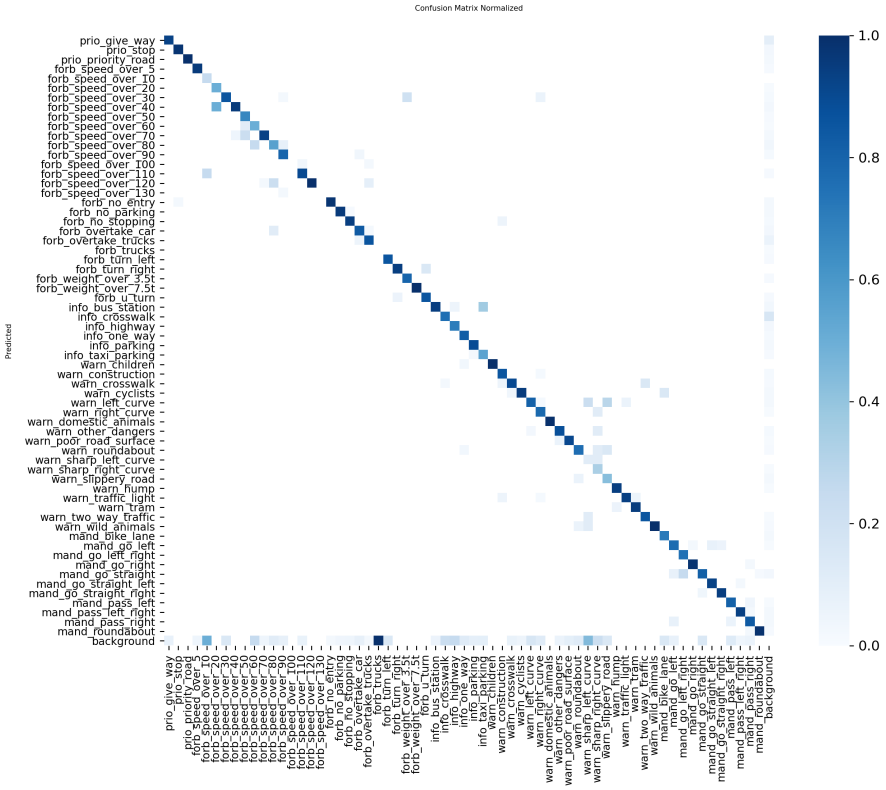


Figura 4.3: Matrice di confusione normalizzata per YOLO12n.

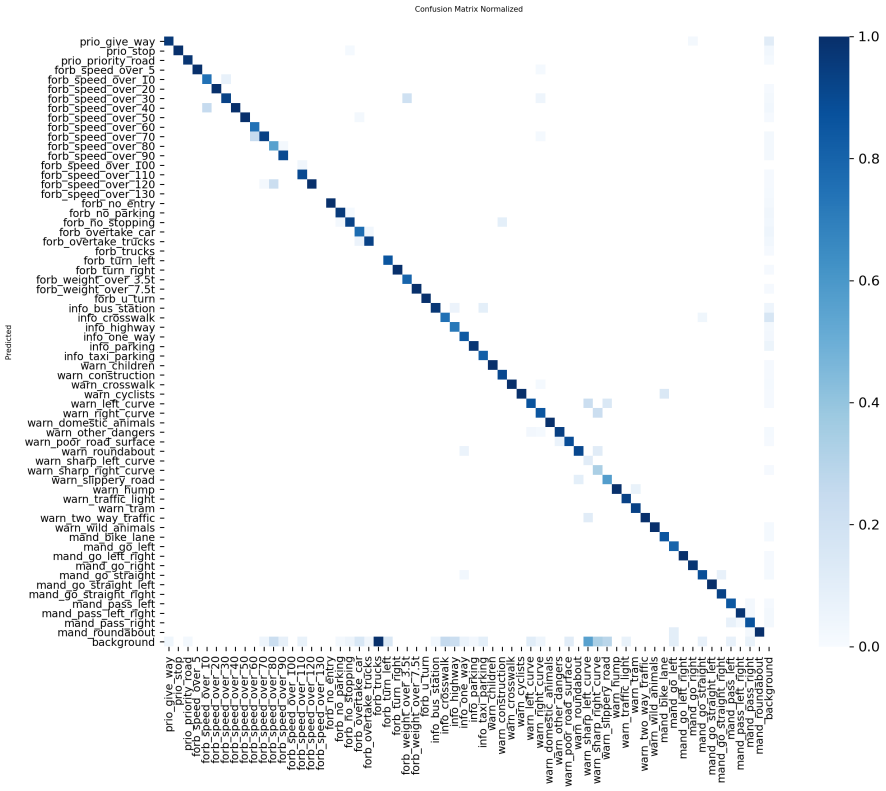


Figura 4.4: Matrice di confusione normalizzata per YOLO12s.

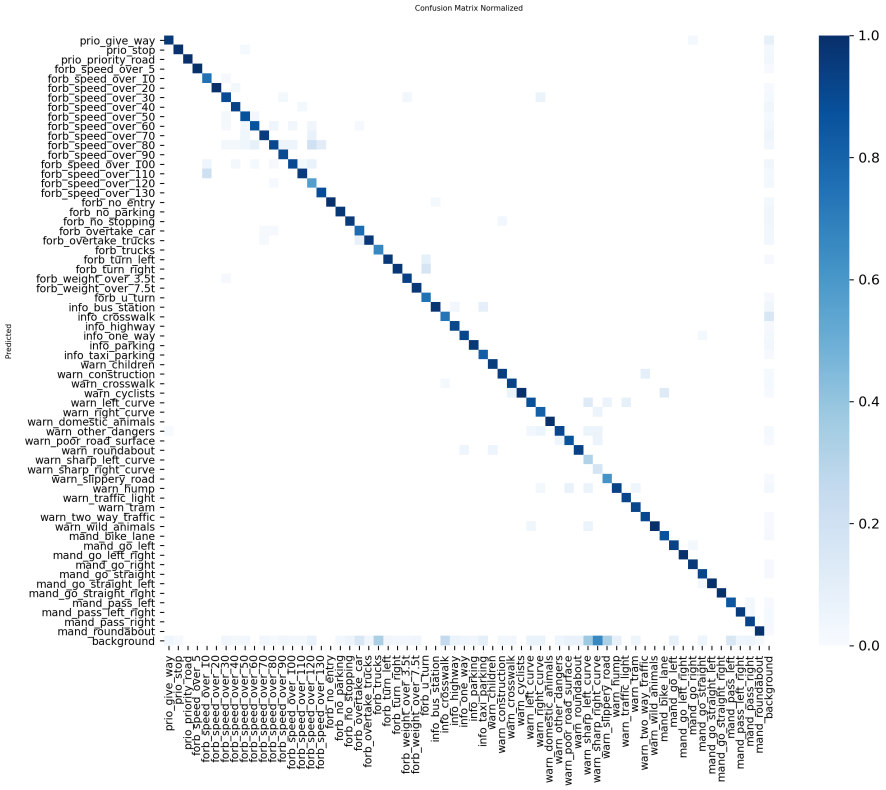


Figura 4.5: Matrice di confusione normalizzata per YOLO12m.

4.3 Analisi Qualitativa

L'ispezione visiva delle predizioni sul test set, incrociata con l'analisi delle matrici di confusione, ha rivelato un comportamento differente tra i vari modelli. Sebbene le metriche globali siano positive, l'analisi qualitativa evidenzia come le prestazioni non siano uniformi, ma variano in funzione della frequenza delle classi nel dataset e della complessità del segnale.

Capacità di Astrazione

Uno degli aspetti positivi emersi durante i test è la sorprendente robustezza del modello in condizioni più o meno critiche.

- **Segnali Parziali e Occlusi:** I modelli hanno dimostrato di saper rilevare correttamente segnali anche quando parzialmente tagliati dai bordi dell'immagine o oscurati da vegetazione o altri veicoli. Questo suggerisce che le reti convoluzionali hanno imparato a riconoscere le classi basandosi su feature parziali piuttosto che necessitare della sagoma intera. Occlusioni che coprono oltre il 50% del segnale vengono rilevate anche dal modello Nano. (Vedi Figura 4.6a)
- **Invarianza alla Scala e Distanza**: Il sistema riesce a localizzare sia segnali molto vicini che occupano una porzione significativa del frame sia oggetti distanti con un'occupazione di pochi pixel nell'immagine, confermando la capacità delle reti di estrarre feature su diverse scale spaziali. Un ruolo determinante è stato svolto dall'architettura multi-scala implementata nel Neck e nell'Head di YOLO12, che si è dimostrata efficace nella gestione delle variazioni di dimensione.
- **Tolleranza alle Prospettive:** La capacità di generalizzazione dei modelli è stata valutata anche rispetto alle prospettive derivate dall'angolazione degli scatti. I segnali con angolazioni laterali, anche se non significative, sono state riconosciuti correttamente.
- **Condizioni Ambientali**: Inoltre, i segnali vengono individuati con coerenza sia in condizioni di controluce che in scenari crepuscolari, dimostrando un'ottima invarianza fotometrica. Tuttavia, in presenza di scarsa illuminazione o contrasti estremi, il confidence score delle predizioni tende a diminuire, pur mantenendo corretta la localizzazione dell'oggetto. (Vedi Figura 4.6b)

(a) Esempio di segnale occluso a sinistra dell'immagine, rilevamento effettuato dal modello nano

(b) Esempio segnale in condizioni ambientali con forte contrasto

Figura 4.6

Impatto della Frequenza delle Classi

Nonostante gli sforzi di bilanciamento e data augmentation, il dataset riflette inevitabilmente la frequenza naturale dei segnali nel mondo reale. L'analisi qualitativa conferma una correlazione diretta tra la frequenza di una classe nel training set e la robustezza del rilevamento:

- **Classi Frequenti:** Segnali più comuni vengono rilevati con elevata confidenza e precisione di localizzazione (bounding box aderenti), anche in condizioni difficili.
- **Classi Sottorappresentate:** Per i segnali più rari, si osserva un calo delle prestazioni. In questi casi, il modello tende a essere più incerto (confidence score bassi) o a mancare del tutto il rilevamento (False Negatives).

Ambiguità e Confusioni Inter-classe

Il sistema dimostra una buona capacità discriminante, evitando generalmente di confondere oggetti simili (come cartelloni pubblicitari rotondi o triangolari) con segnali stradali. Tuttavia gli errori di classificazione sono presenti. L'analisi delle predizioni errate rispecchia le criticità emerse nelle matrici di confusione. Le difficoltà maggiori si concentrano su segnali che condividono una forte somiglianza visiva (forma e colore) ma differiscono per i dettagli interni:

- Limiti di Velocità: Si sono osservate confusioni occasionali tra limiti visivamente simili (es. 80 km/h vs 60 km/h) quando il segnale è distante o la risoluzione è bassa.
- Spesso, l'aggiustamento della soglia di confidenza per l'inferenza permette di filtrare queste predizioni errate, ma in alcuni casi l'errore persiste con score elevati.

Il modello cattura correttamente la semantica generale ("è un limite di velocità"), ma può fallire nella specifica sottoclasse. Si tratta di errori che, in un contesto operativo, potrebbero essere mitigati dal tracciamento del segnale su più frame differenti.
In sintesi, l'analisi qualitativa conferma che il sistema è affidabile per la segnaletica critica e comune, mostrando una notevole capacità di generalizzazione su segnali degradati o parziali, seppur con sporadiche confusioni tra sottoclassi o mancate rilevazioni.

Capitolo 5

Discussione

5.1 Interpretazione dei Risultati e Scenari Applicativi

L'analisi comparativa delle tre varianti di YOLO12 ha evidenziato come la scelta del modello non possa prescindere dal contesto applicativo specifico. I dati confermano che non esiste un vincitore assoluto, ma soluzioni ottimali per vincoli diversi:

- **YOLO12n (Edge-First):** Con un tempo di inferenza di soli 5.4 ms, questo modello si dimostra la scelta migliore per l'edge computing a bassissimo consumo, l'opzione praticabile per dispositivi embedded con memorie limitate, o per sistemi che richiedono frame rate elevatissimi. Sebbene paghi in termini di accuratezza sulle classi più difficili, la sua precisione media resta sorprendentemente alta per la sua dimensione.

- **YOLO12s (Best Balance):** Si conferma il punto di equilibrio ideale per la maggior parte delle applicazioni di assistenza alla guida su hardware standard. Il guadagno di accuratezza rispetto alla variante nano è tangibile, specialmente nella riduzione dei falsi positivi (Precision più alta del gruppo). Preferibile quindi in sistemi di sicurezza o ADAS dove un falso positivo è fastidioso o pericoloso (es. frenate fantasma causate da rilevamenti errati). Mantiene allo stesso tempo una velocità di inferenza pienamente nel range del real-time, compatibile con i requisiti dei moderni sistemi ADAS, in cui i limiti di memoria e di consumo energetico sono trascurabili. Sebbene la variante nano sia stata definita come la più efficiente per l'edge computing, i suoi 185 FPS diventano inutili se si necessita un livello di affidabilità superiore e si dispone di hardware con GPU dedicata.

- **YOLO12m (High-Accuracy):** Rappresenta la scelta obbligata quando la priorità è la massimizzazione del Recall e la localizzazione precisa. Con un tempo medio di inferenza di 31.7 ms, il modello riesce a superare la soglia del real-time, con un margine estremamente ridotto, e fatica a garantire i 30 FPS stabili richiesti per un video fluido. A differenza delle varianti più leggere necessita di hardware dotato di accelerazione GPU dedicata (superiore alla T4), poiché qualsiasi calo di risorse porterebbe il frame rate sotto la soglia di fluidità. Rimane comunque la soluzione di riferimento per scenari di Cloud Computing e di gestione in modalità offline (batch processing), dove la stabilità degli FPS è secondaria rispetto all'accuratezza del rilevamento.

5.2 Analisi delle Criticità Operative

Un aspetto rilevante emerso durante il progetto riguarda la complessità di gestione del training. L'esperienza con vari addestramenti effettuati con **YOLO12m** ha dimostrato che l'affidamento esclusivo agli automatismi del framework come l'*early stopping* può rivelarsi insufficiente in scenari di training frammentati che richiedono la ripresa dello stato (*resume*).

Nello specifico, è stato identificato un comportamento critico legato all'interruzione forzata della prima sessione e il successivo riavvio, che hanno comportato il reset del contatore interno dell'early stopping. Di conseguenza, nella seconda sessione il conteggio della *patience* è ripartito da zero. Questo ha impedito all'algoritmo di arrestare automaticamente il training, che è proseguito fino all'epoca 291 nonostante il picco di performance (best mAP) fosse stato già ottenuto all'epoca 187.

Questo evidenzia l'importanza di una supervisione attiva e di un'analisi dei log. In particolare in progetti complessi distribuiti su più sessioni, la definizione della *patience* e i criteri di stop devono essere calibrati tenendo conto non solo della stabilità delle metriche, ma anche dei meccanismi di reinizializzazione degli stati interni dell'ottimizzatore al momento del ripristino.

5.3 Limiti del Lavoro

Per garantire una corretta interpretazione dei risultati, è doveroso esplicitare i limiti metodologici dello studio:

- **Rappresentatività del Dataset:** Sebbene il dataset *Street Sign Set* (7000+ immagini) sia abbastanza valido, rimane limitato rispetto ai benchmark di dataset che contano decine di migliaia di campioni. Lo sbilanciamento naturale delle classi , seppur mitigato dall'augmentation, influenza le performance sulle categorie di segnali più rari.
- **Vincoli Computazionali e Temporali:** L'utilizzo di piattaforme cloud per l'addestramento ha imposto limiti stringenti sul tempo di esecuzione continuo. Questo vincolo ha impedito un'esplorazione esaustiva dello spazio di ricerca e ha reso necessario frammentare il training dei modelli più complessi, introducendo sfide operative nella gestione della convergenza e nel ripristino degli stati.

5.4 Sviluppi Futuri

Alla luce dei risultati e dei limiti discussi, le direzioni future di ricerca dovrebbero concentrarsi su:

- **Consolidamento del Test Set:** L'attuale set di test, conta 160 immagini e 288 annotazioni, concentrato principalmente sulle classi target. Un obiettivo primario sarà l'acquisizione di un test set più ampio (almeno 500-1000 immagini).
- **Espansione del Dataset:** Acquisizione mirata di nuovi esempi per le classi di coda (quelle con meno di 100 istanze) per livellare le performance di Recall tra le diverse categorie.

- **Copertura degli Scenari Critici:** Inclusione di scenari meteorologici critici attualmente assenti (nebbia fitta, neve, pioggia battente) per testare i limiti di robustezza del modello.
- **Prototipazione:** Implementazione del modello su un dispositivo edge installato su un veicolo per valutare le prestazioni sul campo, analizzando la robustezza rispetto a condizioni non rappresentate nel dataset attuale.

Capitolo 6

Demo Web

L'ultima fase del progetto *Street Sign Sense* ha previsto lo sviluppo di una demo web per dimostrare l'efficacia e la portabilità dei modelli YOLO12 addestrati. L'obiettivo era creare un ambiente di inferenza accessibile, capace di operare interamente *client-side* (direttamente nel browser dell'utente).

Accesso al Codice e alla Demo Live

Il codice sorgente completo dell'applicazione web e l'implementazione live della demo sono stati resi disponibili pubblicamente, garantendo la piena trasparenza e riproducibilità del sistema, accessibili ai seguenti indirizzi:

- **Repository GitHub:**
  `https://github.com/AlessandroFerrante/StreetSignSense/`
- **Demo Live (Web Page):**
  `https://alessandroferrante.github.io/StreetSignSense/`

6.1 Architettura del Sistema

L'applicazione è stata realizzata come una *Single Page Application* (SPA), progettata per essere leggera e responsiva. Il sistema è costituito da **TensorFlow.js**, una libreria open-source che permette l'esecuzione di modelli di Machine Learning in JavaScript. La libreria usufruisce della tecnologia WebGL, TensorFlow.js consente così di accelerare i calcoli matriciali necessari per le reti neurali convoluzionali utilizzando la GPU del dispositivo ospite (smartphone, laptop o desktop), rendendo possibile l'object detection in tempo reale senza hardware dedicato.

6.2 Pipeline di Elaborazione

Il funzionamento della demo segue una pipeline logica, che adatta i vari input alle specifiche richieste dall'architettura YOLO12.

Conversione e Caricamento del Modello

I modelli addestrati in PyTorch (formato `.pt`) sono stati convertiti nel formato `web_model` compatibile con TensorFlow.js (una struttura composta da un file `model.json` contenente la topologia del grafo (o architettura della rete) e i file binari contenenti i pesi).

L'applicazione permette all'utente di selezionare quale variante caricare:

- **YOLO12n (Nano):** Ideale per dispositivi mobili e bassa latenza, privilegia la velocità.
- **YOLO12s (Small):** Un buon compromesso tra velocità e precisione.
- **YOLO12m (Medium):** La versione più accurata, ma necessita di maggiori risorse hardware.

Il caricamento avviene in modalità asincrona tramite la funzione `tf.loadGraphModel`, successivamente viene eseguita un'inferenza su un tensore di zeri per inizializzare gli shader WebGL e allocare la memoria necessaria.

Pre-processing dell'Input

Le reti neurali richiedono che le immagini abbiano una forma e dimensione specifica (in questo caso 640×640 pixel). Prima di essere analizzata, ogni immagine subisce le seguenti trasformazioni:

1. **Ridimensionamento:** L'immagine sorgente (o il frame video) viene ridimensionata mantenendo le proporzioni originali.
2. **Padding:** Vengono aggiunti i bordi per raggiungere la dimensione target di input del modello (640×640 pixel). Fondamentale per evitare distorsioni che comprometterebbero la capacità della rete di riconoscere le forme dei segnali.
3. **Normalizzazione:** I valori dei pixel vengono normalizzati nel range $[0, 1]$ e convertiti in un tensore float a 32 bit, aggiungendo la dimensione del batch ($1 \times 640 \times 640 \times 3$).

Inferenza e Risultati

L'inferenza viene eseguita tramite il metodo `model.executeAsync`. Una volta che il modello ha analizzato l'immagine, produce un output grezzo (tensori contenenti coordinate dei box, punteggi di confidenza e probabilità delle classi) che viene elaborato per ottenere i risultati:

- **Filtraggio:** Vengono scartate le predizioni con un punteggio di confidenza inferiore alla soglia definita dall'utente (default 0.40).
- **Rescaling:** Le coordinate dei bounding box predetti, nell'immagine 640×640 (incluso il padding), vengono rimappate per adattarsi all'immagine originale.
- **Visualizzazione:** I bounding box vengono disegnati attorno ai segnali riconosciuti, etichettandoli con il nome della classe e la percentuale di probabilità.

6.3 Funzionalità e Interfaccia Utente

L'applicazione offre diverse modalità d'uso per testare il modello in vari scenari con un controllo sui parametri di inferenza.

Gestione degli Input

Il sistema supporta tre modalità di acquisizione:

- **Upload File:** Permette di analizzare foto o video salvati sul dispositivo.
- **Webcam/Camera:** Accesso diretto alla fotocamera per analizzare l'ambiente in tempo reale, o scattare una foto all'istante.
- **Demo Image:** Carica diverse immagini di test predefinite, per verificare le capacità dei modelli.

Pannello di Controllo

L'utente può interagire in tempo reale con i parametri del modello tramite slider dedicati:

- **Confidence Threshold:** Regola la sensibilità del modello.
- **IoU Threshold:** Aiuta a gestire il filtraggio delle sovrapposizioni.
- **Max Detections:** Permette di impostare il numero massimo di detection da visualizzare.

Metriche in Tempo Reale

Per valutare le prestazioni, l'interfaccia mostra:

- **FPS:** Indicatore della fluidità dello stream.
- **Tempo di Inferenza:** Tempo in millisecondi per l'inferenza di un singolo frame.
- **Conteggio Oggetti:** Numero di segnali rilevati nel frame corrente.

Figura 6.1: Interfaccia Demo Web

6.4 Considerazioni sulle Prestazioni Web

I test effettuati tramite la demo web confermano le osservazioni del capitolo precedente. La variante nano si dimostra performante su dispositivi standard, garantendo maggiore fluidità, mentre la variante medium, pur essendo più precisa, mostra un incremento del tempo di inferenza, risultando più lenta. La variante small conferma la sua posizione, offrendo un'ottima accuratezza e mantenendo un frame rate accettabile.

Capitolo 7

Conclusioni

Il progetto *Street Sign Sense* è nato con l'intento di esplorare l'object detection applicata alla sicurezza stradale. Al termine dello sviluppo, è possibile affermare che gli obiettivi prefissati non solo sono stati soddisfatti, ma in diversi aspetti le aspettative iniziali sono state superate, in particolare per quanto riguarda l'efficienza operativa dei modelli.

Dal punto di vista applicativo, si può confermare la fattibile integrazione di architetture avanzate come YOLO12 all'interno di sistemi ADAS (Advanced Driver Assistance Systems). La capacità dei modelli di operare in tempo reale, mantenendo un'elevata accuratezza, dimostra che la tecnologia attuale può supportare attivamente il conducente, contribuendo alla sicurezza stradale.

Oltre ai risultati tecnici, questo progetto ha rappresentato un'eccellente esperienza formativa. Affrontare l'intero ciclo di vita di un sistema di Machine Learning, dalla raccolta dei dati al deployment, ha permesso di comprendere in profondità non solo il funzionamento teorico delle Reti Neurali Convoluzionali e dei moderni meccanismi di Attenzione, ma anche le sfide pratiche che caratterizzano questo ambito.

Un aspetto significativo è stato l'apprendimento derivato dalla gestione diretta dei dati. Il lavoro di costruzione, pulizia e arricchimento del dataset e l'attenzione data all'integrazione manuale di oltre 3000 nuove immagini e nelle annotazioni hanno reso evidente come la qualità sia fondamentale per ogni sistema intelligente efficace.

Inoltre, il progetto è stato fondamentale per mettere in pratica e affinare la capacità di leggere e interpretare le metriche con piena consapevolezza. Passare dalla teoria all'analisi critica dei grafici di training e delle matrici ha permesso di sviluppare una visione analitica per comprendere non solo quanto un modello performi bene, ma perché si comporti in un certo modo.

In conclusione, sono pienamente soddisfatto del lavoro svolto, dei risultati tecnici raggiunti e dall'esperienza derivata. Questo percorso ha consolidato il mio interesse nel settore, fornendomi le competenze e una visione critica necessarie per progetti futuri. L'auspicio è che il materiale prodotto in questo studio possa costituire una base valida per nuovi utilizzi e ulteriori sviluppi.

Capitolo 8

Riproducibilità e Risorse del Progetto

Per assicurare la piena trasparenza, accessibilità e riproducibilità del lavoro svolto, tutti i materiali principali del progetto *Street Sign Sense* sono stati resi disponibili pubblicamente.

Il progetto è disponibile sulla piattaforma GitHub (per il codice sorgente e l'applicazione web) e la piattaforma Kaggle (per il dataset e gli ambienti di training originali).

- **Codice Sorgente e Demo Live (GitHub):** Il repository contiene il codice sorgente per l'addestramento, gli script di analisi e l'implementazione completa della demo web.

  Repository GitHub:

  `https://github.com/AlessandroFerrante/StreetSignSense/`

  Demo Web (Pagina GitHub Pages):

  `http://alessandroferrante.github.io/StreetSignSense/`

- **Dataset e Ambienti di Training (Kaggle):** Il materiale ospitato su Kaggle garantisce l'accesso diretto all'ambiente di training utilizzato.

  Dataset *Street Sign Set*: Alessandro Ferrante. (2025). Street Sign Set [Data set]. Kaggle. `https://doi.org/10.34740/KAGGLE/DS/8410752`

  `https://www.kaggle.com/datasets/ferrantealessandro/street-sign-set/`

  Notebooks di Training Originali:

  `https://www.kaggle.com/code/ferrantealessandro/streetsignsense-yolo12n`

  `https://www.kaggle.com/code/ferrantealessandro/streetsignsense-yolo12s`

  `https://www.kaggle.com/code/ferrantealessandro/streetsignsense-yolo12m`

  Modelli:

  `https://www.kaggle.com/models/ferrantealessandro/streetsignsensey12n/`

  `https://www.kaggle.com/models/ferrantealessandro/streetsignsensey12s/`

  `https://www.kaggle.com/models/ferrantealessandro/streetsignsensey12m/`

Bibliografia

[1] Tian, Yunjie and Ye, Qixiang and Doermann, David (2025). *YOLOv12: Attention-Centric Real-Time Object Detectors*. arXiv preprint arXiv:2502.12524. Technical Report available at `https://github.com/sunsmarterjie/yolov12`.

[2] Redmon, J., Divvala, S., Girshick, R., & Farhadi, A. (2016). *You Only Look Once: Unified, Real-Time Object Detection*. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), pp. 779-788.

[3] Jocher, G., Chaurasia, A., & Qiu, J. (2023). *Ultralytics YOLO* (Version 8.3.229) [Software]. Disponibile su `https://github.com/ultralytics/ultralytics` e documentazione su `https://docs.ultralytics.com/`.

[4] Dao, T., et al. (2022). *FlashAttention: Fast and Memory-Efficient Exact Attention with IO-Awareness*. arXiv preprint arXiv:2205.14135. (Tecnologia citata nell'architettura YOLO12).

[5] Lin, T. Y., et al. (2014). *Microsoft COCO: Common Objects in Context*. In European Conference on Computer Vision (ECCV).

[6] Everingham, M., et al. (2010). *The Pascal Visual Object Classes (VOC) Challenge*. International Journal of Computer Vision. (Riferimento per la standardizzazione delle metriche IoU e mAP@50).

[7] Buslaev, A., Iglovikov, V. I., Khvedchenya, E., Parinov, A., Druzhinin, M., & Kalinin, A. A. (2020). *Albumentations: Fast and Flexible Image Augmentations*. Information, 11(2), 125.

[8] Bochkovskiy, A., Wang, C. Y., & Liao, H. Y. M. (2020). *YOLOv4: Optimal Speed and Accuracy of Object Detection*. (Introduce il concetto di Mosaic Augmentation).

[9] Farinella, G. M., & Leonardi, R. (2024). *Teaching material for the Machine Learning course*. Department of Mathematics and Computer Science - University of Catania

[10] Alessandro Ferrante. (2025). Street Sign Set [Data set]. Kaggle. `https://doi.org/10.34740/KAGGLE/DS/8410752`

Sitografia e Risorse Online

- **Ultralytics Documentation**: `https://docs.ultralytics.com/` - Documentazione ufficiale del framework utilizzato per il training.

- **Kaggle**: `https://www.kaggle.com/` - Piattaforma utilizzata per il reperimento del dataset iniziale e l'ambiente di calcolo (GPU T4).
- **Roboflow**: `https://roboflow.com/` - Strumenti utilizzati per l'annotazione, la gestione e il pre-processing del dataset.
- **PyTorch**: `https://pytorch.org/` - Framework di Deep Learning sottostante.

Appendice A

Elenco Completo delle Classi del Dataset

Il dataset *Street Sign Set* comprende le seguenti 63 classi di segnali stradali, come definite nel file `data.yaml`:

- `prio_give_way`: Dare precedenza
- `prio_stop`: Stop
- `prio_priority_road`: Strada con diritto di precedenza
- `forb_speed_over_5`: Limite di velocità 5 km/h
- `forb_speed_over_10`: Limite di velocità 10 km/h
- `forb_speed_over_20`: Limite di velocità 20 km/h
- `forb_speed_over_30`: Limite di velocità 30 km/h
- `forb_speed_over_40`: Limite di velocità 40 km/h
- `forb_speed_over_50`: Limite di velocità 50 km/h
- `forb_speed_over_60`: Limite di velocità 60 km/h
- `forb_speed_over_70`: Limite di velocità 70 km/h
- `forb_speed_over_80`: Limite di velocità 80 km/h
- `forb_speed_over_90`: Limite di velocità 90 km/h
- `forb_speed_over_100`: Limite di velocità 100 km/h
- `forb_speed_over_110`: Limite di velocità 110 km/h
- `forb_speed_over_120`: Limite di velocità 120 km/h
- `forb_speed_over_130`: Limite di velocità 130 km/h
- `forb_no_entry`: Divieto di accesso
- `forb_no_parking`: Divieto di sosta
- `forb_no_stopping`: Divieto di fermata

- `forb_overtake_car`: Divieto di sorpasso (auto)
- `forb_overtake_trucks`: Divieto di sorpasso (camion)
- `forb_trucks`: Divieto di transito (camion)
- `forb_turn_left`: Divieto di svoltare a sinistra
- `forb_turn_right`: Divieto di svoltare a destra
- `forb_weight_over_3.5t`: Divieto di transito (peso $>$ 3.5t)
- `forb_weight_over_7.5t`: Divieto di transito (peso $>$ 7.5t)
- `forb_u_turn`: Divieto di inversione a U
- `info_bus_station`: Fermata autobus
- `info_crosswalk`: Attraversamento pedonale (informazione)
- `info_highway`: Inizio autostrada
- `info_one_way`: Senso unico
- `info_parking`: Parcheggio
- `info_taxi_parking`: Parcheggio taxi
- `warn_children`: Pericolo bambini
- `warn_construction`: Lavori in corso
- `warn_crosswalk`: Attraversamento pedonale (pericolo)
- `warn_cyclists`: Pericolo ciclisti
- `warn_left_curve`: Curva a sinistra
- `warn_right_curve`: Curva a destra
- `warn_domestic_animals`: Pericolo animali domestici
- `warn_other_dangers`: Altri pericoli
- `warn_poor_road_surface`: Strada deformata
- `warn_roundabout`: Intersezione con rotatoria
- `warn_sharp_left_curve`: Doppia curva con la prima a sinistra
- `warn_sharp_right_curve`: Doppia curva con la prima a destra
- `warn_slippery_road`: Strada sdrucciolevole
- `warn_hump`: Dosso
- `warn_traffic_light`: Semaforo
- `warn_tram`: Tram
- `warn_two_way_traffic`: Doppio senso di circolazione

- `warn_wild_animals`: Pericolo animali selvatici
- `mand_bike_lane`: Pista ciclabile
- `mand_go_left`: Obbligo di svoltare a sinistra
- `mand_go_left_right`: Preavviso di direzione obbligatoria sinistra o destra
- `mand_go_right`: Obbligo di svoltare a destra
- `mand_go_straight`: Obbligo di andare dritto
- `mand_go_straight_left`: Direzioni consentite dritto e sinistra
- `mand_go_straight_right`: Direzioni consentite dritto e destra
- `mand_pass_left`: Passaggio obbligatorio a sinistra
- `mand_pass_left_right`: Passaggio consentito a sinistra o destra
- `mand_pass_right`: Passaggio obbligatorio a destra
- `mand_roundabout`: Intersezione con circolazione rotatoria (obbligo)

Appendice B

Dettagli Ambiente di Sviluppo e Codice

In questa appendice vengono riportati i dettagli tecnici relativi alla configurazione dell'ambiente di sviluppo e agli script utilizzati per l'addestramento dei modelli. Il codice completo, incluse le celle di esecuzione e i log dettagliati, è consultabile anche nei notebook originali denominati `streetsignsense-yolo12n`, `streetsignsense-yolo12s` e `streetsignsense-yolo12m`, presenti nella cartella principale del progetto.

B.1 Installazione dipendenze

L'installazione delle dipendenze viene eseguita all'avvio nel seguente modo:

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

B.2 Verifica dell'Ambiente

Prima di avviare qualsiasi operazione, è buona norma verificare la corretta configurazione dell'ambiente e la disponibilità delle risorse hardware (GPU). Il seguente snippet utilizza le API di Ultralytics per un controllo:

```
import ultralytics
ultralytics.checks()
from ultralytics import YOLO
import yaml
```

B.3 Generazione della Configurazione del Dataset

Per garantire la riproducibilità e l'indipendenza dai percorsi locali, il file di configurazione `data.yaml` è stato generato programmaticamente all'interno del notebook. Questo script definisce i percorsi assoluti per i set di training, validation e test, e mappa gli indici numerici ai nomi delle 63 classi.

```
dataset_config = {
    'train': '/StreetSignSet/train/images',
    'val': '/StreetSignSet/valid/images',
    'test': '/StreetSignSet/test/images',
    'nc': 63,
    'names': ['prio_give_way', 'prio_stop', 'prio_priority_road', '
   forb_speed_over_5', 'forb_speed_over_10', 'forb_speed_over_20',
   'forb_speed_over_30', 'forb_speed_over_40', 'forb_speed_over_50'
   , 'forb_speed_over_60', 'forb_speed_over_70', '
   forb_speed_over_80', 'forb_speed_over_90', 'forb_speed_over_100'
   , 'forb_speed_over_110', 'forb_speed_over_120', '
   forb_speed_over_130', 'forb_no_entry', 'forb_no_parking', '
   forb_no_stopping', 'forb_overtake_car', 'forb_overtake_trucks',
   'forb_trucks', 'forb_turn_left', 'forb_turn_right', '
   forb_weight_over_3.5t', 'forb_weight_over_7.5t', 'forb_u_turn',
   'info_bus_station', 'info_crosswalk', 'info_highway', '
   info_one_way', 'info_parking', 'info_taxi_parking', '
   warn_children', 'warn_construction', 'warn_crosswalk', '
   warn_cyclists', 'warn_left_curve', 'warn_right_curve', '
   warn_domestic_animals', 'warn_other_dangers', '
   warn_poor_road_surface', 'warn_roundabout', '
   warn_sharp_left_curve', 'warn_sharp_right_curve', '
   warn_slippery_road', 'warn_hump', 'warn_traffic_light', '
   warn_tram', 'warn_two_way_traffic', 'warn_wild_animals', '
   mand_bike_lane', 'mand_go_left', 'mand_go_left_right', '
   mand_go_right', 'mand_go_straight', 'mand_go_straight_left', '
   mand_go_straight_right', 'mand_pass_left', 'mand_pass_left_right
   ', 'mand_pass_right', 'mand_roundabout']
}

with open('data.yaml', 'w') as f:
    yaml.dump(dataset_config, f)

!cat data.yaml
```

B.4 Script di Addestramento

Di seguito sono riportati gli script completi utilizzati per avviare il training delle tre varianti. Ogni script specifica gli iperparametri discussi nel Capitolo 3. Si noti che il codice è replicato nei rispettivi notebook di progetto per garantire la tracciabilità di ogni esperimento.

```
model = YOLO('yolo12n.pt')

results = model.train(
    data='data.yaml',
    imgsz=640,
    epochs=600,
    patience=100,
    save_period=50,
    device='0,1',
    batch=140,
    fliplr=0.0,
    name='yolo12n_run',
    project='streetsignsense'
)
```

```
model = YOLO('yolo12s.pt')

results = model.train(
    data='data.yaml',
    imgsz=640,
    epochs=600,
    patience=100,
    save_period=50,
    name='yolo12s_run',
    project='streetsignsense',
    device='0,1',
    batch=80,
    fliplr=0.0
)
```

```
model = YOLO('yolo12m.pt')

results = model.train(
    data='data.yaml',
    imgsz=640,
    epochs=600,
    patience=100,
    save_period=50,
    name='yolo12m_run',
    project='streetsignsense',
    device='0,1',
    batch=40,
    fliplr=0.0
)
```

B.5 Script di Validazione

Lo script di validazione utilizzato per valutare le prestazioni del modello sul set di validazione (`valid`) e calcola le metriche riportate nel Capitolo 4.

```
    from ultralytics import YOLO
    modelN = YOLO("/streetsignsense/yolo12n_run/weights/best.pt")
    resultN = modelN.val(data="data.yaml", device='0,1')
    print("\n\nmAP50-95:",resultN.box.map)  # map50-95
    print("mAP-50:",resultN.box.map50)  # map50
    print("mAP-75:",resultN.box.map75)  # map75
    print("Average Precision: ", np.mean(resultN.box.p))  #
precision
    print("Average Recall: ",np.mean(resultN.box.r))  # recall
    print("Average F1: ",np.mean(resultN.box.f1)) # f1score
```

```
    from ultralytics import YOLO
    modelS = YOLO("/streetsignsense/yolo12s_run/weights/best.pt")
    resultS = modelS.val(data="data.yaml", device='0,1')
    print("\n\nmAP50-95:",resultS.box.map)  # map50-95
    print("mAP-50:",resultS.box.map50)  # map50
    print("mAP-75:",resultS.box.map75)  # map75
    print("Average Precision: ", np.mean(resultS.box.p))  #
precision
    print("Average Recall: ",np.mean(resultS.box.r))  # recall
    print("Average F1: ",np.mean(resultS.box.f1)) #f1score
```

```
    from ultralytics import YOLO
    modelM = YOLO("/streetsignsense/yolo12m_run/weights/best.pt")
    resultM = modelM.val(data="data.yaml", device='0,1')
    print("\n\nmAP50-95:",resultM.box.map)  # map50-95
    print("mAP-50:",resultM.box.map50)  # map50
    print("mAP-75:",resultM.box.map75)  # map75
    print("Average Precision: ", np.mean(resultM.box.p))  #
precision
    print("Average Recall: ",np.mean(resultM.box.r))  # recall
    print("Average F1: ",np.mean(resultM.box.f1)) #f1score
```

B.6 Script di Test

La valutazione finale delle performance è stata condotta utilizzando uno script dedicato per l'inferenza sul test set (`test`).

```
    ...

    folder = "/StreetSignSet/test/images"

    #model = YOLO("/streetsignsense/yolov12n_run/weights/best.pt")
    model = YOLO("/streetsignsense/yolo12s_run/weights/best.pt")
    #model = YOLO("/streetsignsense/yolo12m_run/weights/best.pt")

    image_files = sorted([f for f in os.listdir(folder) if f.
endswith(('.jpg', '.png', '.jpeg'))])

    for filename in image_files:
        image_path = os.path.join(folder, filename)
        print(f"\nRisultati per: {filename} ")

        results = model(image_path, device='0,1')

        # Immagine con Predizioni
        prediction_image = results[0].plot()

        ...

        cv2_imshow(prediction_image)
```

B.7 Script per l'Analisi delle Metriche

Per mantenere pulita la pipeline di addestramento, l'elaborazione avanzata dei dati e la generazione dei grafici comparativi sono state delegate a un notebook dedicato. Si rimanda al notebook `yolo12modelanalysis` (presente nella cartella del progetto) per la consultazione degli script specifici utilizzati per:

- **Calcolo e Visualizzazione mAP:** Script per la rappresentazione dei valori di mAP@50, mAP@50-95 e mAP@75 e la generazione delle curve di confronto.
- **Analisi Precision, Recall e F1-Score:** Codice per il calcolo e la visualizzazione grafica dell'andamento di Precisione, Recall e F1 score per tutte le varianti del modello.
- **Confronto Training e Validation Loss:** Script di confronto e la generazione delle curve di loss per tutte le varianti.

Appendice C

Metriche

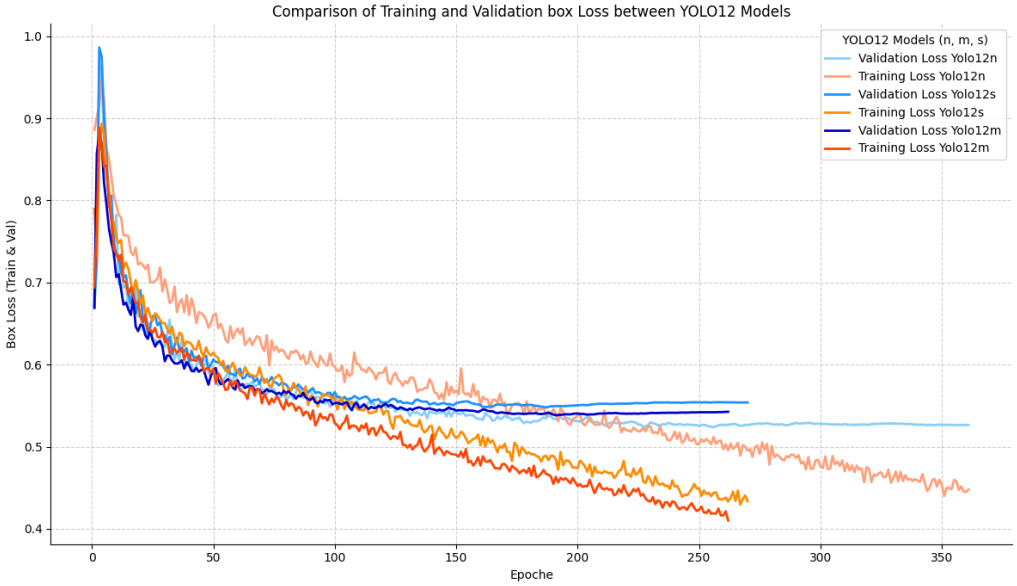


Figura C.1: Confronto training and validation box loss tra le tre varianti

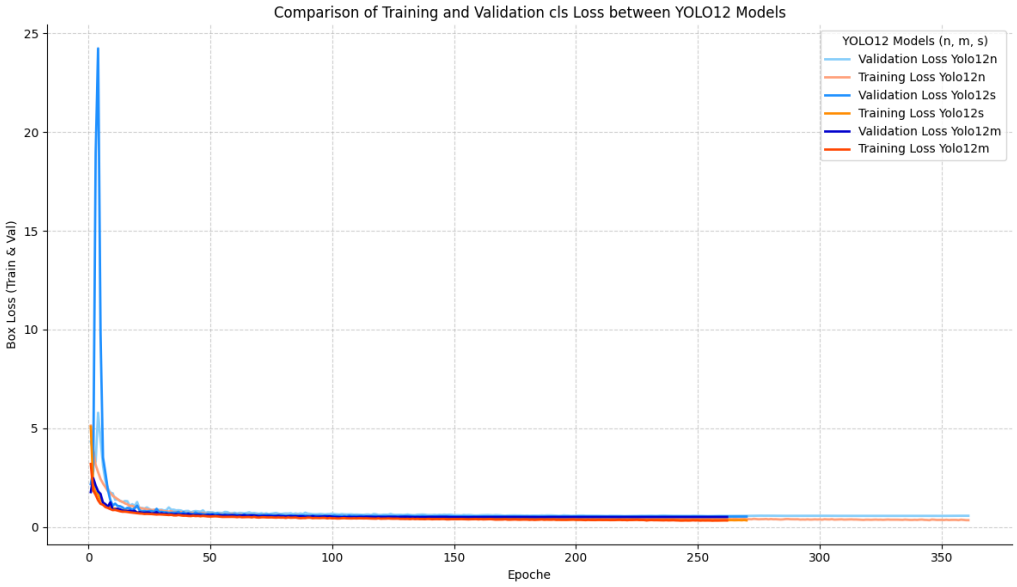


Figura C.2: Confronto training and validation cls loss tra le tre varianti

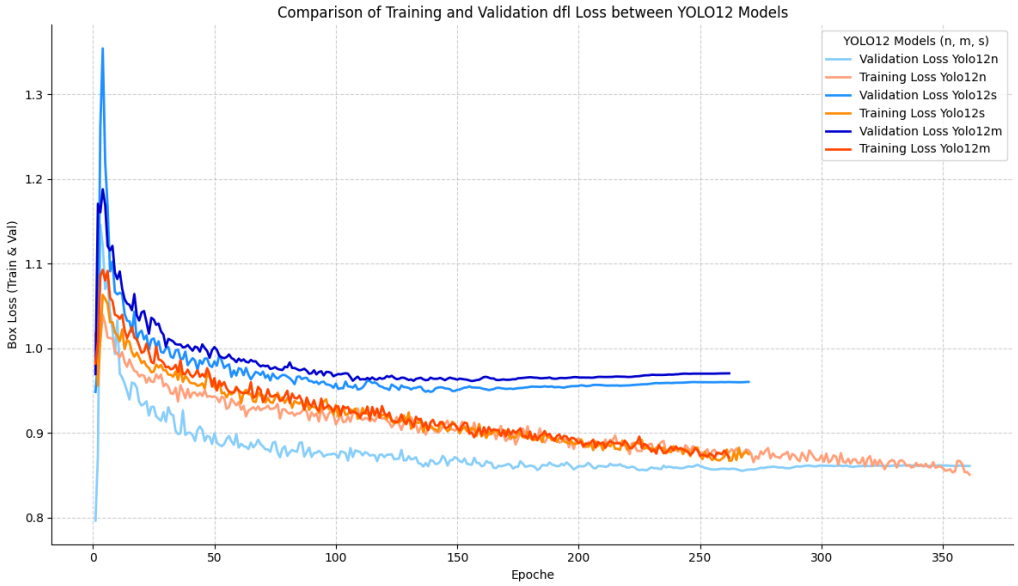
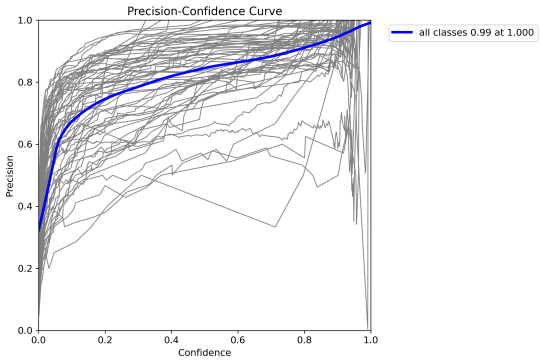
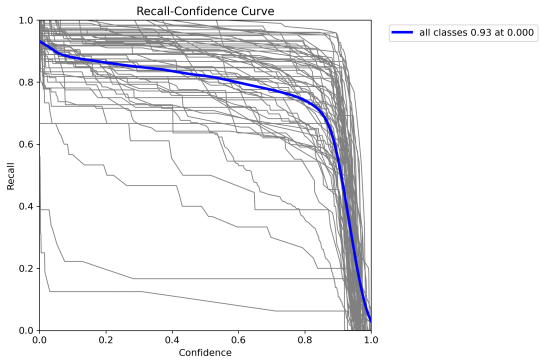


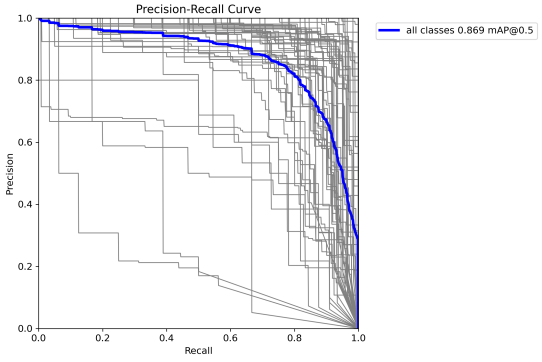
Figura C.3: Confronto training and validation dfl loss tra le tre varianti



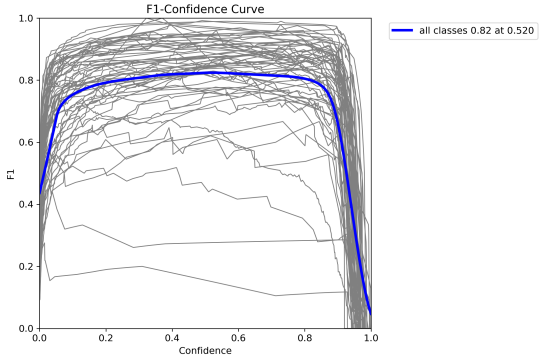
(a) Curva Precision



(b) Curva Recall

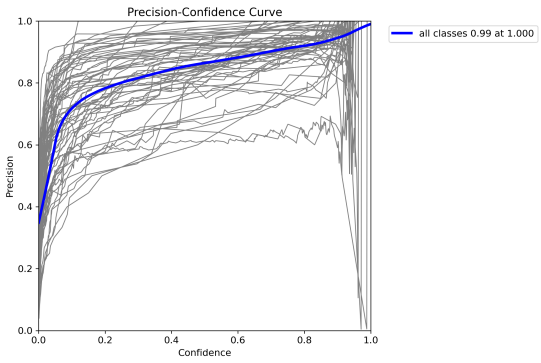


(c) Curva Precision-Recall

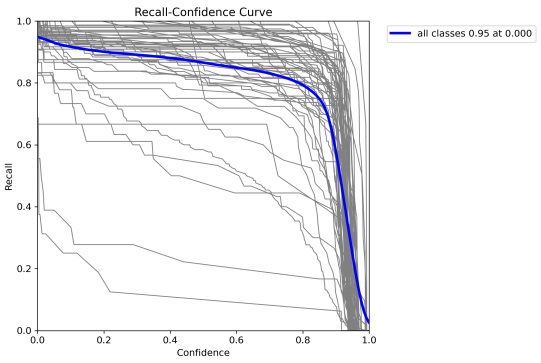


(d) Curva F1-Score

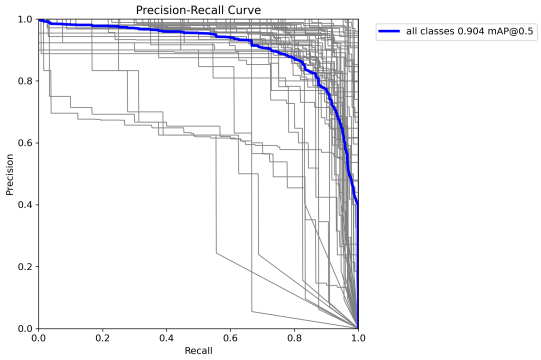
Figura C.4: Curve delle metriche di valutazione per il modello YOLO12 Nano.



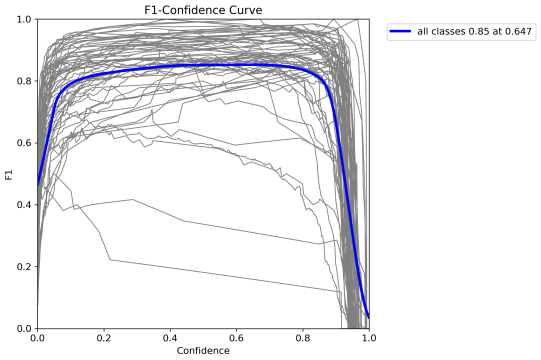
(a) Curva Precision



(b) Curva Recall

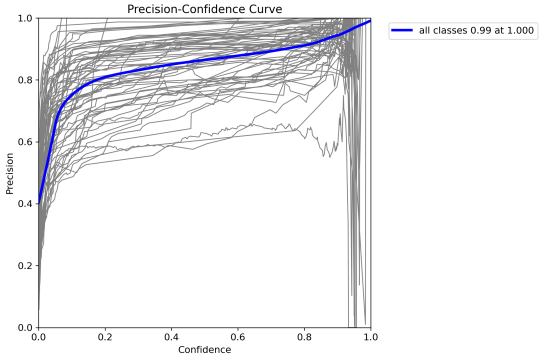


(c) Curva Precision-Recall

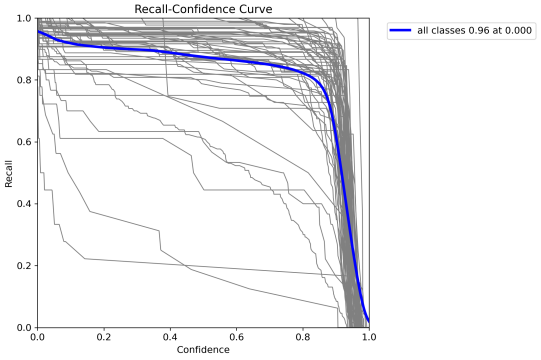


(d) Curva F1-Score

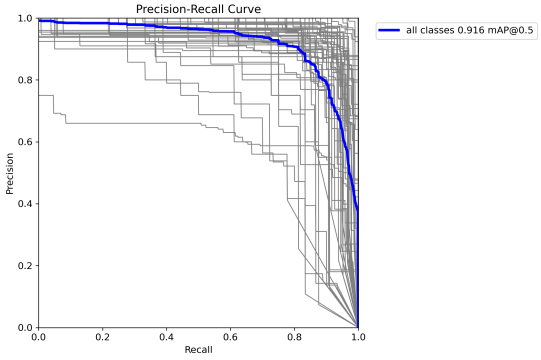
Figura C.5: Curve delle metriche di valutazione per il modello YOLO12 Small.



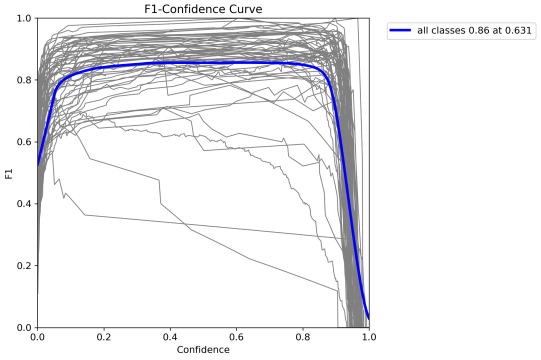
(a) Curva Precision



(b) Curva Recall



(c) Curva Precision-Recall



(d) Curva F1-Score

Figura C.6: Curve delle metriche di valutazione per il modello YOLOv12 Medium.