



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

Denise Cilia
Eleonora Giuffrida
Valeria Platania

*PedestriansDET: Sistema di Detection e Conteggio
dei pedoni*

RELAZIONE PROGETTO MACHINE
LEARNING

*Prof. Giovanni Maria Farinella
Dott. Rosario Leonardi*

Anno Accademico 2024 - 2025

Indice

1	Introduzione	3
2	Problema: Object Detection	4
2.1	Descrizione	4
2.2	Applicazioni pratiche	4
3	Reti Neurali Convoluzionali (CNN)	6
3.1	YOLO - "You Only Look Once"	6
3.1.1	YOLOv8 Medium	7
3.1.2	YOLOv10 Nano	8
3.1.3	YOLOv11 Small	8
3.2	Faster R-CNN: Two-Stage Approach	9
3.2.1	ResNet50	10
4	Dataset	11
4.1	ND4ML (New Dataset For Machine Learning Project)	12
4.2	Dataset acquisito per la fase di test	15
4.2.1	Dati di esempio del test set	16
4.3	Pre-processing dei dataset	18
5	Metodi	22
5.1	Funzionamento YOLO	22
5.1.1	Residual block	22
5.1.2	Bounding box regression	23
5.1.3	Intersection Over the Union	24
5.1.4	Non-Maximum Suppression	25
5.2	Funzionamento Faster R-CNN	26
5.2.1	Feature Extraction	27
5.2.2	Region Proposal Network - RPN	27
5.2.3	Classificazione e Refinement (RoI Pooling)	28
6	Valutazione del modello	29
6.1	Metriche di valutazione	29
6.2	Grafici	31
6.3	Loss functions	33

7 Fase di Training e Validazione	34
7.1 YOLO	34
7.1.1 Parametri di training	34
7.1.2 Organizzazione della directory di training	35
7.1.3 Fase di training	36
7.1.4 Fase di validazione	38
7.2 Faster R-CNN con Resnet50	40
7.2.1 Parametri di training	40
7.2.2 Organizzazione della directory per il training	41
7.2.3 Fase di training	41
7.2.4 Fase di validazione	42
7.3 Demo e funzionamento	45
8 Risultati	47
8.1 Performance dei modelli YOLO	48
8.1.1 Grafici dei risultati sul test set	49
8.1.2 Analisi dei risultati	53
8.2 Performance del modello Faster R-CNN (Resnet50)	54
8.2.1 Grafici dei risultati sul train/val set	55
8.2.2 Analisi dei risultati	55
8.3 Link ai video del test set con annotazioni e ground truth	56
8.4 Considerazioni complessive e impatto dei fattori esterni	56
9 Conclusione	57
9.1 Lezioni apprese	58
9.2 Possibili miglioramenti	59

1 Introduzione

Nel panorama in continua evoluzione dell'intelligenza artificiale, il Machine Learning si afferma come una disciplina fondamentale, con applicazioni che trovano spazio in numerosi settori. Tra questi, la Computer Vision riveste un ruolo di primaria importanza, e al suo interno, l'Object Detection si distingue come una delle sfide più rilevanti e complesse.

Il presente lavoro propone di esplorare e documentare le fasi di sviluppo di un sistema incentrato sul riconoscimento e la localizzazione di esseri umani all'interno di sequenze video e fotogrammi. Il task verrà affrontato sfruttando la comprovata efficacia degli algoritmi della famiglia YOLO (You Only Look Once). Questi modelli si sono affermati per la loro notevole capacità nell'eseguire la rilevazione degli oggetti in tempo reale, caratteristica cruciale per applicazioni che richiedono risposte immediate.

In particolare, verranno impiegati e valutati alcuni dei modelli più recenti della serie YOLO, tra cui YOLOv8m, YOLOv10n e YOLOv11s, al fine di identificare la soluzione più idonea in termini di *trade-off* tra accuratezza e performance computazionali per il task specifico. Ai precedenti menzionati seguirà l'uso di Faster RCNN con backbone ResNet50. Le sezioni successive della relazione illustreranno in dettaglio le metodologie adottate, la preparazione del dataset, le fasi di addestramento e le metriche di valutazione impiegate per analizzare le prestazioni dei modelli su un dataset acquisito manualmente.

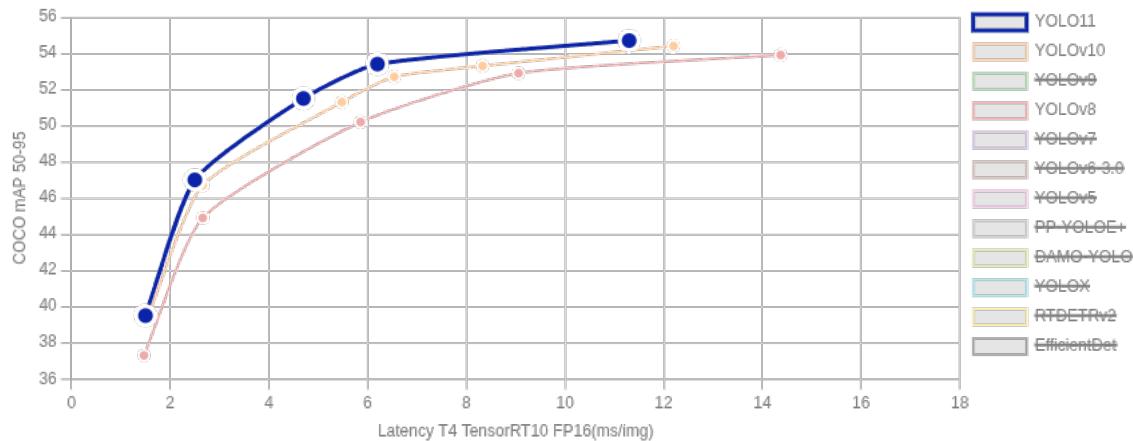


Figure 1: Confronto delle misure di performance dei modelli YOLO

2 Problema: Object Detection

2.1 Descrizione

L'Object Detection identifica con precisione la posizione di ogni singolo oggetto di interesse presente in un'immagine e lo classifica assegnando una classe di appartenenza. Si prenda come esempio pratico il voler individuare tutte le persone presenti in un filmato: l'Object Detection non solo dirà che sono presenti delle persone, ma disegnerà un riquadro, la cosiddetta *bounding box*, attorno a ciascuna di esse, fornendo le loro coordinate esatte. Dunque, un algoritmo di Object Detection può essere visto come l'unione di due componenti fondamentali:

- Un algoritmo di classificazione: il suo compito è determinare a quale categoria appartiene l'oggetto individuato. Nel nostro caso, si tratterà di stabilire se un determinato elemento è un essere umano.
- Un algoritmo di regressione: questo componente si occupa di definire con esattezza la posizione e le dimensioni della *bounding box*, ovvero il riquadro che racchiude l'oggetto. È grazie a questa parte che possiamo localizzare precisamente ogni individuo all'interno di un frame.

2.2 Applicazioni pratiche

L'importanza di questa task si evince in una moltitudine di contesti pratici. Si prenda d'esempio la sicurezza e la sorveglianza, dove sistemi intelligenti possono monitorare aree, rilevare presenze non autorizzate o analizzare comportamenti sospetti, come nel caso di presenza di intrusi in aree riservate o con accesso vietato. Oppure ancora, nel settore della robotica e automazione nel quale i robot possono essere dotati di un sistema visivo per interagire con l'ambiente circostante, riconoscendo ostacoli o la presenza e la posizione di operatori umani per garantire la sicurezza e facilitare la collaborazione. Anche nel campo dell'analisi del comportamento umano, l'Object Detection di persone in video permette di studiare le interazioni sociali.

Lo studio, focalizzato sul riconoscimento di esseri umani in sequenze video e fotogrammi, si inserisce pienamente nel contesto di *pedestrian detection* che trova estensione anche nel campo dell'**ADAS** (Advanced driver-assistance system) per il supporto ai sistemi di sicurezza delle macchine con guida autonoma.

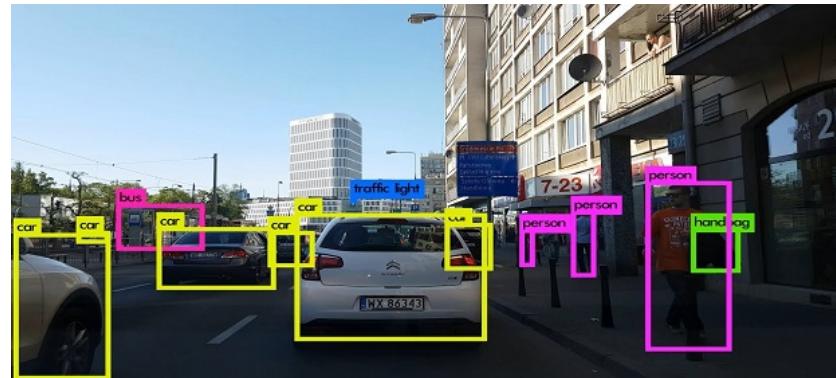


Figure 2: Object Detection applicata alla guida autonoma



Figure 3: Object Detection applicata al riconoscimento di pedoni

3 Reti Neurali Convoluzionali (CNN)

E' essenziale comprendere le fondamenta su cui poggiano gran parte degli algoritmi di Computer Vision moderni: le Reti Neurali Convoluzionali (CNN). Le CNN agiscono come degli "occhi" artificiali estremamente sofisticati, capaci di imparare a riconoscere pattern e caratteristiche visive direttamente dai dati. A differenza delle reti neurali tradizionali, le CNN sono state progettate specificamente per elaborare dati strutturati come le immagini, sfruttando strati convoluzionali che applicano filtri per estrarre caratteristiche gerarchiche, dai bordi a texture più semplici fino a forme e oggetti complessi. Mentre YOLO ha rivoluzionato il campo con il suo approccio *single-stage*, è importante riconoscere che esistono anche architetture *two-stage* che hanno contribuito enormemente allo sviluppo dell'Object Detection, offrendo spesso un'accuratezza superiore, seppur a discapito dei tempi di elaborazione. Tra queste, Faster R-CNN si è affermata come una pietra miliare, rappresentando un'evoluzione significativa rispetto ai suoi predecessori (R-CNN e Fast R-CNN). Segue la trattazione nel dettaglio di entrambi gli approcci.

3.1 YOLO - "You Only Look Once"

Nel campo dell'Object Detection, l'introduzione di YOLO[6] (You Only Look Once) ha segnato una vera e propria svolta, ridefinendo gli standard di velocità ed efficienza. A differenza degli approcci precedenti, che spesso prevedevano più passaggi per identificare e classificare gli oggetti, YOLO ha introdotto il paradigma *single-stage*. Questo significa che il modello è in grado di predire contemporaneamente sia le posizioni degli oggetti (precisamente le bounding box) sia le loro classi di appartenenza, il tutto in un unico passo. Questa caratteristica intrinseca ha permesso a YOLO di eccellere nella detection.

Dalla sua prima apparizione nel 2015, la filosofia di YOLO ha dato vita a una serie di evoluzioni, ognuna delle quali ha introdotto miglioramenti significativi in termini di accuratezza, velocità e robustezza. Nello studio vengono confrontate alcune delle versioni più recenti e performanti di questa famiglia, ognuna con le sue peculiarità e ottimizzazioni.

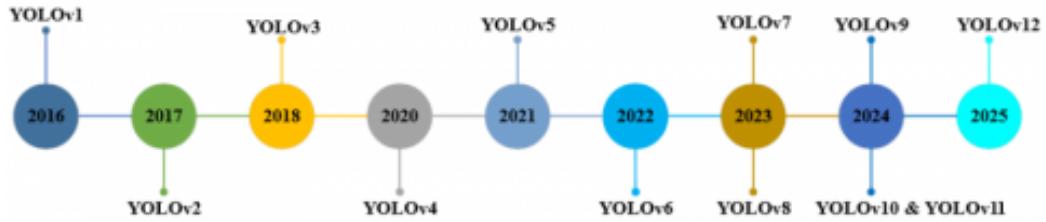


Figure 4: Release timeline dei modelli YOLO

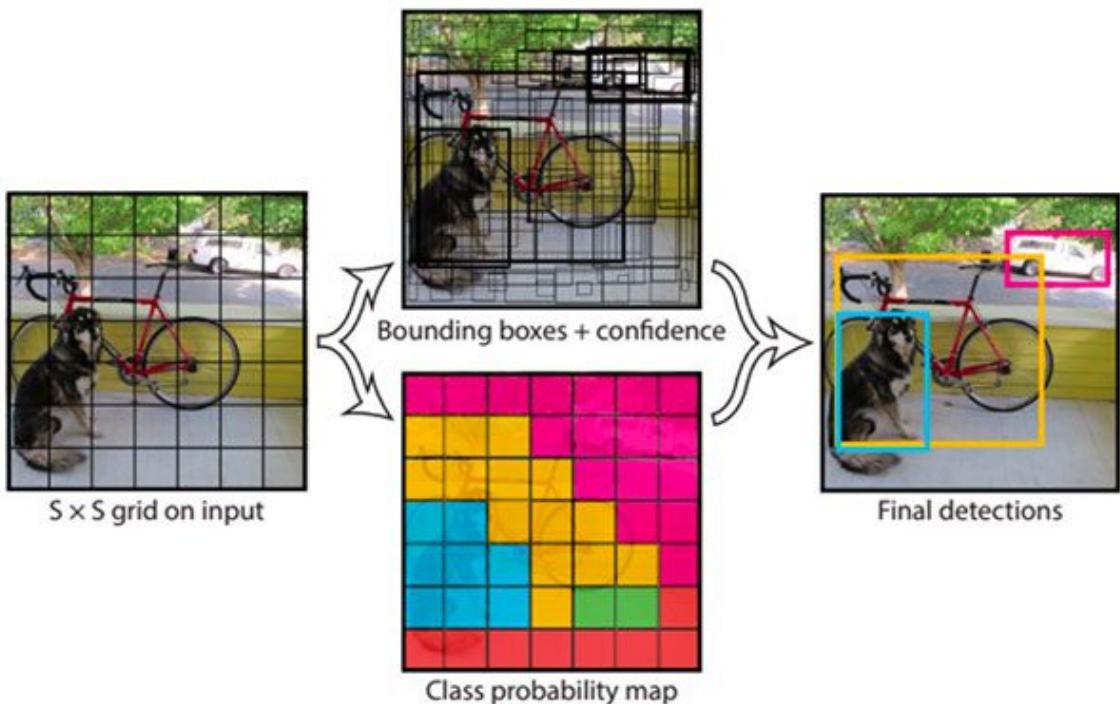


Figure 5: Funzionamento single-stage in YOLO

3.1.1 YOLOv8 Medium

Basandosi sulla filosofia "You Only Look Once" (YOLO), che processa l'intera immagine in un'unica passata per predire contemporaneamente posizioni dei *bounding box* e classi di appartenenza, il modello porta avanti questa tradizione con significative innovazioni.

Questa implementazione si distingue per una serie di miglioramenti architettonici e funzionali che ne aumentano l'efficienza e l'accuratezza. Presenta un'architettura *Anchor-Free*: anziché la tradizionale detection basata sulle anchor, YOLOv8 utilizza un approccio che permette di rendere la fase di *training* ancora più semplice e aiuta il modello a lavorare bene su dataset diversi. È stato progettato per essere un modello versatile, capace di gestire non solo l'Object Detection, ma anche compiti correlati come l'*Image Segmentation* e la *Pose Estimation*, rendendolo una soluzione più completa per diverse applicazioni di Computer Vision. L'approccio di questa versione si concentra sull'offrire un framework robusto e facile da usare, che permette agli sviluppatori di implementare rapidamente soluzioni di rilevamento oggetti ad alte prestazioni.

3.1.2 YOLOv10 Nano

Con l'arrivo di YOLOv10, i ricercatori della Tsinghua University hanno introdotto un approccio innovativo, mirato a superare alcune delle limitazioni delle versioni precedenti, in particolare nella post-elaborazione e nell'architettura del modello. La versione *nano* (YOLOv10n) è stata progettata per offrire prestazioni elevate anche in ambienti con risorse computazionali limitate, pur mantenendo un'ottima accuratezza. Un aspetto distintivo di YOLOv10 è l'eliminazione del meccanismo di Non-Maximum Suppression (**NMS**) durante la fase di training, un processo tradizionalmente utilizzato per rimuovere le predizioni ridondanti. Questa innovazione è resa possibile dalla strategia chiamata Dual Label Assignment with NMS-free training. In sintesi, YOLOv10 combina un approccio *one-to-one matching*, ovvero assegna una singola predizione a ogni oggetto, eliminando la necessità di NMS con la più tradizionale *one-to-many assignment*. Quest'ultima garantisce performance migliori ma richiede NMS. L'introduzione di un *one-to-one head* aggiuntivo, ottimizzato congiuntamente al *one-to-many head* tramite una *Consistent Matching Metric*, permette di ottenere i benefici di entrambi gli approcci, migliorando l'accuratezza e la convergenza.

3.1.3 YOLOv11 Small

Il modello si concentra sull'affinare l'equilibrio tra velocità e accuratezza, un aspetto cruciale per le applicazioni pratiche. Tra le sue innovazioni, spicca l'aumento di accuratezza rispetto ai modelli precedenti. YOLOv11[5] utilizza

loss functions più complesse, migliorando le *anchor boxes* e evidenziando la tecnica di *data augmentation*. Con quest'ultime, il modello riesce ad individuare gli oggetti anche in contesti non semplici. YOLOv11 migliora anche l'uso delle risorse hardware, riducendo sensibilmente i tempi di training e costi computazionali.

3.2 Faster R-CNN: Two-Stage Approach

L'approccio *two-stage* nell'Object Detection si distingue per la sua metodologia di rilevamento degli oggetti, suddivisa in due fasi principali e sequenziali. Nella prima fase, il sistema si concentra sull'identificazione delle aree dell'immagine che più probabilmente contengono un oggetto, generando un insieme di "proposte di regione". La seconda fase prende queste proposte e le analizza in dettaglio, classificando l'oggetto all'interno di ciascuna regione e affinando la posizione del suo bounding box. Questo processo in due passaggi, pur introducendo una maggiore complessità computazionale rispetto ai metodi *single-stage*, consente di raggiungere un'accuratezza superiore, rendendo questi modelli ideali per applicazioni dove la precisione è la priorità assoluta.

Questo approccio affonda le sue radici nella ricerca sull'Object Detection, nascendo dalla necessità di superare le inefficienze dei metodi precedenti. Inizialmente, i sistemi si affidavano a tecniche di *sliding window*, che scansionavano l'intera immagine con finestre di diverse dimensioni e proporzioni per classificare il contenuto di ogni porzione. Questo processo, tuttavia, si rivelava computazionalmente proibitivo e inefficiente.

La svolta si è avuta con l'introduzione delle architetture basate su Region-based Convolutional Neural Networks (R-CNN). L'idea innovativa consisteva nel generare un numero limitato di proposte di regione e concentrare la classificazione e la regressione dei bounding box solo su queste. Questo ha ridotto drasticamente il carico computazionale. Le successive evoluzioni, come Fast R-CNN, hanno ulteriormente ottimizzato questo processo, ma persistevano ancora colli di bottiglia legati alla generazione separata delle proposte e alla loro elaborazione.

È in questo contesto di continua ricerca di efficienza e precisione che Faster R-CNN[4] si è affermata come una pietra miliare. Questa architettura ha perfezionato l'approccio *two-stage* integrando la generazione delle proposte di regione direttamente all'interno della rete neurale, attraverso il Region Proposal Network (RPN). Ciò ha reso l'intero processo *end-to-end* e signifi-

ficativamente più efficiente. Il funzionamento dettagliato di questo approccio, in particolare di Faster R-CNN, verrà approfondito nel Capitolo 5.

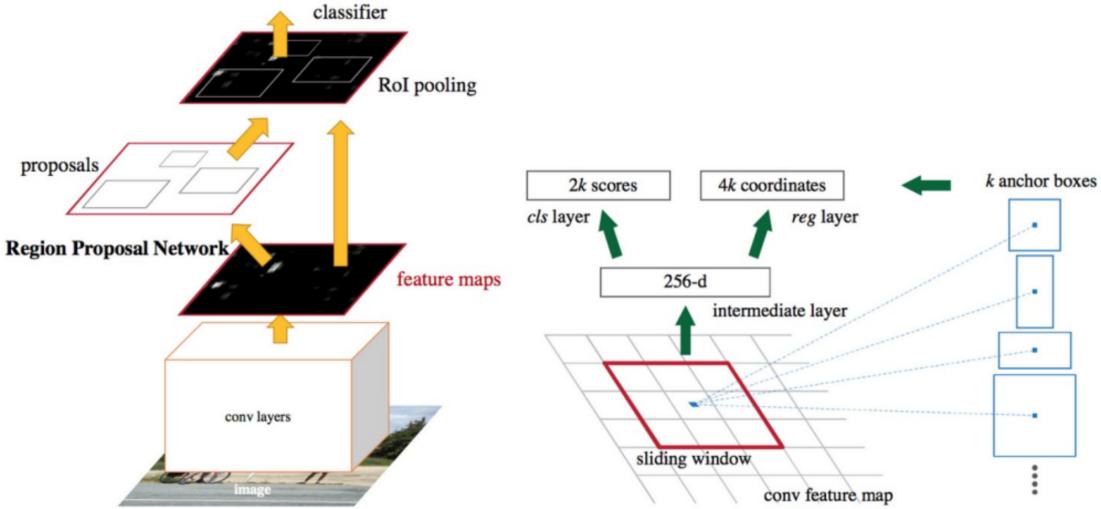


Figure 6: Architettura Faster RCNN

3.2.1 ResNet50

ResNet50[3] è una rete neurale convoluzionale profonda composta da 50 strati, appartenente alla famiglia delle *Residual Networks*, introdotte da He et al. nel 2015. La sua architettura è progettata per affrontare il problema della degradazione delle prestazioni nelle reti molto profonde, grazie all'introduzione delle **connessioni residue** (*skip connections*). Queste connessioni permettono di far fluire l'informazione direttamente da uno strato all'altro, saltando uno o più layer intermedi, favorendo così un addestramento più stabile e una convergenza più rapida.

La struttura di ResNet50 si basa su una sequenza iniziale di convoluzione 7×7 con 64 filtri e *stride* pari a 2, seguita da un livello di *Max Pooling* 3×3 . L'architettura è quindi suddivisa in quattro blocchi principali, ciascuno composto da più unità residuali. Ogni unità è implementata tramite uno schema a **bottleneck**, con tre convoluzioni in sequenza: una 1×1 per la riduzione della dimensionalità, una 3×3 per l'estrazione delle caratteristiche spaziali, e una 1×1 per l'espansione dei canali. Questo design consente di mantenere un buon compromesso tra efficienza computazionale e capacità espressiva.

I blocchi successivi aumentano progressivamente il numero di canali: 256, 512, 1024 e 2048, mentre la risoluzione spaziale viene ridotta tramite *stride* pari a 2 nelle prime convoluzioni di ciascun blocco (eccetto il primo). Tutte le unità bottleneck sono collegate alle loro rispettive *skip connections*, implementate tramite addizione element-wise, che permettono un flusso diretto del gradiente anche nei livelli più profondi.

Per via della sua architettura bilanciata, ResNet50 rappresenta una scelta solida come **backbone** nei modelli di *object detection* come Faster R-CNN, in quanto fornisce feature map ricche e gerarchiche senza un costo computazionale eccessivo. In questo progetto, è stata adottata per la sua comprovata efficacia nel riconoscimento visivo e per la sua ampia disponibilità nei principali framework di deep learning.

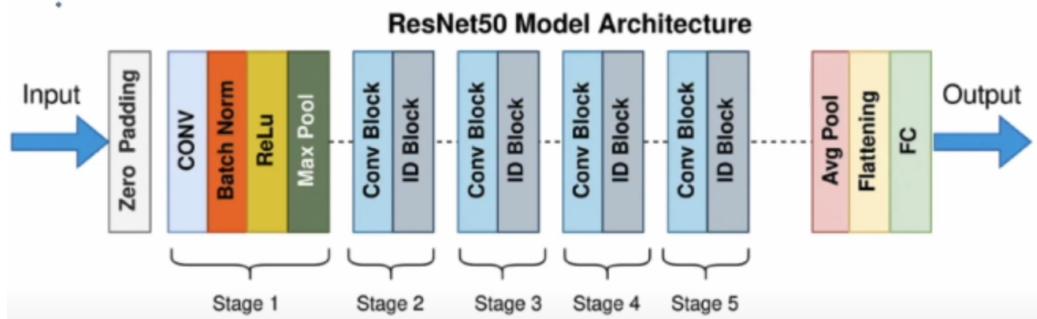


Figure 7: Architettura ResNet50

4 Dataset

Per la corretta esecuzione dello studio sono stati previsti **due diversi** dataset elaborati manualmente per adattarli meglio al progetto. In ordine:

- Il primo dataset, chiamato **New Dataset For Machine Learning Project**, è stato utilizzato per la fase di **training** dei diversi modelli.
- Il secondo dataset è stato acquisito personalmente, ed è stato utilizzato per la fase di **testing** dei modelli presentati nello studio.

4.1 ND4ML (New Dataset For Machine Learning Project)

Il dataset usato per la fase di training dei modelli è il dataset **New Dataset For Machine Learning Project** [2]. Questo dataset è la combinazione di quattro dataset che contengono frame estratti da brevi video, incentrati sul rilevamento di pedoni, e persone più in generale. Si tratta di dataset accessibili pubblicamente sulla piattaforma **Roboflow**, che mette a disposizione diversi tools per poter annotare frame attraverso interfaccia grafica, e non solo. I dataset, in ordine, possono essere trovati con i seguenti nomi: **MOT17-03-DPM, Human Detection, People Detection** ed infine **GTA_dataset**. L'ultimo dataset è stato personalmente acquisito mediante registrazione schermo sul videogioco **GTA V** e successivamente annotato manualmente mediante Roboflow. Si tratta di un dataset sintetico che prende fortemente ispirazione dall'articolo **MOTSynth**[1]; l'idea è quella di aumentare ulteriormente le dimensioni del dataset finale in modo da evitare l'overfitting dei modelli, fornendo anche un'ampia diversificazione nei frame che verranno utilizzati per l'allenamento. Difatti, i frame presenti nel dataset finale sono tutti in diverse condizioni di illuminazione, posa, e orientamento.

Nel dataset sono presenti un totale di **2985** immagini di cui **2015** immagini vengono utilizzate per il training, **531** per la fase di validation e **439** per la fase di testing. Tuttavia, sebbene sia presente lo split per il test la fase di testing verrà comunque effettuata su un altro dataset personalmente acquisito. Tutte le immagini sono state salvate in formato .jpg e hanno una dimensione variabile, generalmente **1920x1080**. Inoltre sono presenti diverse classi: 0, pedestrian, 1, 2, 3, 4, 5, 6, 7. Nella sottosezione pre-processing verrà approfondito come queste classi sono state riformulate per poterle meglio adattare allo studio. In particolare, per ogni annotazione verrà generato un corrispondente file nel formato desiderato (ad esempio YOLO oppure COCO JSON) che indicherà come nel caso di YOLO la bounding box:

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

Di seguito è possibile visualizzare degli esempi di frame del dataset utilizzato.



Figure 8: Frame sintetico del dataset senza ground-truth



Figure 9: Frame sintetico del dataset con ground-truth



Figure 10: Frame del dataset senza ground-truth



Figure 11: Frame del dataset con ground-truth

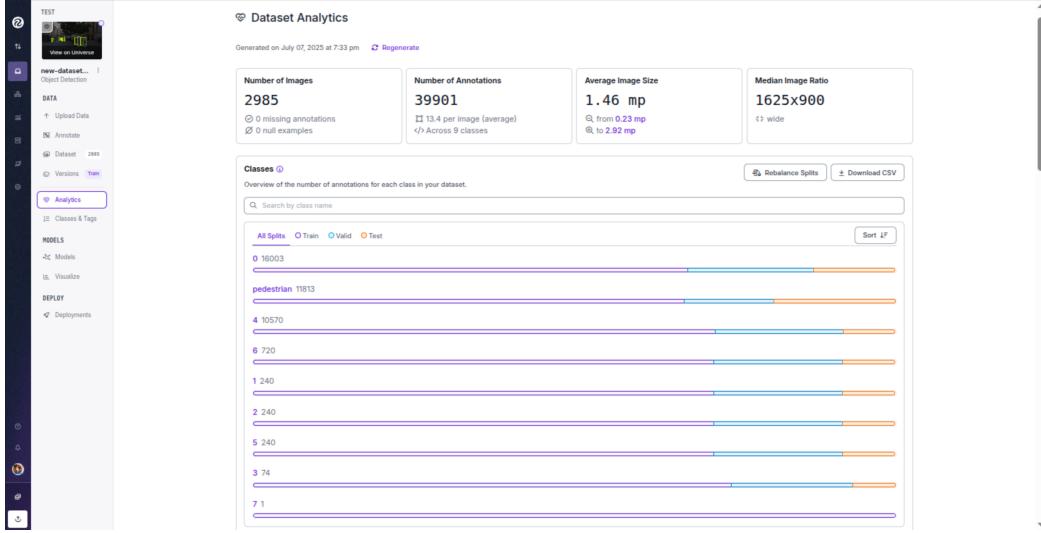


Figure 12: Statistiche sul dataset calcolate su Roboflow

4.2 Dataset acquisito per la fase di test

Allo scopo di effettuare la fase di testing del modello su dati mai visti prima, è stato realizzato un nuovo dataset formato da tre brevi video successivamente frammentati in singole immagini. Il dataset è stato acquisito nel centro di **Catania**. Contiene al suo interno immagini acquisite attraverso un dispositivo **iPhone 15** a risoluzione originale **1920x1080** camminando per le strade del centro catanese. I video sono stati acquisiti con diverse condizioni di illuminazione e cercando di ottenere diverse angolazioni, pur sempre simili alle angolazioni presenti invece nel dataset usato per il training.

I frame ottenuti dai video sono stati annotati attraverso **Roboflow**.

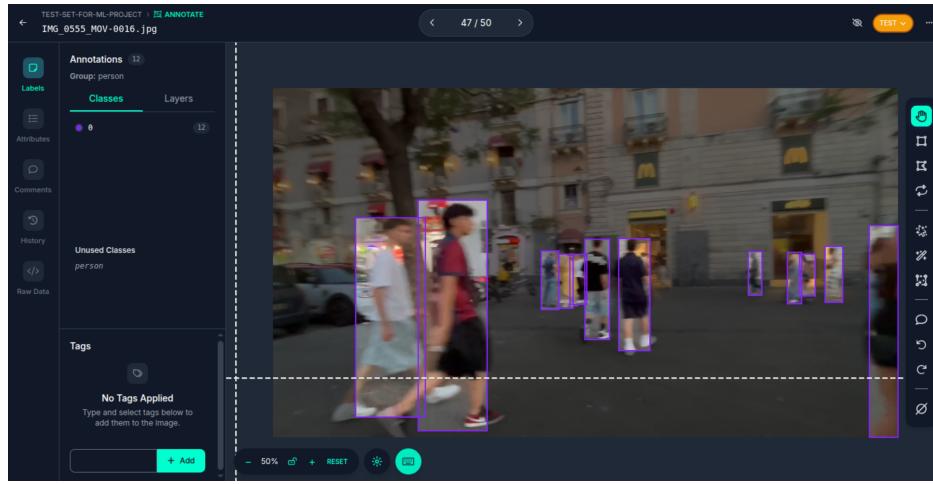


Figure 13: Esempio di labelling con Roboflow

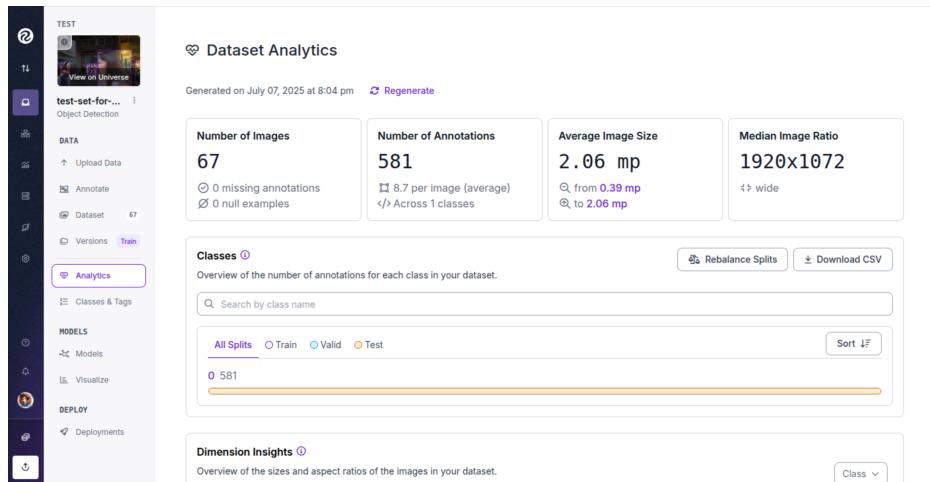


Figure 14: Statistiche sul test set calcolate su Roboflow

4.2.1 Dati di esempio del test set

Per poter comprendere meglio la tipologia dei frame utilizzati come test set, sono di seguito riportate un'immagine di esempio con la rispettiva **ground truth**: le reali etichette annotate mediante Roboflow.



Figure 15: Frame del test set senza ground-truth

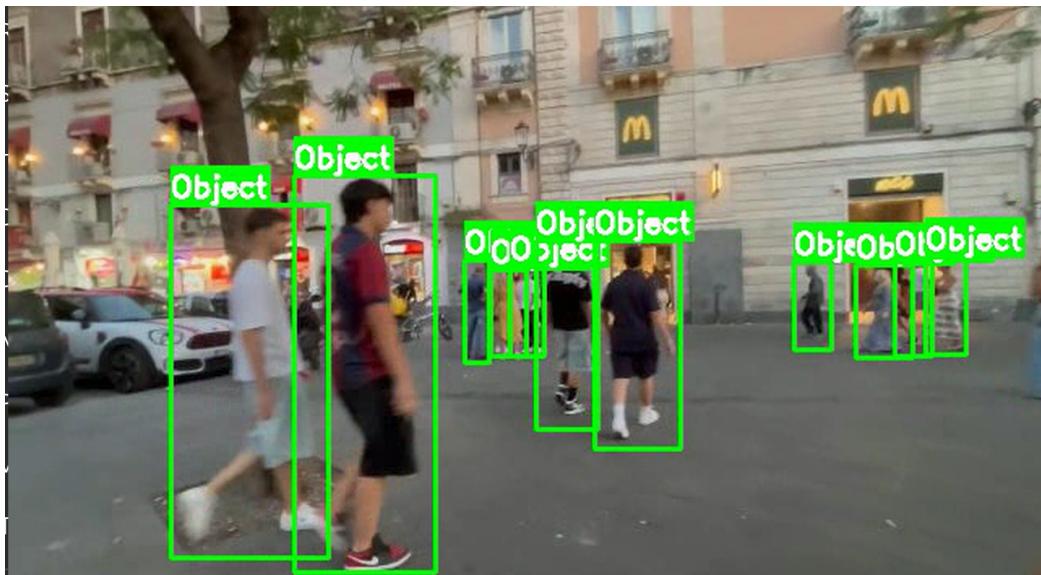


Figure 16: Frame del test set con ground-truth

4.3 Pre-processing dei dataset

Sono state necessarie diverse operazioni di pre-processing su entrambi i dataset per poter ottenere i migliori risultati possibili in termini di prestazioni e qualità nel training e nel test set dei modelli.

Le principali operazioni di pre-processing sui dataset sono:

- **Resize** a dimensioni 640x640, ovvero la dimensione consigliata nei modelli YOLO.
- **Re-mapping dei Class-ID**. Per poter gestire al meglio la classe *pedestrians* all'interno dei modelli è stato necessario rimuovere le righe inerenti a delle classi (1, 2, 3, 5, 6, 7) poiché non rappresentavano pedoni o persone; inoltre è stato necessario convertire le classi 4 e 8 in classi 0. Quest'ultima è la classe che identifica correttamente un pedone. Nel caso particolare del formato COCO JSON la classe 0 è solitamente associata al *background*, pertanto in questo caso verrà associata al pedone la classe 1.

La seconda operazione, il re-mapping, è stata effettuata utilizzando degli script Python per rendere il tutto più rapido ed automatico. Di seguito è possibile visualizzarne il codice sia per il formato YOLO che per il formato COCO JSON necessario per il modello Faster R-CNN.

Listing 1: Esempio di codice per il remapping del formato YOLO

```
1 import sys
2 import os
3 import argparse
4 from pathlib import Path
5
6 def convert_yolo_annotations(input_file):
7     converted_lines = 0
8     removed_lines = 0
9     output_lines = []
10
11    # Leggi il file di input
12    with open(input_file, 'r', encoding='utf-8') as f:
13        lines = f.readlines()
14
15    for line_num, line in enumerate(lines, 1):
16        line = line.strip()
17
18        # Salta righe vuote
```

```

19     if not line:
20         continue
21
22     # Dividi la riga in componenti
23     parts = line.split()
24
25     try:
26         class_id = int(parts[0])
27     except ValueError:
28         print(f"AttentioneRigaConClasseIDnonValido")
29         continue
30
31     # Applica le regole di conversione
32     if class_id in [4, 8]:
33         # Converte classe 4 e 8 -> classe 0
34         parts[0] = '0'
35         output_lines.append(''.join(parts))
36         converted_lines += 1
37
38     elif class_id == 0:
39         # Mantiene classe 0 invariata
40         output_lines.append(line)
41
42     elif class_id in [1, 2, 3, 5, 6, 7]:
43         # Rimuove le righe con queste classi
44         removed_lines += 1
45
46
47     # Scrivi il file di output
48     with open(input_file, 'w', encoding='utf-8') as f:
49         for line in output_lines:
50             f.write(line + '\n')
51
52     return converted_lines, removed_lines
53
54
55 def process_directory(directory_path, recursive=False):
56     directory = Path(directory_path)
57
58     # Trova tutti i file .txt
59     if recursive:
60         txt_files = list(directory.rglob("*.txt"))
61     else:
62         txt_files = list(directory.glob("*.txt"))
63

```

```

64     if not txt_files:
65         print(f"NessunFileTrovato")
66         return
67
68     total_converted = 0
69     total_removed = 0
70     processed_files = 0
71
72     for txt_file in txt_files:
73         try:
74             print(f"Processando:{txt_file}")
75             converted, removed = convert_yolo
76                 _annotations(str(txt_file))
77
78             total_converted += converted
79             total_removed += removed
80             processed_files += 1
81
82             if converted > 0 or removed > 0:
83                 print(f"-Convertite:{converted}-righe-(4,8->0)-")
84                 print(f"-Rimosse:{removed}-"
85                     "righe-(classi--1,2,3,5,6,7)")
86
87         except Exception as e:
88             print(f"Errore:{e}")

```

Listing 2: Esempio di codice per il remapping del formato COCO JSON

```

1 import json
2 from pathlib import Path
3
4 def convert_annotations_inplace(json_file):
5     # Leggi il file JSON
6     with open(json_file, 'r', encoding='utf-8') as f:
7         data = json.load(f)
8
9     # ID delle categorie da eliminare e convertire
10    categories_to_remove = {2, 3, 4, 6, 7, 8}
11    categories_to_convert = {5, 9}
12
13    # Statistiche iniziali
14    original_annotations = len(data['annotations'])
15
16    # 1. Filtra e converti le annotazioni
17    filtered_annotations = []
18    for annotation in data['annotations']:

```

```

19     category_id = annotation['category_id']
20
21     # Salta le annotazioni da eliminare
22     if category_id in categories_to_remove:
23         continue
24
25     # Converti category_id 5 e 9 in 1
26     if category_id in categories_to_convert:
27         annotation['category_id'] = 1
28
29     filtered_annotations.append(annotation)
30
31 data['annotations'] = filtered_annotations
32
33 # 2. Aggiorna le categorie
34 new_categories = []
35 for category in data['categories']:
36     cat_id = category['id']
37
38     # Salta le categorie da rimuovere
39     if cat_id in categories_to_remove:
40         continue
41
42     # Aggiorna la categoria 1 per riflettere la fusione
43     if cat_id == 1:
44         category['name'] = "pedestrian"
45         category['supercategory'] = "pedestrians-"
46
47     # Mantieni solo le categorie necessarie
48     if cat_id in {0, 1}:
49         new_categories.append(category)
50
51 data['categories'] = new_categories
52
53
54 # Salva il file modificato
55 with open(json_file, 'w', encoding='utf-8') as f:
56     json.dump(data, f, indent=2, ensure_ascii=False)
57
58 # Statistiche finali
59 print(f"Annotazioni:{original_annotations}
60 ->{len(data['annotations'])}")
61 print(f"Categorie:{len(new_categories)}")

```

5 Metodi

5.1 Funzionamento YOLO

Il principio fondamentale di YOLO risiede nella sua strategia di suddividere l’immagine in una griglia. Ogni cella di questa griglia è incaricata di prevedere la presenza di un oggetto il cui centro ricade al suo interno. Per ogni cella, il modello genera delle previsioni relative ai riquadri di delimitazione (bounding box), al grado di certezza della previsione (punteggio di confidenza) e alle probabilità delle diverse classi di oggetti. Questa architettura integrata consente a YOLO di raggiungere velocità di elaborazione elevate, rendendolo particolarmente adatto per applicazioni che richiedono risposte immediate. Le successive evoluzioni di YOLO hanno costantemente affinato questo meccanismo di base, migliorando l’accuratezza senza compromettere la rapidità. Di seguito, verranno spiegati in dettaglio i meccanismi specifici di funzionamento di YOLO.

5.1.1 Residual block

Un residual block è un componente utilizzato per facilitare l’addestramento di reti molto profonde, ampiamente utilizzato nelle architetture YOLO moderne. Il concetto chiave è la skip connection: l’output di un gruppo di layer convoluzionali $F(x)$ viene sommato all’input originale x , realizzando un collegamento identitario diretto. Matematicamente, il blocco residuo può essere rappresentato come:

$$y = F(x) + x,$$

dove x è l’input del blocco, $F(x)$ rappresenta la trasformazione appresa e y l’output. Questa formulazione preserva il gradiente durante il backpropagation anche in reti molto profonde, mitigando problemi di vanishing gradient. In sintesi, i blocchi residui migliorano la propagazione delle informazioni e la stabilità del training nei modelli YOLO, consentendo reti più profonde e prestazioni migliori grazie alla riutilizzazione dei segnali a vari livelli della rete.

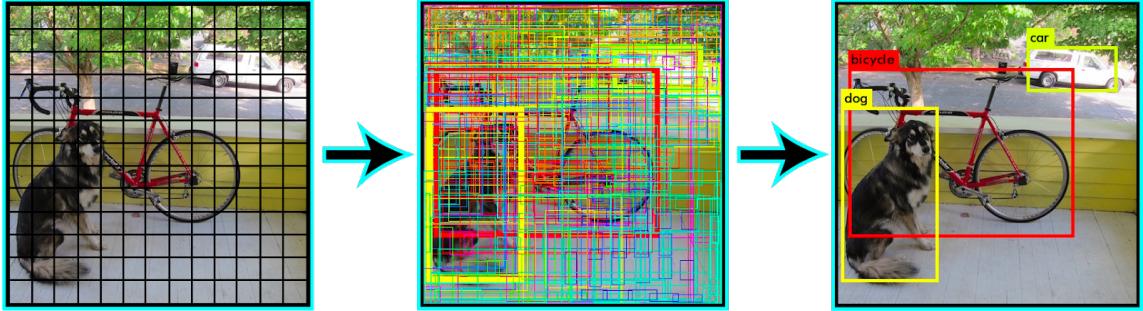


Figure 17: Esempio di Residual Block

5.1.2 Bounding box regression

Rappresenta il cuore di un detector YOLO. Data un’immagine, questa può contenere un numero variabile di oggetti, ciascuno dei quali viene rappresentato da una bounding box predetta dal modello. YOLO ne determina i valori tramite un modulo di regressione nel seguente formato come segue:

$$Y = [pc, bx, by, bh, bw, c1, \dots, cn].$$

dove:

- pc: indica la probabilità che un oggetto sia presente,
- bx, by: sono le coordinate normalizzate del centro,
- bw, bh: larghezza e altezza,
- c_i : sono le probabilità di classe.

Ogni bounding box è comunemente rappresentata dal punto centrale (x, y) , dalla larghezza w e dall’altezza h del riquadro rispetto all’immagine. I modelli YOLO originari (v1-v3) utilizzavano anchor boxes come riferimenti predefiniti di forma: la rete in uscita prediceva delle quantità t_x, t_y, t_w, t_h che venivano poi trasformate nelle coordinate finali del box aggiustando l’ancora assegnata.

Durante l’addestramento, YOLO confronta ogni box predetto con i ground truth corrispondenti usando una loss multicomponente. Ad esempio in Yolov8 si ha la combinazione di CIoU (Complete IoU loss)+ Distribution Focal Loss.

Questi accorgimenti assicurano che la bounding box regression produca riquadri più accurati e stabili durante il training e inferenza, tenendo conto non solo dell'errore assoluto sulle coordinate ma anche della misura di overlap e dell'allineamento geometrico tra predizione e ground truth.

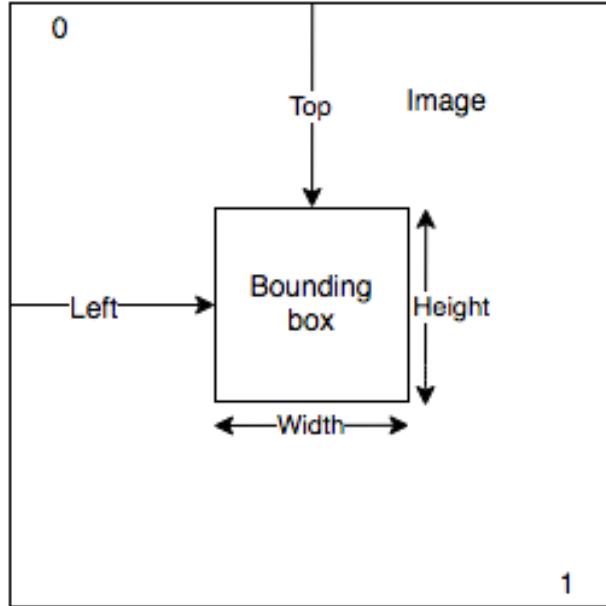


Figure 18: Esempio di Bounding Box

5.1.3 Intersection Over the Union

L'Intersection over Union (IoU) è la metrica standard per valutare la sovrapposizione tra due bounding box, ed è cruciale sia in fase di training che di valutazione dei modelli YOLO. Data una bounding box predetta B_p e la bounding box ground truth B_{gt} , l'IoU è definita come il rapporto tra l'area di intersezione e l'area di unione dei due riquadri:

$$\text{IoU}(B_p, B_{gt}) = \frac{\text{Area}(B_p \cap B_{gt})}{\text{Area}(B_p \cup B_{gt})}$$

In particolare, IoU varia da 0 (inteso come nessuna intersezione) a 1 (inteso come sovrapposizione perfetta).

Durante la fase di addestramento, questa viene utilizzata per determinare quale anchor (o cella) è responsabile di predire un dato oggetto. Mentre, per la fase di valutazione, l’IoU viene impiegato per decidere se una predizione è considerata un True Positive: ad esempio una detection è ritenuta corretta se l’IoU con il ground truth supera una certa soglia (generalmente 0.5). Inoltre, molte funzioni di loss avanzate (come GIoU, DIoU, CIoU, SIoU) estendono o pesano la metrica IoU per guidare la regressione delle box in modo più efficace.

Infine, nel contesto della NMS (Non-Maximum Suppression), l’IoU è il criterio con cui si valuta la ridondanza di due predizioni: se due box predetti per lo stesso oggetto hanno un IoU elevato, uno dei due verrà soppresso.

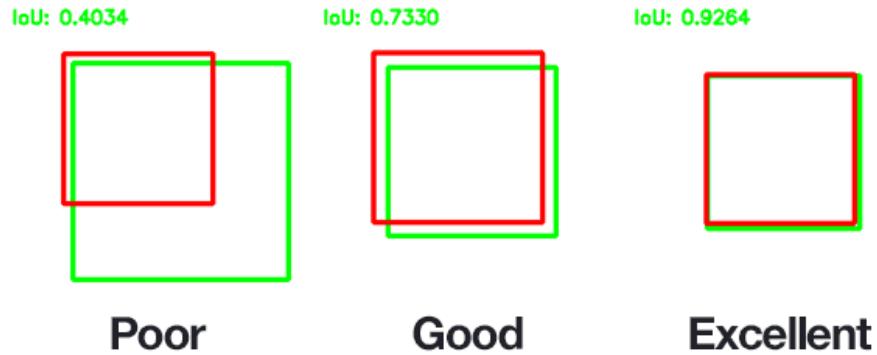


Figure 19: Esempio di Intersection Over Union

5.1.4 Non-Maximum Suppression

E’ una procedura di post-processing utilizzata tradizionalmente dai modelli YOLO (e dai detector in generale) per eliminare le predizioni duplicate relative allo stesso oggetto.

Il suo impiego risulta necessario poiché le bounding box non sopresse nel passaggio precedente, possono essere molto vicine tra loro e creare rumore. In pratica, per ciascuna classe si ordinano le bounding box predette per punteggio di confidenza; si prende la box con confidenza più alta e la si emette come rilevazione finale, quindi si sopprimono tutte le altre box che hanno con essa un IoU superiore a una certa soglia (di solito 0.5). Si ripete poi il processo con la successiva box rimanente in ordine di score, fino ad esaurire le predizioni. Questo semplice procedimento garantisce che per ogni oggetto

venga riportata idealmente una sola bounding box.

Tuttavia, è necessario portare l'attenzione su YOLOv10 che rappresenta un cambio di paradigma poiché, tramite la strategia di consistent one-to-one assignment in training. Quest'ultimo elimina la necessità di NMS poiché la rete impara a produrre al massimo una box per oggetto. Ogni oggetto viene assegnato alla migliore predizione (e non a molteplici anchor come avveniva prima), riducendo drasticamente le duplicazioni in output. Questo approccio NMS-free non solo semplifica la pipeline end-to-end, ma riduce anche la latenza e migliora potenzialmente la precisione in scenari affollati, dove NMS talvolta poteva sopprimere rilevazioni valide.

In conclusione, si ritiene comunque corretto attenzionare la Non-Maximum Suppression poiché è stata a lungo un elemento fondamentale nell'oggettistica di YOLO per ottenere output puliti e privi di duplicati; le ultime evoluzioni della famiglia mirano però a incorporarne gli effetti direttamente nel processo di training, rendendo il modello end-to-end senza post-processing esplicito.

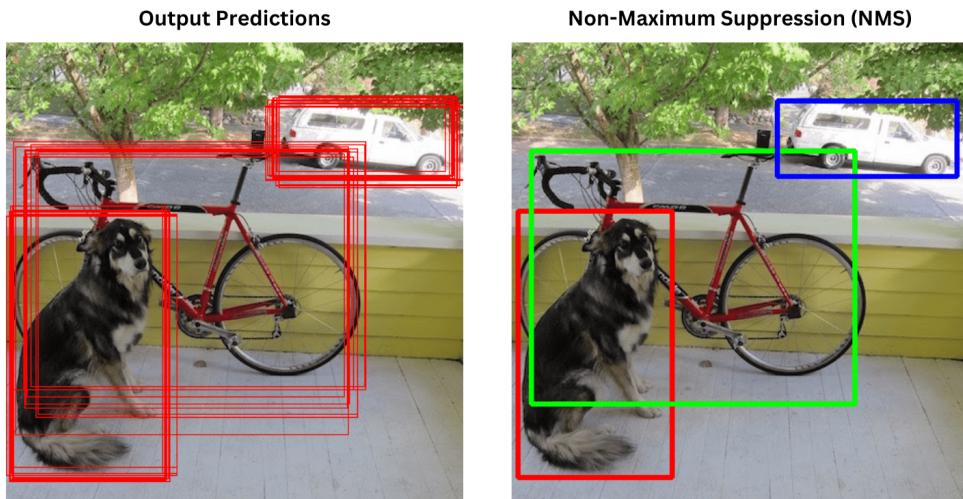


Figure 20: Funzionamento di NMS

5.2 Funzionamento Faster R-CNN

Faster R-CNN si distingue per la sua architettura a due stadi, che le permette di raggiungere un'elevata precisione nella localizzazione e classificazione degli oggetti. Opera attraverso un processo a due fasi distinte e consecutive,

che insieme garantiscono un'elevata precisione nell'identificazione e classificazione degli oggetti.

Di seguito si entrerà nei dettagli del suo funzionamento.

5.2.1 Feature Extraction

Il primo stadio di Faster R-CNN si concentra sull'identificazione delle aree dell'immagine che potrebbero contenere un oggetto. Inizialmente, l'immagine viene processata da una *backbone network*, ovvero una rete neurale convoluzionale profonda. Quest'ultima viene spesso pre-addestrata su vasti dataset di immagini come ImageNet; questo processo permette di imparare il riconoscimento di un'ampia varietà di caratteristiche visive. La backbone, dunque, estrae una mappa di caratteristiche ad alto livello dall'intera immagine.

Nel contesto di studi come il presente, l'utilizzo di **Resnet50** come backbone network per Faster R-CNN è una scelta comune. Resnet50 (**Residual Network con 50 strati**) è una CNN profonda e molto efficiente, rinomata per la sua capacità di superare il *vanishing gradient problem* (un ostacolo comune nelle reti molto profonde) grazie all'introduzione di connessioni residue (*skip connections*). Queste connessioni permettono ai gradienti di fluire più liberamente attraverso la rete, facilitando l'addestramento di modelli molto profondi e consentendo a Resnet50 di apprendere rappresentazioni complesse e ricche, fondamentali per il riconoscimento accurato degli oggetti.

5.2.2 Region Proposal Network - RPN

Successivamente, entra in gioco il **Region Proposal Network** (RPN). L'RPN è una piccola rete neurale che scansiona la mappa di caratteristiche estratta. Il suo compito non è ancora quello di classificare gli oggetti, ma di generare una serie di proposte di regione (*Region Proposals*), ovvero dei bounding box preliminari che indicano dove è probabile che sia presente un oggetto. L'RPN valuta ogni potenziale area e assegna un punteggio di oggettività, indicando quanto è probabile che quell'area contenga qualcosa di interessante (un oggetto) piuttosto che semplice sfondo. Questo processo è cruciale perché riduce drasticamente il numero di aree da analizzare nel dettaglio, concentrando l'attenzione del modello solo sulle porzioni più promettenti dell'immagine.

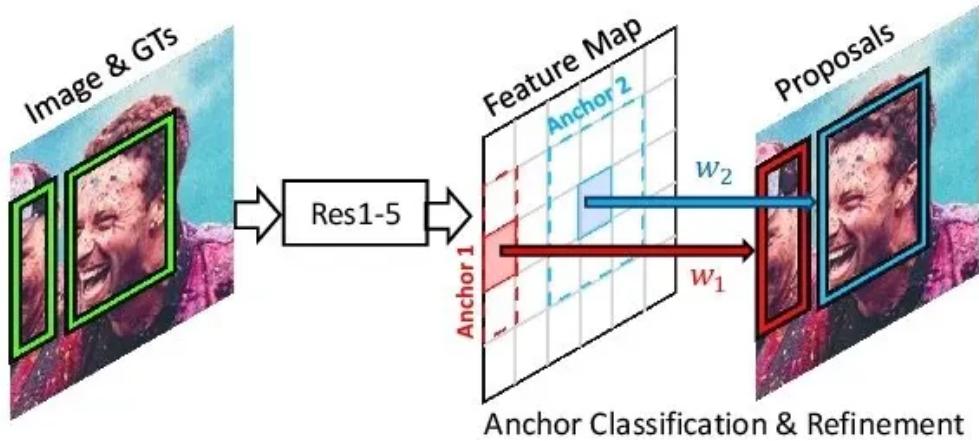


Figure 21: Esempio di Region Proposal Network

5.2.3 Classificazione e Refinement (RoI Pooling)

Una volta che l'RPN ha generato le sue proposte di regione, queste vengono passate al secondo stadio per un'analisi ancora più approfondita. Per prima cosa, un meccanismo chiamato **RoI Pooling** (*Region of Interest Pooling*, o *RoI Align* nelle versioni più recenti per una maggiore precisione) standardizza le dimensioni di tutte le proposte di regione, indipendentemente dalla loro dimensione originale. Questo è necessario perché i successivi strati di classificazione e regressione richiedono input di dimensione fissa. Le regioni standardizzate vengono poi alimentate a due "teste" parallele:

- **Classificatore:** Una rete neurale che prende ogni proposta di regione e ne determina la classe specifica (ad esempio, "persona", "veicolo", "sfondo"). È qui che il modello decide cosa c'è all'interno del bounding box.
- **Regressore del Bounding Box:** Un'altra rete neurale che affina le coordinate del bounding box generato dall'RPN. Questo passaggio è fondamentale per rendere il riquadro il più preciso possibile, adattandolo perfettamente all'oggetto rilevato.



Figure 22: Esempio di RoI Align

6 Valutazione del modello

Nel presente capitolo, vengono elencate e discusse le metriche di valutazione, sia in termini di valori numerici che di rappresentazioni grafiche, utilizzate per analizzare le performance dei modelli di Object Detection impiegati, inclusi i modelli YOLO e Faster R-CNN.

6.1 Metriche di valutazione

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

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

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

- **Recall:** detta anche "sensitivity", misura la proporzione di rilevamenti positivi corretti rispetto a tutte le istanze effettivamente positive. Viene

definita come il rapporto tra i True Positive e la somma dei True Positive e False Negative:

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

- **F1-score:** è una metrica che tiene conto sia della precision che della recall. È molto utile nel caso in cui si vogliano bilanciare precision e recall, in quanto penalizza valori fortemente negativi di precision o recall. Viene calcolata come una media armonica tra precision e recall:

$$F1 = 2 \times \frac{Precision \times Recall}{Precision + Recall}$$

- **Valore di confidenza:** detto anche confidence score, è un numero compreso tra 0 e 1, associato ad ogni predizione effettuata da un modello di Machine Learning, che indica la probabilità che quest'ultima sia corretta. Nel caso di un modello di Object Detection come YOLO, essa indica la probabilità che la classe associata dal modello all'oggetto individuato sia corretta.
- **Curva Precision-Recall:** grafico utilizzato per valutare le prestazioni di un modello. Questa curva mostra la variazione della precision in funzione della recall.
- **Average Precision:** misura l'area al di sotto della curva precision-recall. Viene calcolata come segue:

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

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

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

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

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

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

$$mAP = \frac{1}{N} \sum_{i=1}^N \text{AP}_i$$

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

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

6.2 Grafici

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

- **Matrice di confusione:** è una tabella che mostra il numero di previsioni corrette e errate effettuate da un modello su un set di dati. Viene in genere utilizzato per valutare le prestazioni di un modello di classificazione binaria, ma può anche essere applicato a attività di classificazione multiclasse. In particolare, l'asse delle ascisse rappresenta le classi reali, mentre l'asse delle ordinate rappresenta le classi predette. Nel caso di un problema di classificazione binario, all'interno della matrice di confusione è possibile distinguere:

- Veri positivi (TP): il numero di esempi che sono stati previsti come positivi e sono effettivamente positivi.
- Veri negativi (TN): il numero di esempi che sono stati previsti come negativi e sono effettivamente negativi.
- Falsi positivi (FP): il numero di esempi previsti come positivi ma in realtà negativi.
- Falsi negativi (FN): il numero di esempi previsti come negativi ma in realtà positivi.

	Predicted Positive	Predicted Negative
Actual Positive	True Positive (TP)	False Negative (FN)
Actual Negative	False Positive (FP)	True Negative (TN)

Figure 23: Esempio di matrice di confusione

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

- **Recall-Confidence Curve:** è una rappresentazione visuale della relazione tra recall e confidenza. In particolare, mostra come varia la recall in funzione della variazione della soglia di confidenza. Utilizzando tale grafico, si nota come quando la threshold di confidenza diminuisce, più predizioni sono considerate valide.
- **F1-Confidence Curve:** è una rappresentazione visuale della relazione tra F1 score e confidenza. In particolare, mostra come varia il F1 score in funzione della variazione della soglia di confidenza. Un valore alto di F1 score indica performance migliori, e di conseguenza, la soglia di confidenza per cui si ottiene tale valore solitamente è la migliore per effettuare predizioni quanto più accurate possibile.

6.3 Loss functions

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

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

7 Fase di Training e Validazione

Nella presente sezione vengono descritti gli aspetti principali delle fasi di training e validazione dei diversi modelli YOLO impiegati (**YOLOv8m**, **YOLOv10n**, **YOLOv11s**), utilizzando il dataset e le metriche di valutazione definiti nei capitoli precedenti. In particolare, verranno elencati i parametri di training adottati, i passi seguiti per effettuare il training e la validazione (inclusi i frammenti di codice utilizzati) e saranno mostrati i principali risultati attraverso grafici e tabelle.

Si assume che l’addestramento avvenga in un ambiente *Kaggle* (o analogo) con una distribuzione *Python* già configurata e che il dataset sia stato correttamente preparato nel formato richiesto (per YOLO o per Faster RCNN).

Nota: Tutti i codici e i file menzionati in questa sezione sono disponibili nel *notebook* Kaggle del progetto. Sarà inoltre messo a disposizione un *demo notebook* su Kaggle sulla quale effettuare le predizioni su immagini o video a scelta. Il link di accesso al progetto Kaggle è il seguente: [Link](#).

7.1 YOLO

7.1.1 Parametri di training

Di seguito sono riportati i parametri principali adottati per l’addestramento dei modelli **YOLOv8m**, **YOLOv10n** e **YOLOv11s**. I valori scelti tengono conto della disponibilità computazionale offerta dalla piattaforma Kaggle (**GPU P100**) e sono stati mantenuti coerenti tra i modelli al fine di garantire una valutazione comparabile.

In particolare:

- Il numero di **epoch**e e il **batch size** sono stati scelti per massimizzare l’apprendimento pur rispettando i limiti di tempo imposti dalle GPU.
- Il **learning rate**, il tipo di **ottimizzatore** e i parametri di regolarizzazione seguono le indicazioni generali dei paper ufficiali YOLOv8 e YOLOv10.
- L’input è stato ridimensionato a 640×640 , risoluzione ottimale per il compromesso tra accuratezza e prestazioni.

Parametro	Valore utilizzato	Descrizione
epochs	100	Numero di epoche (iterazioni complete sul dataset di train)
patience	n/a	Epoche di tolleranza per <i>early stopping</i> in assenza di miglioramenti
lr0 (Learning Rate iniziale)	0.003	Passo di apprendimento iniziale per l'ottimizzatore
weight decay	0.0005	Coefficiente di regolarizzazione L2 (<i>Weight Decay</i>)
imgsz	640×640	Dimensione di input delle immagini di training (in pixel)
batch	32	Dimensione del batch (numero di immagini per iterazione)
ottimizzatore	Adam	Algoritmo di ottimizzazione utilizzato (es. Adam, SGD, ecc.)

Table 1: Parametri principali utilizzati per il training dei modelli

7.1.2 Organizzazione della directory di training

I file necessari sono stati caricati nella directory `/kaggle/input/new-dataset-for-ml-project-last-version/`, che contiene l'intero dataset annotato e il file di configurazione `data.yaml`, indispensabile per il training con i modelli Ultralytics.

La struttura della directory rispetta lo standard previsto dai modelli YOLOv8/v10-/v11, con suddivisione tra **train**, **valid** e **test**, ciascuno contenente le rispettive sottocartelle **images** e **labels**. Di seguito viene mostrata la struttura completa:

```
/kaggle/input/new-dataset-for-ml-project-last-version/
|-- train/
|   |-- images/
|   |-- labels/
|-- val/
|   |-- images/
```

```

|   |-- labels/
|-- test/
|   |-- images/
|   |-- labels/
`-- data.yaml

```

Il file `data.yaml` specifica i percorsi relativi alle directory di training, validation e testing, il numero di classi, e l'elenco delle etichette di ciascuna classe. Nel caso dello studio si ricorda la presenza di una sola classe: persona. Tutti i file di annotazione sono forniti nel formato YOLO (file `.txt` con annotazioni normalizzate per ogni immagine) e rispettano le specifiche richieste dal framework Ultralytics, ovvero le bounding box nel formato `class-index x-center y-center width height`. È importante sottolineare che ciascun file di annotazione **deve** avere lo stesso nome del file di immagine corrispondente al fine di una corretta etichettatura.

7.1.3 Fase di training

La fase di training dei modelli è stata eseguita in ambiente *Kaggle*, sfruttando GPU P100 per ridurre i tempi di addestramento. Tutti i modelli YOLO (YOLOv8m, YOLOv10n, YOLOv11s) sono stati addestrati utilizzando la stessa pipeline, modificando unicamente il peso preaddestrato di partenza.

Il codice di addestramento per ciascun modello è il seguente:

Listing 3: Esempio di codice per il training di YOLOv10n

```

1 from ultralytics import YOLO
2
3 # Inizializzazione del modello con pesi preaddestrati
4 model = YOLO('yolov10n.pt')
5
6 # Training del modello sul dataset personalizzato
7 model.train(
8     data = "/kaggle/input/new-dataset-for-ml-project
9 -last-version/data.yaml",
10    epochs = 100,
11    imgsz = (640, 640),
12    batch = 32,
13    optimizer = "Adam",
14    lr0 = 1e-3
15 )

```

Per il training di YOLOv8m e YOLOv11s è stato sufficiente sostituire il peso iniziale con '`yolov8m.pt`' e '`yolov11s.pt`' rispettivamente. La struttura del dataset è conforme al formato YOLOv5/Ultralytics (equivalente per le versioni di YOLO successive), con il file `data.yaml` che definisce il percorso per le immagini e le annotazioni, oltre al numero di classi.

Durante la fase di addestramento, il framework Ultralytics ha generato automaticamente la cartella `/runs/detect/train/` contenente:

- I pesi ottimali del modello (`best.pt`) e quelli all'ultima epoca (`last.pt`);
- Grafici di loss, precision, recall e mAP (visualizzabili da `results.png`);
- File di log contenenti metriche dettagliate per ogni epoca;
- Esempi di predizioni effettuate sul validation set (in `val_batch_images.jpg`).

Tutti i modelli hanno mostrato una regolare curva di convergenza della loss e un incremento progressivo delle metriche di valutazione nel tempo, come approfondito nelle sezioni successive.



Figure 24: Esempi di etichette su training

7.1.4 Fase di validazione

La fase di validazione dei modelli viene eseguita su un test set acquisito e annotato manualmente, in modo da poter visualizzare più concretamente le capacità dei modelli precedentemente addestrati. Di seguito è possibile visualizzare il codice di validazione per ciascun modello:

Listing 4: Esempio di codice per la validazione dei modelli YOLO

```
1 from ultralytics import YOLO  
2  
3 model = YOLO("best-model-weights-to-evaluate.pt")  
4  
5 results = model.val(data="/kaggle/input/test-set-for-ml-  
6 project/test-set-for-ml-project/data.yaml", split = "test")  
7  
8 results.results_dict
```

È possibile sostituire il `best-model-weights-to-evaluate.pt` con i pesi migliori del modello di cui si vuole effettuare la validazione. Anche in questo caso viene fornito un file `.yaml` che permette di individuare il path corrispondente al test comprensivo di immagini e annotazioni. Il campo `.results_dict` permette di visualizzare il dizionario delle metriche: precision, recall, mAP(50), mAP(50-95), fitness. I risultati verranno salvati dentro la cartella `run/detect/val[n]`.



Figure 25: Esempi di validazione su split = Validation Set con YOLOv11s



Figure 26: Esempi di validazione su split = Test Set con YOLOv11s

7.2 Faster R-CNN con Resnet50

7.2.1 Parametri di training

Di seguito sono riportati i parametri principali adottati per l'addestramento del modello *two-stage* **Faster RCNN** con backbone **Resnet50**, architettura con 50 layer convoluzionali. I valori scelti tengono conto della disponibilità computazionale data dalla piattaforma Kaggle (**GPU P100**).

Parametro	Valore utilizzato	Descrizione
<code>epochs</code>	30	Numero di epoche (iterazioni complete sul dataset di train)
<code>patience</code>	n/a	Epoche di tolleranza per <i>early stopping</i> in assenza di miglioramenti
<code>lr0</code> (Learning Rate iniziale)	0.001	Passo di apprendimento iniziale per l'ottimizzatore
<code>weight decay</code>	0.0005	Coefficiente di regolarizzazione L2 (<i>Weight Decay</i>)
<code>momentum</code>	0.9	Consente di mantenere l'inerzia del movimento dei pesi nel tempo
<code>gamma</code>	0.1	Controlla quanto peso hanno i singoli punti di addestramento nel determinare il <i>decision boundary</i>
<code>step_size</code>	3	Determina la dimensione dei passi fatti durante la discesa del gradiente
<code>imgsz</code>	640×640	Dimensione di input delle immagini di training (in pixel)
<code>batch</code>	4	Dimensione del batch (numero di immagini per iterazione)
<code>ottimizzatore</code>	SDG	Algoritmo di ottimizzazione utilizzato (es. Adam, SGD, ecc.)

Table 2: Parametri principali utilizzati per il training del modello

7.2.2 Organizzazione della directory per il training

I file necessari sono stati caricati nella directory `/kaggle/input/new-dataset-for-ml-project-coco`, che contiene l'intero dataset annotato e i file di annotazione `_annotations.coco.json`. La struttura della directory rispetta lo standard previsto dal modello Faster RCNN con suddivisione tra **train**, **valid** e **test**, ciascuno contenente le rispettive sottocartelle **images**. Al loro interno si trovano non solo i vari frame, ma anche i file di annotazione in formato `.json`, seguendo lo standard del dataset COCO. Di seguito viene mostrata la struttura completa:

```
/kaggle/input/new-dataset-for-ml-project-coco/
|-- train/
|   |-- images/
|   |   |-- _annotations.coco.json
|   |   |-- img.png
|-- val/
|   |-- images/
|   |   |-- _annotations.coco.json
|   |   |-- img.png
|-- test/
    |-- images/
        |-- _annotations.coco.json
        |-- img.png
```

Le annotazioni in formato COCO JSON rappresentano uno standard ampiamente utilizzato nel campo dell'Object Detection. Forniscono un modo strutturato per memorizzare immagini includendo informazioni sugli oggetti, le loro bounding boxes, categorie e altri metadati. Tipicamente all'interno di un file di annotazione COCO è possibile trovare cinque campi principali: `info`, `licences`, `images`, `annotations`, `categories`. Viene sempre indicata una supercategoria consistente nel *background* dell'immagine come nel caso dello studio e, ancora, una categoria a seguire: 1 (persona). *Si noti invece che nel caso di YOLO, la categoria persona viene codificata con 0.*

7.2.3 Fase di training

La fase di training del modello è stata eseguita in ambiente *Kaggle*, sfruttando GPU P100 per una runtime totale di 2h 54m.

Il codice di addestramento per il modello è il seguente:

Listing 5: Esempio di codice per il training del modello Resnet

```
1 model.train()
2 total_loss = 0
3 num_batches = len(data_loader)
4
5 progress_bar = tqdm(data_loader, desc=f"Training-
6 Epoch{epoch+1}")
7
8 for batch_idx, (images, targets) in enumerate(progress_bar):
9     # Move data to device
10    images = [img.to(device) for img in images]
11    targets = [{k: v.to(device) for k, v in t.items()}]
12    for t in targets]
13
14    # Forward pass
15    loss_dict = model(images, targets)
16    losses = sum(loss for loss in loss_dict.values())
17
18    # Backward pass
19    optimizer.zero_grad()
20    losses.backward()
21    optimizer.step()
22
23    total_loss += losses.item()
24
25    # Update progress bar
26    progress_bar.set_postfix({
27        'Loss': f'{losses.item():.4f}',
28        'AvgLoss': f'{total_loss/(batch_idx+1):.4f}'
29    })
30
31 return total_loss / num_batches
```

Durante la fase di allenamento è possibile visualizzare lo status del training mediante il widget della progress bar. Inoltre, per ogni epoca verrà stampata a video la misura di Train Loss, Val Loss, Epoca corrente e infine Learning Rate per l'epoca in questione.

7.2.4 Fase di validazione

La fase di validazione del modello è eseguita su un test set acquisito e annotato manualmente, in modo da poter visualizzare e mettere a confronto con le capacità dei modelli YOLO precedentemente valutati. Di seguito è possibile visualizzare il codice di validazione per il modello:

Listing 6: Esempio di codice per la validazione del modello Resnet

```
1 def evaluate_model(model, data_loader, device,
2 confidence_threshold = 0.5):
3     # Evaluate the model on a dataset
4     model.eval()
5
6     all_predictions = []
7     all_targets = []
8
9     with torch.no_grad():
10         for images, targets in tqdm(data_loader, desc="Evaluating"):
11             images = [img.to(device) for img in images]
12
13             predictions = model(images)
14
15             # Process predictions and targets
16             for pred, target in zip(predictions, targets):
17                 scores = pred['scores']
18                 keep = scores >= confidence_threshold
19
20                 pred_filtered = {
21                     'boxes': pred['boxes'][keep].cpu().numpy(),
22                     'scores': pred['scores'][keep].cpu().numpy(),
23                     'labels': pred['labels'][keep].cpu().numpy()
24                 }
25
26                 target_processed = {
27                     'boxes': target['boxes'].cpu().numpy(),
28                     'labels': target['labels'].cpu().numpy()
29                 }
30
31             all_predictions.append(pred_filtered)
32             all_targets.append(target_processed)
33
34     return all_predictions, all_targets
35
36
37 def calculate_iou(box1, box2):
38     # Calculate intersection coordinates
39     x1 = max(box1[0], box2[0])
40     y1 = max(box1[1], box2[1])
41     x2 = min(box1[2], box2[2])
42     y2 = min(box1[3], box2[3])
43
44     # Calculate intersection area
```

```

45     if x2 <= x1 or y2 <= y1:
46         return 0.0
47
48     intersection = (x2 - x1) * (y2 - y1)
49
50     # Calculate union area
51     area1 = (box1[2] - box1[0]) * (box1[3] - box1[1])
52     area2 = (box2[2] - box2[0]) * (box2[3] - box2[1])
53     union = area1 + area2 - intersection
54
55     return intersection / union if union > 0 else 0.0
56
57
58 def calculate_metrics(predictions, targets, iou_threshold = 0.5):
59     # Calculate detection metrics
60
61     all_pred_boxes = []
62     all_pred_scores = []
63     all_gt_boxes = []
64
65     # Collect all predictions and ground truths
66     for pred, target in zip(predictions, targets):
67         all_pred_boxes.extend(pred['boxes'])
68         all_pred_scores.extend(pred['scores'])
69         all_gt_boxes.extend(target['boxes'])
70
71     # Sort predictions by confidence score
72     sorted_indices = np.argsort(all_pred_scores)[::-1]
73     sorted_pred_boxes = [all_pred_boxes[i] for i in sorted_indices]
74
75     # Calculate matches
76     tp = 0 # True positives
77     fp = 0 # False positives
78     matched_gt = set() # Keep track of matched ground truth boxes
79
80     for pred_box in sorted_pred_boxes:
81         best_iou = 0.0
82         best_gt_idx = -1
83
84         for gt_idx, gt_box in enumerate(all_gt_boxes):
85             if gt_idx in matched_gt:
86                 continue
87
88             iou = calculate_iou(pred_box, gt_box)
89             if iou > best_iou:

```

```

90         best_iou = iou
91         best_gt_idx = gt_idx
92
93     if best_iou >= iou_threshold:
94         tp += 1
95         matched_gt.add(best_gt_idx)
96     else:
97         fp += 1
98
99     fn = len(all_gt_boxes) - tp # False negatives
100
101    precision = tp / (tp + fp) if (tp + fp) > 0 else 0.0
102    recall = tp / (tp + fn) if (tp + fn) > 0 else 0.0
103    f1_score = 2 * (precision * recall) / (precision + recall)
104    if (precision + recall) > 0 else 0.0
105
106    return {
107        'precision': precision,
108        'recall': recall,
109        'f1_score': f1_score,
110        'true_positives': tp,
111        'false_positives': fp,
112        'false_negatives': fn,
113        'total_predictions': len(all_pred_boxes),
114        'total_ground_truth': len(all_gt_boxes)
115    }

```

I risultati di validazione del modello permettono di concentrarsi su *precision*, *recall*, la *F1 score*, i *True Positives*, *False Positives*, *False Negatives*, *Total Predictions*, *Total Ground Truth*. A differenza dei modelli YOLO, non verranno generati dei grafici. Tuttavia è possibile effettuare e visualizzare le detection di frame e video mediante gli appositi metodi definiti.

7.3 Demo e funzionamento

La presente sezione è interamente dedicata alla demo implementata per poter testare i modelli su nuovi dati. La demo è stata scritta interamente in Python, appoggiandosi sulla piattaforma *Kaggle* per poter usufruire liberamente delle risorse computazionali senza necessità di installare librerie nel proprio ambiente. È pronta all'uso, basterà cliccare su **Copy & Edit** per poter modificare le celle del notebook. Presenta le seguenti funzionalità:

- Caricamento di un'immagine o di un video su cui effettuare la predi-

zione tramite tasto di Upload

- Scelta del modello con cui effettuare la predizione. Sarà possibile caricare il dataset con i migliori pesi reso pubblico. Si potrà selezionare dunque tra YOLOv8m, YOLOv10n, YOLOv11s, Resnet50.
- Visualizzazione grafica dei risultati: una volta terminato il processo di predizione sarà possibile visualizzare nella cartella `working` di *Kaggle* il risultato in formato `.png` e in formato `.avi|.mp4` per i video.
- Conteggio delle persone presenti per un'immagine, e una media di persone presenti per un video fornito in input.

Per poter utilizzare la demo basterà avviare la sessione prestando attenzione a modificare il path relativo al dataset contenente una o più immagini, oppure contenente uno o più video, o entrambi.

Listing 7: Path relativo al dataset

```
1 import torch
2
3 device = torch.device('cuda' if torch.cuda.is_available()
4 else 'cpu')
5
6 INPUT_FILE_PATH = "/kaggle/input/test-video-set/IMG_0555.MP4"
7 WORKING_PATH = "/kaggle/working/"
```

In questo modo sarà, dunque, possibile procedere alla fase successiva in cui quelle differenti del *notebook* potranno essere eseguite al fine di produrre il risultato desiderato. Nel caso dell'esempio sopra riportato la modalità è di video prediction dunque si avrà:

Listing 8: Esempio di video predict su YOLOv8m

```
1 model = YOLO("/kaggle/input/best-weights-for-model/best
2 yolov8m.pt")
3 results = model.predict(source = INPUT_FILE_PATH, save =
4 True, project = 'runs/detect', name = 'exp')"
```

È possibile importare il dataset `best-weights-for-model` con i migliori pesi per i modelli allenati nello studio tramite il seguente link: Best Weights Dataset

È possibile visualizzare un breve video tutorial sulla demo mediante questo

link: [Demo](#)

Nota: Si tratta di una demo su nuovi dati. Non è necessario caricare le bounding box relative a ciascun file poiché non verrà calcolata alcuna metrica. È una visualizzazione delle potenzialità dei modelli addestrati in questo studio.

Si consiglia inoltre di effettuare il pre-processing dei dati con una resize delle immagini o dei video alle dimensioni **640x640** utilizzate nello studio. Il link è seguente: [Demo notebook](#)



Figure 27: Esempio di output della Demo su Prediction Frame di Faster R-CNN

8 Risultati

Questo capitolo è dedicato all'analisi dei risultati conseguiti grazie alle fasi di training e validazione dei modelli di Object Detection impiegati nello studio. Il training è stato effettuato utilizzando il dataset denominato "nome del dataset", con l'ausilio di una **GPU P100** attraverso l'ambiente di sviluppo offerto dalla piattaforma *Kaggle*.

Nelle sezioni successive, verranno presentati in dettaglio grafici e tabelle che

riassumono le principali misure di valutazione calcolate sui diversi modelli.

8.1 Performance dei modelli YOLO

Di seguito, viene presentata una tabella riassuntiva delle performance ottenute dai modelli della famiglia YOLO sul test set acquisito.

Modello	Precision (B)	Recall (B)
YOLOv10n	0.8437	0.4830
YOLOv11s	0.8035	0.5594
YOLOv8m	0.8965	0.5368

Table 3: Performance dei modelli YOLO sul Test Set (Precisione e Recall)

Modello	mAP50 (B)	mAP50-95 (B)	Fitness
YOLOv10n	0.5871	0.2981	0.3270
YOLOv11s	0.6463	0.3333	0.3646
YOLOv8m	0.6555	0.3276	0.3604

Table 4: Performance dei modelli YOLO sul Test Set (mAP e Fitness)

8.1.1 Grafici dei risultati sul test set

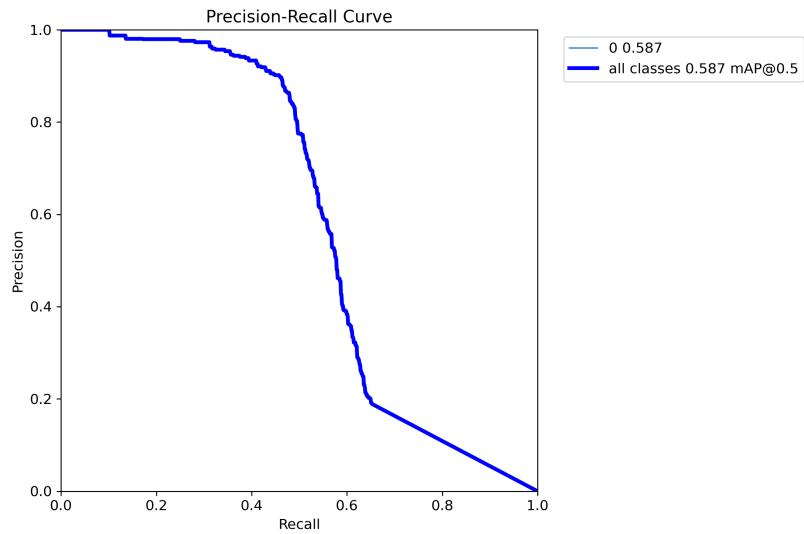


Figure 28: Curva di Precision-Recall per YOLOv8m

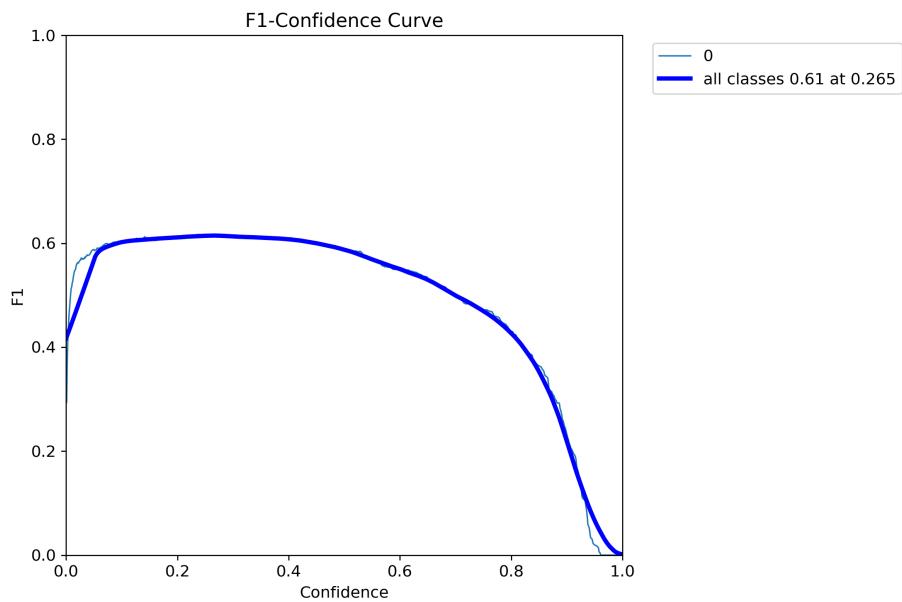


Figure 29: F1-Score per YOLOv8m

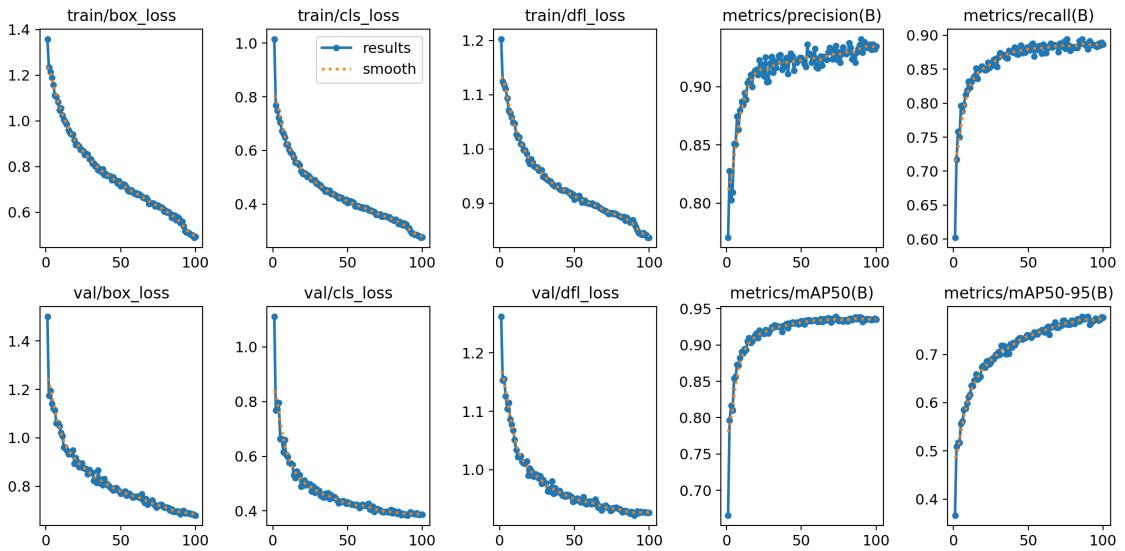


Figure 30: Risultati completi per YOLOv8m

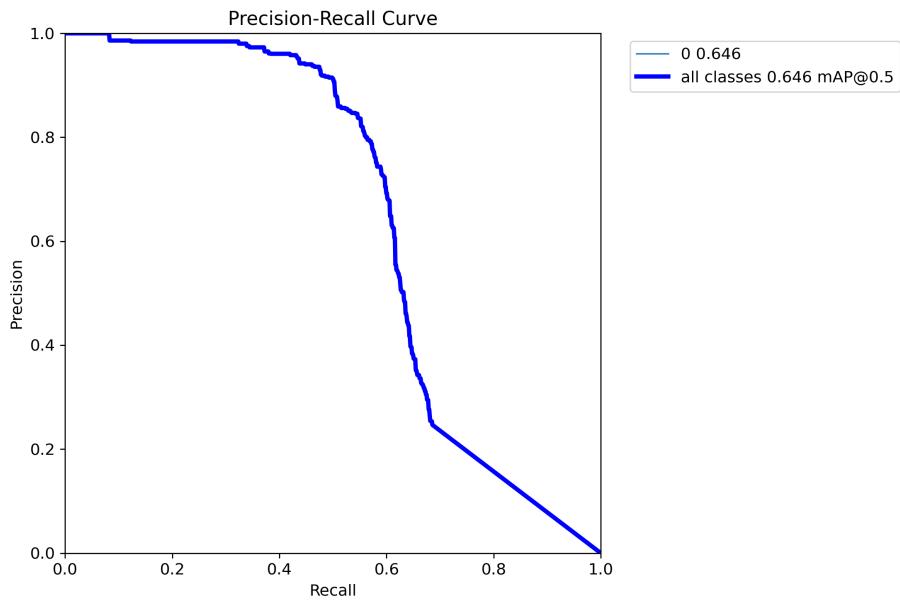


Figure 31: Curva di Precision-Recall per YOLOv10n

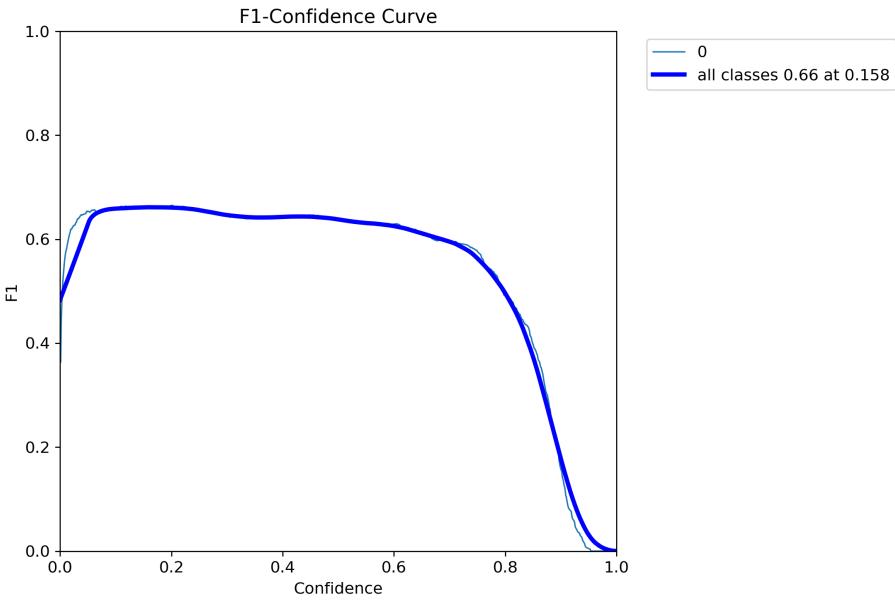


Figure 32: F1-Score per YOLOv10n

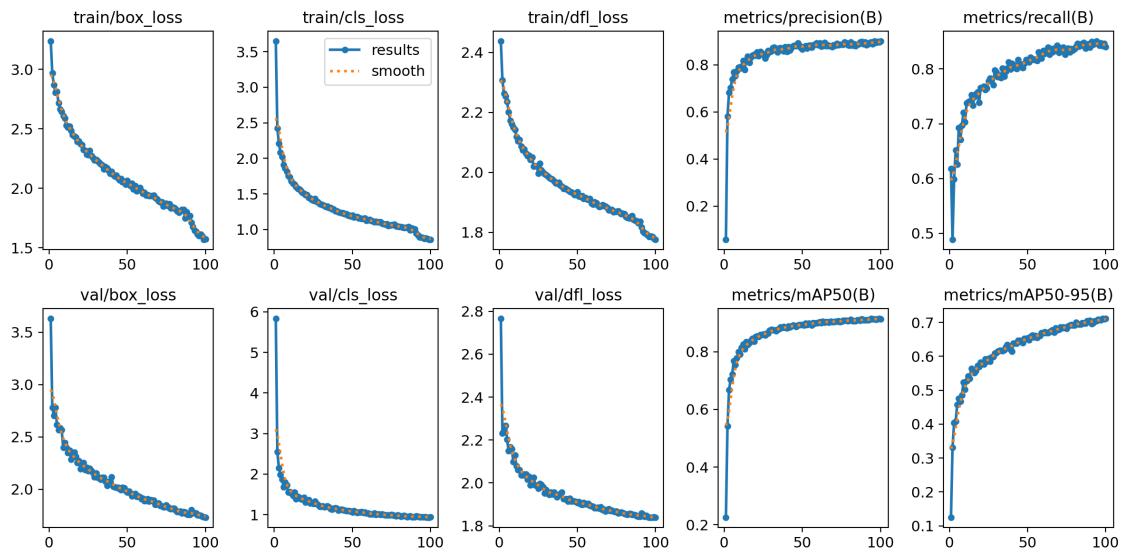


Figure 33: Risultati completi per YOLOv10n

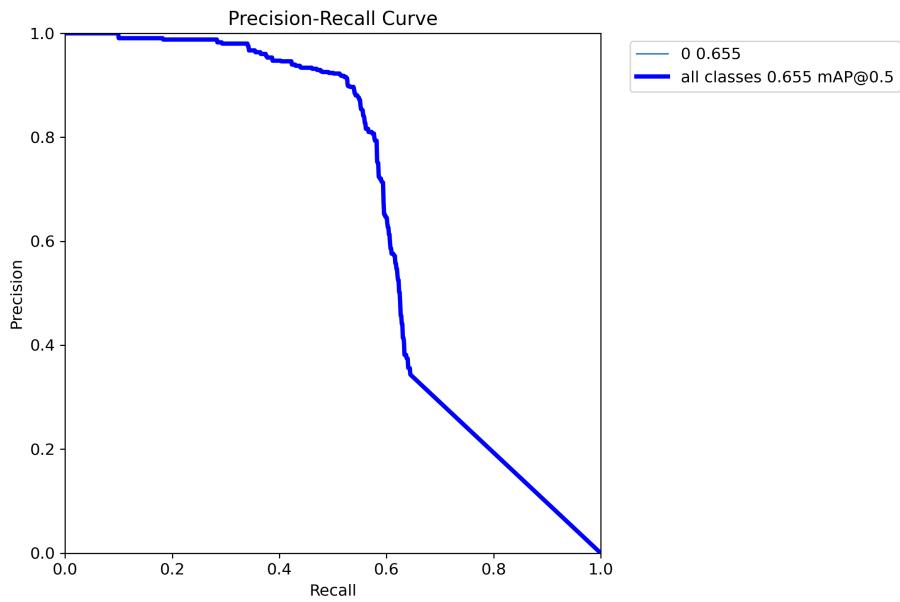


Figure 34: Curva di Precision-Recall per YOLOv11s

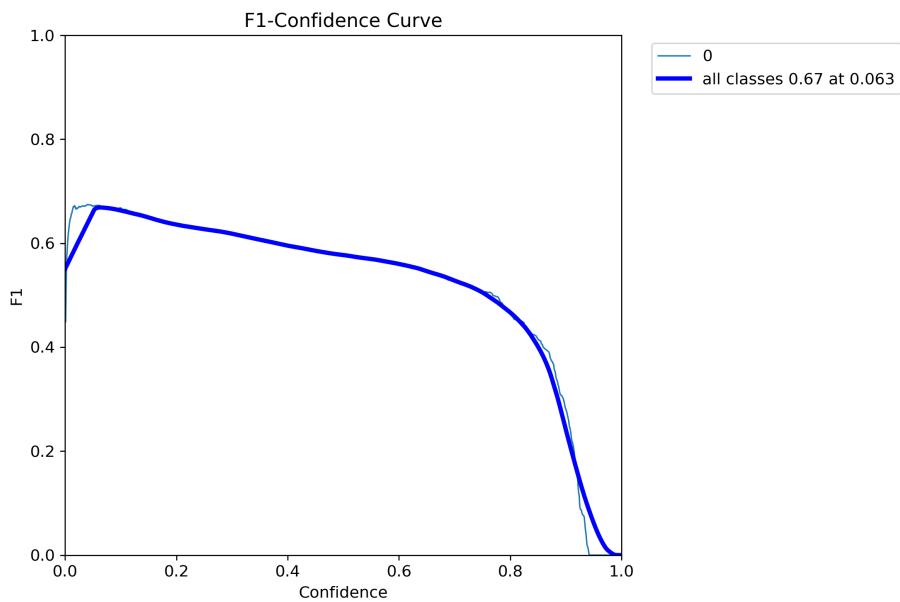


Figure 35: F1-Score per YOLOv11s

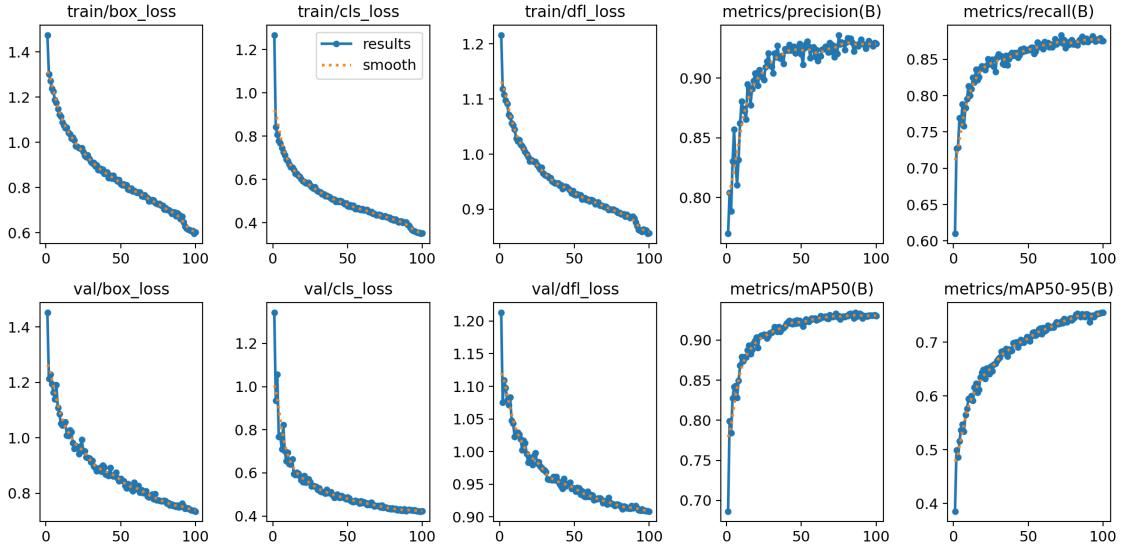


Figure 36: Risultati completi per YOLOv11s

8.1.2 Analisi dei risultati

Analizzando i dati presentati nelle tabelle, emerge un quadro interessante per i modelli YOLO, tutti addestrati per 100 epochhe. Nel dettaglio:

- **YOLOv8m**

Questo modello si distingue per la sua **Precisione (0.8965)** particolarmente alta. Ciò significa che quasi il 90% delle volte in cui rileva un oggetto e lo etichetta come "umano", la sua predizione è corretta. È un risultato notevole per evitare falsi positivi. Anche le metriche di **mAP (mAP50: 0.6555, mAP50-95: 0.3276)** sono le migliori del gruppo YOLO, indicando una buona capacità di rilevamento complessiva. Tuttavia, il suo punto debole, comune a tutti i modelli YOLO in questo contesto, è la **Recall (0.5368)**. Questo valore suggerisce che, pur essendo molto preciso nelle sue rilevazioni, riesce a identificare solo poco più della metà degli umani effettivamente presenti nel test set.

- **YOLOv11s**

Questo modello mostra un comportamento simile al precedente, con una **Precisione (0.8035)** comunque elevata. La sua **Recall (0.5594)**, sebbene leggermente superiore a quella della versione *nano*, rimane un

aspetto critico. Le sue metriche di **mAP** (**mAP50: 0.6463, mAP50-95: 0.3333**) sono molto competitive, quasi alla pari con il modello più performante del gruppo. Questo modello rappresenta un buon equilibrio tra le versioni più piccole e quelle più grandi di YOLO, ma la sua tendenza a mancare un numero significativo di umani potrebbe essere un limite in applicazioni dove è cruciale non perdere nessuna rilevazione.

- **YOLOv10n**

La versione *nano* di YOLOv10 è progettata per essere estremamente leggera e veloce, ideale per ambienti con risorse limitate. Le sue metriche riflettono questo compromesso: pur avendo una **Precisione (0.8437)** ancora molto buona, la sua **Recall (0.4830)** è la più bassa tra i modelli YOLO testati. Questo indica non individua quasi la metà degli umani presenti nel test set. Anche le sue metriche di **mAP** (**mAP50: 0.5871, mAP50-95: 0.2981**) sono le più basse. È una scelta valida solo se la velocità è l'unica priorità e si può tollerare di perdere molte rilevazioni, accettando un compromesso significativo sull'accuratezza complessiva.

I modelli YOLO, pur eccellendo in precisione e mantenendo la loro intrinseca velocità, mantengono una recall medio-bassa. Sembrano essere più cauti, rilevano solo gli oggetti di cui il modello pare essere molto sicuro, ignorando quelli più difficili da individuare.

8.2 Performance del modello Faster R-CNN (Resnet50)

Di seguito, viene presentata una tabella riassuntiva delle performance ottenute dal modello Faster R-CNN con backbone Resnet50 sul test set acquisito.

Modello	Precisione	Recall	F1-Score
Faster R-CNN (ResNet50)	0.7956	0.7435	0.7687

Table 5: Risultati del modello Faster RCNN con backbone Resnet50

8.2.1 Grafici dei risultati sul train/val set

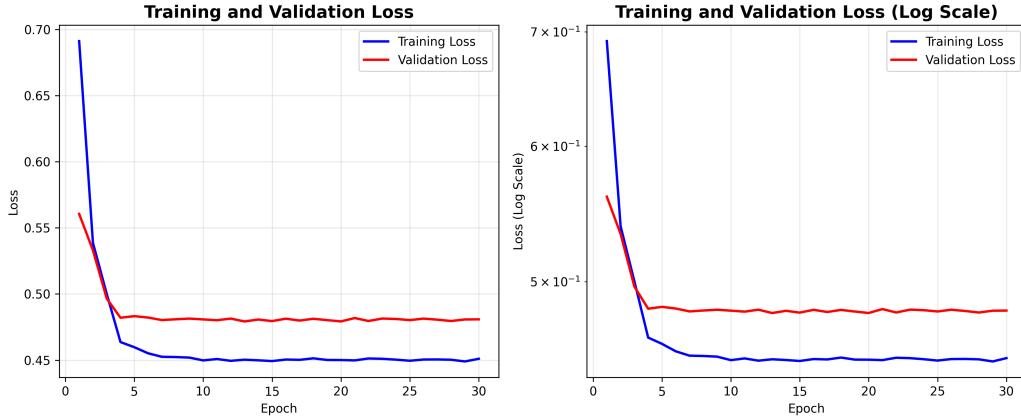


Figure 37: Curve di Training e Validation Loss per ResNet50

8.2.2 Analisi dei risultati

Il modello Faster R-CNN, equipaggiato con una backbone Resnet50, è stato addestrato per **30 epoch**. Pur essendo stato addestrato per un numero di epoch inferiori rispetto ai modelli YOLO, dimostra una performance notevole, soprattutto in termini di recall.

La sua precisione (**0.7956**) indica che quasi l'80% delle volte in cui il modello rileva un oggetto come umano/pedone, la sua predizione è corretta. Bisogna, però, spostare l'attenzione verso la sua recall (**0.7435**). Questo valore, significativamente più alto rispetto a quello dei modelli YOLO, indica che Faster R-CNN è stato in grado di identificare circa il 74% di tutti gli esseri umani effettivamente presenti nel test set. In applicazioni dove è cruciale non perdere nessuna rilevazione (ad esempio nella sorveglianza per la sicurezza o nella guida autonoma per la rilevazione di pedoni), una recall elevata è spesso preferibile. L'F1-Score (0.7687), che bilancia precisione e recall, conferma l'equilibrio complessivo delle prestazioni del modello. I conteggi dettagliati ci danno un'idea più precisa: con **432 True Positives**, il modello ha rilevato correttamente un gran numero di umani. I **149 False Negatives** sono gli umani che il modello non è riuscito a rilevare. Questo quadro suggerisce che Faster R-CNN è più "aggressivo" nel trovare gli oggetti, a costo di qualche falso positivo in più, ma con il grande vantaggio di mancare meno soggetti.

8.3 Link ai video del test set con annotazioni e ground truth

Di seguito sarà possibile trovare i link diretti ai tre video acquisiti e utilizzati come test set. In ordine abbiamo: **IMG_0555**, **IMG_0559**, **IMG_0564**. La predizione su questi video è stata fatta con YOLOv11s che ha avuto una performance migliore rispetto gli altri modelli della famiglia, e Faster R-CNN. Questo è l'elenco ai video originali:

- IMG_0555
- IMG_0559
- IMG_0564

Qui di seguito la predizione dei risultati YOLOv11s:

- IMG_0555
- IMG_0559
- IMG_0564

Ed infine i risultati di Faster R-CNN con ResNet50:

- IMG_0555
- IMG_0559
- IMG_0564

8.4 Considerazioni complessive e impatto dei fattori esterni

I risultati ottenuti si presentano promettenti. Tuttavia, si ritiene possibile che vi siano margini di miglioramento. I risultati potrebbero essere influenzati dalla differenza di angolazione delle riprese tra il set di training e quello di test. La differenza di angolazione delle riprese tra il training set e il test set è un fattore significativo. I modelli tendono a performare meglio su dati simili a quelli su cui sono stati addestrati. Inoltre, il ridimensionamento dei video potrebbe aver introdotto un deterioramento delle caratteristiche originali e della qualità iniziale, incidendo sulle performance.

9 Conclusione

In conclusione, i risultati ottenuti evidenziano come tutti i modelli analizzati siano stati in grado di rilevare con successo la presenza di persone nelle immagini, sebbene con differenze significative nelle rispettive prestazioni. La famiglia **YOLO**, nelle sue versioni **v8m**, **v10n** e **v11s**, ha dimostrato un'elevata efficacia in contesti real-time, in particolare grazie ad un alto livello di **precisione** (fino a circa **0.8965** con YOLOv8m), utile a contenere il numero di falsi positivi. Tuttavia, la **recall**, generalmente inferiore (attorno al 50–55%), suggerisce la mancata rilevazione di alcuni soggetti, soprattutto quelli meno evidenti.

Il modello **YOLOv8m** si è rivelato il più accurato in termini assoluti, mentre **YOLOv11s** ha offerto un buon compromesso tra accuratezza e efficienza computazionale, con un bilanciamento efficace tra precisione e recall. **YOLOv10n**, pur essendo molto efficiente dal punto di vista computazionale, ha mostrato le prestazioni più basse, specialmente nella capacità di rilevamento, risultando più adatto a scenari embedded o con risorse limitate.

Risulta diverso l'approccio *two-stage* adottato con **Faster R-CNN** e backbone **ResNet50** il quale ha permesso di raggiungere una **recall** sensibilmente più alta (circa **0.74**), evidenziando la capacità del modello di individuare la maggior parte dei soggetti presenti nelle immagini. Il compromesso è rappresentato da una precisione leggermente inferiore, ma comunque accettabile, e da un maggior costo computazionale.

La scelta dei modelli dipende fortemente dal contesto applicativo:

- i modelli **YOLO**, per la loro architettura single-stage, sono ideali per applicazioni real-time come sorveglianza, robotica o guida autonoma;
- **Faster R-CNN**, invece, è indicato in contesti in cui è fondamentale non perdere alcuna rilevazione, anche a costo di maggiori risorse computazionali.

Nel complesso, il sistema sviluppato ha confermato l'**attuabilità e l'efficacia** nell'impiego di tecnologie di Object Detection avanzate per il riconoscimento di pedoni in ambienti reali. L'utilizzo di un dataset acquisito manualmente ha permesso di testare i modelli in condizioni meno controllate rispetto ai benchmark standard, confermando la capacità dei modelli più recenti di adattarsi a contesti concreti.

Un ulteriore aspetto affrontato in questo lavoro è stato il **conteggio delle**

persone (*people counting*) nei video, funzione di rilievo in ambiti di sorveglianza e analisi del comportamento. Tale attività è stata realizzata a partire dai risultati dell'object detection, combinando le bounding box generate nei singoli frame. Sebbene la soluzione implementata non includa ancora un sistema avanzato di tracking multi-oggetto (come DeepSORT), il metodo di conteggio frame-by-frame ha fornito una prima valutazione dell'affidabilità dei modelli anche in contesto dinamico. Il risultato ha confermato che la **recall** è il parametro più critico ai fini del conteggio accurato, poiché ogni persona non rilevata comporta una sottostima sistematica.

Pertanto, i modelli con una recall più elevata, come **Faster R-CNN**, risultano più adatti per applicazioni dove è fondamentale evitare mancate rilevazioni, come il conteggio affidabile di flussi pedonali in ambienti reali.

9.1 Lezioni apprese

Nel corso di questo progetto, sono emerse diverse considerazioni che hanno contribuito a rafforzare la comprensione pratica dei modelli di Object Detection e delle loro implicazioni nel mondo reale:

- **Importanza della qualità e varietà dei dati:** la fase di raccolta manuale del dataset ha evidenziato quanto sia fondamentale disporre di immagini realistiche, differenti per angolazione, qualità e contesto ambientale. L'assenza di varietà nei dati di addestramento può compromettere significativamente la generalizzazione dei modelli, portandoli in uno stato di overfitting.
- **Trade-off tra accuratezza e prestazioni computazionali:** il confronto tra modelli YOLO e Faster R-CNN ha permesso di osservare direttamente come scelte architettoniche diverse influenzino la velocità, la precisione e la capacità di rilevamento. In particolare, YOLO si è rivelato più adatto a contesti real-time, mentre Faster R-CNN ha eccelso nella recall, elemento cruciale per il people counting.
- **Criticità della recall per il conteggio:** nell'implementazione del people counting basato su Object Detection, si è notato come una bassa recall sia correlata a una sottostima del numero di persone rilevate. Ciò ha messo in luce la necessità di bilanciare meglio le metriche di valutazione in funzione dell'applicazione.

- **Limiti dell'assenza di tracking:** l'adozione di una soluzione frame-by-frame, priva di tracking, ha mostrato i suoi limiti nei video dinamici, in cui la stessa persona può essere conteggiata più volte o, al contrario, non rilevata in frame successivi. Questo evidenzia il ruolo fondamentale che può avere un modulo di tracking multi-oggetto (es. DeepSORT) nei sistemi di conteggio affidabili.
- **Apprendimento sull'intero ciclo di sviluppo:** l'esperienza maturata ha fornito una panoramica completa del workflow di un progetto di Machine Learning: dalla definizione del problema, alla raccolta e annotazione dei dati, passando per l'addestramento, la valutazione e l'impiego di tecniche di interpretazione dei risultati. Questo ha permesso di consolidare le conoscenze teoriche con un forte approccio applicativo.

In sintesi, questo progetto non solo ha permesso di mettere in pratica tecniche all'avanguardia per l'Object Detection, ma ha anche fornito una visione critica sull'importanza del contesto applicativo e delle scelte progettuali, stimolando lo sviluppo di soluzioni più complete e robuste per sfide reali di Computer Vision.

9.2 Possibili miglioramenti

Sulla base dell'analisi condotta, si propongono i seguenti possibili sviluppi futuri:

- **Espansione del dataset e numero di epoche:** includere immagini con angolazioni, risoluzioni e scenari differenti per migliorare la generalizzazione del modello. Inoltre, sarebbe interessante aumentare il numero di epoche durante l'apprendimento per vedere come questo migliora le prestazioni.
- **Data Augmentation mirata:** introdurre trasformazioni prospettiche, modifiche di scala e simulazioni di condizioni reali (blur, rumore, oscuramento) per rendere il training più robusto.
- **Ottimizzazione dei parametri di inferenza:** calibrare le soglie di confidenza e le strategie di NMS (dove presenti) in funzione del contesto operativo.

- **Esplorazione di architetture avanzate:** sperimentare modelli YOLO più grandi o tecniche ibride, come DETR o modelli Transformer-based.
- **Introduzione del tracking temporale:** integrare algoritmi di tracking (es. SORT o DeepSORT) per migliorare la continuità della rilevazione nei video.
- **Ottimizzazione per ambienti embedded:** valutare tecniche di quantizzazione, pruning e accelerazione hardware per un deployment efficiente su dispositivi edge.

In definitiva, questo lavoro ha fornito un confronto approfondito tra approcci *single-stage* e *two-stage*, mettendo in evidenza vantaggi e limiti di ciascuno. I risultati ottenuti costituiscono una solida base per applicazioni reali e futuri miglioramenti, orientati verso sistemi di rilevamento sempre più **affidabili, veloci e adattabili** a diversi scenari della Computer Vision moderna.

References

- [1] Matteo Fabbri et al. *MOTSynth: How Can Synthetic Data Help Pedestrian Detection and Tracking?* , ICCV 2021.
- [2] Denise Cilia, Eleonora Giuffrida, Valeria Platania: *New Dataset For Machine Learning Project*, Link
- [3] He, K., Zhang, X., Ren, S., Sun, J. (2016) *Deep Residual Learning for Image Recognition*, In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)
- [4] Ren, S., He, K., Girshick, R., Sun, J. (2015) *Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks* NIPS'15: Proceedings of the 29th International Conference on Neural Information Processing Systems - Volume 1
- [5] Khanam, R. Hussain, M. (2024) *YOLOv11: An Overview of the Key Architectural Enhancements*.
- [6] Redmon, J. et al. (2016) *You Only Look Once: Unified, Real-Time Object Detection*.