



Università
di Catania

DIPARTIMENTO DI MATEMATICA E INFORMATICA
CORSO DI LAUREA MAGISTRALE IN INFORMATICA

Lorenzo La Rocca, Edoardo Tantari, Raffaele Terracino

Object detection su cartelli stradali europei: un confronto
fra modelli

RELAZIONE PROGETTO DI
MACHINE LEARNING

Prof: Giovanni Maria Farinella
Prof: Rosario Leonardi

Anno Accademico 2024 - 2025

Indice

1	Introduzione	3
2	Dataset	6
2.1	Roboflow	7
2.2	Kaggle	8
2.3	German Traffic Sign Detection Benchmark Dataset	8
2.4	Acquisizioni	9
2.5	Unione dei dataset	12
2.6	Statistiche del dataset	12
2.7	Organizzazione delle cartelle	15
3	Metodi	16
3.1	Data augmentaton	17
3.2	Risoluzione delle immagini	18
3.3	Architetture scelte	19
3.3.1	YOLO	19
3.3.2	Faster R-CNN	19
3.4	Risorse computazionali	20
3.5	Flusso di lavoro	20
4	Valutazione	22

<i>INDICE</i>	2
5 Esperimenti	25
5.1 YOLO	25
5.1.1 Parametri di augmentation	25
5.1.2 Parametri di training	27
5.1.3 No Augmentation	27
5.1.4 Default Augmentation	30
5.1.5 Personalized Augmentation	32
5.1.6 Confronto tra i modelli YOLO	35
5.2 Faster R-CNN	36
5.2.1 Parametri di training	37
5.2.2 Modelli addestrati	37
5.2.3 Confronto con YOLO	39
6 Demo	40
7 Codice	42
7.1 Script dataset	42
7.2 YOLO	43
7.3 Faster R-CNN	44
7.4 Demo	44
8 Conclusione	46
Bibliografia	48

Capitolo 1

Introduzione

Una delle principali applicazioni moderne dell'intelligenza artificiale si ritrova in ambito automobilistico. Si pensi per esempio alla guida assistita o autonoma, settori in continua crescita e che pongono numerose sfide in ambito legale, etico e tecnico [1]. Un veicolo a guida autonoma deve essere capace di elaborare l'ambiente che lo circonda e prendere decisioni che garantiscano la sicurezza del passeggero, di altri conducenti e dell'ambiente circostante. Algoritmi avanzati di computer vision, basati su modelli di deep learning, permettono a un veicolo di identificare persone ed oggetti in tempo reale, in modo tale da prendere decisioni sulla dinamica del veicolo [2]. Si pensi per esempio all'attraversamento pedonale: se il semaforo diventa rosso, il veicolo deve rallentare e fermarsi per permettere ai pedoni di attraversare. Deve essere anche capace di rilevare i pedoni, in quanto potrebbe succedere che un pedone attraversi la strada in un punto in cui non sono presenti strisce. In questo caso è di fondamentale importanza il tempo di elaborazione e la successiva capacità di rallentare, fermarsi o cambiare strada per evitare danni al pedone. La capacità di identificare oggetti in una scena e classificarli richiede l'utilizzo di algoritmi di object detection. In particolare, identificare

segnali stradali è fondamentale per ridurre il rischio di incidenti e non violare le regole della stada. Il problema che si pone per lo sviluppo del progetto è quindi di costruire e confrontare diversi modelli di object detection che siano in grado di rilevare e classificare segnali stradali. Nel contesto del progetto è stato scelto un sottoinsieme di 19 classi tra tutti i possibili segnali stradali europei. Questa scelta è motivata dalle risorse di calcolo a disposizione dei membri del team e dall'importanza dei segnali scelti, che sono i seguenti:

- segnale di direzione obbligatoria a sinistra;
- segnale di direzione obbligatoria a destra;
- segnale di dare precedenza;
- segnale di divieto di accesso;
- segnale di stop;
- semaforo di colore verde;
- semaforo di colore rosso;
- segnale di limite di velocità da 20 a 120;

Nel problema dell'object detection, si definisce bounding box una sottoimmagine in cui è presente l'oggetto di interesse. Un bounding box è descritto da una quadrupla (x_1, y_1, x_2, y_2) contenente le coordinate di inizio e fine del bounding box. Pertanto, data in input un'immagine RGB, per ogni segnale stradale presente nell'immagine il modello deve predire le coordinate del bounding box e la classe di appartenenza relativa. La figura 1.1 mostra un esempio per il problema descritto.



Figura 1.1: Un esempio di riconoscimento di due cartelli stradali. Ad ogni oggetto individuato corrisponde uno score, che è il più alto tra quelli individuati dal modello tra tutte le classi. La foto è stata scattata nei pressi di Acireale (CT).

Capitolo 2

Dataset

Il primo passo di ogni progetto di machine learning è sempre la raccolta dati. La raccolta di un corposo dataset di qualità è fondamentale per ottenere buoni risultati in fase di addestramento di un modello di machine learning. Si è cercato di raccogliere immagini di segnali stradali a diversa distanza, in modo tale da avere più dati realistici. Difatti, un veicolo a guida autonoma deve essere in grado di riconoscere il segnale a lunga distanza, in quanto riconoscerlo solo quando è vicino potrebbe avere conseguenze fatali. Si pensi per esempio all’attraversamento pedonale citato nella sezione precedente. Per quanto riguarda la nomenclatura delle classi si è deciso di usare la terminologia inglese, separando con un underscore (_) i nomi di segnali composti da due parole. Per esempio, la classe relativa al segnale ”dare precedenza” è stata denominata *make_way*. I dati raccolti per lo sviluppo dei modelli si possono dividere in due categorie. La prima è costituita da dati raccolti per il web, in particolare dalle piattaforme Kaggle e Roboflow e dal sito web del German Traffic Sign Benchmark [3]. La seconda categoria consiste in acquisizioni effettuate dal team, da usare in fase di test dei modelli costruiti.

2.1 Roboflow

Roboflow è una piattaforma online che facilita lo sviluppo di applicazioni di Computer Vision, permettendo di caricare dataset ed etichettarli facilmente [4]. Roboflow ha a disposizione una grande community che pubblica giornalmente dataset accessibili a tutti, secondo licenze aperte. Da tale piattaforma sono stati raccolti numerosi dataset già pronti. Per ogni dataset sono state selezionate le immagini delle classi di interesse. La seguente lista indica i dataset raccolti su roboflow e il numero di immagini selezionate:

- ”Road sign” [5], 499 immagini;
- ”3423s3r” dataset [6], 87 immagini;
- ”car” dataset [7], 956 immagini;
- ”road sign detection” dataset [8], 353 immagini;
- ”Traffic Signs and Traffic Lights” dataset [9], 375 immagini;
- ”Mapillary” dataset [10], 1964 immagini;
- ”Trafic_Signal_Collect” dataset [11], 547 immagini;
- ”speed” dataset [12], 108 immagini;
- ”Traffic and Road Signs” dataset [13], 1355 immagini;

Per esempio, dal dataset ”speed” sono state selezionate soltanto le immagini raffiguranti il limite di velocità 110, segnale poco presente negli altri dataset.

2.2 Kaggle

Kaggle è una delle più importanti piattaforme dedicate al Machine Learning, che permette di caricare dataset e progetti accessibili secondo varie licenze [14]. Il dataset raccolto per il progetto da tale piattaforma è il "Traffic sign detection" dataset disponibile al link [15], distribuito con licenza CC BY 4.0 International che ne permette l'utilizzo con il dovere di citare l'autore. Il dataset consiste in 4969 immagini etichettate secondo il formato Ultralytics YOLO [16]. Le classi presenti sono: segnale di stop, semaforo verde e rosso, limiti di velocità da 10 a 120. In accordo con le classi scelte per il progetto, le immagini relative al limite di velocità 10 sono state eliminate.

2.3 German Traffic Sign Detection Benchmark Dataset

Il German Traffic Detection Sign Benchmark (GTDSB)[3] è un progetto di ricerca del 2013 riguardo la classificazione di segnali stradali. Il progetto offre un dataset gratuito di 900 immagini di segnali stradali tedeschi, con immagini in formato PPM e annotazioni in formato CSV. Tutte le immagini raffigurano segnali stradali scattati a media o lunga distanza. Dal dataset sono state selezionate 117 immagini in modo tale da avere più dati disponibili per alcune classi, in particolare per le classi *make-way*, *pedestrian-crossing*, *no-entry*, *stop*, *turn-left* e *turn-right*.

2.4 Acquisizioni

La seconda categoria di dati raccolti consiste in foto scattate dal team per diversi comuni della Sicilia, per un totale di 491 immagini raccolte. 486 foto sono state scattate nei comuni e nei dei dintorni di:

- Acireale (CT)
- Campofelice di Roccella (PA)
- Catania
- Cefalù (PA)
- Gagliano Castelferrato (EN)
- Termini Imerese (PA)

Oltre i comuni citati, 4 foto provengono dalle città di Livorno e una da Pisa, concesse da una residente del luogo. Dove possibile si è cercato di effettuare 3 tipi di scatti: da vicino, a media distanza e da lontano. Si è cercato anche di scattare foto dove fossero presenti più segnali, anche di classi diverse. Per alcune classi, non è stato possibile acquisire molti esempi a causa della rarità del segnale o nella difficoltà nella sua acquisizione. Per esempio i segnali di limite di velocità superiori al 90 sono principalmente presenti in autostrada, pertanto è risultato difficile per i membri del team reperire esempi di queste classi. Per motivi di privacy, le immagini sono state modificate per oscurare targhe e volti, dove necessario, mediante i software GIMP ed Apple Intelligence. La figura 2.1 riporta degli esempi di acquisizioni effettuate. Le foto scattate sono state caricate su Roboflow ed etichettate grazie agli strumenti messi a disposizione dalla piattaforma. In accordo con gli obiettivi concordati per il progetto, le foto acquisite non sono state utilizzate utilizzate in

fase di training dei modelli. L’obiettivo è di testare i modelli costruiti su immagini “reali”, come se venissero da una camera montata su un veicolo. I dati acquisiti dal team si prestano a questo compito. La tabella che segue riporta il numero di esempi acquisiti per ogni classe. Le classi meno numerose risultano essere gli *speed_limit* da 80 a 110, comunemente presenti in autostrada. Sfortunatamente non è stato possibile effettuare scatti del segnale *speed_limit_120*.

Class Name	Count
no_entry	112
stop	107
green_light	35
pedestrian_crossing	64
speed_limit_30	62
red_light	56
make_way	55
turn_left	35
turn_right	31
speed_limit_50	35
speed_limit_40	24
speed_limit_70	10
speed_limit_20	8
speed_limit_90	6
speed_limit_80	3
speed_limit_110	2
speed_limit_100	1
speed_limit_60	1

Tabella 2.1: Campioni acquisiti per ogni classe



(a) Divieto di accesso da vicino



(b) Divieto di accesso a media distanza



(c) Divieto di accesso da lontano



(d) Doppio divieto di accesso da lontano

Figura 2.1: Esempi di foto scattate nei pressi del centro di Catania. Le targhe sono state oscurate mediante il filtro "effetto pixel" del software GIMP.

2.5 Unione dei dataset

I dati raccolti da Kaggle, dal sito web del GTSDB e da Roboflow sono caricati su quest'ultima piattaforma, permettendo così di gestirli con una interfaccia comune. Facendo ciò è stato possibile rinominare le classi, visualizzare le immagini raccolte, correggere alcune immagini che non erano state etichettate correttamente ed esportare i dati nei formati desiderati. Roboflow organizza i progetti, di cui i dataset fanno parte, in workspace. Ogni workspace ha un limite di 10000 immagini. Poichè il numero totale di immagini dalle fonti citate supera questo limite, sono stati creati più workspace, ognuno contenente un sottoinsieme del dataset. I dataset sono stati successivamente scaricati e uniti mediante script Python. Il dataset risultante è stato poi suddiviso randomicamente mediante script come: training set 70%, validation set 20% e test set 10%. Infine, al test set sono state aggiunte le 491 aquisite dal team, costruendo così il dataset finale utilizzato durante il progetto.

2.6 Statistiche del dataset

Il dataset finale è suddiviso come segue:

- training set: 8.117 immagini;
- validation set: 2.319 immagini;
- test set: 1650 immagini.

La tabella e i grafici che seguono riportano il numero di esempi per ogni classe e per ogni split.

Class	Train	Valid	Test
green_light	743	203	123
make_way	583	167	121
no_entry	283	71	151
pedestrian_crossing	528	152	137
red_light	678	180	167
speed_limit_100	344	86	49
speed_limit_110	173	56	14
speed_limit_120	321	85	44
speed_limit_20	284	80	45
speed_limit_30	705	209	176
speed_limit_40	326	75	67
speed_limit_50	651	195	125
speed_limit_60	461	119	75
speed_limit_70	505	156	91
speed_limit_80	509	181	68
speed_limit_90	569	176	85
stop	588	156	215
turn_left	358	103	97
turn_right	504	131	113

Tabella 2.2: Distribuzione dei dati per classe nei set di training, validazione e test.

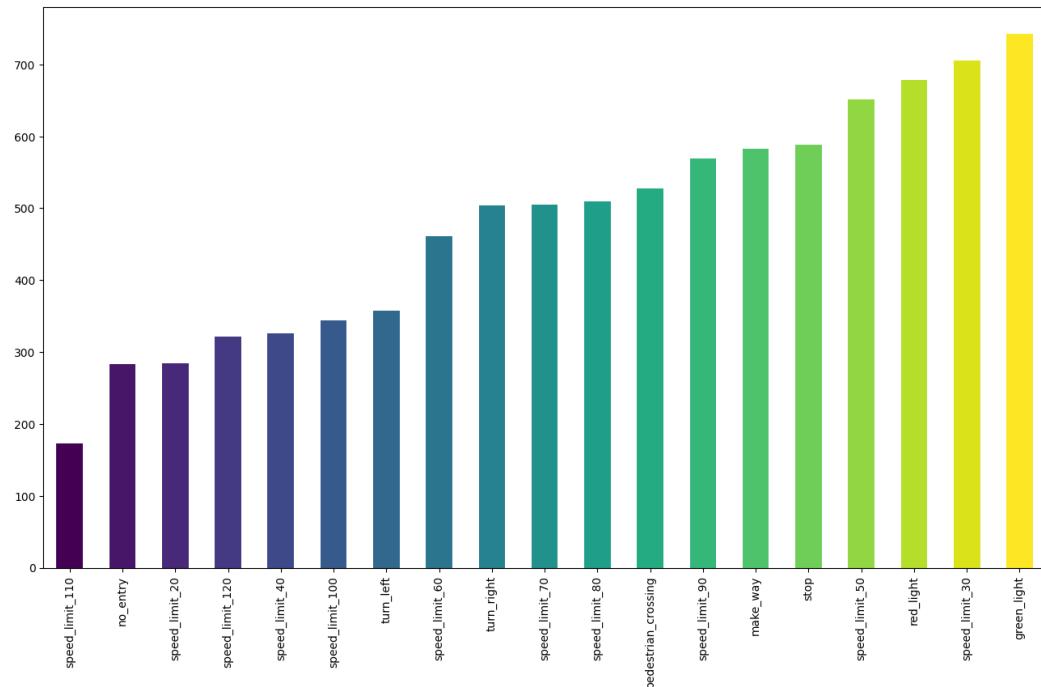
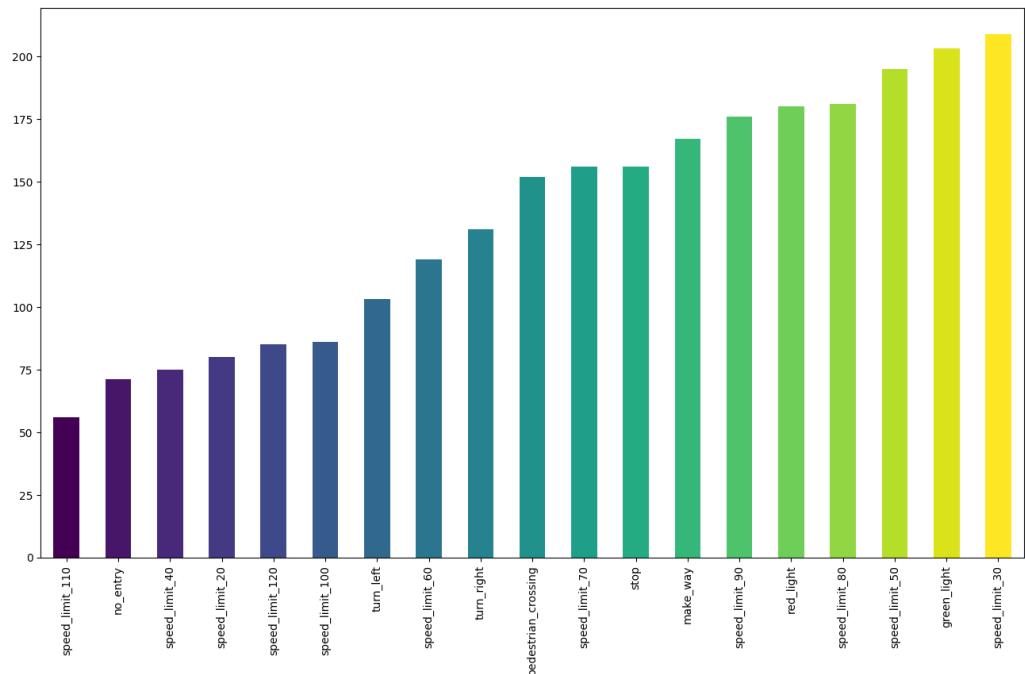
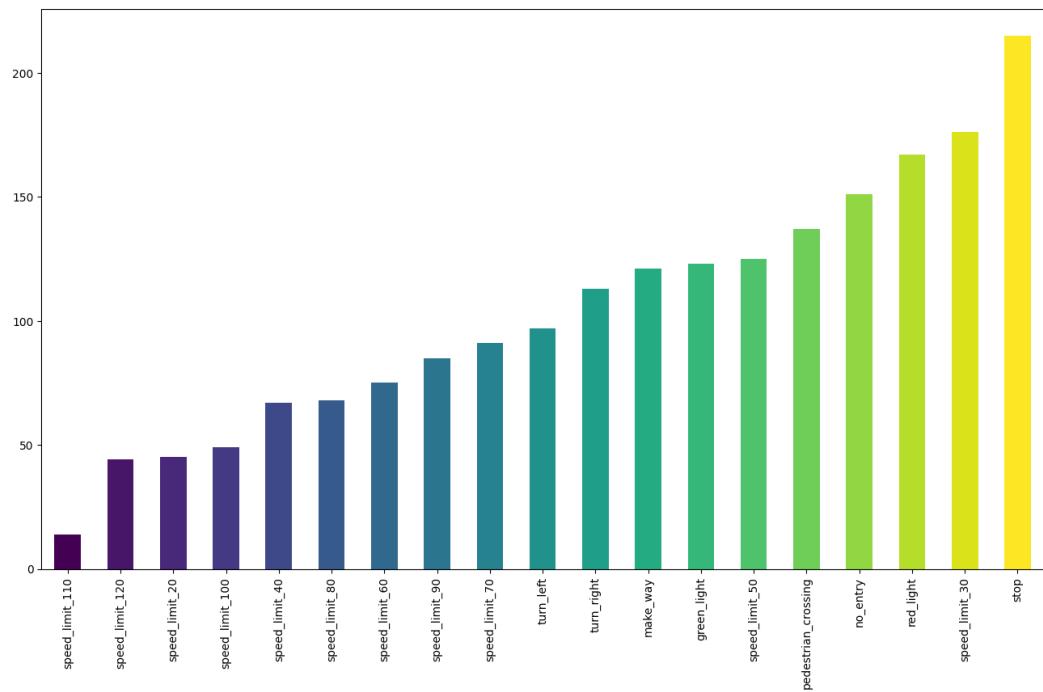


Figura 2.2: Distribuzioni degli esempi per classe nel training set

**Figura 2.3:** Distribuzioni degli esempi per classe nel validation set**Figura 2.4:** Distribuzioni degli esempi per classe nel test set

2.7 Organizzazione delle cartelle

L’organizzazione delle directory consiste in una cartella ”dataset” contenente due versioni dei dati raccolti. La prima, chiamata dataset_yolo, contiene il dataset in formato YOLO. A sua volta, tale cartella include le cartelle ”train”, ”valid” e ”test” e il file ”data.yaml”, contenente i nomi delle classi. Ognuna delle tre cartelle è suddivisa in due cartelle, ”images” e ”labels”. La cartella ”images” contiene le immagini mentre la cartella ”labels” consiste di file txt contenenti le etichette per ogni immagine. Ad ogni immagine corrisponde un file txt con lo stesso nome. La seconda cartella, ”dataset_PVOC”, contiene i dati in formato Pascal VOC XML. Al suo interno vi sono tre cartelle: JPEGImages, contenente le immagini, Annotations, contenente le annotazioni e, infine, ImageSets, contenente dettagli sulla suddivisione del dataset in training set, validation set e test set.

Capitolo 3

Metodi

Il problema dell’object detection si può affrontare mediante due principali strategie: one-stage e two-stage. Nelle architetture one-stage, una singola rete neurale predice sia i bounding box che le probabilità delle classi, direttamente dall’immagine in una sola valutazione. In quelle two-stage, dapprima si generano delle ”region proposals”, ovvero delle regioni candidate a contenere oggetti, indipendentemente dalle classi di appartenenza. Successivamente, a partire da queste proposte, si effettua la classificazione e si affinano le coordinate dei bounding box. In generale le architetture one-stage risultano molto più veloci ma meno precise. Al contrario, le architetture two-stage sono computazionalmente pesanti ma offrono migliori risultati. In contesti come la guida autonoma è fondamentale trovare un compromesso tra velocità di calcolo e correttezza delle operazioni. Un veicolo a guida autonoma deve anche effettuare varie operazioni di diagnostica, fondamentali per il corretto funzionamento del veicolo, pertanto non può impiegare decine di secondi per identificare un singolo oggetto. Per questo motivo, considerate anche le risorse computazionali del team, per il progetto è stato deciso di concentrarsi maggiormente su architetture one-stage, costruendo vari modelli di questo tipo e

due modelli two-stage. Per la costruzione dei modelli one stage si sono tenuti in considerazione due importanti aspetti, fondamentali in fase di training: data augmentation e risoluzione delle immagini.

3.1 Data augmentation

Per data augmentation si intende una tecnica mirata ad affrontare il problema dell'overfitting. Consiste nel generare dati sintetici in fase di training a partire dai dati a disposizione. Per esempio, se l'input è un'immagine è possibile fare data augmentation applicando una o più trasformazioni affini, per esempio un ribaltamento orizzontale e/o verticale, mantendendo la stessa etichetta originale. Così facendo è possibile aumentare le dimensioni del training set. Poichè avere più dati di training potrebbe aiutare a prevenire l'overfitting, la data augmentation è una tecnica molto diffusa. Tuttavia nel contesto del problema del progetto, alcune tecniche di augmentation potrebbero essere dannose. Si pensi per esempio al ribaltamento orizzontale: poichè viene mantenuta la stessa etichetta dell'immagine originale, ribaltando orizzontalmente un'immagine di classe *turn_right* e mantenendo la stessa etichetta sull'immagine creata, si avrebbe in realtà un'immagine *turn_left* etichettata come *turn_right*. Similmente, ribaltare verticalmente un'immagine di classe *speed_limit_90* produrrebbe un'immagine di classe *speed_limit_60* etichettata come *speed_limit_90*. Non facendo attenzione a quali tecniche di augmentation possano provocare simili effetti, si introduce nel training set un insieme di immagini mal etichettate che potrebbero incidere sulla qualità del modello allenato. Tuttavia, per alcune classi è chiaro che l'utilizzo di tali tecniche potrebbe invece migliorare il modello, come nel caso delle classi *pedestrian_crossing* e *no_entry*. In particolare, essendo il segnale di

no_entry un rettangolo bianco su sfondo rosso, fare augmentation con ribaltamenti verticali e orizzontali non cambia la forma del cartello. Ogni modello one-stage costruito è basato su diverse configurazioni di data augmentation e di risoluzione delle immagini in fase di training. In particolare si considerano tre configurazioni. La prima è denominata *no_augmentation* ed esclude l'utilizzo di qualsiasi tecnica di data augmentation in fase di training. La seconda, *default_augmentation*, consiste nell'impostazione predefinita di data augmentation consigliate dagli sviluppatori dell'architettura scelta. La terza, *personalized_augmentation*, consiste in parametri di augmentation selezionati dal team del progetto. Ogni configurazione è stata salvata in un file YAML, in modo tale da avere una singola interfaccia comune per caricare la configurazione scelta ed allenare il modello. Tutti i parametri configurati sono spiegati nel dettaglio nel capitolo 5.

3.2 Risoluzione delle immagini

Effettuare un ridimensionamento delle immagini prima del training permette di avere le immagini a una risoluzione comune, parametro che incide sul tempo di addestramento dei modelli. Per ognuno dei modelli one-stage si è scelto di effettuare due addestramenti separati: il primo con immagini a risoluzione 416x416 e il secondo con immagini di risoluzione 640x640. Entrambe le risoluzioni sono comuni nel contesto dell'object detection. Non sono state scelte risoluzioni superiori a causa delle risorse computazionale a disposizioni del team. I modelli vengono testati su immagini a risoluzione concorde con quella usata in fase di training.

3.3 Architetture scelte

3.3.1 YOLO

Tra le architetture one-stage allo stato dell'arte, è stato scelto di utilizzare YOLO[17]. In particolare si è scelta la versione 12, rilasciata a febbraio 2025[18], nella dimensione "medium", consistente in 292 layers e 20.199.168 parametri[19]. YOLOv12 è stato addestrato sul dataset COCO[20] 2017 ed è disponibile attraverso la libreria Ultralytics , che consente di scaricare il modello pre-addestrato e utilizzarlo attraverso codice Python. La data augmentation in Ultralytics YOLO vengono applicate in modalità "online", cioè durante le epoche di training in modo dinamico, generando nuove varianti casuali di ogni immagine in tempo reale. Questo significa che il modello vede sempre nuove varianti di ogni immagine a ogni epoca, massimizzando la diversità dei dati senza aumentare permanentemente le dimensioni del dataset [21]. La funzione di loss di YOLO è composta da 3 parti. La prima, detta box loss, misura quanto bene il modello sia in grado di predire i bounding box. La seconda, detta cls loss, indica quanto bene il modello sia in grado di classificare gli oggetti del bounding box. Infine, la terza parte, detta dfl loss, permette di raffinare le coordinate del bouding box.

3.3.2 Faster R-CNN

Come architettura two-stage è stata scelta Faster R-CNN[22], il cui checkpoint è messo a disposizione da Pytorch, libreria Python per il Deep Learning. Similmente a YOLO, la funzione di loss di Faster R-CNN è composta da due parti. La prima parte è utile alla classificazione, mentre l'altra è utile alla predizione dei bounding box.

3.4 Risorse computazionali

Per l’addestramento dei modelli sono state utilizzate due tipi di risorse a disposizione dei membri del team. La prima è Google Colab, una piattaforma online che permette di eseguire notebook Jupyter, scritti in Python, sfruttando potenti risorse computazionali messe a disposizione da Google, gratuitamente o a pagamento. Con la versione Pro, il team ha avuto accesso alle schede video NVIDIA A100 e NVIDIA T40. L’accesso a tali GPU è limitato, pertanto si è utilizzato Colab per gli addestramenti meno onerosi. La seconda risorsa consiste in due macchine messe a disposizione dal membro del team Tantari, le cui specifiche sono elencate in tabella 3.1. Queste due macchine sono state utilizzate per i modelli più costosi computazionalmente.

Componente	PC 1	PC 2
CPU	Intel i9-11900F	Intel i7-7700X
RAM	64 GB	32 GB
GPU	RTX 4090 (24 GB VRAM)	RTX 3080 Ti (12 GB VRAM)

Tabella 3.1: Componentistica delle macchine utilizzate per alcuni modelli

3.5 Flusso di lavoro

L’input del training per un modello è composto da:

- training set;
- numero di epoche;
- parametro di resize delle immagini;
- parametri di training, come batch size e learning rate

- per i modelli YOLO, file di configurazione per la data augmentation.

Per ognuna delle due risoluzioni scelte per il ridimensionamento e per ognuna delle tre configurazioni di augmentation, si è allenato un modello YOLO, arrivando a un totale di 6 modelli YOLO costruiti. La libreria Ultralytics YOLO opera nel modo seguente. Alla fine del training vengono prodotti due file: il primo, `last.pt`, è l'insieme dei pesi del modello allenato all'ultima epoca; il secondo, `best.pt`, è l'insieme dei pesi del modello relativamente all'epoca che ha fornito i migliori risultati sul validation set. Questa strategia aiuta a prevenire l'overfitting, pertanto nel contesto del progetto i pesi considerati sono quelli del `best.pt`. Per quanto riguarda Faster R-CNN, a causa delle ingenti risorse computazionali richieste per il training, sono stati allenati due modelli con questa architettura, relativamente ai due parametri di ridimensionamento. Ogni modello è stato allenato monitorando la convergenza delle loss attraverso le piattaforme Tensorboard e Weight and Biases. A ogni epoca sono state salvate le metriche sul validation set. Una volta concluso l'addestramento di un singolo modello, sono state effettuate delle osservazioni sulle curve di loss per decidere se allenare ulteriormente oppure no. Infine, sono state calcolate le metriche sul test set. In tutto, sono stati allenati 8 modelli di object detection.

Capitolo 4

Valutazione

Le misure di valutazione utilizzate nella fase sperimentale sono le seguenti:

- Precision e Recall
- F_1 score
- Mean Average Precision.

Per il calcolo di queste metriche si parte dalla Intersection over Union (IoU)[23].

A partire da una predizione, per il calcolo dell'IoU si considerano il bounding box predetto e il bounding box reale. Si calcola dapprima l'area della sovrapposizione dei due bounding box e successivamente l'area dell'unione. Il rapporto tra le due aree dà la IoU:

$$IoU = \frac{\text{Area of overlap}}{\text{Area of union}}$$

Fissata una soglia t , si ha che una predizione è un:

- True Positive se $IoU > t$ e la classe predetta corrisponde a quella reale;
- False Positive se: $IoU < t$, oppure la classe predetta è errata oppure il modello predice un bounding box in una regione priva di oggetti reali;

- False Negative se il modello non effettua nessuna detection in una regione in cui è presente un oggetto reale. [23]

Precision e Recall vengono pertanto calcolate come:

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

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

Un'alta precision indica una bassa percentuale di falsi positivi, mentre un'alta recall indica una bassa percentuale di falsi negativi. Precision e Recall vengono calcolate per ogni classe e mediate. In base al contesto d'utilizzo, si può scegliere di prediligere l'una o l'altra. La media geometrica di Precision e Recall è chiamata F_1 score e si calcola come:

$$F_1 = 2 * \frac{\text{precision} * \text{recall}}{\text{precision} + \text{recall}}$$

Per ottenere un alto F_1 score è necessario avere un'alta Precision e un'alta Recall, pertanto questa metrica consente di valutare Precision e Recall insieme, con un singolo valore. Nel contesto del progetto, è importante avere un valore alto di F_1 score. Per comprenderne il motivo, si supponga di costruire un modello con un'alta Precision ma una bassa Recall. Ciò vuol dire che il modello rileverebbe solamente i segnali per i quali è particolarmente certo, finendo però per non riconoscere molti dei segnali presenti nella strada. Implementando tale modello in un veicolo a guida autonoma, esso potrebbe violare le norme del codice stradale e causare incidenti. Al contrario, si supponga di costruire un modello con una bassa Precision ma un'alta Recall. Questo modello riconoscerebbe la maggior parte dei segnali presenti, anche a costo di sbagliare. Per esempio, un segnale di stop potrebbe essere scambiato per il

segna di divieto di accesso e questo potrebbe portare, anche in questo caso, ad incidenti. Pertanto la metrica F_1 si presta ad essere un buon indicatore. L'ultima metrica considerata per la valutazione dei modelli è la Mean Average Precision (MAP), parecchio utilizzata nel campo dell'object detection. Il suo calcolo si basa sui concetti di curva Precision-Recall e di Average Precision (AP). La curva Precision-Recall è una curva che mostra l'andamento della Precision in funzione della Recall al variare della soglia del classificatore. L'Average Precision è l'area sotto questa curva, calcolata come:

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

dove $p(r)$ è la funzione che rappresenta la Precision al variare della Recall. Pertanto la AP è un indicatore della bontà del classificatore indipendentemente dalla soglia di classificazione. La MAP si ottiene calcolando la AP per ogni classe e facendo la media. Supponendo di avere K classi $\{1, \dots, K\}$ la MAP si calcola con la seguente formula

$$MAP = \frac{1}{K} \sum_{i=1}^K AP_i$$

Si distinguono principalmente due tipi di MAP. La prima è la MAP-50, calcolata fissando una soglia t di IoU pari a 0.5. La seconda è la MAP 50-95 che è la media delle AP calcolate su 10 soglie di IoU nell'intervallo [0.50, 0.95] con passo 0.05. Le metriche descritte sono tenute in considerazione per la descrizione dei risultati ottenuti, riportati nella sezione successiva.

Capitolo 5

Esperimenti

In questa sezione si vanno a descrivere nel dettaglio i parametri utilizzati nel training di ognuno dei modelli descritti. Per ognuno dei modelli vengono riportate le metriche di valutazione descritte nella sezione precedente. **L'obiettivo, per i modelli YOLO, è capire come diverse tecniche di data augmentation influiscano sui risultati, a parità di parametri di training.** Il miglior modello YOLO viene poi confrontato con il miglior modello Faster R-CNN. Per ogni modello prodotto vengono riportate le curve di loss su training e validation set, le metriche e le curve Precision-Recall, per poter confrontare nel dettaglio tutti i modelli prodotti. Le metriche vengono calcolate usando come valore di *confidence* per l'inferenza 0.25.

5.1 YOLO

5.1.1 Parametri di augmentation

I parametri di augmentation previsti da Ultralytics YOLO sono riportati nella tabella seguente, che include anche una descrizione e i valori di default previsti dalla libreria.

Parametro	Descrizione	Default
hsv_h	Modifica il valore di tonalità (H) nel spazio colore HSV	0.015
hsv_s	Modifica il valore di saturazione (S) nel spazio colore HSV	0.7
hsv_v	Modifica il valore di luminosità (V) nello spazio colore HSV	0.4
degrees	Ruota l'immagine casualmente nell'intervallo specificato in gradi	0.0
translate	Trasla l'immagine in orizzontale e verticale di una frazione della dimensione originale, in base alla magnitudo della traslazione specificata	0.1
scale	Ridimensiona casualmente l'immagine di un fattore nel range specificato	0.5
shear	Inclina l'immagine con angolo nel range specificato	0.0
perspective	Applica una trasformazione prospettica all'immagine secondo la magnitudo specificata.	0.0
flipud	Capovolge verticalmente l'immagine secondo la probabilità specificata.	0.0
fliplr	Capovolge orizzontalmente l'immagine secondo la probabilità specificata.	0.5
bgr	Inverte i canali RGB dell'immagine secondo la probabilità specificata	0.0
mosaic	Combina 4 immagini in una sola, secondo la probabilità specificata	1.0
mixup	Effettua una media di due immagini secondo la probabilità specificata	0.0
cutmix	Sovrappone una regione di un'immagine su un'altra secondo la probabilità specificata	0.0
auto_augment	Applica una policy di augmentation automatica	randaugment
erasing	Erode porzioni dell'immagine secondo la probabilità specifica	0.4

Tabella 5.1: Parametri di data augmentation in Ultralytics YOLO con valori di default

5.1.2 Parametri di training

I parametri utilizzati per il training dei modelli YOLO sono i seguenti:

Parametro	Descrizione	Valore fissato
epoch	Numero di epoche	100
lr0	Learning rate iniziale	0.01
lrf	Learning rate finale come frazione di quello iniziale ($lr0 * lrf$)	0.01
batch	Dimensione del mini batch utilizzato nella SGD	-1 (auto)
momentum	Fattore di momentum per la SGD	0.937
weight_decay	Termine di regolarizzazione L_2	0.0005

Tabella 5.2: Parametri di training fissati per i modelli YOLO

5.1.3 No Augmentation

La configurazione No Augmentation corrisponde all'utilizzo di nessuna tecnica di data augmentation in fase di training, impostando ogni parametro di augmentation al valore nullo. Questo esperimento consente di valutare la capacità del modello di generalizzare a partire dai dati originali, prima di introdurre tecniche di data augmentation che ne alterino la distribuzione. Un buon comportamento del modello in questo caso potrebbe indicare che il dataset sia già sufficientemente rappresentativo. Inoltre, questo esperimento permette di quantificare l'effetto netto delle tecniche di data augmentation dei prossimi esperimenti. Come descritto precedentemente, questa configurazione è stata usata per allenare due modelli YOLO, uno con risoluzione di immagini di training pari a 416x416, chiamato *NoAug416*, e l'altro con risoluzione 640x640, chiamato *NoAug640*. Entrambi i modelli sono stati allenati su Google Colab, utilizzando la GPU NVIDIA A100-SXM4-40GB. NoAug416 ha richiesto 1 ora e 21 minuti per il training, mentre NoAug640 ha impiegato 4 ore e 10 minuti.

Le loss su training set e validation set per entrambi i modelli sono mostrate in figura 5.1, prodotta dal software "Weights and Biases" durante il training. Si consideri NoAug416. Osservando le loss di training per tale modello, si nota che le curve presentano un andamento decrescente e regolare. In particolare, la box_loss non è ancora arrivata a convergenza, mentre la classification_loss e la dfl_loss sembrano essersi stabilizzate. Sul validation set, nelle fasi finali del training la cls_loss e la box_loss per il validation set tendono a stabilizzarsi, con fluttuazioni contenute, mentre la curva dfl_loss assume un comportamento in salita a partire dalla settantesima epoca. Pertanto, proseguire con il training potrebbe non essere conveniente. Si consideri adesso il secondo modello, chiamato NoAug640. Per questo modello, dopo 100 epochhe la box_loss e la dfl_loss sul training set non ancora arrivate a convergenza, a differenza della cls_loss invece sembra essersi stabilizzata. Tuttavia, le 3 loss sul validation set sono arrivate a convergenza, con un comportamento stabile già a partire dalla settantesima epoca. Anche in questo caso è giustificato interrompere il training. Per quanto riguarda le metriche, vengono riportati i risultati in tabella 5.3. NoAug640 produce risultati di gran lunga migliori rispetto a NoAug416. Vi è infatti un incremento della MAP0.5 di 0,049 sul test set. Questo incremento delle performance potrebbe essere dovuto al fatto che, aumentando la risoluzione delle immagini, ovvero la quantità di informazioni spaziali, risulta più probabile riuscire a distinguere oggetti lontani o piccoli. In generale ogni metrica risulta migliore. Pertanto, al costo di 3 ore in più di utilizzo della GPU per il training, si è ottenuto un modello migliore. Questo evidenzia un chiaro trade-off tra performance e costo computazionale: l'aumento della risoluzione comporta un carico maggiore in termini di risorse e tempo di addestramento, ma può risultare vantaggioso in termini di capacità predittiva, soprattutto in contesti in cui la rilevazione di oggetti piccoli o distanti è

cruciale come nel caso di un veicolo a guida autonoma. Il miglioramento delle performance viene ulteriormente confermato dalle curve Precision-Recall, mostrate in figura 5.2. Si osserva, infine, una discrepanza tra le metriche ottenute sul validation set e quelle sul test set. Tale differenza è verosimilmente attribuibile alla presenza, nel test set, di immagini acquisite direttamente dal team, caratterizzate da una risoluzione superiore rispetto a quella del resto del dataset. Durante il calcolo delle metriche, sia per il validation set che per il test set, è stata mantenuta la stessa risoluzione utilizzata in fase di addestramento (416×416 o 640×640), al fine di garantire coerenza tra training e inferenza. Tuttavia, nel caso delle immagini ad alta risoluzione, un aumento del parametro di resize durante l'inferenza potrebbe permettere una migliore preservazione dei dettagli e, di conseguenza, un incremento delle performance. Anche una calibrazione mirata della soglia di confidence potrebbe contribuire al miglioramento dei risultati.

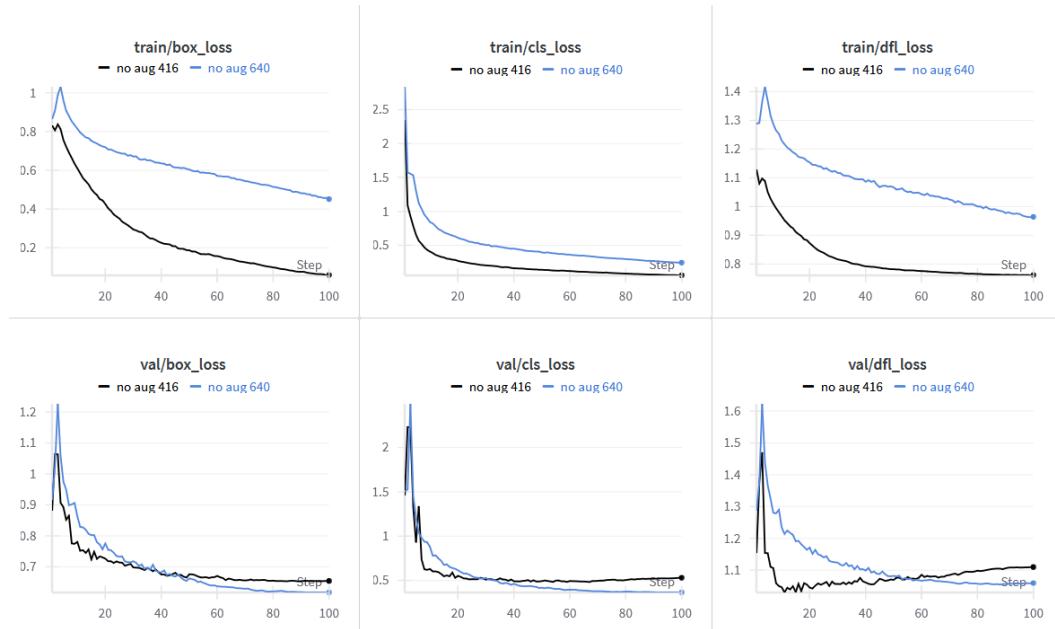


Figura 5.1: Curve di loss per l'esperimento NoAug

Validation Set					
Modello	Precision	Recall	F_1 Score	mAP@0.5	mAP@50-95
YOLOv12-NoAug 416	0.933	0.831	0.879	0.898	0.772
YOLOv12-NoAug 640	0.921	0.909	0.915	0.945	0.832
Test Set					
Modello	Precision	Recall	F_1 Score	mAP@0.5	mAP@50-95
YOLOv12-NoAug 416	0.875	0.665	0.755	0.797	0.664
YOLOv12-NoAug 640	0.892	0.744	0.811	0.846	0.723

Tabella 5.3: Metriche dei modelli YOLO no-augmentation calcolate su validation set e test set

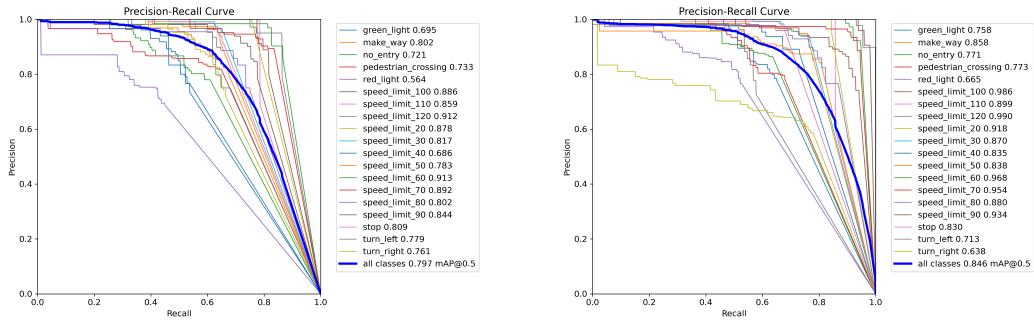


Figura 5.2: Curve Precision Recall per NoAug416 (sinistra) e NoAug640 (destra)

5.1.4 Default Augmentation

La configurazione Default Augmentation consiste nell'utilizzare i parametri di augmentation impostati di default dagli sviluppatori di YOLO, così come descritto nella tabella 5.2. Pertanto questa configurazione di data augmentation include anche operazioni potenzialmente dannose per alcune classi, come il ribaltamento orizzontale. Il modello allenato su immagini a risoluzione 416x416 è denominato *DefaultAug416*, mentre quello allenato su immagini 640x640 prende il nome di *DefaultAug640*. I modelli sono stati allenati rispettivamente sulle macchine della tabella 3.1. La figura 5.3 riporta le tre loss su training e validation set, per i due modelli alle risoluzioni scelte, chia-

mati rispettivamente DefaultAug416 e DefaultAug640. Si nota subito che, sia sul training set che sul validation set, le box_loss e le cls_loss dei due modelli quasi si sovrappongono. Al contrario, sebbene il comportamento sia simile, la dfl_loss è diversa per i due modelli su entrambi i set. Tutte e 3 le curve di validation si sono stabilizzate, pertanto non vi sono segni di overfitting. Dalle metriche della tabella 5.4, si nota che DefaultAug640 presenta performance migliori di DefaultAug416. Pertanto, anche in questo esperimento, allenare il modello su immagini a maggior risoluzione ha portato ad un miglioramento delle performance.

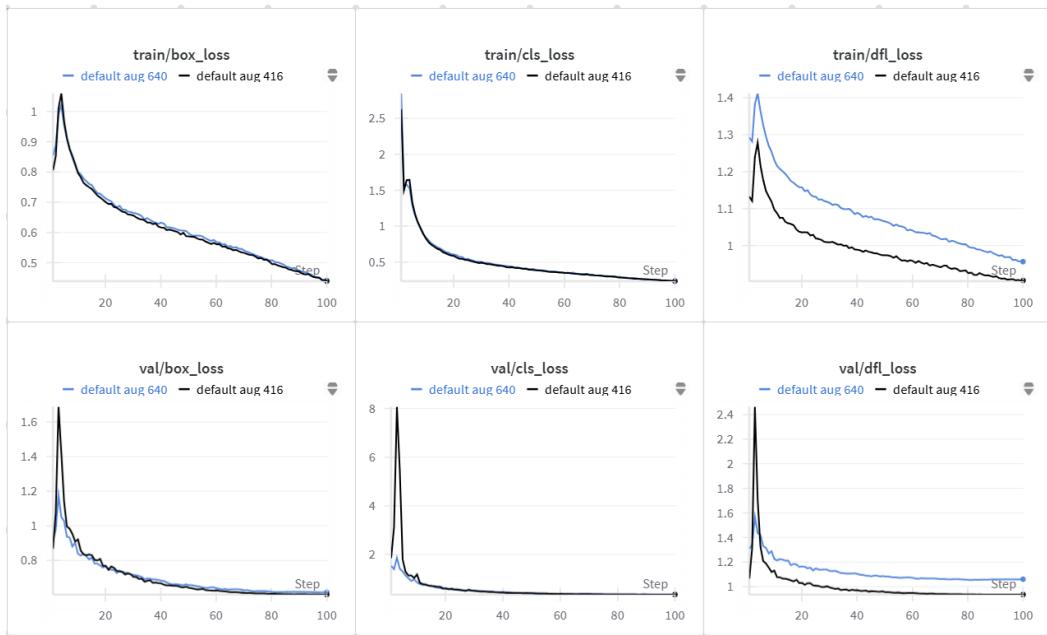


Figura 5.3: Curve di loss per l'esperimento DefaultAug

Validation Set					
Modello	Precision	Recall	F_1 Score	mAP@0.5	mAP@50-95
YOLOv12-DefaultAug 416	0.929	0.873	0.900	0.932	0.820
YOLOv12-DefaultAug 640	0.922	0.909	0.915	0.943	0.832
Test Set					
Modello	Precision	Recall	F_1 Score	mAP@0.5	mAP@50-95
YOLOv12-DefaultAug 416	0.873	0.733	0.797	0.834	0.713
YOLOv12-DefaultAug 640	0.845	0.780	0.811	0.848	0.726

Tabella 5.4: Metriche dei modelli YOLO default-augmentation calcolate su validation set e teset set

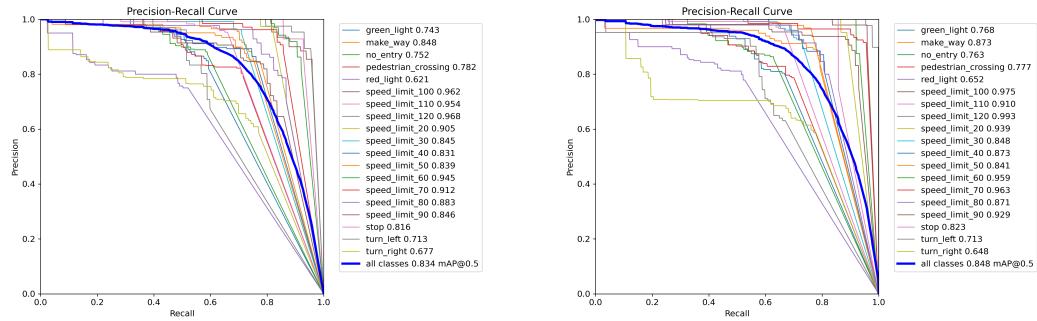


Figura 5.4: Curve Precision Recall per DefaultAug416 (sinistra) e DefaultAug640 (destra)

5.1.5 Personalized Augmentation

La configurazione Personalized Augmentation è stata costruita a partire dalla precedente, facendo ipotesi su quali tecniche potrebbero risultare dannose per alcune classi. In particolare, analizzando la tabella 5.2 si sono individuati i seguenti parametri da attenzionare:

- flipud;
- fliplr;
- shear;

- mixup;
- perspective;
- erasing,

Gli unici parametri che hanno valore di default diversi da 0 sono *erasing* e *flplr*. Pertanto, impostando tali parametri a 0 e lasciando il resto dei parametri invariato si ottiene la configurazione PersonalizedAugmentation. In linea con i precedenti esperimenti, i due modelli relativi alle due risoluzioni di immagini sono denominati PersonalizedAug416 e PersonalizedAug640. I due modelli sono stati rispettivamente allenati su Google Colab con GPU NVIDIA A100 e sul PC1 della tabella 3.1 La figura 5.5 mostra le loss dei due modelli su training e validation set. Per entrambi i modelli, sul training set soltanto la *cls_loss* inizia a stabilizzarsi nelle ultime epoche, al contrario di *box_loss* e *dfl_loss* che dopo 100 epoche non hanno ancora raggiunto la convergenza. Sul validation set, *box_loss* e *cls_loss* hanno invece raggiunto la convergenza, mentre la *dfl_loss* nelle ultime 20 epoche mostra un andamento crescente. Anche in questo caso è giustificata l'interruzione del training. Si considerino adesso le metriche della tabella 5.5. Sul test set, ogni metrica calcolata con PersonalizedAug640 è migliore di PersonalizedAug416. Si ha un incremento della mAP@0.5 di 0.012. Similmente all'esperimento con configurazione No Augmentation, aumentare la risoluzione delle immagini di training ha portato a migliori risultati. Si considerino adesso le curve Precision-Recall in figura 5.6. PersonalizedAug640 presenta performance significativamente migliori per le classi dei limiti di velocità, per i due tipi di semafori e per il segnale di stop. La classe per cui il modello risulta meno performante è *red_light*.

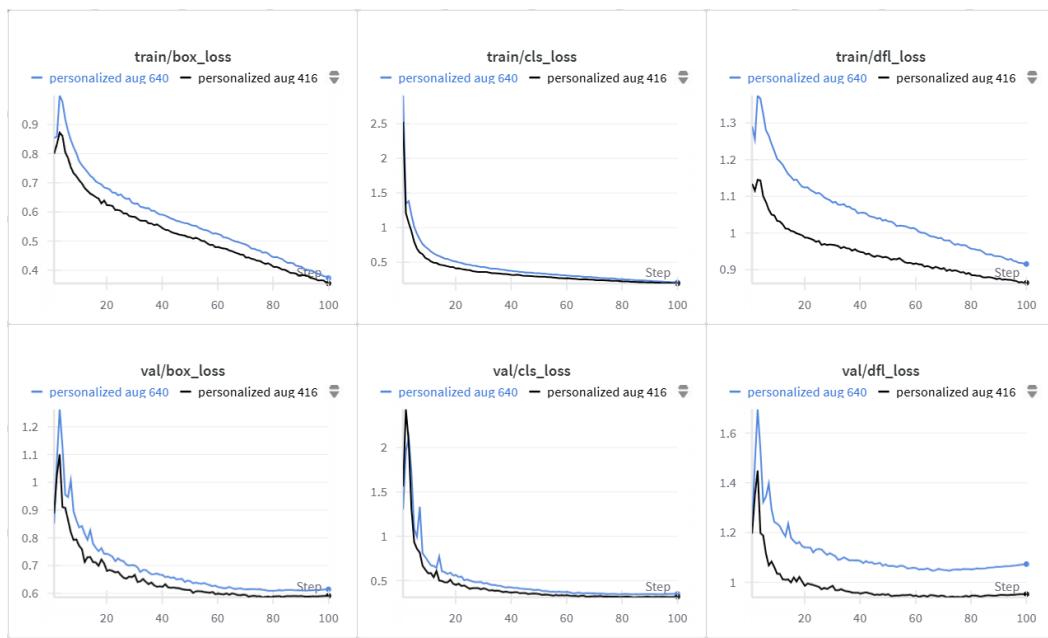


Figura 5.5: Curve di loss per l'esperimento PersonalizedAug

Validation Set					
Modello	Precision	Recall	F_1 Score	mAP@0.5	mAP@50-95
YOLOv12-PersonalizedAug 416	0.939	0.892	0.915	0.939	0.830
YOLOv12-PersonalizedAug 640	0.927	0.919	0.923	0.949	0.841
Test Set					
Modello	Precision	Recall	F_1 Score	mAP@0.5	mAP@50-95
YOLOv12-PersonalizedAug416	0.889	0.760	0.820	0.855	0.727
YOLOv12-PersonalizedAug640	0.893	0.772	0.828	0.867	0.741

Tabella 5.5: Metriche dei modelli YOLO personalized-augmentation calcolate su validation set e test set

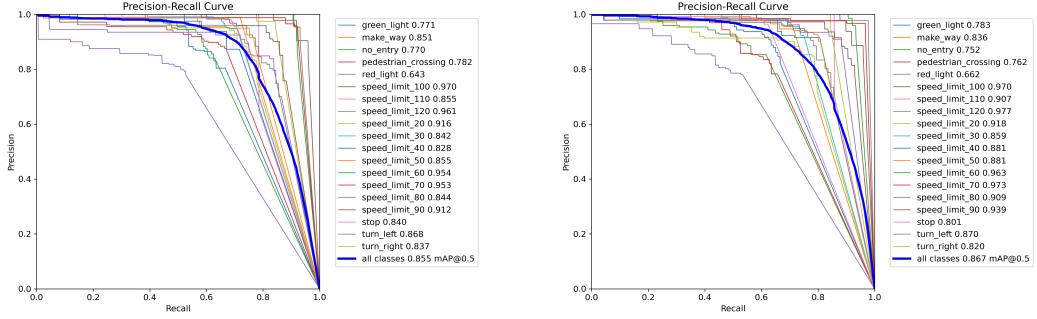


Figura 5.6: Curve Precision Recall per PersonalizedAug416 (sinistra) e PersonalizedAug640 (destra)

5.1.6 Confronto tra i modelli YOLO

Nella tabella 5.6 vengono riportati i migliori modelli prodotti per ognuna delle tre configurazioni di augmentation. Si nota subito come DefaultAug640 abbia performance migliori rispetto a NoAug640. Quindi utilizzare tecniche di augmentation ha migliorato i risultati, anche se sono state incluse le tecniche considerate dannose. Evitando tali tecniche, ovvero facendo uso di PersonalizedAug640, si ha un ulteriore miglioramento delle performance rispetto agli altri due modelli. Questo conferma l'ipotesi che alcune tecniche di augmentation sarebbero potute risultare dannose per alcune classi. La figura 5.7 mostra le due curve Precision-Recall di DefaultAug640 e PersonalizedAug640. Rispetto a DefaultAug640, PersonalizedAug640 presenta un significativo miglioramento per le classi *turn_left* e *turn_right*, per le quali si era supposto che il ribaltamento orizzontale potesse creare problemi. Anche per alcuni limiti di velocità si ha un netto miglioramento. Vi è invece un leggero decremento di performance per le classi *pedestrian_crossing*, *no_entry*, *stop*, *make_way*, *speed_limit_100*, *speed_limit_110*, *speed_limit_120* e *speed_limit_20* poichè per queste classi il ribaltamento orizzontale risulterebbe utile. Pertanto, usando PersonalizedAug640, 7 classi su 19 presentano un'area sotto la curva legger-

mente peggiore rispetto a DefaultAug640. Si ha un invece un miglioramento per le restanti 12 classi. Concludendo, il miglior modello YOLO prodotto è **PersonalizedAug640**. Tale modello andrà confrontato con il miglior modello Faster R-CNN, descritto successivamente.

Test Set					
Modello	Precision	Recall	F_1 Score	mAP@0.5	mAP@50-95
YOLOv12-NoAug 640	0.892	0.744	0.811	0.846	0.723
YOLOv12-DefaultAug 640	0.845	0.780	0.811	0.848	0.726
YOLOv12-PersonalizedAug640	0.893	0.772	0.828	0.867	0.741

Tabella 5.6: Metriche dei modelli migliori per ogni configurazione

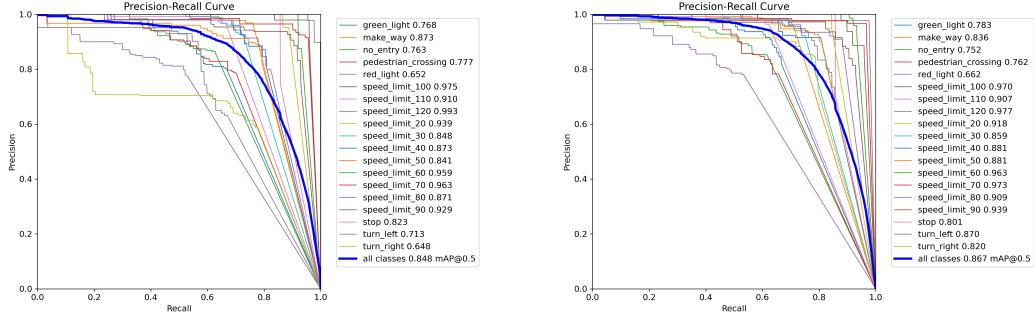


Figura 5.7: Curve Precision Recall per DefaultAug640 (sinistra) e PersonalizedAug640 (destra)

5.2 Faster R-CNN

A causa delle ingenti risorse computazionali richieste da questa architettura, sono stati allenati soltanto due modelli Faster R-CNN: il primo su immagini a risoluzione 416x416, chiamato *Frc416*, mentre il secondo su immagini a risoluzione 640x640, chiamato *Frc640*. Entrambi i modelli sono stati allenati utilizzando il PC1 della tabella 3.1. Il formato di dataset utilizzato da Faster R-CNN è il formato Pascal VOC XML. Pertanto, per addestrare i due modelli,

è risultato necessario convertire il dataset nel formato citato. Ciò viene fatto attraverso l'uso di script Python, a partire dal dataset in formato YOLO.

5.2.1 Parametri di training

I parametri di training utilizzati per i due modelli sono elencati nella seguente tabella.

Parametro	Descrizione	Valore fissato
epoches	Numero di epoche	15
lr	Learning rate iniziale	0.005
γ	Fattore di decadimento del learning rate	0.1
stepsize	Numero di epoche necessarie per il decadimento	5
batch	Dimensione del mini batch utilizzato nella SGD	2
momentum	Fattore di momentum per la SGD	0.900
weight_decay	Termine di regolarizzazione L_2	0.0005

Tabella 5.7: Parametri di training fissati per i modelli YOLO

5.2.2 Modelli addestrati

La figura 5.8 mostra la loss dei due modelli Frc416 e Frc640 su training set e validation set. Per semplicità di implementazione, si è deciso di utilizzare Tensorboard piuttosto che Weights and Biases. Il primo modello ha richiesto 1 ora e 47 minuti per il training, mentre il secondo ha impiegato 2 ore. Sul training set la loss di entrambi i modelli è arrivata a convergenza. Similmente le loss di validation tendono a stabilizzarsi, con piccole oscillazioni nelle ultime epoche con Frc640 che si stabilizza su una loss più bassa rispetto a Frc416, indicando una migliore generalizzazione. La tabella 5.7 riporta le metriche su validation e test set. Il modello con le prestazioni migliori è chiaramente

Frcnn640, che raggiunge una mAP@0.5 di 0.760 sul test set. Tuttavia, la precision si attesta su un valore basso (0.600), mentre la recall è leggermente più alta (0.697). Tale discrepanza si riflette in un F_1 score pari a 0.645. Pertanto il modello mostra una discreta capacità di individuare i segnali stradali presenti nelle immagini (recall), commettendo però diversi errori nella classificazione (precision). Il modello è in grado di localizzare i segnali, ma non sempre riesce a identificarli correttamente.

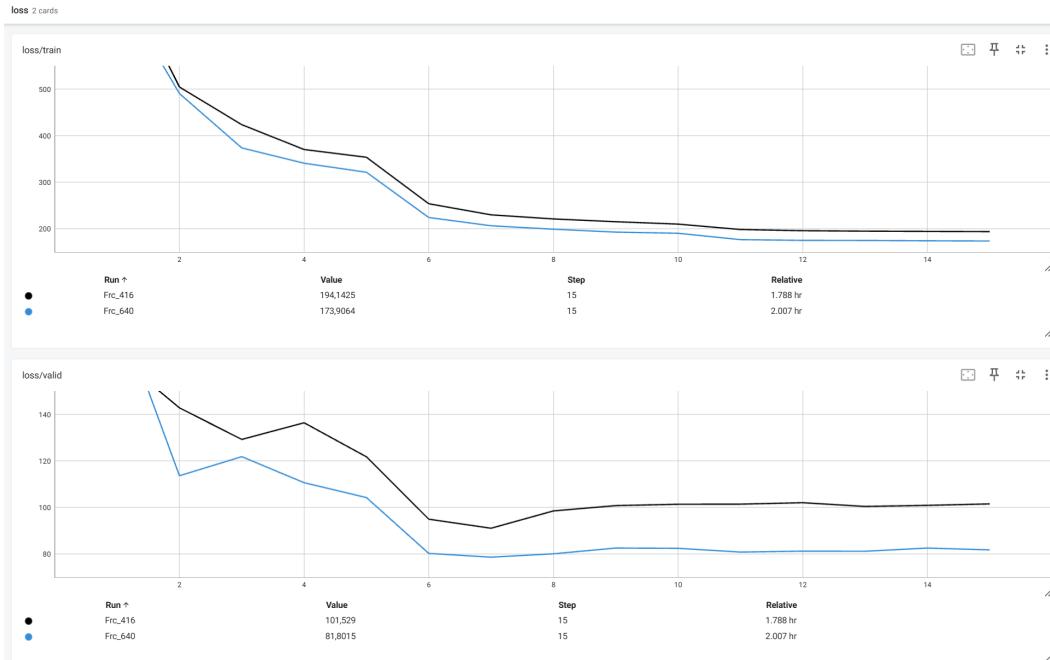


Figura 5.8: Curve di loss su training e validation set

Validation Set					
Modello	Precision	Recall	F_1 Score	mAP@0.5	mAP@50-95
Frcnn416	0.700	0.770	0.733	0.875	0.700
Frcnn640	0.723	0.794	0.757	0.903	0.723
Test Set					
Modello	Precision	Recall	F_1 Score	mAP@0.5	mAP@50-95
Frcnn416	0.572	0.668	0.616	0.732	0.572
Frcnn640	0.600	0.697	0.645	0.760	0.600

Tabella 5.8: Metriche dei modelli Faster R-CNN calcolate su validation set e test set

5.2.3 Confronto con YOLO

La tabella 5.9 mostra le metriche calcolate usando il miglior modello YOLO e il miglior modello Faster R-CNN. Il modello YOLOv12-PersonalizedAug640 risulta nettamente superiore al miglior modello Frcnn640 in ogni metrica. Sebbene in generale le architetture two stage risultino più potenti, l'assenza di una strategia di data augmentation per il modello Faster R-CNN ha probabilmente limitato le sue capacità di generalizzazione. Anche i modelli Yolo costruiti senza alcuna tecnica di augmentation risultano migliori di entrambi i modelli Faster R-CNN. Pertanto le motivazioni vanno cercate anche dietro i 10 anni di differenza delle due architetture. Infatti, la versione 12 di YOLO fa utilizzo di architetture avanzate come i meccanismi di attenzione [18], non esistenti all'epoca del rilascio di Faster R-CNN.

Test Set					
Modello	Precision	Recall	F_1 Score	mAP@0.5	mAP@50-95
YOLOv12-PersonalizedAug640	0.893	0.772	0.828	0.867	0.741
Frcnn640	0.600	0.697	0.645	0.760	0.600

Tabella 5.9: Metriche dei due migliori modelli addestrati

Capitolo 6

Demo

La demo consiste in una web app che consente di effettuare inferenza su immagini e video, utilizzando il miglior modello YOLO o il miglior modello Faster R-CNN. La demo è stata realizzata usando la libreria Streamlit, che permette di scrivere facilmente web-app per applicazioni di analisi dati. Uno screenshot dell'applicazione è mostrato nella figura seguente.

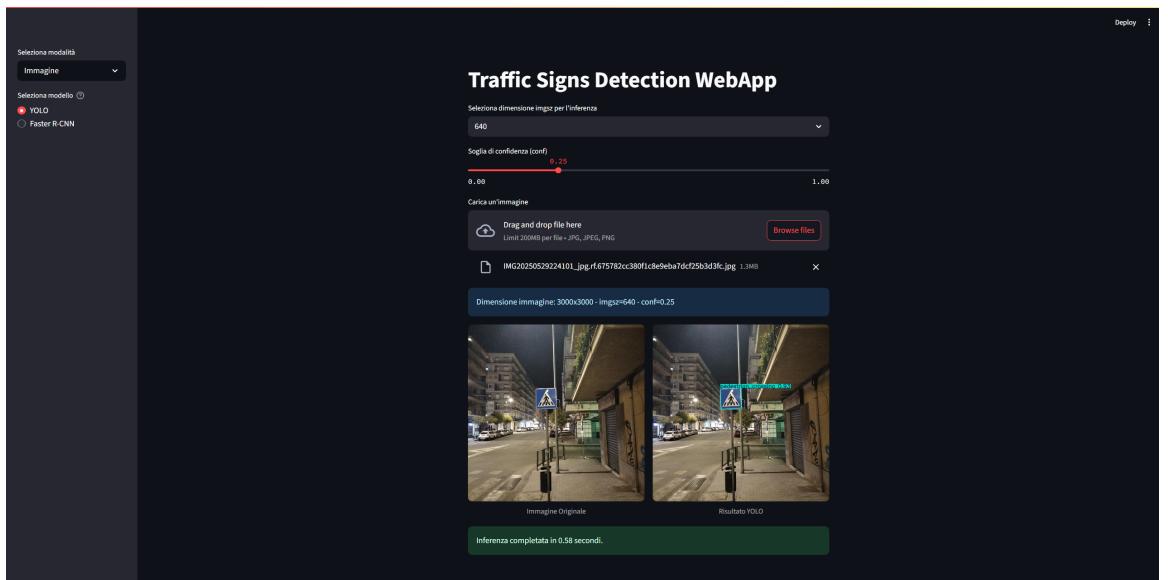


Figura 6.1: Esempio di inferenza utilizzando la demo

Il **primo** avvio dell'applicazione richiede qualche minuto perchè la demo scarica da HuggingFace il miglior modello Faster R-CNN addestrato. A sinistra è possibile scegliere la modalità di inferenza tra "Immagine" e "Video", oltre che il modello tra i due migliori. Al centro è presente una semplice interfaccia utente che permette di fare inferenza. Vi è una prima barra che consente di impostare la risoluzione dell'immagine prima dell'inferenza, seguita da uno slider che consente di scegliere la confidence. Subito dopo è presente un pulsante che permette di selezionare l'immagine o il video per cui fare inferenza. Selezionata l'immagine, l'inferenza parte automaticamente. Le due immagini prima e dopo l'inferenza vengono riportate una di fianco all'altra. Infine, viene mostrato il tempo impiegato per l'inferenza. Per quanto riguarda l'inferenza su video, una volta caricato è necessario cliccare sul pulsante "Avvia Elaborazione". Così facendo viene mostrato all'utente il risultato in tempo reale per ogni frame, il tempo previsto per la fine dell'inferenza e una barra che indica la percentuale di completamento. Il pulsante "Interrompi elaborazione" permette di arrestare l'inferenza. Sia in caso di interruzione che nel caso in cui l'elaborazione viene portata a termine, vengono salvate, nella cartella della demo, le inferenze su ogni frame e il video finale prodotto. Un video dimostrativo della demo viene allegato nella consegna del progetto.

Capitolo 7

Codice

Il linguaggio utilizzato per script, training dei modelli e realizzazione della demo è il linguaggio Python. Il codice del progetto è consultabile nel repository del progetto su GitHub. Il repository contiene anche un file *requirements.txt*, che è possibile passare come argomento al software *pip* per installare tutte le librerie necessarie. Ogni file Python è accuratamente commentato. Il codice è organizzato in base al suo contesto d'utilizzo, come descritto di seguito.

7.1 Script dataset

Gli script Python utilizzati per la produzione del dataset sono i seguenti:

- *gtsdb_to_yolo.py*: script per la conversione del dataset GTSDDB nel formato YOLO;
- *filter_classes_roboflow.py*: script per selezionare solo le classi di interesse da un dataset roboflow, scaricato in formato YOLO;
- *merge_dataset.py*: script per unire i dataset in formato YOLO, scaricati dai vari workspace roboflow;

- *split_dataset_yolo.py*: script per effettuare lo split training/validation/-test set del dataset in formato YOLO;
- *merge_PVOC.py*: script per unire i dataset scaricati da roboflow nel suo formato VOC;
- *split_PVOC.py*: script per effettuare lo split training/validation/test set del dataset creato con il precedente script;
- *roboflow_to_pascal_VOC.py*: script per effettuare la conversione dal formato Roboflow VOC a Pascal VOC XML;

7.2 YOLO

Il codice per l’addestramento e la valutazione dei modelli YOLO è organizzato sotto forma di notebook Jupyter. Attraverso la libreria Ultralytics, il codice risulta snello e semplice. Dopo aver importato le librerie necessarie, viene effettuato il login alla piattaforma Weights and Biases utilizzando l’API presente nel file *.env*. Successivamente viene istanziato un oggetto di classe YOLO, caricando il file dei pesi del modello *YOLOv12 – medium* pre-addestrato sul dataset COCO. Una volta fatto ciò, si utilizza il metodo *train* per addestrare il modello sul dataset in formato YOLO, passando come unico parametro il file di configurazione del training. Questo file è chiamato *train_settings.yaml* e contiene tutti i parametri di training e augmentation descritti nella sezione Esperimenti 5. Ogni modello ha il proprio file di configurazione, disponibile nella cartella *models/nome_modello_yolo/args.yaml*. Una volta concluso il training, attraverso il *val* vengono calcolate le metriche sul test set. Quanto descritto è presente nel file *training_template.ipynb*, che rappresenta un template da modificare passando il corretto file *args.yaml* del modello scelto.

7.3 Faster R-CNN

Il codice prodotto per Faster R-CNN consiste in un implementazione del modello mediante la libreria Pytorch. L'implementazione consiste nei seguenti moduli:

- dataset.py: contiene la classe del dataset e i metodi per interagire con esso;
- faster_rcnn.py: contiene un metodo per caricare il modello pre-addestrato Faster R-CNN con backbone ResNet-50-FPN.
- train.py: contiene i metodi per tutta la fase di training del modello;
- engine.py: contiene i metodi per il caricamento del modello, per inferenza e valutazione;
- eval.py: contiene un metodo che combina gli altri moduli per effettuare la valutazione sull'intero dataset;
- main.ipynb: contiene esempi di pipeline di training, valutazione ed inferenza su dataset, singole immagini o video.

Per ripetere il training e la valutazione con Faster R-CNN, bisogna eseguire il notebook *main.ipynb* specificando il percorso del dataset.

7.4 Demo

Il codice della demo è realizzato mediante la libreria Streamlit. Il codice è organizzato in 3 moduli:

- main.py: contiene la business logic dell'applicazione, comprensiva della creazione dei componenti grafici e delle loro funzionalità;

- `models.py`: contiene la logica per il download di Frcnn640 da HugginFace e il suo caricamento nell'applicazione.
- `inference.py`: contiene la logica per l'inferenza su immagini e video.

Capitolo 8

Conclusione

A partire dal dataset costruito, sono stati prodotti 6 modelli YOLO, basati su diverse tecniche di augmentation e di risoluzione delle immagini in fase di training, oltre che 2 modelli Faster R-CNN. Si è effettuato un confronto tra i modelli YOLO, scegliendo il migliore. Tale modello è stato poi confrontato con il miglior modello Faster R-CNN. Si è così concluso che la miglior strategia per effettuare detection su segnali stradali è usare YOLO con una configurazione di data augmentation personalizzata e risoluzione delle immagini in fase di training 640x640. Tale modello è chiamato PersonalizedAug640. Il progetto ha fatto capire quanto sia importante scegliere le giuste tecniche di Data Augmentation per risolvere il problema posto. Sono state individuate alcune tecniche potenzialmente dannose per la detection e si è dimostrato che evitandole si ottengono migliori risultati. Si è notato che il modello ha performance peggiori sulle immagini acquisite dai membri del team. Questo problema potrebbe essere risolto aggiungendo al dataset immagini ad alta risoluzione oppure aggiustando, in base alla necessità, la soglia di confidenza e la risoluzione in fase di inferenza. Si potrebbe anche espandere il progetto aggiungendo ulteriori classi. Per esempio, si è notato che sul segnale di

rotatoria il modello tende a rispondere con etichette *turn_left* o *turn_right*, segnali con una morfologia simile. Per introdurre il modello sviluppato in un sistema a guida autonoma si dovrebbero aggiungere ulteriori classi, tornando alla fase di raccolta dati e ripetendo la fase sperimentale.

Bibliografia

- [1] M. Martínez-Díaz and F. Soriguera, “Autonomous vehicles: theoretical and practical challenges,” *Transportation Research Procedia*, vol. 33, pp. 275–282, 2018, xIII Conference on Transport Engineering, CIT2018. [Online]. Available: <https://www.sciencedirect.com/science/article/pii/S2352146518302606>
- [2] X. Dong, M. Cappuccio, H. A. Jassmi, F. Alnajjar, E. Debie, M. Ghasrikhouzani, A. Lanteri, A. Luqman, T. McGregor, O. Molloy, A. Plebe, M. Regan, and D. Zhang, “Why autonomous vehicles are not ready yet: A multi-disciplinary review of problems, attempted solutions, and future directions,” 2025. [Online]. Available: <https://arxiv.org/abs/2311.09093>
- [3] “The german traffic sign detection benchmark.” [Online]. Available: https://benchmark.ini.rub.de/gtsdb_news.html
- [4] “Roboflow.” [Online]. Available: <https://roboflow.com/>
- [5] ““road signs” dataset.” [Online]. Available: <https://universe.roboflow.com/roboflow-100/road-signs-6ih4y>
- [6] ““3423s3r” dataset.” [Online]. Available: <https://universe.roboflow.com/p-h-didkk/3423s3r/browse>

- [7] ““car” dataset.” [Online]. Available: <https://universe.roboflow.com/project-4poby/car-o4yu1/browse?queryText=class%3A4&pageSize=50&startIndex=0&browseQuery=true>
- [8] “Khulna university road sign detection.” [Online]. Available: <https://universe.roboflow.com/khulna-university-uraug/road-sign-detection-3jl2n>
- [9] “Traffic signs and traffic lights dataset.” [Online]. Available: <https://universe.roboflow.com/traffic-signs-detection-aojvk/traffic-signs-and-traffic-lights>
- [10] “Mapillary dataset.” [Online]. Available: <https://universe.roboflow.com/class-bz1ka/mapillary-50bjc>
- [11] “Traffic_signal_collect dataset.” [Online]. Available: https://universe.roboflow.com/imagetranningpractice/trafic_signal_collect/dataset/4
- [12] “Speed dataset.” [Online]. Available: <https://universe.roboflow.com/speedlimit-nndx3/speed-xxelu>
- [13] “Traffic and road signs dataset.” [Online]. Available: <https://universe.roboflow.com/usmangaudhry622-gmail-com/traffic-and-road-signs/dataset/1>
- [14] “Kaggle.” [Online]. Available: <https://www.kaggle.com/>
- [15] P. K. Darab, “Traffic signs detection.” [Online]. Available: <https://www.kaggle.com/datasets/pkdarabi/cardetection/data>
- [16] “Ultralytics yolo format.” [Online]. Available: <https://docs.ultralytics.com/datasets/detect/>

- [17] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, “You only look once: Unified, real-time object detection,” 2016. [Online]. Available: <https://arxiv.org/abs/1506.02640>
- [18] Y. Tian, Q. Ye, and D. Doermann, “Yolov12: Attention-centric real-time object detectors,” 2025. [Online]. Available: <https://arxiv.org/abs/2502.12524>
- [19] “Yolo122 yaml file.” [Online]. Available: <https://github.com/ultralytics/ultralytics/blob/main/ultralytics/cfg/models/12/yolo12.yaml>
- [20] T.-Y. Lin, M. Maire, S. Belongie, L. Bourdev, R. Girshick, J. Hays, P. Perona, D. Ramanan, C. L. Zitnick, and P. Dollár, “Microsoft coco: Common objects in context,” 2015. [Online]. Available: <https://arxiv.org/abs/1405.0312>
- [21] “How augmentation works in yolo.” [Online]. Available: <https://github.com/orgs/ultralytics/discussions/10018>
- [22] S. Ren, K. He, R. Girshick, and J. Sun, “Faster r-cnn: Towards real-time object detection with region proposal networks,” 2016. [Online]. Available: <https://arxiv.org/abs/1506.01497>
- [23] S. S. A. Zaidi, M. S. Ansari, A. Aslam, N. Kanwal, M. Asghar, and B. Lee, “A survey of modern deep learning based object detection models,” 2021. [Online]. Available: <https://arxiv.org/abs/2104.11892>