VISIONE E PERCEZIONE

VEHICLE TRACKING AND COUNTING USING YOLOV8, BYTETRACK & SUPERVISION

Studenti:
MANUELE CAPECE 67468
GIANFRANCO MANFREDA 68856

Docente:
ING. DOMENICO BLOISI

ANNO ACCADEMICO 2022/2023

Capitolo 1

Introduzione

1.1 Abstract

Il conteggio di oggetti in movimento è uno dei casi d'uso più popolari nella visione artificiale (CV). Viene utilizzato, tra l'altro, nell'analisi del traffico e come parte dell'automazione dei processi di produzione.

1.2 Stato dell'arte

Il conteggio, la classificazione e il rilevamento dei veicoli stanno diventando sempre più significativi nel campo della gestione stradale. In ogni caso, a causa delle diverse dimensioni dei veicoli, la loro identificazione rimane un test che influenza direttamente l'esattezza dei conteggi dei veicoli. Per risolvere questo problema, proponiamo un framework per il conteggio, la classificazione e il rilevamento dei veicoli. Nel sistema proposto utilizziamo **Yolov8** per il rilevamento dei veicoli autostradali ripresi da un cavalcavia, **ByteTrack** per il loro tracciamento e l'ultima libreria Python di Roboflow - **Supervision** per il conteggio dei veicoli.

Capitolo 2

Strumenti utilizzati

2.1 Yolov8

La serie di modelli *YOLO (You Only Look Once)* è diventata famosa nel mondo della visione artificiale. La fama di *YOLO* è attribuibile alla sua notevole precisione pur mantenendo una piccola dimensione del modello. Questi modelli possono essere addestrati su una singola GPU, il che li rende accessibili a una vasta gamma di sviluppatori. I professionisti del machine learning possono implementarlo per hardware edge a basso costo o nel cloud.

YOLO è stato apprezzato dalla comunità della computer vision sin dal suo primo lancio nel 2015 da *Joseph Redmond*. All'inizio (versioni 1-4), *YOLO* veniva mantenuto in codice C in un framework di deep learning personalizzato scritto da Redmond chiamato Darknet .

Il campo della visione artificiale avanza con il rilascio del 10 gennaio 2023 di *YOLOv8* [1], un modello all'avanguardia che definisce un nuovo stato dell'arte per il rilevamento di oggetti, la classificazione delle immagini e le attività di segmentazione delle istanze. Insieme ai miglioramenti all'architettura del modello stesso, *YOLOv8* introduce gli sviluppatori in una nuova interfaccia intuitiva tramite un pacchetto PIP per l'utilizzo del modello *YOLO*.

YOLOv8 è stato sviluppato da *Ultralytics*, che ha anche creato l'influente modello *YOLOv5* che definisce il settore; include numerose modifiche e miglioramenti dell'architettura e dell'esperienza degli sviluppatori rispetto a *YOLOv5*.

YOLOv8 è in fase di sviluppo attivo al momento della stesura di questo di questa tesina, poiché Ultralytics lavora su nuove funzionalità e risponde al feedback della community. Infatti, quando Ultralytics rilascia un modello, gode di un supporto a lungo termine: l'organizzazione collabora con la comunità per rendere il modello il migliore possibile.

2.2 ByteTrack

Negli ultimi decenni la computer vision è cresciuta enormemente sia nel mondo accademico che in quello industriale. Con l'avvento del deep learning e il miglioramento della capacità di calcolo di processori come le GPU e le TPU, i ricercatori hanno compiuto progressi significativi nella risoluzione di una gamma sempre più ampia di compiti.

I metodi tradizionali di visione artificiale che utilizzano tecniche basate su regole e caratteristiche per il rilevamento degli oggetti sono stati messi in ombra dagli approcci di apprendimento profondo basati sulle rivoluzionarie reti neurali convoluzionali (CNN). Le reti neurali profonde hanno raggiunto prestazioni all'avanguardia per compiti come la classificazione, il rilevamento di oggetti e la segmentazione di immagini/istanze. Con il rapido progresso delle capacità della visione artificiale, i ricercatori stanno ora spostando la loro attenzione su compiti di visione artificiale più complessi, come il tracciamento di oggetti multipli.

A questo punto sorge spontanea la domanda: Che cos'è il tracciamento di oggetti multipli?
Il *Multiple Object Tracking (MOT)* è un'attività di computer vision che prevede il tracciamento dei movimenti di più oggetti nel tempo in una sequenza video. L'obiettivo è determinare l'identità, la posizione e la traiettoria di ciascun oggetto nel video, anche nei casi in cui gli oggetti sono parzialmente o completamente occlusi da altri oggetti nella scena. Il MOT è un problema importante nella computer vision, in quanto ha numerose applicazioni pratiche. Ad esempio, nel campo della sorveglianza, la MOT può essere utilizzata per rilevare e seguire comportamenti sospetti in una folla o per monitorare il movimento dei veicoli in un parcheggio. Nella robotica, può essere utilizzato per guidare veicoli autonomi, identificare ostacoli e pianificare percorsi sicuri in un ambiente dinamico. Altre applicazioni della MOT sono l'analisi dello sport, l'imaging medico e molte altre.

In generale, il tracciamento di oggetti multipli avviene in due fasi: il rilevamento degli oggetti e l'associazione degli oggetti. Il rilevamento degli oggetti è il processo di identificazione di tutti i potenziali oggetti di interesse nel fotogramma corrente, utilizzando rilevatori di oggetti come *Faster-RCNN* o *YOLO*. L'associazione di oggetti è il processo di collegamento tra gli oggetti rilevati nel fotogramma corrente e gli oggetti corrispondenti dei fotogrammi precedenti, denominati tracklet. L'associazione dell'oggetto o dell'istanza viene solitamente eseguita prevedendo la posizione dell'oggetto nel fotogramma corrente in base ai tracklet dei fotogrammi precedenti, utilizzando il filtro di Kalman, seguito da un'assegnazione lineare

uno-a-uno, in genere utilizzando l'algoritmo ungherese per ridurre al minimo le differenze totali tra i risultati della corrispondenza.

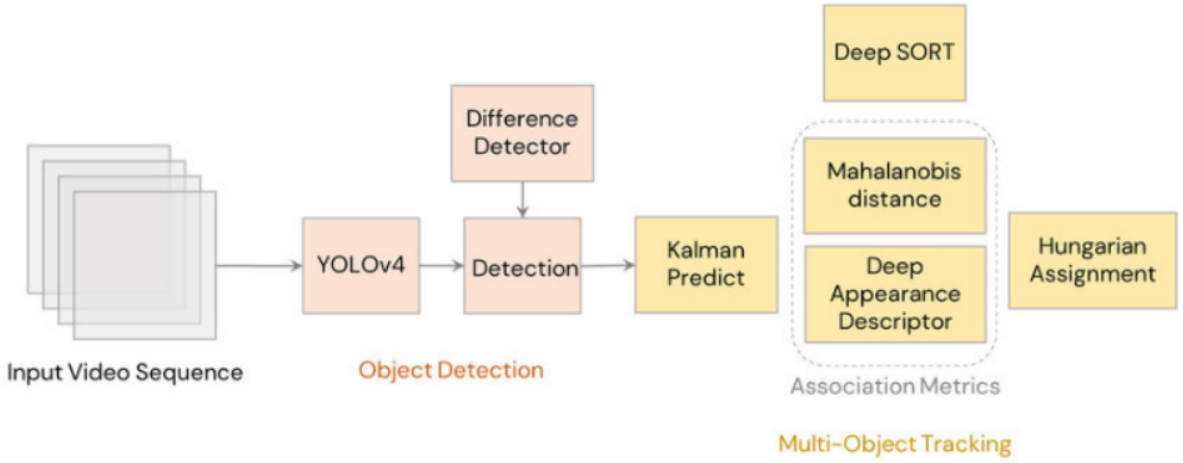


Figura 1: ***DeepSORT Algorithm***

Nonostante i numerosi progressi compiuti nel corso degli anni, la MOT rimane un compito impegnativo. Di seguito sono descritti alcuni problemi critici che hanno impedito prestazioni di alta qualità e che sono stati la fonte di motivazione per approcci recenti.

Le complicazioni possono derivare dai dati visivi stessi. Ad esempio, il movimento e l'aspetto di uno stesso oggetto possono variare notevolmente nel corso della sequenza video. Gli oggetti possono muoversi a velocità e direzioni diverse, cambiare di dimensione o forma e possono essere parzialmente o completamente occlusi da altri oggetti nella scena. Questi problemi contribuiscono alle imprecisioni del tracking MOT, come il cambio di ID dell'oggetto o l'assegnazione di più tracklet allo stesso oggetto.

A livello architettonico, i tracker multiclasse possono incontrare problemi di commutazione di classe che aggravano i problemi di associazione degli oggetti tra i vari fotogrammi, se il rilevatore non riesce a classificare accuratamente i riquadri di delimitazione a causa di problemi di classificazione, come classi simili che confondono il rilevatore. Pertanto, l'algoritmo di tracking deve essere in grado di gestire queste variazioni e di associare in modo affidabile ogni oggetto nel fotogramma corrente con l'oggetto corrispondente nel fotogramma precedente.

Per l'implementazione pratica, le architetture MOT possono essere necessarie per eseguire previsioni in tempo reale. Molti metodi MOT possono avere difficoltà con la velocità di inferenza dei video, soprattutto per le previsioni in diretta su video con frame rate elevati, perché la complessità dei modelli utilizzati richiede più calcoli e quindi più tempo di calcolo. In particolare, i video tracker multicomponente o a più stadi con architetture robuste incontreranno sicuramente difficoltà nell'eseguire l'inferenza su feed video in diretta.

Gli approcci MOT più recenti hanno utilizzato algoritmi complessi basati principalmente sulla previsione della traiettoria con il filtro di Kalman, seguita dall'associazione di bounding box con l'algoritmo ungherese. Nel progetto in questione, utilizzeremo un tracker semplice ma potente: **ByteTrack** [2]. Questo algoritmo mira principalmente a eseguire il tracciamento di più oggetti, nel caso di studio in esame, di veicoli autostradali.
BYTETrack: Multi-Object Tracking by Associating Every Detection Box è un documento presentato all'*ECCV2022* da *Yifu Zhang et al*. Grazie alla sua struttura universale e alla relativa semplicità, è stato adottato da molti ricercatori successivi per i loro tracker MOT (Bot-SORT, SMILEtrack).

L'innovazione principale di ByteETrack consiste nel mantenere le bounding box di rilevamento non di sfondo a bassa confidenza, che di solito vengono scartate dopo il filtraggio iniziale dei rilevamenti, e di utilizzare queste caselle a basso punteggio per una fase di associazione secondaria. In genere, le caselle di rilevamento occluse hanno punteggi di confidenza inferiori alla soglia, ma contengono comunque alcune informazioni sugli oggetti che rendono il loro punteggio di confidenza più alto rispetto alle caselle puramente di sfondo. Pertanto, queste caselle a bassa confidenza sono ancora significative da tenere sotto controllo durante la fase di associazione.

2.3 Supervision

Con il supporto della libreria python **Supervision** [3] di *Roboflow*, (piattaforma di computer vision che consente agli utenti di costruire modelli di computer vision in modo più rapido e accurato grazie a migliori tecniche di raccolta dati, pre-elaborazione e formazione dei modelli), è stato possibile rilevare e contare senza problemi gli oggetti in base alle rispettive regioni o zone. L'algoritmo YOLOv8 aiuta a rilevare e classificare gli oggetti, ma se combinato con la libreria di supervisione, possiamo anche specificare con precisione la regione o la zona dell'oggetto.

Capitolo 3

Sviluppo del codice

3.1 Overview

L'obiettivo che ci siamo preposti è quello di costruire un *'object detection tracking and counting system'* . Utilizzeremo l'ultima versione di Yolo, ovvero *Yolov8,* per la fase di *object detections*, il tracker d'avanguardia *ByteTrack* per il tracciamento e l'ultima librearia python di Roboflow chiamata *Supervision* per il conteggio degli oggetti, nel nostro caso di veicoli autostradali.

3.2 Setting up the Python Environment for the vehicle tracking

Il primo passo è eseguire ''*!nvidia-smi* '' per confermare che abbiamo accesso alla GPU, perchè i modelli di computer vision funzionano più velocemente se li esegui su una GPU piuttosto che su una CPU.
Poi creiamo una variabile *HOME* per rendere più facile la gestione dei percorsi.
I video sui quali effettueremo le predizioni si troveranno nella cartella '*content/video'.*
I video col tracciamento e conteggio dei veicoli si troveranno nella cartella '*content/video/result',* mentre il percorso per con il tracciamento e il conteggio dei veicoli suddivisi per classi_id si troveranno nel percorso '*content/video/result/per_casistiche'*
Ora è possibile installare *Yolov8* ma, considerando che l'ultima versione è ancora in fase di costruzione, è meglio specificare una versione specifica bloccata così da avere un notebook funzionante con la versione determinata indipendentemente dal momento in cui lo si va ad eseguire.
Si procede con l'installazione di ByteTrack (0.1.0 version) e della libreria Supervision di Roboflow (0.1.0 version).
Per apportare le modifiche necessarie al nostro caso di studi abbiamo effettuato una fork del repository di supervision creando un nuovo branch nominato ***svUnibas,*** disponibile al link [4] . Le modifiche effettuate mirano alla realizzazione delle classi che si occupano del tracking and counting delle categorie di veicoli di nostro interesse, ovvero: car, bus, track and motorcycle. Inoltre nel repository git creato si trova anche il Notebook in cui vengono generati i video del progetto.
La fase successiva è quella riguardante il caricamento del modello pre-addestrato:

```
MODEL = 'yolov8x.pt'
```

Infine i video che utilizzeremo vengono scaricati e salvati sul notebook, dove sono raffigurati veicoli che transitano su un'autostrada.

3.3 Using Yolov8 for vehicle detection

Ora che sono disponibili tutti gli strumenti, possiamo eseguire la CLI di Yolov8 sul video in questione per constatare se il nostro modello è abbastanza forte è affidabile, con l'istruzione:

```
%cd {HOME}
!yolo task=detect mode=predict model=yolov8x.pt conf=0.25 source={SOURCE_VIDEO_PATH} save=True
```

I rilevamenti sembrano essere abbastanza stabili quindi possiamo continuare.
Useremo Supervision per ricreare un ciclo d'inferenza più dettagliato e Yolov8 per eseguire l'inferenza nel ciclo principale.

3.4 Building Custom Inference Pipeline with Supervision for a single frame with Supervision

Iniziamo col creare il generatore di frame. Questo è un pezzo di codice che ci permetterà di leggere i frame dal video uno per uno; quindi, invece di caricarli nello stesso momento in memoria con una buona probabilità di saturare la RAM velocemente, caricheremo solo un frame alla volta.
La libreria Supervision ha un metodo dedicato chiamato : `get_video_frames_generator`

```python
from typing import Generator

import cv2


def get_video_frames_generator(video_path: str) -> Generator[int, None, None]:
    """
    Returns a generator that yields the frames of the video.

    :param video_path: str : The path of the video file.
    :return: Generator[int, None, None] : Generator that yields the frames of the video.
    """
    video = cv2.VideoCapture(video_path)
    if not video.isOpened():
        raise Exception(f"Could not open video at {video_path}")
    success, frame = video.read()
    while success:
        yield frame
        success, frame = video.read()
    video.release()

```

Figura 2: *get_video_frames_generator method*

che prende in input un solo parametro, ovvero il percorso del video sorgente.

Come primo step svilupperemo una pipeline di inferenza personalizzata con Supervision per un singolo frame.
Per scegliere un solo frame bisogna innanzitutto creare un iteratore, estrapolare il fotogramma successivo ed eseguire il modello pre-addestrato sul frame, che restituirà un elenco di previsioni che associamo alla variabile '*results*'.
Se richiamo a *results* il metodo *.boxes.xyxy* e faccio una stampa *print(results.boxes.xyxy)* mi restituirà il tensore python contenente le posizioni degli oggetti sul frame:

```
tensor([[ 935.,  895., 1245., 1308.],
        [2940., 1271., 3226., 1498.],
        [1436., 1085., 1621., 1231.],
        [1477., 1005., 1631., 1131.]], device='cuda:0')
```

Figura 3: *tensor Python*

Possiamo anche vedere le *confidence* (con valori tra 0 e 1) e le classi degli oggetti rilevati dal modello nel frame:

```
tensor([0.92847, 0.89655, 0.86285, 0.78417], device='cuda:0')
```

Figura 4: ***confidence tensor***

```
tensor([7., 2., 2., 2.], device='cuda:0')
```

Figura 5: classes tensor

Per lavorare con la libreria Supervision dovremo convertire questi tensori in array nunmpy.

Supervision è una libreria generica, perciò bisognerà convertire ciò che otteniamo da Yolov8 in un oggetto che sia comprensibile da Supervision; questi oggetti sono i *detections.*

Detection API

xyxy `(np.ndarray)`: An array of shape `(n, 4)` containing the bounding boxes coordinates in format `[x1, y1, x2, y2]`
mask: `(Optional[np.ndarray])`: An array of shape `(n, W, H)` containing the segmentation masks.
confidence `(Optional[np.ndarray])`: An array of shape `(n,)` containing the confidence scores of the detections.
class_id `(Optional[np.ndarray])`: An array of shape `(n,)` containing the class ids of the detections.
tracker_id `(Optional[np.ndarray])`: An array of shape `(n,)` containing the tracker ids of the detections.

Figura 6: Detection API

La classe *Detections* può essere utilizzata per visualizzare i bounding box sul frame; dopo aver importato le utilità *Detections* e *BoxAnnotator* possiamo crearci un'istanza di delimitazione di BoxAnnotator che prende in input il metodo *ColorPalette()* (rappresenta un insieme di colori usati da quel particolare annotatore).
A questo punto l'unica cosa di cui abbiamo bisogno è annotare il frame richiamando all'annotatore *box_annotaor* il metodo *.annotate* che prende in input il frame non elaborato (ndarray) e i detections. Se si desidera bounding box più dettagliati, ad esempio stampare l'id della classe, posso passare un parametro aggiuntivo all'annotatore chiamato 'labels'. Considerando che gli esseri umani lavorano meglio con il testo piuttosto che con i numeri, i modelli Yolov8 contengono proprietà chiamate names:

model.model.names

e posso mappare gli ID di classe in quei nomi.
Perciò è stata creata la costante *CLASS_NAMES_DICT = model.model.names* potendo così mappare le nostre etichette in labels in modo diverso, stampando il nome della classe invece che l'id.

3.5 Building Custom Inference Pipeline with Supervision for a whole video with Supervision

Nella fase successiva si è proceduto a convertire l'elaborazione di un singolo frame nell'elaborazione dell'intero video, aggiungendo ulteriori livelli di complessità con tracker e contatore.
Per farlo dobbiamo eseguire un loop sui video frames provenienti da generator. Il modo più semplice è dichiarare un ciclo for e indentare il codice già scritto. Il codice ottenuto stamperà tutti i frame che dovremmo salvare sotto forma di video; a tal proposito ci siamo serviti di un'altra classe di Supervision: *VideoSink*, specificando il percorso del file di output e l'istanza della classe VideoInfo (mi restituisce: risoluzione, FPS e quantità totale di frames).
Si indenta un'ulteriore volta il codice scritto sotto la chiamata:

```
with VideoSink(TARGET_VIDEO_PATH, video_info) as sink
```

Rimuoviamo *show_frame_in_notebook* perchè vogliamo salvare i frame in un video, sostituendo la riga di codice con:

```
sink.write_frame(frame)
```

e quindi sfruttando il metodo *write_frame* di VideoSink.

3.6 Tracking Detections with ByteTrack

Il principale obiettivo, arrivati a questo punto, è collegare il tracker, già installato nell'ambiente. Si crea un'istanza:

```
byte_tracker = BYTETracker(BYTETrackerArgs())
```

Il tracker *BYTETracker* accetta argomenti che sono sotto forma di classe python e nel ciclo for uso questa istanza per acquisire gli ID tracker e fare matching con i detections.
Ora si possono aggiornare le labels includendo non solo l'ID della classe e la confidence, ma anche l'ID del tracker.
L'ultima fase è quella che riguarda il tracciamento della linea e il conteggio degli oggetti che l'attraversano completamente.

3.7 Counting objects crossing the line with Supervision

I due punti della linea vengono creati come istanze, che abbiamo chiamato *LINE_START* e *LINE_END* della classe *Point* di Supervision.
Per visualizzare la linea abbiamo bisogno di un altro annotatore che creo come istanza della classe *LineCounterAnnotator* di Supervision.
Aggiorniamo la variabile line_counter con l'istruzione:

```
line_counter.update(detections=detections)
```

in modo che acquisisca i rilevamenti e determini se è stata attraversata o meno ed infine annoteremo quella linea mostrando su display l'incremento dei contatori in real-time con l'istruzione:

```
line_annotator.annotate(frame=frame, line_counter=line_counter)
```

Ricordo che i detections che passo in input al line_counter sono stati filtrati attraverso la creazione di una *mask* (un ndarray) che seleziona solo le classi_id di nostro interesse, salvate nella variabile *CLASSE_ID* :

```
CLASS_ID = [2, 3, 5, 7] #2: 'car', 3: 'motorcycle', 5: 'bus', 7: 'truck'
```

effettuando un filtraggio dei detections attraverso il metodo *filter* della classe Detection:

```
mask = np.array([class_id in CLASS_ID for class_id in detections.class_id],
dtype=bool)
detections.filter(mask=mask, inplace=True)
```

3.8 Counting objects by categories crossing the line with Supervision

Un'ulteriore passo è stato quello di effettuare il conteggio dei veicoli autostradali che attraversano la linea, suddivisi per categoria.
Il metodo impiegato è stato quello di creare 4 script .py inseriti nel percorso *supervision* → *tools* della cartella di progetto, ognuno di questi contenenti le proprie specifiche classi di LineCounter e LineCounterAnnotator adattate al determinato veicolo autostradale, che sia esso 'car', 'motorcycle', 'bus' o 'truck'.
All'interno del notebook sono stati definiti quattro contatori e quattro annotatori di linea come segue:

```
# create LineCounter instances
line_counter_car = LineCounterCar(start=LINE_START, end=LINE_END)
line_counter_moto = LineCounterMoto(start=LINE_START, end=LINE_END)
line_counter_bus = LineCounterBus(start=LINE_START, end=LINE_END)
line_counter_truck = LineCounterTruck(start=LINE_START, end=LINE_END)
# create LineCounterAnnotator instances
```

```
line_annotator_car = LineCounterAnnotatorCar(thickness=2, text_thickness=2,
text_scale=1)
line_annotator_moto = LineCounterAnnotatorMoto(thickness=2, text_thickness=2,
text_scale=1)
line_annotator_bus = LineCounterAnnotatorBus(thickness=2, text_thickness=2,
text_scale=1)
line_annotator_truck = LineCounterAnnotatorTruck(thickness=2,
text_thickness=2, text_scale=1)
```

Per aggiornare i contatori di linea specifici per categoria è stata creata, per ognuno di essi, una deepcopy dell'istanza detections e del ndarray mask (creata precedentemente per filtrare i rilevamenti delle classi indesiderate).
In seguito viene riportato un pezzo di codice relativo al filtraggio per l'update del contatore di linea della classe 'car':

```
# filtering updating line counter car and filtering out detections with
class_id = 3, 5, 7
 detections_car =  deepcopy(detections)
 mask_car = deepcopy(mask)
 mask_car = np.array([class_id in [2] for class_id in
detections_car.class_id],    dtype=bool)
 detections_car.filter(mask=mask_car, inplace=True)
 line_counter_car.update(detections=detections_car)
```

Infine vengono aggiornati gli annotatori di linea passando in input i rispettivi contatori di linea:

```
# annotate
line_annotator_car.annotateCar(frame=frame,line_counter_car=line_counter_car)
line_annotator_moto.annotateMoto(frame=frame,line_counter_moto=line_counter_m
oto)
line_annotator_bus.annotateBus(frame=frame,line_counter_bus=line_counter_bus)
line_annotator_truck.annotateTruck(frame=frame,line_counter_truck=line_counte
r_truck)
```

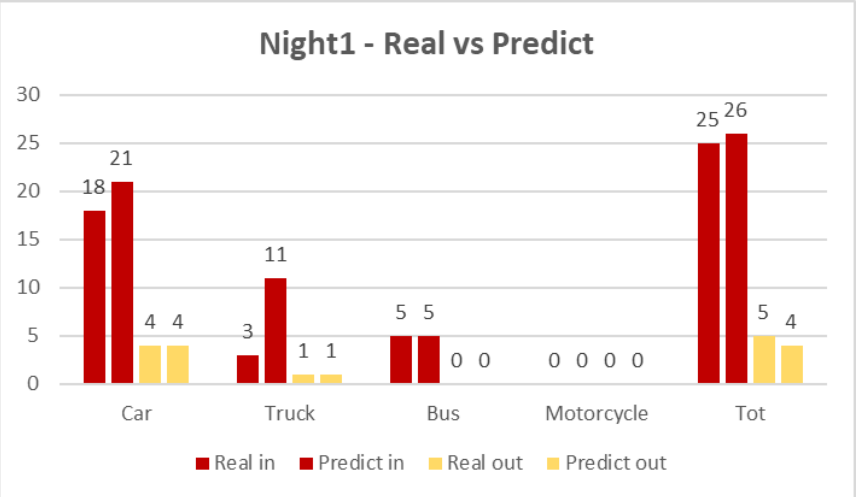
Capitolo 4

Conclusioni

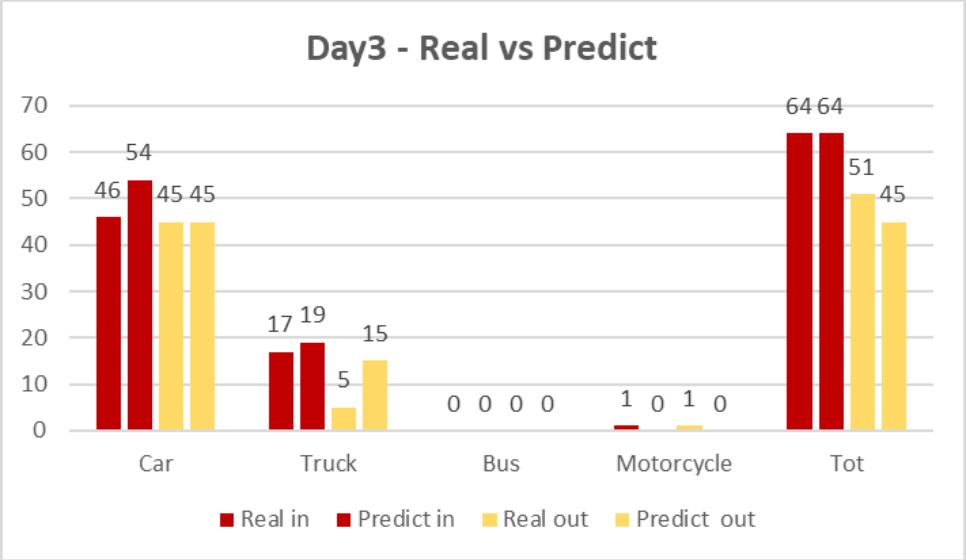
4.1 Analisi dei risultati ottenuti

4.1.1 Funzionamento diurno

Per quanto riguarda il funzionamento del modello di detection, in condizioni diurne quest'ultimo è in grado di rilevare in modo molto efficace i veicoli transitanti sulla carreggiata. Tuttavia yolov8 a volte confonde i SUV con dei truck, soprattutto se questi sono dotati di dispositivi portapacchi. Inoltre è anche indeciso sui veicoli pick-up con la quale cambia decisione più volte durante il loro transito. Questo comportamento è possibile notarlo nel video *'night1'* in cui si nota subito dal grafico che vengono contati più truck di quanti ne passano realmente.



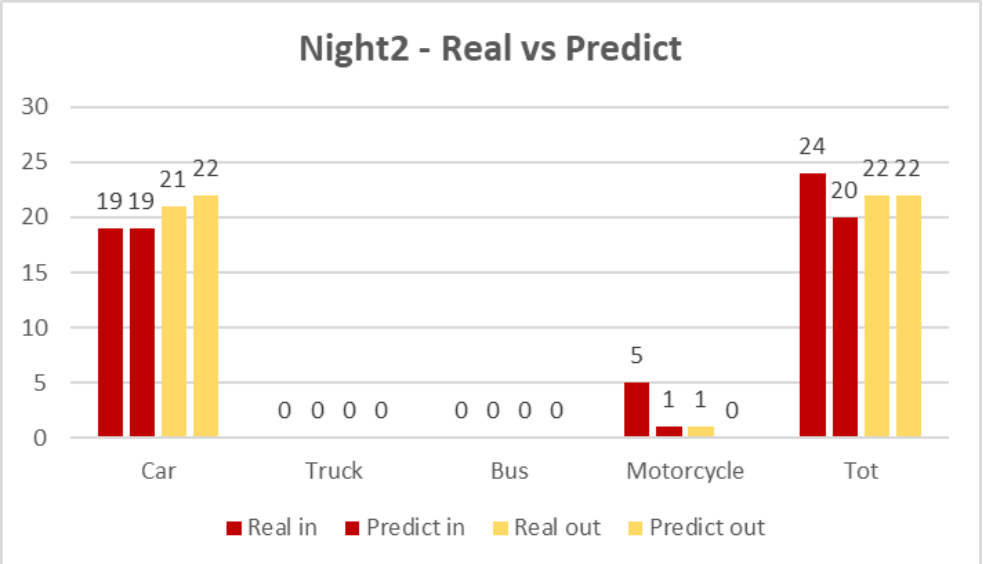
Inoltre segnaliamo che nel rilevamento dei bus il modelle tende ad essere indeciso se classificarli come bus oppure truck. Difatti raramente è possibile che venga conteggiato un bus in entrambe le categorie. Un'ulteriore difficoltà del modello è quella relativa al rilevamento delle moto.



In particolare nel video ‘day3’ è possibile notare come la rete non rilevi per nulla le due moto in entrata e in uscita. Bisogna anche considerare che è una categoria che passa di rado e quindi ci vorrebbe un video con più moto transitanti per trarre altre conclusioni.

4.1.2 Funzionamento notturno

In sostanza i problemi di tracciamento e conteggio che abbiamo riscontrato nelle condizioni di sole sono le stesse che si possono riscontrare nelle condizioni di buio.



Nel caso notturno il problema del rilevamento delle moto è ancora più evidente. In merito a questo nel video ‘night 2’ delle 5 moto transitanti ne viene contata soltanto una.

4.2 Conclusioni finali e sviluppi futuri

In conclusione il modello preaddestrato yolov8x.pt risulta essere performante e accurato al netto dei pochi problemi citati sopra. Per migliorare ancora i conteggi effettuati si potrebbe addestrare questo modello con ulteriori dati notturni. E per risolvere il problema dei SUV si potrebbe implementare questa categoria nel riconoscimento per evitare confusione con i truck. Per quanto riguarda la linea di counting questa potrebbe essere posizionata con un margine maggiore dai fine carreggiata dato che a volte i mezzi pesanti che transitano nella corsia più a destra risultano essere troppo al limite con la linea tracciata, venendo quindi esclusi dal conteggio.

References

[1] Glenn Jocher and Ayush Chaurasia and Jing Qiu (n.d.). *Ultralytics YOLOv8*.

https://github.com/ultralytics/ultralytics?ref=blog.roboflow.com

[2] Zhang, Y. (2022). ByteTrack: Multi-Object Tracking by Associating Every Detection Box.

[3] Roboflow. (n.d.). *Supervision-0.1.0*. roboflow / supervision.

https://github.com/roboflow/supervision/tree/0.1.0

[4] https://github.com/manuelecapece/supervision/tree/svUnibas