Alma Mater Studiorum · Università di Bologna

SCUOLA DI SCIENZE
Corso di Laurea in Informatica

Rilevamento dell'utilizzo dello smartphone durante la guida tramite l'object detection: un approccio avanzato

Relatore:
Dott. Federico Montori

Correlatore:
Prof. Luca Bedogni

Presentata da:
Juan Guillermo Jaramillo Saa

Sessione II
Anno Accademico 2022/2023

A mia madre e mia sorella,
per il loro amore infinito

Alla mia metà, Valeria

Sommario

Lo scopo di questo elaborato è quello di fornire una base da cui costruire un sistema di rilevazione di utilizzo dello smartphone durante la guida. Noto anche come texting and driving, si tratta di una forma di distrazione alla guida molto pericolosa, che aumenta significativamente le probabilità di incidenti stradali. La letteratura in questo campo offre diverse soluzioni, a partire da approcci che utilizzano i sensori inerziali del dispositivo mobile, fino ad arrivare all'uso di object detection con la fotocamera frontale dello smartphone. Vogliamo migliorare lo stato dell'arte di quest'ultimo approccio. Per far ciò, sono necessari modelli più robusti, che permettano il riconoscimento anche di passeggeri posteriori. Sotto le dovute assunzioni, introdurremo nuovi oggetti che ci aiuteranno a superare questo limite. Inoltre, per migliorare l'accuratezza e la robustezza delle classificazioni delle immagini come conducente o passeggero, introdurremo due nuove euristiche basate sulle confidence delle bounding box.

Indice

Capitolo 1

Introduzione

L'espressione *texting and driving* è spesso utilizzata per riferirsi all'atto di comporre, mandare o leggere messaggi su un cellulare durante la guida di un autoveicolo. Si tratta di un'azione estremamente pericolosa perchè aumenta significativamente le probabilità del conducente di essere coinvolto in un incidente stradale.

Molti lavori in letteratura, si avvalgono dell'utilizzo del telefono con l'obiettivo di costruire un sistema ideale che non permetta l'utilizzo dei cellulari durante la guida. Tra questi lavori troviamo [1] e [8].

Lo scopo di questa tesi è quello di progettare un sistema che permetta la rilevazione dell'utilizzo dello smartphone durante la guida, basandosi in particolare sul lavoro [8], cercando di espanderlo e di superarne alcune criticità. Utilizzeremo la fotocamera frontale dello smartphone per ottenere immagini da dare in pasto a un modello YOLO [9] pre-addestrato sul dataset COCO [7].

1.1 Object detection

L'object detection (o rilevamento di oggetti) è una tecnologia nel campo di studio della computer vision (o visione artificiale). Questa tecnologia permette il rilevamento di oggetti predeterminati in immagini o video.

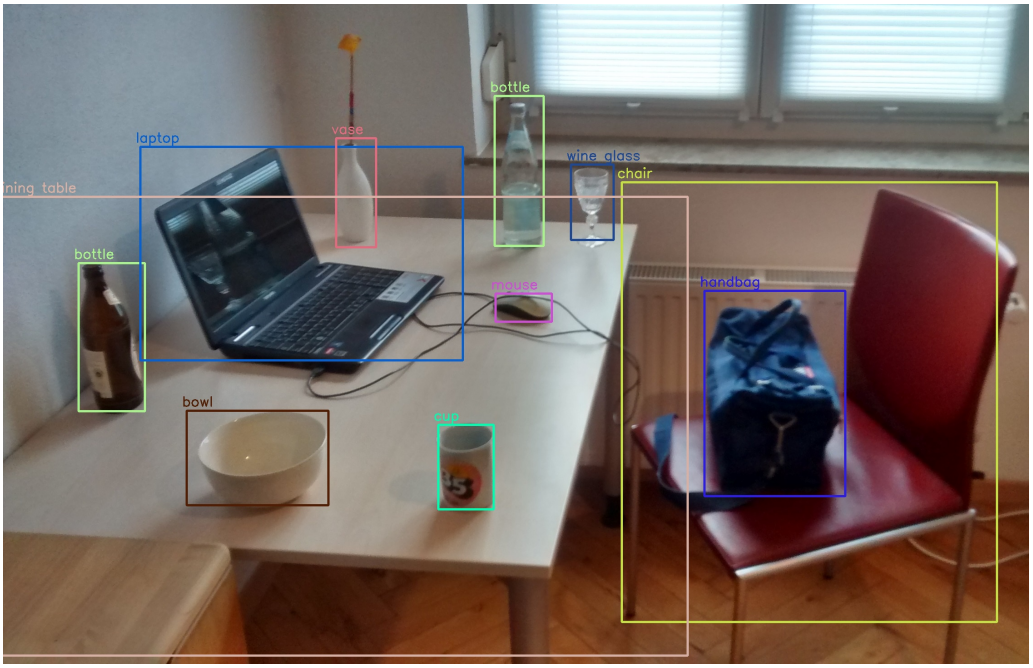

Figura 1.1: Oggetti rilevati con il modulo Deep Neural Network (dnn) della libreria OpenCV utilizzando un modello YOLOv3 addestrato su set di dati COCO in grado di rilevare 80 tipi comuni di oggetti. [3]

La differenza principale tra gli algoritmi di classificazione e gli algoritmi di object detection, sta nel fatto che negli algoritmi di detection, si cerca di disegnare delle **bounding box** che contengono gli oggetti. Inoltre **non è conosciuto** a priori il **numero di bounding box** che possono essere riconosciute in un'immagine e, proprio per questa motivazione, non è possibile costruire una tradizionale rete convoluzionale seguita da un layer denso.

Tra i primi approcci di object detection, vi sono le **R-CNN** (*Region-based Convolutional Neural Networks*, [6]). Si tratta di un algoritmo basato sulla classificazione di 2000 *proposte di regione* (ovvero porzioni di un'immagine, stabilite da un algoritmo non basato su alcun tipo di apprendimento). Ogni regione proposta viene poi data in pasto (dopo essere state ridimensionate ad una dimensione fissa) ad una CNN pre-addestrata che si comporta da *feature-extractor*. Tra i principali problemi di questa soluzione troviamo il fatto che, l'algoritmo di selezione delle regioni, potrebbe "candidare" regioni poco rilevanti. Successivamente sono state introdotte le **Fast R-CNN** e **Faster R-CNN**, che sono riuscite a superare i problemi di performance delle R-CNN e ottenere una maggiore qualità delle regioni candidate [5].

YOLO [9] (You Only Look Once) è un algoritmo per l'object detection che invece di dividere l'immagine in regioni, considera tutta l'immagine come una griglia $S \times S$ dove in

ogni cella vengono computate m bounding box, insieme alle coordinate di queste ultime, la larghezza, l'altezza e un valore di confidence calcolato sull'**IOU** (Intersection Over Union) tra l'anchor box candidata e la bounding box reale. Ogni cella predice, oltre alla bouding box e la confidence, anche la classe dell'oggetto della box. Tuttavia è importante notare che, anche se ogni cella può computare m bounding box, predice soltanto una classe. Questo è il principale limite dell'algoritmo YOLO: **se più oggetti di classi differenti si trovassero all'interno di una cella, l'algoritmo non riuscirebbe a classificare entrambi gli oggetti correttamente**. Quindi ogni predizione da una cella ha dimensione $c + m * 5$, dove c è il numero delle classi e m è moltiplicato per 5 perchè per ogni box ci si aspetta una tupla $(x, y, w, h, \text{confidence})$. Successivamente vengono selezionate le bounding box, sulla base della loro confidence e del valore di IOU con le altre box.

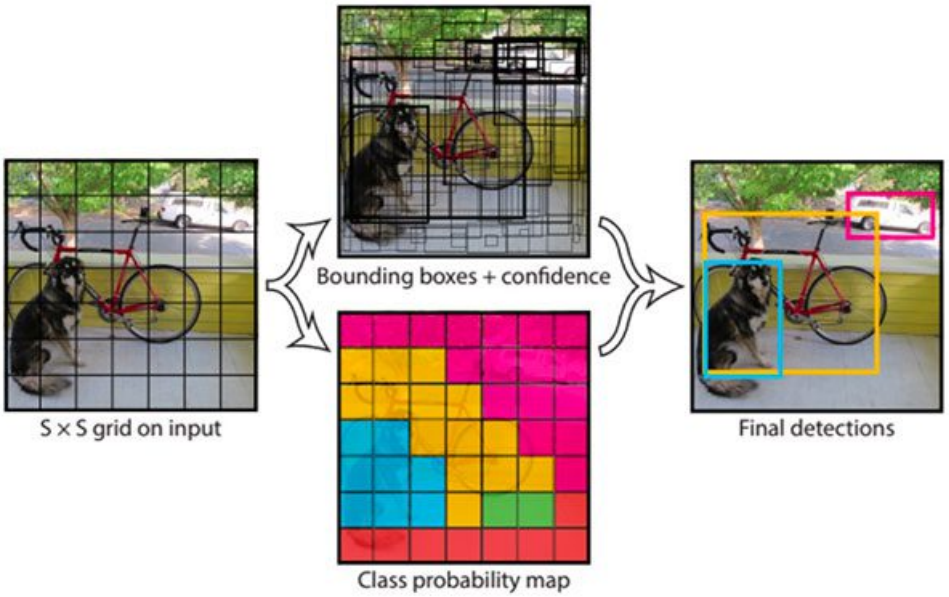

Figura 1.2: Illustrazione semplificata della pipeline di YOLO. [9]

1.2 Strumenti per l'object detection

Nel contesto della rilevazione di oggetti, la scelta degli strumenti e delle risorse giuste svolge un ruolo cruciale nella pianificazione, implementazione e analisi dei risultati. Nelle sezioni a seguire ci dedicheremo all'esplorazione e alla discussione dei vari strumenti e delle risorse utilizzate per questo progetto.

1.2.1 YOLO Ultralytics

Ultralytics è un'azienda specializzata nello sviluppo di software di computer vision e l'apprendimento automatico.

YoloV8 è l'ultima versione di YOLO sviluppata da Ultralytics (una panoramica sulla sua architettura può essere trovata su https://blog.roboflow.com/whats-new-in-yolov8). Per questo progetto, abbiamo utilizzato questa versione di YOLO, anche grazie alla sua ottima documentazione (https://docs.ultralytics.com/).

In Python, è possibile utilizzarla con il pacchetto *ultralytics*. Ultralytics offre i suoi prodotti software sotto licenza *GNU Affero General Public License v3.0* per progetti accademici e open source (più informazioni possono essere trovate su https://ultralytics.com/license).

Possiamo caricare i pesi di un modello pre-addestrato nel seguente modo:

```
1 from ultralytics import YOLO
2 # Load a pretrained YOLO model
3 model = YOLO('yolov8n.pt')
4 # Train the model using the 'coco128.yaml' dataset for 20 epochs
5 results = model.train(data='coco128.yaml', epochs=20)
```

1.2.2 LabelImg

La creazione di dataset per l'apprendimento supervisionato è un processo dispendioso in termini di risorse, in quanto richiede che gli esseri umani annotino le *ground truth*. Nell'object detection, esistono diversi strumenti che ci possono aiutare in questo compito, uno di questi è **LabelImg**, ditribuito sotto licenza *MIT*.

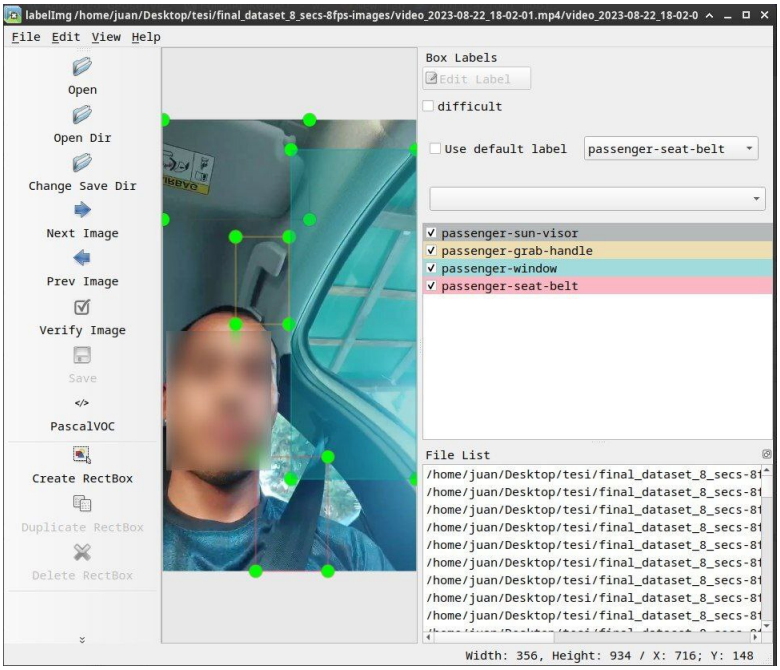

Figura 1.3: Interfaccia per le annotazioni di bounding box su LabelImg

LabelImg fornisce una semplice e intuitiva interfaccia grafica per l'annotazione di immagini. Offre la possibilità di salvare le annotazioni secondo il formato *PascalVOC* oppure *YOLO*. Il formato *YOLO* consiste nel salvare le annotazioni in file `.txt` che hanno la seguente struttura:

```
<object-class> <x> <y> <width> <height>
```

1.2.3 Roboflow

Una volta effettuate le annotazioni, è necessario preparare il dataset da dare in pasto al modello di object detection. Per questo compito possiamo utilizzare gli strumenti offerti da **Roboflow** per il preprocessing dei dati e per operazioni di *data augmentation*.

In particolare Roboflow, ci permette di caricare il dataset annotato, scegliere la percentuale di dati da dedicare al training, al test e alla validation e scaricarne una versione in un formato di nostra scelta. Per YoloV8, il formato dei dati è lo stesso che viene utilizzato per YoloV5 (un'ulteriore versione di YOLO).

Tra le operazioni che Roboflow permette di effettuare in fase di preprocessing, vi è il **ridimensionamento** e il padding delle immagini ad una specifica dimensione, che dovrà matchare le dimensioni che il modello si aspetta di ricevere. Tra le augmentations invece troviamo svariate opzioni, tra cui:

- **Rotazioni**: vengono aggiunte rotazioni variabili alle immagini.

- **Inclinazioni**: vengono applicate inclinazioni casuali.
- **Cutout**: si tratta di una forma di augmentation dove l'immagine in input è ricoperta casualmente di quadrati neri, permettendo quindi al modello di riconoscere oggetti parzialmente occcultati.

Più informazioni possono essere trovate su https://docs.roboflow.com/datasets/image-augmentation

1.3 Overfitting e come evitarlo

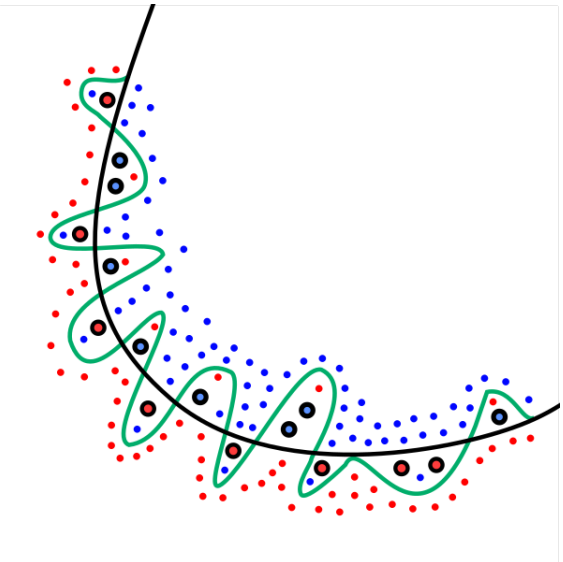

Figura 1.4: La linea verde rappresenta un modello che ha fatto overfitting sui dati. [4]

L'overfitting è un comportamento che si verifica quando il modello fornisce previsioni accurate per i dati di addestramento ma non per i nuovi dati (si guardi Figura 1.4).

Uno dei principali motivi che può portare ad overfitting è quello di allenare a lungo il modello su un insieme di dati piccolo. Questo avviene perchè, avendo pochi dati a disposizione, il modello non riesce a catturare l'insieme di tutti i possibili valori che può ricevere, il chè lo porta ad imparare cose inutili ma che lo aiutano ad ottenere un'accuratezza maggiore.

Ci sono diverse strategie per **evitare l'overfitting**:

1. Usare un modello più piccolo.
2. **Ottenere più dati**.

3. Usare early stopping: si tratta di un metodo che permette di specificare quante iterazioni possono essere eseguite prima inizi l'overfitting.

4. Usare altre forme di regolarizzazione (come il *dropout*).

Ottenere più dati non è sempre possibile per una serie di motivi legati principalmente ai **costi** di collezione e annotazione di dati extra. Nella sezione a seguire, introduciamo una nota tecnica per ridurre l'overfitting, basata sull'espansione dei dati esistenti.

1.4 Data augmentation

La **data augmentation** è una tecnica per ridurre l'overfitting che consiste nell'applicare trasformazioni casuali in modo da ottenere dei dati extra.

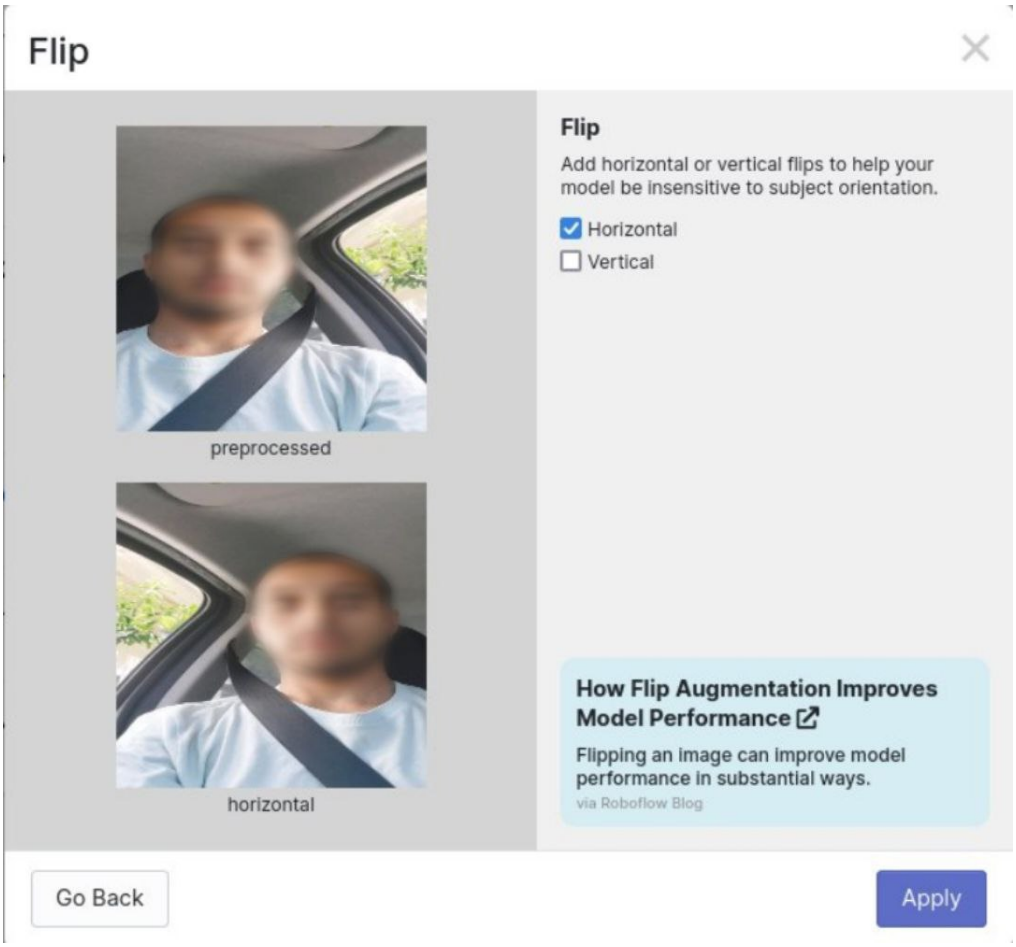

Figura 1.5: Esempio di trasformazione di Roboflow, flip orizzontale

Tra i vantaggi di utilizzo della data augmentation, troviamo:

- Permette, in genere, di ottenere un'accuratezza molto più alta.
- Insegna al modello che alcune cose non importano.

Dall'altra parte troviamo il principale svantaggio: alcune trasformazioni potrebbero essere sbagliate (e.g. nel nostro caso, invertire le immagini orizzontalmente ha come conseguenza quella di cambiare le classi degli oggetti: la cintura del passeggero diventa la cintura del conducente e viceversa), quindi dobbiamo essere cauti con le trasformazioni che decidiamo di applicare (si guardi la Figura 1.5).

1.5 Transfer learning

Nella pratica, poche persone addestrano da zero un'intera rete CNN (quindi senza alcuna inizializzazione), perchè è relativamente difficile e raro avere un dataset di grandi dimensioni.

Piuttosto si utilizzano tecniche di **transfer learning** per sfruttare reti già pre-addestrate su dataset grandi.

In particolare, si ha transfer learning quando un modello che è stato allenato e pensato per uno specifico task, viene riutilizzato per lavorare un un altro task. Si tratta quindi di un modo per trasferire le conoscenze acquisite da un modello ad un altro.
Nel deep learning, uno degli approcci comuni al transfer learning è quello di prendere un modello pre-addestrato e allenarlo su un nuovo insieme di dati (**fine-tuning**). Può essere fatto su un'intera rete oppure su un sottinsieme dei suoi layer, "congelando" i restanti layer. Questa scelta è generalmente motivata dal fatto che i primi layer della rete tendono ad estrarre feature generiche, che possono essere molto utili per diverse applicazioni (e.g. bordi, colori, texture). Man mano che si procede attraverso gli strati successivi della rete, essi diventano sempre più specializzati alle classi di oggetti presenti nel dataset di addestramento originale (più dettagli su https://cs231n.github.io/transfer-learning/).

Capitolo 2

Stato dell'arte

Il rilevamento dell'utilizzo dello smartphone durante la guida rimane ancora un problema aperto. Molti sistemi e molte applicazioni sono stati progettati per affrontare questa problematica, ottenendo ottimi risultati.

Tra questi lavori troviamo [1]. In particolare in questo lavoro è stato utilizzato un metodo completamente diverso da quello che utilizzeremo noi: l'idea era quella di stimare l'accelerazione centripeta associata alle svolte e, confrontandole con le precedenti, rilevare in quale lato della macchina si trova il cellulare. Il tutto è basato sul fatto che durante le svolte a sinistra, l'accelerazione centripeta percepita nella parte sinistra dell'auto è inferiore a quella percepita al lato destro dell'auto. Tra i limiti di questa soluzione troviamo che, essendo essa basata sull'accelerazione centripeta, l'attivazione del **meccanismo** di rilevazione è fortemente **legato ai cambiamenti di direzione**.

Inoltre, è importante considerare che, in generale, gli approcci basati sull'utilizzo dei **sensori inerziali** possono essere sensibili a **comportamenti imprevisti** o a tentativi di "**ingannare**" il sistema. Ad esempio, in [1], un guidatore potrebbe intenzionalmente posizionare il telefono sul sedile del passeggero, creando una situazione fuorviante nel processo di classificazione del sistema.

In altri lavori come [2], [10] vengono utilizzate fotocamere esterne che però richiedono intrinsecamente l'**installazione** di **hardware dedicato** all'interno dell'abitacolo delle auto. Questo le rende soluzioni complesse (maggiore manutenzione) oltre ad essere costose. Inoltre il meccanismo di rilevazione può essere facilmente compromesso da ostacoli che però possono essere tranquillamente rilevati e segnalati.

Figura 2.1: Differenza tra guidatore (sinistra) e passeggero (destra)

Il lavoro [8] nasce per superare alcune delle criticità di [1], riuscendo a raggiungere ottimi risultati. Si basa tutto sull'utilizzo della fotocamera frontale dello smartphone insieme all'object detection. Si sono individuati due oggetti caratterizzanti per la distinzione di un frame (o immagine) come *passeggero* o *conducente*: la cintura e il finestrino. Il finestrino del guidatore è chiaramente diverso dal finestrino del passeggero: sono uno il "flip" dell'altro. Stesso discorso vale per la cintura (si guardi la Figura 2.1). Il sistema che svilupperemo trarrà ispirazione da questo lavoro.

2.1 Analisi del lavoro di riferimento

Nelle sezioni a seguire analizzeremo [8] nel dettaglio, descrivendone i limiti e i possibili miglioramenti.

2.1.1 CNN

Inizialmente, gli autori hanno proposto una soluzione che prevedeva l'utilizzo di una CNN per la rilevazione di conducenti e passeggeri. I risultati hanno visto che, per questo tipo di task, l'utilizzo di una CNN ha un'accuratezza pari al lancio di una moneta. Questo li ha spinti a considerare l'object detection, che, come vedremo nella Sezione 2.1.4, ha raggiunto degli ottimi risultati.

2.1.2 Dataset *Continuous*

Sono stati registrati 8 video di 10 secondi ciascuno, di cui:

- 4 in posizione da conducente
- 4 in posizione da passeggero anteriore

Da questi video sono state successivamente estratte le immagini con un frame rate di 15 fps, ottenendo per ogni video 150 immagini, per un totale di 1200 frame estrapolati. Nonostante la buona dimensione del dataset, avrebbe giovato alla **diversificazione** dei dati avere un maggior numero di immagini, in modo da aumentare la **robustezza** del modello, aiutandolo a generalizzare meglio.

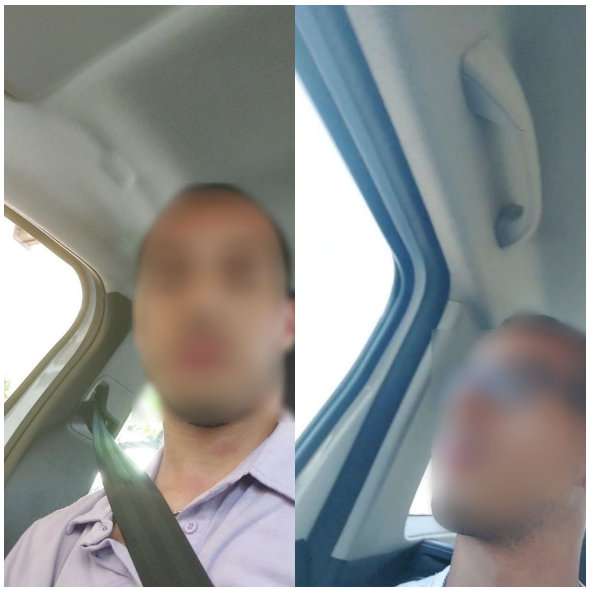

Figura 2.2: Conducente (sinistra) e passeggero posteriore (destra)

Il dataset è stato annotato considerando 4 oggetti: cintura del passeggero, cintura del conducente, finestrino del passeggero e finestrino del conducente. Utilizzando soltanto questi 4 oggetti **limitiamo** il nostro modello al **riconoscimento** dei guidatori e dei **passeggeri anteriori**. Infatti, se dovessimo applicare lo stesso modello ad un'immagine di un passeggero posteriore al lato di guida, verrebbero rilevati soltanto oggetti caratterizzanti dei conducenti (vedi Figura 2.2). Questo può far pensare che ci sia la necessità di trovare nuovi oggetti per aiutare a discriminare correttamente le immagini.

2.1.3 Training e classificazione dei video

Durante il lavoro, la fase training, è stata affidata a Roboflow Train (https://docs.roboflow.com/train/tra
Questo ha introdotto nuovi costi, poichè il suo utilizzo è vincolato alla sottoscrizione di piani tariffari personalizzabili in base alle specifiche esigenze dell'applicazione. Inoltre,

si è soggetti di una forte **dipendenza** che può causare problemi se il servizio dovesse interrompersi o subire variazioni di costi.

Una volta effettuato l'addestramento del modello, è stato progettato un sistema per la classificazione di sequenze di immagini. Il sistema riceveva in input i video e riconosceva se si trattava di un conducente oppure di un passeggero. Per costruirlo è stato necessario affrontare il problema della **classificazione** di un'**immagine** come passeggero o guidatore. Per fare tale discriminazione, è stato deciso di utilizzare la seguente **euristica**:

- se ci sono più elementi riconducibili al guidatore, allora si tratta dell'immagine di un guidatore.
- se ci sono più elementi riconducibili al passeggero, allora si tratta dell'immagine di un passeggero.
- altrimenti l'immagine non può essere classificata.

Si tratta di un'euristica che di fatto non utilizza le informazioni sulla *confidence* delle bounding box, facendo affidamento esclusivamente sulle **quantità di oggetti rilevati**. Vediamo come si comporta sul seguente esempio (utilizzando una pseudocodifica):

```
[
  {
    class: "driver-window",
    confidence: "0.30",
    bbox: bbox1,
  },
  {
    class: "driver-seat-belt",
    confidence: "0.20",
    bbox: bbox2,
  },
  {
    class: "passenger-window",
    confidence: "0.91",
    bbox: bbox3,
  },
  {
    class: "passenger-seat-belt",
    confidence: "0.92",
    bbox: bbox4,
  },
]
```

In questo caso, il frame non verrebbe classificato anche se avrebbe senso classificarlo come passeggero, per le maggiori confidence. Questo specifico caso suggerisce che potrebbero essere studiate nuove euristiche che riescano a sfruttare al meglio le informazioni offerte dal modello.

2.1.4 Risultati

I risultati che sono stati raggiunti sono molto buoni. L'accuracy sulla classificazione dei frame, raggiunta utilizzando come dataset di test il dataset *Continuous*, è dell'80%.

Il calcolo dell'accuracy è stato poi generalizzato ($l = 1$) con la valutazione di quest'ultima, disponendo i frame in gruppi di lunghezza l, utilizzando un meccanismo di finestra scorrevole, riuscendo ad ottenere fino all'88.30% di accuratezza.

Inoltre, sono stati determinati ottimi tempi di rilevazione della classe del soggetto nei video (per dettagli su come è stato calcolato il tempo, si guardi la sezione 3.1). I risultati sono eccellenti, tuttavia sarebbe interessante sapere se, con un modello più robusto ottenuto da un dataset più vario, possono essere migliorati.

2.2 Conclusioni

In questo capitolo, abbiamo tentato di fornire un'ampia panoramica riguardo al problema della rilevazione dell'uso dello smartphone durante la guida, esaminandone quali sono le sfide.

In particolare abbiamo analizzato gli ottimi risultati di [8], sottolineandone allo stesso tempo alcune criticità, tra cui:

- La poca diversificazione del dataset.
- La focalizzazione della classificazione sui passeggeri anteriori e sui conducenti.
- La dipendenza da Roboflow Train.
- L'euristica per la classificazione delle immagini, che non tiene in considerazione le informazioni sulle confidence delle bounding box.

In merito a quanto emerso, possiamo farci le seguenti domande, a cui cercheremo di rispondere nel corso di questa tesi:

- **Aumentando la dimensione del dataset, riusciamo comunque ad ottenere dei buoni risultati?**
- **Possiamo estendere questo sistema anche per i passeggeri posteriori?**
- **Possiamo utilizzare al meglio le informazioni sulle confidence?**
- **In generale, è possibile migliorare i risultati ottenuti?**

Capitolo 3

Verso l'implementazione

In questo capitolo, ci prepariamo ad affrontare la costruzione del nostro sistema, stabilendo un linguaggio comune e preciso, che ci consentirà di comunicare in modo efficace. Questo passo preliminare è essenziale per garantire una chiara comprensione dei concetti chiave e delle variabili che utilizzeremo nel corso della nostra ricerca.

3.1 Tempi di attivazione

Il lavoro [8] introduce un meccanismo per quantificare l'intervallo di tempo da registrare con lo smartphone per raggiungere una classificazione affidabile. Per fare ciò, gli autori del lavoro definiscono γ come la finestra di immagini di lunghezza l dalla quale cominciare a registrare le *confidence*. Le *confidence* sono calcolate nel seguente modo:

> *"Poi introduciamo le confidence k come il massimo numero di classificazioni della stessa classe, in percentuale al numero totale di classificazioni."*[8]

Per chiarezza, nel corso dell'elaborato ci riferiremo al concetto di *confidence* definito dagli autori, con il termine *classification-confidence*. Un algoritmo per il suo calcolo potrebbe essere il seguente:

1. si ottengono γ classificazioni.
2. si calcola la *classification-confidence* k su di esse e si valuta $k \geq K$, dove K è una soglia di *classification-confidence* prefissata.
3. se vale $k \geq K$ allora registra il tempo $\frac{1}{fps} * \gamma$.
4. altrimenti $\gamma = \gamma + 1$ e torniamo al punto 1.

La nozione di *classification-confidence* diventa fondamentale quando parliamo di sistemi di rilevamento reali: possiamo classificare un flusso di γ immagini soltanto se siamo "abbastanza" (K) sicuri che il flusso si riferisca ad un passeggero o un conducente.

3.2 Definizioni

A seguire presentiamo alcune definizioni e notazioni che ci saranno utili per definire al meglio le euristiche di classificazione delle immagini, nel capitolo di implementazione.

- Sia A l'insieme degli oggetti che possono apparire in un frame.

  Esempio: $A = \{\text{passenger-seat-belt}, \text{driver-seat-belt}, \text{passenger-window}, \text{driver-window}\}$

- Sia C l'insieme di tutte le classi.

  Esempio: $C = \{\text{passenger}, \text{driver}\}$

- Sia A_c l'insieme degli oggetti che possono apparire in un frame, ristretti ad una specifica classe c.

  Esempio: $A_{\text{driver}} = \{\text{driver-seat-belt}, \text{driver-window}\}$

- Sia Ω^k l'insieme che contiene tutte le bounding box riconosciute applicando il modello M sull'immagine k.

- Sia Ω_c^k l'insieme contenente tutte le bounding box, riconosciute da un modello M, di soli oggetti in A_c nell'immagine k.

  Esempio: Sia $A_{\text{passenger}} = \{\text{passenger-seat-belt}, \text{passenger-window}\}$, allora $\Omega^k_{\text{passenger}}$ conterrà soltanto bounding box degli oggetti in $A_{\text{passenger}}$.

- Sia conf_M la funzione che, data una bounding box riconosciuta nell'immagine k, restituisce la confidence del modello M su di essa, quindi $\text{conf}_M : \Omega^k \to [0, 1]$

- Sia $a \in A$. Con $\text{conf}_M(\text{a})$ indichiamo la somma delle confidence di tutte le bounding box, identificate nell'immagine k dal modello M, che rappresentano l'oggetto a.

3.3 Accuracy, Precision e Recall

Esistono diversi modi per misurare l'accuratezza di un modello M. Prima però dobbiamo capire cosa sono TP (True Positive), TN (True Negative), FP (False Positive) e FN (False Negative).

Per esempio, possiamo definire:

- "driver" come la classe positiva.

- "non-driver" (i.e. passeggero) come la classe negativa.

E riassumere in una confusion matrix quali siano le possibili classificazioni:

True positive. Realtà: Si tratta di un conducente Il modello lo ha classificato come conducente	**False positive**. Realtà: Si tratta di un passeggero Il modello lo ha classificato come conducente
False negative. Realtà: Si tratta di un conducente Il modello lo ha classificato come passeggero	**True negative**. Realtà: Si tratta di un passeggero Il modello lo ha classificato come passeggero

Tabella 3.1: Confusion matrix dei possibili esiti

Un **true positive** è un esito dove il modello classifica correttamente la classe positiva. Similmente, un **true negative** è un esito dove il modello ne classifica correttamente una negativa. D'altro canto, si verifica un **false negative** quando il modello sbaglia a classificare una classe negativa mentre un **false positive** quando il modello sbaglia nella classificazione della classe positiva.

Questi indicatori sono utili quando andiamo a parlare di precision e recall:

- **Precision**: indica la frazione di predizioni positive corrette rispetto al numero totale di predizioni positive effettuate dal modello: $\frac{\text{TP}}{\text{TP+FP}}$
- **Recall**: indica la frazione di predizioni positive corrette rispetto al numero totale di positivi: $\frac{\text{TP}}{\text{TP+FN}}$
- Infine, c'è anche l'**accuracy**, che misura il numero di predizioni corrette rispetto alle predizioni totali: $\frac{\text{correct\_classifications}}{\text{all\_classifications}}$

In seguito, interpreteremo queste metriche per determinare quali di esse potrebbero avere un peso maggiore nel contesto del nostro studio. Inoltre faremo un'osservazione sull'insieme delle classi C e le classi positive e negative.

Capitolo 4

Implementazione

Nella Sezione 2.1 abbiamo analizzato attentamente il lavoro di riferimento su cui baseremo la costruzione del nostro sistema. In questo capitolo ci occuperemo della vera e propria implementazione di quest'ultimo, motivando in dettaglio le scelte di progettazione e i risultati ottenuti.

4.1 Dataset

Proponiamo un nuovo dataset che vuole essere un'espansione del dataset *Continuous*, del lavoro [8].

In particolare sono stati registrati, in **condizioni di buona luce**, 34 video da 8 secondi, di cui:

- 13 **da passeggero anteriore**
- 8 **da passeggero posteriore** (4 lato guidatore, 4 lato passeggero)
- 13 **da conducente**

Da ogni video sono state estrapolate 64 immagini con un frame rate di 8 fps, ottenendo 2176 immagini in totale. Ho cercato di utilizzare una maglia diversa per ogni video registrato nella stessa postazione e sono state coinvolte 2 automobili.

D'ora in poi n sarà utilizzato per riferirci al numero di video, sec per la durata in secondi e fps per il frame rate utilizzato per estrarne le immagini.
La raccolta dei video è stata seguita da una fase di preparazione iniziale prima di cominciare il processo di annotazione. In particolare è stato necessario tagliarli alla durata fissata (sec) per poi estrapolarne i $sec * fps$ frame.

Per scopi puramente organizzativi, ho deciso di dare la seguente struttura alle cartelle contenenti i frame dei video:

```
$videos_directory
  video_1
    video_1_frame_1.png
    video_1_frame_2.png
    video_1_frame_3.png
    ...
  video_2
    video_2_frame_1.png
    video_2_frame_2.png
    video_2_frame_3.png
    ...
  ...
```

Per ottenere ciò, ho creato uno script in bash che, data in input la cartella contenente i video non elaborati, li tagliasse e ne estraesse i frame secondo tale struttura. A seguire ne riporto una porzione:

```
1 for filename in $trimmed_videos_dirname/*.mp4; do
2   mkdir "${folder}/${filename##*/}"
3   ffmpeg
4    -i $filename
5    -vf fps=$fps "${folder}/${filename##*/}/${filename##*/}-%d.png"
6 done
```

Figura 4.1: I nuovi oggetti da annotare

Una volta terminata questa preparazione iniziale, è cominciata la vera e propria fase di annotazione delle immagini. A differenza del dataset *Continuous*, sono stati aggiunti due

nuovi oggetti da annotare, l'**aletta parasole** e la **maniglia del passeggero** (si guardi la Figura 4.1). La scelta di aggiungere nuovi oggetti è stata fatta sia per cercare di migliorare la robustezza del modello, in quanto in tal modo è in grado di riconoscere una gamma più ampia di oggetti, sia per cercare di risolvere il problema della classificazione dei passeggeri posteriori.
Quindi sono stati annotati i seguenti oggetti (tra parentesi indichiamo il corrispondente elemento di A):

- il finestrino del passeggero (**passenger-window**)
- il finestrino del conducente (**driver-window**)
- la cintura del passeggero (**passenger-seat-belt**)
- la cintura del conducente (**driver-seat-belt**)
- l'aletta parasole del passeggero (**passenger-sun-visor**)
- l'aletta parasole del conducente (**driver-sun-visor**)
- la maniglia auto del passeggero (**passenger-grab-handle**)

Si noti come:

$$\text{driver-grab-handle} \notin A_{\text{driver}}$$

Questo è conseguenza del fatto che nelle auto in cui sono state scattate le foto, al posto di guida non erano presenti maniglie. In seguito, esploreremo come possiamo **sfruttare** questa caratteristica **a nostro vantaggio**.

4.2 Euristiche

Per poter effettuare la classificazione tra passeggero e guidatore guardando soltanto l'insieme Ω^k che contiene le bounding box riconosciute dal modello nell'immagine k, è necessario introdurre una o più euristiche che possano guidarci in questo processo. Presentiamole in ordine:

4.2.1 Cardinalità dei Ω_c^k (o baseline)

Si tratta dell'euristica che è stata utilizzata anche da [8]. Un'immagine k viene classificata come classe $c_k \in C$ dal modello M, utilizzando il seguente criterio:

$$c_k = \underset{c \in C}{\text{argmax}} \, |\Omega_c^k|$$

Nel caso di parimeriti (i.e. $|\Omega^k_{\text{passenger}}| = |\Omega^k_{\text{driver}}|$), non viene assegnata alcuna classe. Il principale svantaggio di questa soluzione è dato dal fatto che **non vengono utilizzate le informazioni sulla confidence date dal modello** M.
A seguire ne riportiamo il codice Python:

```
def object_cardinalities(results):
  # | \Omega^k_{\text{passenger}} |
  driver_objects = 0
  # | \Omega^k_{\text{driver}} |
  passenger_objects = 0

  for r in results:
    for c in r.boxes.cls:
      # basta controllare se la classe è sottostringa
      if "driver" in A[int(c)]:
        driver_objects += 1
      elif "passenger" in A[int(c)]:
        passenger_objects += 1
  # classifichiamo il frame
  if driver_objects > passenger_objects:
    return "driver"
  elif passenger_objects > driver_objects:
    return "passenger"
  else:
    return "none"
```

Notiamo come i parimeriti si traducano in classificazioni "none".

4.2.2 Somma massima delle confidence (o MCS, Maximum confidence sums)

Data una classe c, definiamo la seguente somma:

$$\text{c\_conf\_sum} = \Sigma_{a \in A_c} \text{conf}_M(a)$$

Un'immagine k viene classificata come classe $c_k \in C$ dal modello M, utilizzando il seguente criterio:

$$c_k = \underset{c \in C}{\text{argmax}}(\text{c\_conf\_sum})$$

Questo metodo permette di sfruttare le informazioni sulla confidence delle bounding box, rendendo la classificazione più significativa. Nel caso di parimeriti (anche se saranno rari), come per la *baseline*, non classifichiamo il frame. Lo svantaggio principale di questa

soluzione è data dal fatto che **viene effettuata una classificazione** nella **maggior parte dei casi**, anche quando sarebbe più conveniente non classificare l'immagine.
A seguire ne riportiamo il codice Python:

```
def compute_sum_confidence(A_k, results):
  '''
    In questa funzione calcoliamo c_conf_sum,
    iterando sulle confidence delle bounding
    box riconosciute dal modello nell'immagine k
  '''
  driver_conf_sum = 0
  passenger_conf_sum = 0
  ...
  return (driver_conf_sum, passenger_conf_sum)

def MCS(results):
  '''
   A_k è il multiinsieme che contiene
   gli oggetti in A che compaiono in k,
   ci servirà  per il calcolo di c_conf_sum
  '''
  A_k = compute_A_k(A, results)
  driver_conf_sum, passenger_conf_sum =
    compute_sum_confidence(A_k, results)

  # Classifichiamo il frame
  if driver_conf_sum > passenger_conf_sum:
    return "driver"
  elif passenger_conf_sum > driver_conf_sum:
    return "passenger"
  else:
    return "none"
```

4.2.3 MCS con soglia di attivazione T

L'idea alla base è uguale a quella della MCS. Quel che cambia è il fatto che viene utilizzata *mcs* soltanto se vale:

$$|\text{driver\_conf\_sum} - \text{passenger\_conf\_sum}| > T$$

Altrimenti viene applicata la *baseline*. T è un numero reale positivo (noi utilizzeremo $T = 0.25$). In questo modo possiamo sfruttare i benefici di entrambe le euristiche **senza**

correre il rischio di commettere errori nella classificazione dei casi borderline. A seguire ne riportiamo il codice Python:

```
def MCS_with_treshold_activation(results, T=0.25):
  ...
  # Verifichiamo la soglia T
  if (abs(driver_conf_sum - passenger_conf_sum) > T):
    # Classifichiamo il frame come MCS
    if driver_conf_sum > passenger_conf_sum:
      return "driver"
    elif passenger_conf_sum > driver_conf_sum:
      return "passenger"
    else:
      return "none"
  else:
    return object_cardinalities(results)
```

4.3 Preprocessing

Per fare in modo che il nostro dataset fosse pronto per essere dato in pasto al modello YoloV8, è stato deciso di utilizzare Roboflow per partizionarlo in train, validation e test, contenendo rispettivamente l'80%, il 10% e il 10% dei dati. Oltre a permettere di dividere il dataset come appena descritto, Roboflow permette anche di ridimensionare le immagini in modo tale da fissare una dimensione valida per tutti i dati, oltre che ad applicare trasformazioni per ottenere più dati (1.4).

Inizialmente decisi di creare due versioni del dataset: uno che utilizzava le augmentations e uno senza. Purtroppo dopo alcuni risultati ho scartato il dataset che utilizzava le trasformazioni perchè, nonostante l'addestramento impiegasse molto più tempo, otteneva risultati che si discostavano di poco da quelli ottenuti con il dataset originale, senza augmentation.

Successivamente, per effettuare un **confronto** accurato tra le **classificazioni** ottenute utilizzando **più oggetti**, **ho creato due versioni del dataset**: una in cui escludevo le annotazioni dei nuovi oggetti e l'altra in cui includevo tutte le classi annotate. Chiameremo queste due versioni rispettivamente Dataset A e Dataset B.

Una volta creata la versione del dataset, Roboflow ci offre la possibilità di esportarla scegliendo un formato per le annotazioni. Il formato deve essere scelto sulla base della rete che si vorrà utilizzare. Noi abbiamo deciso di addestrare un modello YoloV8 che, oltre a richiedere il formato tradizionale delle annotazioni YOLO, si aspetta anche un file `data.yaml` di configurazione. Alcuni parametri che possiamo trovare al suo interno, sono:

- I percorsi delle immagini di train, test e validation.
- I nomi degli oggetti (`names`) delle annotazioni.
- Il numero di classi degli oggetti `nc`.

Il codice fornito da Roboflow per scaricare il dataset direttamente da Python, è il seguente:

```
#pip install roboflow
from roboflow import Roboflow
rf = Roboflow(api_key=my_api_key)
project = rf.workspace(my_workspace).project(my_project)
dataset = project.version(my_version).download("yolov5")
```

4.4 Training

Come già detto nella sezione precedente, noi utilizzeremo un modello YoloV8 pre-addestrato, `YOLOv8s`.

```
# Carichiamo il modello pre-addestrato sul dataset COCO.
model = YOLO('yolov8s.pt')

# Fine tuning
results = model.train(
   data='data.yaml',
   imgsz=640,
   epochs=epochs
)
```

L'addestramento prevede di default che tutti i pesi possano essere migliorati continuando la *backpropagation*. È possibile congelare i pesi dei layer iniziali, ma noi ci limiteremo ad effettuare fine-tuning sull'intera rete.

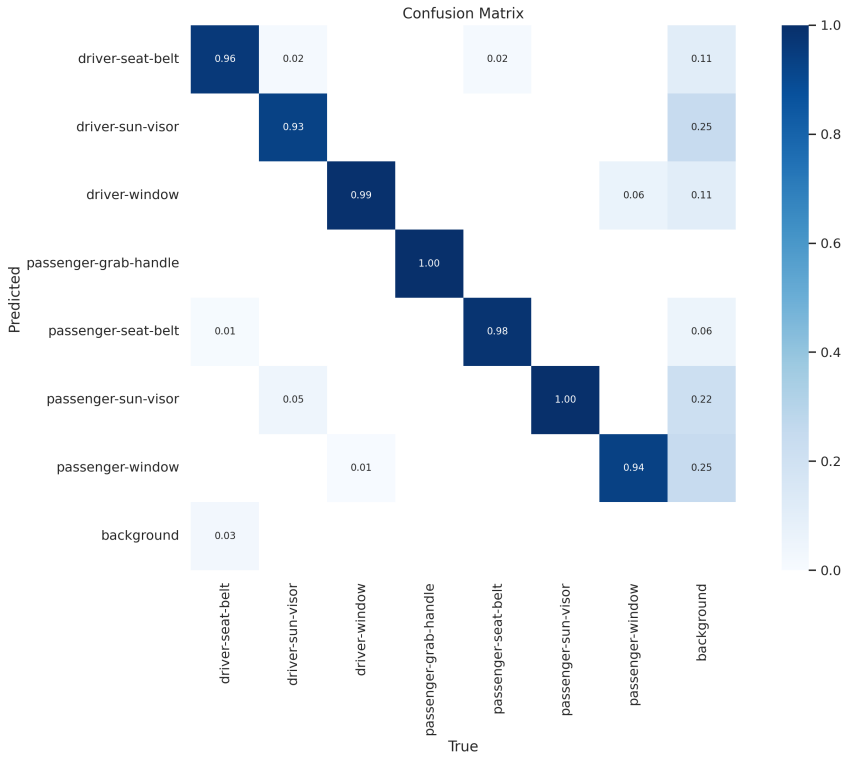

Figura 4.2: Confusion matrix risultante da un addestramento sul Dataset B

Nell'object detection, le confusion matrix (si guardi la Figura 4.2) possono essere molto utili. Questa matrice ci fornisce un'indicazione delle associazioni tra le classi degli oggetti predette e le reali classi degli oggetti, espresse in percentuale rispetto al numero totale di predizioni. Si tratta di un ottimo strumento per capire quali classi possono creare confusione al modello. Nella matrice della Figura 4.2, possiamo notare anche come sia presente una classe *background* che ci fornisce informazioni sui FN e FP. Alcune osservazioni che possiamo fare leggendo la Figura 4.2, sono:

- l'oggetto *driver-seat-belt*, è stato riconosciuto nel 96% dei casi correttamente. Nei restanti:
  - il 3% delle volte non viene classificato
  - l'1% delle volte viene scambiato per un *passenger-seat-belt*
- nel 6% dei casi, il modello ha riconosciuto bounding box per *passenger-seat-belt* anche in assenza di questi ultimi.

Figura 4.3: Esempio predizione del modello addestrato

Una volta ottenuti i pesi del modello allenato sul Dataset A e Dataset B, possiamo salvarli come meglio vogliamo. Avendo sviluppato l'intero progetto sfruttando Colab, è stato preferito caricarli su Google Drive. Per riuscire a distinguere tra loro i pesi salvati, ho adottato la seguente naming convention:

`{dataset_version}_{epochs}_pretrained_weights_YYYY-MM-DD_H-M-S.pt`

dove:

- `dataset_version` indica la versione del dataset su cui è stato addestrato il modello (Dataset A o Dataset B)
- `epochs` indica per quante epoche il dataset è stato addestrato
- `YYYY` è l'anno, `MM` il mese, `DD` il giorno, `H` l'ora, `M` i minuti e `S` i secondi

4.5 Classificazione video

Facciamo un riepilogo di quanto fatto finora:

- Abbiamo costruito il dataset delle immagini, creandone due varianti per confrontare il modello allenato su oggetti in $A^{\text{Dataset A}}$ e $A^{\text{Dataset B}}$
- In seguito abbiamo effettuato delle operazioni di preparazione in vista della fase di training

- Infine abbiamo salvato i pesi ottenuti dall'addestramento del modello sui dataset

Il seguente passo è quello di costruire il vero e proprio sistema. L'idea è che, per effettuare una classificazione più accurata, il sistema si aspetta di ricevere in input una sequenza di immagini dall'utente. In questo modo, anche se dovessero essercene alcune in cui il modello fatica a rilevare gli oggetti (e quindi sbagliare a classificarlo o non riuscire proprio), siamo comunque in grado di fornire una classe. In particolare, si tratterà della classe c associata al maggior numero di frame.

A seguire riportiamo il codice Python della funzione che, data la cartella contenente i frame, **ritorna la classe** associata da ogni euristica sull'**intero flusso di immagini**.

```
def predict_on_one_video(
    video_directory,
    window_size = 15
    ):
  '''
    Numero di frame classificati come "driver",
    con le finestre scorrevoli
  '''
  driver_frames = {}

  driver_frames[object_cardinalities_name] = 0
  driver_frames[MCS_name] = 0
  driver_frames[MCS_with_activation_name] = 0

  '''
    Numero di frame classificati come "passenger",
    con le finestre scorrevoli
  '''
  passenger_frames = {}
  ...

  '''
    Numero di frame classificati come "none",
    con le finestre scorrevoli
  '''
  none_frames = {}
  ...

  # Classificazioni dei frame, con le euristiche
  frame_classifications = {}
  ...
```

```
32
33  # Lista dei file della directory, ordinata
34  directory_files = natsorted(os.listdir(video_directory))
35  # Indice del frame sotto osservazione
36  frame_counter = 0
37
38  for filename in directory_files:
39    '''
40      Prima di dare in pasto l'immagine al modello,
41      le ridimensioniamo alla dimensione utilizzata
42      per il training, per far lavorare meglio il modello
43    '''
44    img = cv2.imread(f"{video_directory}/{filename}")
45    img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
46    img = cv2.resize(img, (TRAINING_WIDTH, TRAINING_HEIGHT))
47    '''
48      Diamo in input al modello il frame
49      su cui riconoscere gli oggetti in A^{DatasetVersion}
50    '''
51    results = model(f"{video_directory}/{filename}", conf=0.2)
52
53    # Otteniamo le classificazioni c_k dalle euristiche
54
55    frame_classifications[object_cardinalities_name].append(
56        object_cardinalities(results)
57    )
58    frame_classifications[MCS_name].append(
59        MCS(results)
60    )
61    frame_classifications[MCS_with_activation_name].append(
62        MCS_with_treshold_activation(results, 0.25)
63    )
64
65    if frame_counter > (window_size - 1):
66      # Otteniamo le ultime window_size classificazioni
67      for counter in range(1, window_size+1):
68        windowd_classifications, windowp_classifications =
69        get_window_classifications(
70            frame_classifications,
71            heuristics,
72            frame_counter,
```

```
            counter
          )
        '''
          Classifichiamo il frame corrente con le
          finestre scorrevoli e aggiorniamo i contatori
        '''
        driver_frames, passenger_frames, none_frames =
        update_classifications(
          windowd_classifications,
          driver_frames,
          windowp_classifications,
          passenger_frames,
          heuristics,
          none_frames
        )
      frame_counter += 1
    # Otteniamo la classe dell'intero video
    pred_video_class = get_video_classification(
      driver_frames,
      passenger_frames,
      heuristics
      )
    return pred_video_class
```

L'esecuzione di `predict_on_one_video("videos/video1", window_size)` potrebbe, per esempio, restituire le seguenti classi:

```
{
    "baseline": "driver",
    "mcs": "passenger",
    "mcsT": "passenger"
}
```

Questo implica che applicando la *baseline* come euristica di classificazione delle immagini, alla maggior parte di queste ultime è stata associata la classe "driver". Similmente per *mcs* e *mcsT*.

A questo punto, per rispondere alle domande che ci eravamo posti nella Sezione 2.2, abbiamo bisogno di arricchire il codice della `predict_on_one_video` per stimarne le prestazioni. In particolare vogliamo tenere traccia delle seguenti metriche:

- I tempi di attivazione.
- Accuracy, precision e recall delle classificazioni dei frame.

Nelle sezioni a seguire riporteremo i passaggi chiave per implementarle.

4.5.1 Implementazione: tempi di attivazione

La Sezione 3.1 spiega in dettaglio come è stata definita questa metrica. Abbiamo bisogno di tenere traccia di un contatore delle finestre analizzate, in modo tale da cominciare a registrare le *classification-confidence* dalla finestra γ-esima in poi:

```
def predict_on_one_video(...):
  ...
  end = 0
  '''
    Tracciamo il tempo che impieghiamo a
    raggiungere una soglia di classification-confidence K
  '''
  frame_sec = 1 / fps
  for filename in directory_files:
    ...
    if frame_counter > (window_size - 1):
      ...
      # window_counter = frame_counter + 1 - window_size
      window_counter += 1
      frame_sec += 1 / fps
      '''
        Iniziamo ad osservare la classification-confidence
        dopo la \gamma-esima finestra
      '''
      if (window_counter >= start_recording_window):
        confidence = compute_classification_confidence(
            driver_frames,
            passenger_frames,
            heuristics,
            window_counter
        )
        for heuristic in heuristics:
          for k in range(0, len(confidence_list)):
            # Se non è stata ancora raggiunta l'accuracy
            if classification_confidence[k].get(heuristic) == None:
              '''
                Registriamo il tempo se viene raggiunta
                altrimenti aumentiamo \gamma e riproviamo
              '''
              if confidence[heuristic] > confidence_list[k]:
                end = time.time()
```

```
37                 classification_confidence[k][heuristic] = frame_sec
38       ...
39     ...
40     return (..., classification_confidence, ...)
```

In questo modo abbiamo aggiunto la possibilità di ottenere anche il momento in cui viene raggiunto un arbitrario valore di *classification-confidence* k, ma resta ancora un problema: potremmo ritrovarci un **video** in cui **non viene mai raggiunta la *classification-confidence*** k. Poichè vogliamo ottenere degli oggetti che possano essere facilmente confrontabili tra loro, in caso di *classification-confidence* non raggiunta, è stato deciso di considerare come tempo di attivazione, *sec*. Un modo di gestire questo caso in un'applicazione reale potrebbe essere quello di stabilire un tempo massimo `MAX_ACTIVATION_TIME` per la classificazione del flusso di immagini.

4.5.2 Implementazione: accuracy, precision e recall

Per tracciare queste metriche di valutazione, dobbiamo fissare le seguenti classi:

- passeggero, come classe negativa.
- conducente, come classe positiva.

È **importante** notare che, in **alcune istanze**, le **euristiche** potrebbero non riuscire a effettuare **alcuna classificazione**, introducendo quindi immagini "none", ovvero non classificate. Teniamo presente che nel calcolo di precision e recall, questa classe non verrà inclusa, poichè ci concentreremo principalmente sul confronto tra guidatore e passeggero.

Nel nostro specifico caso d'uso, queste metriche possono essere interpretate nel seguente modo:

- Accuracy: **Quanti frame sono stati classificati correttamente?**
- Precision: **Dei frame classificati come guidatori, quanti erano appartenenti a video di guidatori veri?** Ricordiamoci che, poichè non stiamo tenendo in considerazione i frame senza classe, potremmo avere un'ottima precision con un'accuratezza molto bassa (pensiamo ad un caso estremo: TP = 1 e FP = 0 ma con $sec * fps = 64$).
- Recall: **Dei frame appartenenti a video di guidatori, quanti siamo riusciti a classificare correttamente come guidatori?** Anche qua, considerando che la classe "none" non è inclusa nel calcolo, avrebbe più senso la seguente interpretazione: "Dei frame appartenenti a video di guidatori, quanti siamo riusciti a classificare correttamente come guidatori **e non come passeggeri**?"

Per implementare queste metriche è necessario prevedere un parametro aggiuntivo alla `predict_on_one_video` che ci permetta di catturare la *ground truth* del video in analisi. Per evitare di dover specificare manualmente questo parametro, ho pensato di aggiungere un file nella cartella delle immagini, con la seguente naming convention: c`.class`.

Esempio:

```
videos
  video_1
    driver.class
    video_1_frame_1.png
    video_1_frame_2.png
    video_1_frame_3.png
    ...
  video_2
    passenger.class
    video_2_frame_1.png
    video_2_frame_2.png
    video_2_frame_3.png
    ...
  ...
```

4.6 Risultati

È giunto il momento di testare il nostro sistema, utilizzando i video presentati nella Sezione 4.1, **valutando le sue prestazioni** sui due **modelli** ottenuti dal **dataset** A **e** B. In questo contesto, **esamineremo anche il comportamento delle euristiche** applicate durante il processo di classificazione.

Questi test ci permetteranno di rispondere alle domande che ci eravamo posti nella Sezione 2.2, oltre che a valutare se le scelte progettuali da noi effettuate abbiano portato cambiamenti alle performance. Anticipiamo che il nostro sistema è riuscito a predire correttamente nella maggior parte dei casi la classe dei video. Per i restanti casi, siamo comunque riusciti ad ottenere le classificazioni giuste solo dopo aver arricchito il codice della `predict_on_one_video` (i dettagli saranno spiegati nella Sezione 4.6.3).

Tutti i risultati riportati, sono stati ottenuti allenando il modello per 5 epoche, con $\gamma = 20$ e $l = 10$.

4.6.1 Risultati: tempi di attivazione

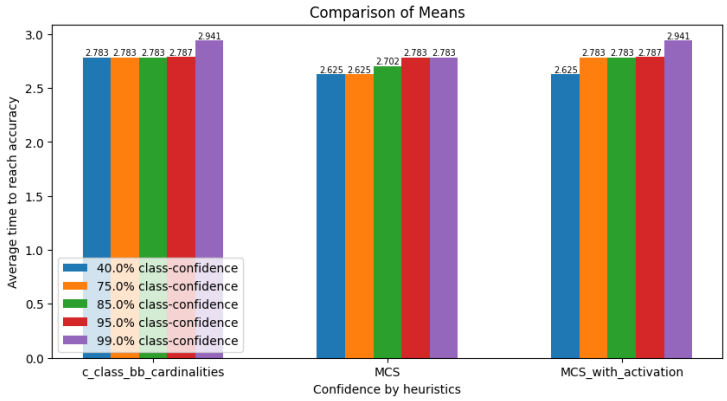

Figura 4.4: Grafico a barre dei tempi medi impiegati per raggiungere le *classification-confidence* dal sistema che utilizza il modello addestrato sul Dataset A, uno per ogni euristica

Prima di effettuare il confronto dei tempi di attivazione, ho deciso di definire una lista delle *classification-confidence* da tracciare (`confidence_list`), in modo da avere una chiara visualizzazione delle prestazioni dei modelli all'aumentare delle richieste. Si noti che lo sviluppatore di un'applicazione che utilizza questo sistema di rilevazione, prima di mettere in produzione il suo prodotto software, deve fissare la *classification-confidence* K, la finestra dalla quale cominciare a registrare i tempi γ e l'ampiezza della finestra l, in base alle sue esigenze [8] (e.g. se volesse minimizzare il tempo di rilevazione mantenendo allo stesso tempo delle classificazioni affidabili, potrebbe configurare una soglia K alta e mantenere γ e/o l bassi). Inoltre, è importante sottolineare che i tempi di cui discuteremo **non rappresentano i tempi di esecuzione dell'algoritmo**. Come già spiegato nella Sezione 3.1, il tempo che registriamo, è calcolato in base agli fps del flusso di immagini. Nel nostro specifico caso, per ogni iterazione sui frame di un video, paghiamo un "prezzo" di $\frac{1}{8}$ secondi (0.125).

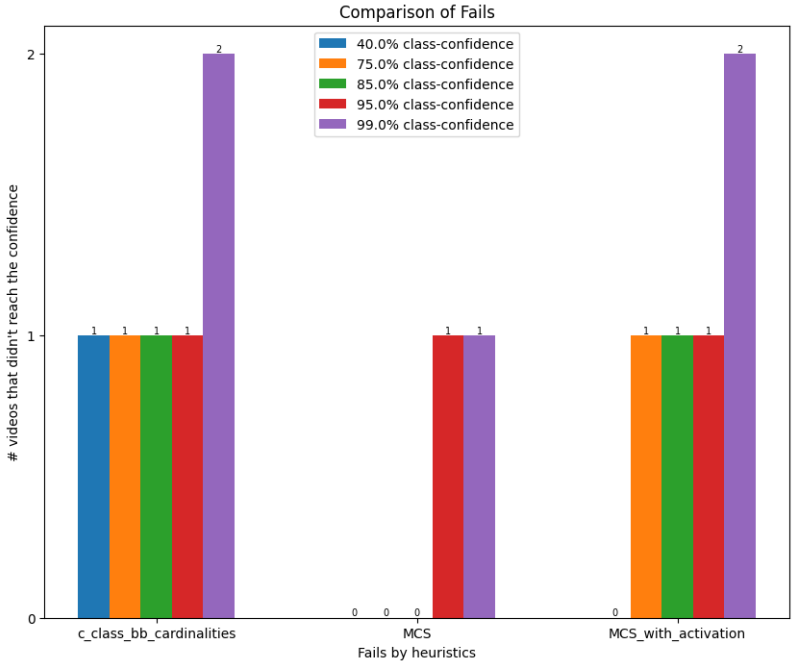

Figura 4.5: Grafico a barre del numero di video che non hanno raggiunto le *classification-confidence* utilizzando il sistema addestrato sul Dataset A, uno per ogni euristica

La Figura 4.4 mostra come, all'aumentare del valore delle *classification-confidence*, aumentano anche i tempi necessari per raggiungerle. Inoltre possiamo vedere come sembrano non esserci delle differenze veramente significative tra le performance delle varie euristiche.

Ricordiamoci che, nella Sezione 3.1, abbiamo fatto notare che potrebbero esserci dei flussi di immagini che non raggiungono la soglia K. Nella Figura 4.5 vediamo come ***mcs*** sia l'euristica che **fallisce** nel **minor numero di video**. Non è un risultato sorprendente, sappiamo dalla Sezione 4.2.2, che la *mcs* è caratterizzata dal fatto di rendere **rare le classificazioni in "parimeriti"**. Di conseguenza, l'algoritmo della Sezione 3.1 potrà raggiungere *classification-confidence* più alte grazie alla minor presenza di classi "none", rispetto alle altre euristiche.

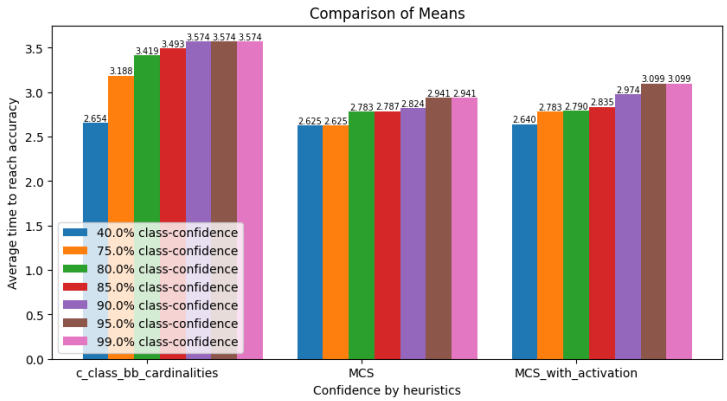

Figura 4.6: Grafico a barre dei tempi medi impiegati per raggiungere le *classification-confidence* dal sistema che utilizza il modello addestrato sul Dataset B, uno per ogni euristica

La Figura 4.6 mostra invece il comportamento del sistema del modello addestrato sul Dataset B. Possiamo notare come, l'introduzione di nuovi oggetti, non sembra portare miglioramenti particolarmente significativi per i tempi di attivazione. In particolare le prestazioni della *baseline*, sembrano essere leggermente peggiorate. Questo comportamento sarà spiegato meglio nella sezione a seguire, per ora ci basta sapere che è legato alle classificazioni nulle.

4.6.2 Risultati: accuracy, precision e recall

In una delle sezioni precedenti (4.5.2), abbiamo cercato di interpretare accuracy, precision e recall adattandole al nostro caso d'uso. Sulla base di queste interpretazioni possiamo fare la seguente osservazione: **è molto importante non scambiare un conducente per un passeggero**, se vogliamo prevenire incidenti dovuti al *texting and driving*. Questo implica che, nel nostro specifico caso, a parità di performance vogliamo assolutamente prediligere sistemi che abbiano un'alta recall. Per fortuna, in quasi tutti i confronti che vedremo, siamo riusciti ad ottenere una recall uguale o molto vicina al 100%. Questo risultato è estremamente positivo e ci consente di concentrare il nostro studio sulle altre due metriche.

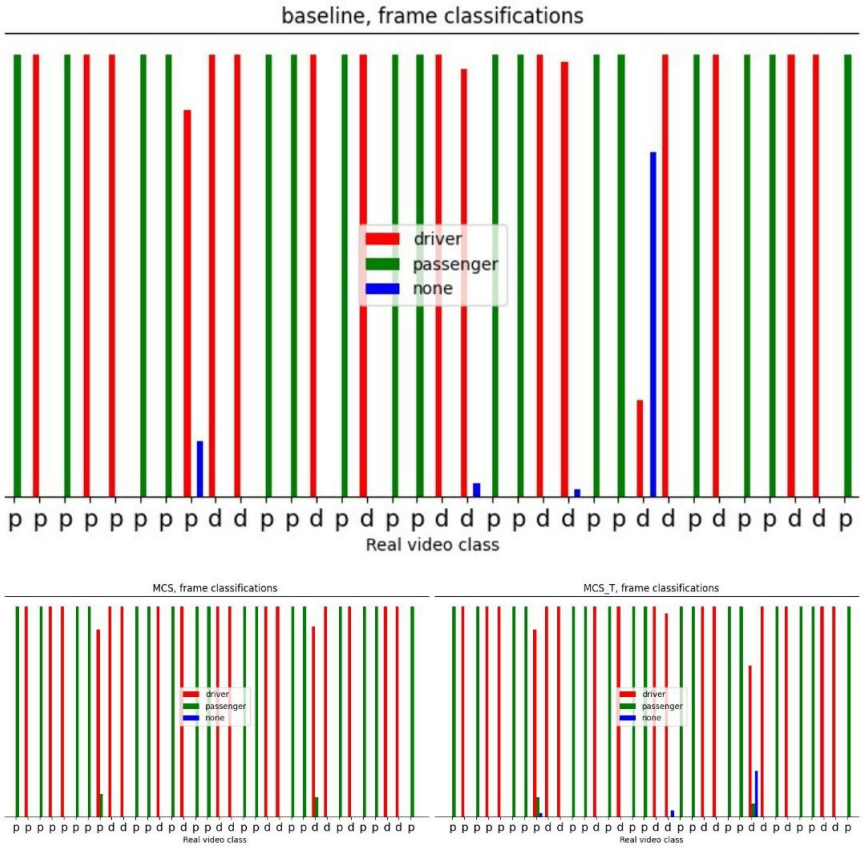

Figura 4.7: Grafico a barre delle classificazioni delle immagini effettuate dal sistema che utilizza il modello addestrato sul Dataset A, uno per ogni euristica

La Figura 4.7 ci mostra la distribuzione delle classificazioni delle $sec * fps$ immagini, di tutti i video. Ognuna delle n lettere sull'asse delle x, indica la *ground truth* del video:

- p sta per *passenger*
- d sta per *driver*

Possiamo vedere che, come fatto notare nelle criticità del lavoro [8], i video da passeggero posteriore al lato conducente sono stati classificati come conducenti da tutte le euristiche.

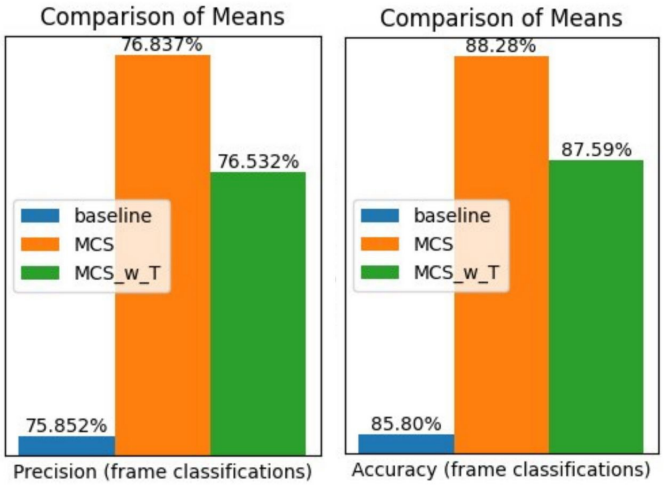

Figura 4.8: Precision (sinistra) e accuracy (destra) delle classificazioni, Dataset A

Dalla Figura 4.8 possiamo confermare quello che si intravedeva nel grafico precedente: *mcs* e *mcsT*, anche se di poco, hanno delle performance migliori rispetto alla *baseline*, almeno sulla versione A del dataset.

Ora è il momento di vedere come si comporta il sistema con l'aggiunta dei nuovi oggetti. Come anticipato anche a fine delle sezione precedente, guardando la Figura 4.10 non sembrano esserci stati miglioramenti con l'introduzione di nuove classi. In linea generale, il comportamento delle euristiche in tutti i casi è abbastanza buono (riusciamo a raggiungere degli alti livelli di accuracy), ma il sistema non riesce ancora a fare una distinzione chiara tra le immagini di un conducente e quelle di un passeggero posteriore.

Proprio per quest'ultimo motivo, nella Figura 4.6, i tempi di attivazione non sono migliorati sul Dataset B.

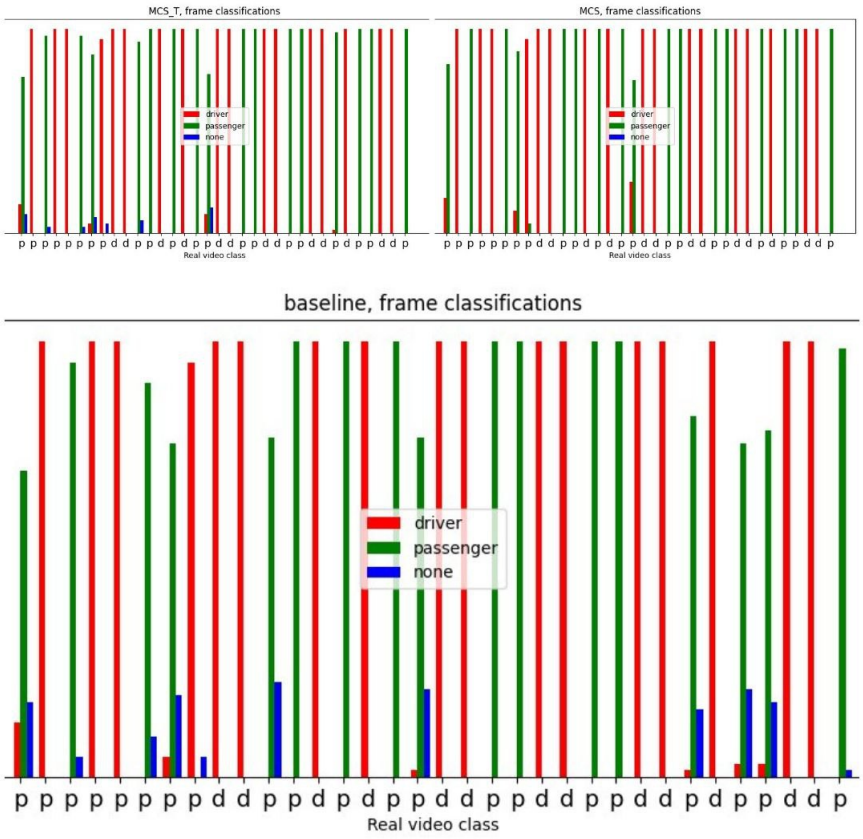

Figura 4.9: Grafico a barre delle classificazioni delle immagini effettuate dal sistema che utilizza il modello addestrato sul Dataset B, uno per ogni euristica

È importante ricordarsi che la *baseline* è un'euristica basata sulla quantità di oggetti rilevati all'interno di un'immagine. Nella Sezione precedente abbiamo fatto un'osservazione: i tempi di attivazione della *baseline* sul Dataset B sono leggermente peggiori rispetto ai risultati ottenuti sul Dataset A. Dalla Figura 4.9 emerge chiaramente che con l'introduzione di nuovi oggetti l'euristica fatica a classificare alcuni frame, aumentando quindi i tempi di raggiungimento di un'arbitraria soglia K.
Questo potrebbe indicare che, in un futuro, il dataset potrebbe essere ulteriormente espanso, aggiungendo più occorrenze dei nuovi oggetti in modo tale da diminuire le classificazioni "none".

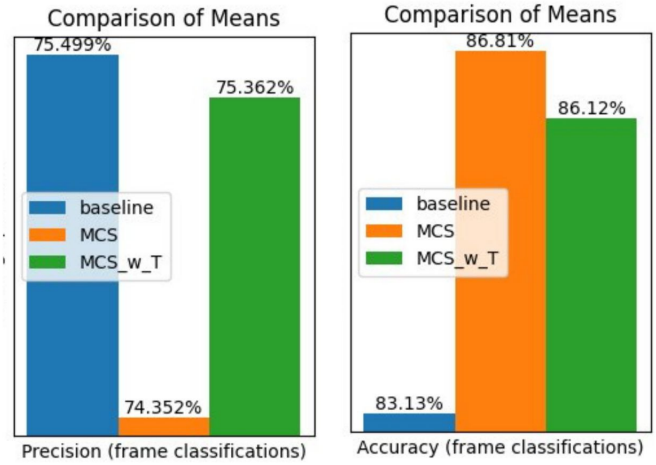

Figura 4.10: Precision (sinistra) e accuracy (destra) delle classificazioni, Dataset B

4.6.3 Risultati: passeggeri posteriori

Come già fatto notare in 4.1, sappiamo che driver-grab-handle $\notin A_{\text{driver}}$. Potremmo utilizzare questa informazione a nostro vantaggio modificando il codice della `predict_on_one_video`, per fare in modo che, non appena venga riconosciuta una maniglia con una confidence maggiore di una soglia `SPECIAL_OBJ_TRESHOLD` tra gli oggetti di un'immagine k, venga assegnata la classe *passenger* a c_k, ancor prima di applicare le euristiche. Si tratta di un'assunzione molto forte: se il conducente guidasse un'auto dove al posto di guida fosse presente una maniglia al di sopra del finestrino, riuscirebbe a "ingannare" il sistema di rilevazione. Tuttavia, in generale, possiamo fare le seguenti osservazioni:

- Le persone che guidano non possono utilizzare le maniglie durante la guida, perchè dovrebbero avere entrambe le mani sul volante.
- Le maniglie sono utilizzate anche come supporto per uscire/entrare dal veicolo, ma i conducenti hanno già il volante a cui aggrapparsi o spingersi

Dunque non è così raro trovare auto che, come nel caso di quella utilizzata per registrare i video di test, non siano provviste di maniglie al posto di guida.

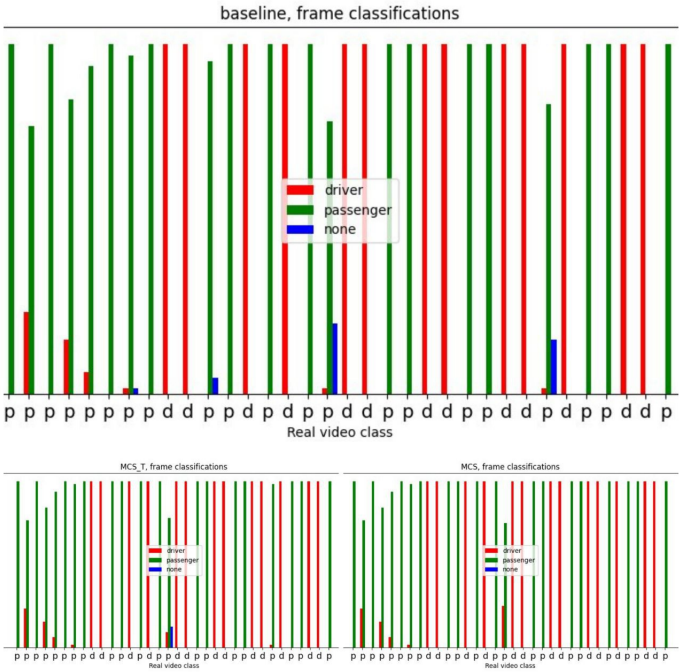

Figura 4.11: Grafico a barre delle classificazioni delle immagini, Dataset *B*

Come previsto (si guardi la Figura 4.11 e 4.12), con queste modifiche siamo riusciti a riconoscere tutti i passeggeri, classificandoli correttamente e riuscendo a raggiungere un'accuratezza sulle classificazioni dei video del 100%. Un risultato straordinario da cui possiamo fare la seguente considerazione: un sistema come il nostro, che utilizza **esclusivamente** la fotocamera frontale e l'object detection, può risolvere il problema della classificazione dei passeggeri posteriori, soltanto con l'introduzione di nuovi oggetti che lo aiutino a distinguere questi ultimi dal conducente (e.g. maniglia), e viceversa.

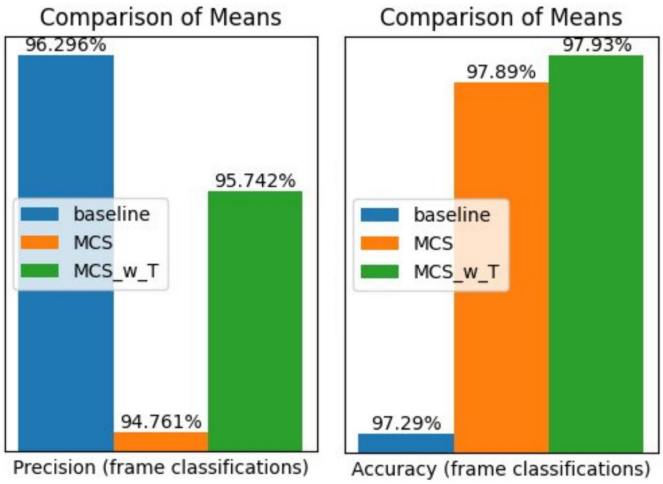

Figura 4.12: Precision (sinistra) e accuracy (destra) del sistema, Dataset B

4.6.4 Risultati: considerazioni finali

Ricapitoliamo quanto osservato grazie ai grafici:

- In media, fissato un valore K di *classification-confidence*, l'euristica che lo raggiunge in meno tempo è *mcs*, oltre a essere quella che fallisce nel minor numero di video a raggiungerla, sui dati di test.
- I tempi di raggiungimento di una soglia di *classification-confidence* K utilizzando il modello basato sul Dataset B, sono leggermente superiori a quelli del modello basato sul Dataset A. Questo potrebbe indicare la necessità di aumentare le occorrenze dei nuovi oggetti nei dataset di train, per aiutare il modello ad aumentare la sua robustezza.
- Nella maggioranza dei casi, le nostre euristiche hanno prodotto ottimi valori di recall, un risultato interessante in vista di una possibile applicazione reale.
- Sul dataset di test, si è scoperto che:
  - In media, la *mcs* è l'euristica con il miglior valore di accuracy tra tutte (90.99% contro i 90.54% della *mcsT* e 88.74% della *baseline*).
  - Le precision invece, sono molto simili tra di loro.
- Per riuscire a discriminare i passeggeri posteriori dal conducente, ci siamo avvalsi di una caratteristica del nostro dataset, senza la quale non avremmo potuto raggiungere gli ottimi risultati presentati.

Capitolo 5

Conclusioni e sviluppi futuri

In questo progetto di tesi abbiamo presentato una soluzione al problema dell'uso dello smartphone durante la guida (*texting and driving*). Il sistema prevede l'uso di un modello YOLO per effettuare object detection sulle immagini provenienti dalla fotocamera frontale dello smartphone, per poi classificare l'intero flusso di immagini ricevute grazie all'uso di euristiche e finestre scorrevoli sui frame. Il lavoro nasce con l'intento di contribuire ad un avanzamento dello stato dell'arte. In particolare abbiamo preso come riferimento il lavoro [8] e ne abbiamo esteso il dataset, eliminato la dipendenza da Roboflow Train integrando YOLO e tentato di risolvere la classificazione dei passeggeri posteriori oltre che a cercare di ottenere risultati migliori attraverso l'introduzione di nuove euristiche di classificazione. Abbiamo studiato in dettaglio il comportamento delle euristiche, testandole prima sul sistema del lavoro [8] e poi sul sistema esteso al riconoscimento di nuovi oggetti, riuscendo ad ottenere dei risultati leggermente migliori rispetto alla classificazione della *baseline*. Siamo stati in grado di risolvere la classificazione del passeggero al lato di guida, introducendo un "oggetto discriminante" che ci permettesse di distinguere chiaramente i conducenti dai non-conducenti. Questo ci ha portati a raggiungere dei livelli di precision e accuracy ottimi, riuscendo inoltre, per il dataset di test su cui sono state tracciate le performance del sistema, ad ottenere un'accuratezza del 100% sulle classificazioni dei video.
Nel futuro, potrebbero essere fatti dei lavori in cui vengono rilassate alcune delle assunzioni che abbiamo fatto durante lo sviluppo di questo progetto. In primis, per poter sfruttare la rilevazione in un sistema reale, dobbiamo capirne il comportamento anche in situazioni di scarsa luminosità.
Le maniglie ci hanno aiutato a distinguere tra conducente e passeggero durante il corso di questa ricerca, ma in uno scenario reale questo metodo potrebbe non essere applicabile (pensiamo alle auto sprovviste di maniglie). Per questo un altro possibile sviluppo è quello di, partendo dal nostro sistema di rilevazione, costruirne un altro che sfrutti, oltre alla fotocamera, anche i sensori inerziali dello smartphone, come visto nel lavoro [1]. In questo modo evitiamo di doverci affidare esclusivamente alla presenza di "ogget-

ti discriminanti", che dovrebbero essere indicati volontariamente da parte dell'utente, rendendolo facilmente "ingannabile" da chiunque.

Bibliografia

[1] Luca Bedogni, Octavian Bujor e Marco Levorato. "Texting and Driving Recognition Exploiting Subsequent Turns Leveraging Smartphone Sensors". In: *2019 IEEE 20th International Symposium on "A World of Wireless, Mobile and Multimedia Networks" (WoWMoM)*. 2019, pp. 1–9. DOI: 10.1109/WoWMoM.2019.8793032.

[2] José Celaya-Padilla et al. ""Texting Driving" Detection Using Deep Convolutional Neural Networks". In: *Applied Sciences* 9 (lug. 2019), p. 2962. DOI: 10.3390/app9152962.

[3] Wikipedia contributors. *Objects detected with OpenCV's Deep Neural Network module (dnn). Reading a network model stored in Darknet model files.It uses a YOLOv3 model trained on COCO dataset capable of detecting 80 common objects in context.* 2019. URL: https://en.wikipedia.org/wiki/File:Detected-with-YOLO--Schreibtisch-mit-Objekten.jpg.

[4] Wikipedia contributors. *The green line represents an overfitted model and the black line represents a regularized model. While the green line best follows the training data, it is too dependent on that data and it is likely to have a higher error rate on new unseen data illustrated by black-outlined dots, compared to the black line.* 2008. URL: https://commons.wikimedia.org/wiki/File:Overfitting.svg.

[5] Rohith Gandhi. *R-CNN, Fast R-CNN, Faster R-CNN, YOLO — Object Detection Algorithms*. 2018. URL: https://towardsdatascience.com/r-cnn-fast-r-cnn-faster-r-cnn-yolo-object-detection-algorithms-36d53571365e.

[6] Ross B. Girshick et al. "Rich feature hierarchies for accurate object detection and semantic segmentation". In: *CoRR* abs/1311.2524 (2013). arXiv: 1311.2524. URL: http://arxiv.org/abs/1311.2524.

[7] Tsung-Yi Lin et al. "Microsoft COCO: Common Objects in Context". In: *Computer Vision – ECCV 2014*. A cura di David Fleet et al. Cham: Springer International Publishing, 2014, pp. 740–755. ISBN: 978-3-319-10602-1.

[8] Federico Montori, Marco Spallone e Luca Bedogni. "Texting and Driving Recognition leveraging the Front Camera of Smartphones". In: *2023 IEEE 20th Consumer Communications Networking Conference (CCNC)*. 2023, pp. 1098–1103. DOI: 10.1109/CCNC51644.2023.10060838.

[9] Joseph Redmon et al. *You Only Look Once: Unified, Real-Time Object Detection*. 2016. arXiv: 1506.02640 [cs.CV].

[10] Lichao Yang et al. "Recognition of visual-related non-driving activities using a dual-camera monitoring system". In: *Pattern Recognition* 116 (2021), p. 107955. ISSN: 0031-3203. DOI: https://doi.org/10.1016/j.patcog.2021.107955. URL: https://www.sciencedirect.com/science/article/pii/S0031320321001424.

Ringraziamenti

Per primi, vorrei ringraziare il Dott. Federico Montori e il Prof. Luca Bedogni, miei relatori, per la loro disponibilità e per avermi indirizzato durante lo sviluppo del progetto.

Ringrazio Michelangelo, per essersi preso cura della mia famiglia, anche se a volte ci siamo scontrati.

Ringrazio Alex perchè senza di lui questi anni (non solo quelli dell'uni) sarebbero stati pesanti. È grazie a te se ho scoperto la passione per l'informatica, grazie a quella scelta delle superiori.

Ringrazio tutte le persone che mi hanno visto crescere in questi anni.

Anche se è già dedicata a voi, volevo comunque ringraziarvi:

Ringrazio mia sorella per avermi sempre sopportato e supportato da tutta la vita, e per essere la migliore sorella del mondo. Questo traguardo è anche tuo.

Ringrazio il mio amore, il "bruco". In te ho sempre avuto modo di rifugiarmi quando pensavo di non potercela più fare, quando mi sentivo al limite. Grazie, ti amo.

Infine ringrazio mia madre, che ammiro. Farò del mio meglio per renderti fiera, non scorderò mai tutti i sacrifici che hai fatto per portarci qui.

Senza tutti loro non ce l'avrei fatta. Grazie.