Dipartimento di Informatica
Corso di Laurea in INFORMATICA

L'Intelligenza Artificiale nella elaborazione delle immagini: tecniche di visione artificiale per il monitoraggio della guida di motocicli

Relatore:
Prof. Antonio Carta

Presentata da:
Davide Bulotta

Co-Relatori:
Ing. G. Riccardo Leone
Prof. Marco Righi

Sessione estiva
Anno Accademico 2022/2023

Contents

1 Introduzione

Il tirocinio si è svolto presso l'Istituto di Scienza e Tecnologie dell'Informazione del Consiglio Nazionale delle Ricerche e ha richiesto circa cinque mesi. Il tirocinio si è svolto nell'ambito del progetto di ricerca Artificial Intelligence driven RIding Distributed Eye (AI-RIDE) che ha come obiettivo lo studio e la realizzazione prototipale di un framework di Intelligenza Artificiale integrato nel contesto della formazione dei motociclisti, mirando in particolare ad una verifica standard, misurabile e imparziale dell'esame della patente di guida per motocicli. L'esito di un esame di patente di guida dipende da diversi fattori, come il tempo di prestazione, la precisione della traiettoria, la gestione della velocità e l'esecuzione di un percorso netto senza effettuare penalità. La maggior parte di tali fattori può essere misurata utilizzando strumenti di analisi delle immagini ottenute per mezzo di telecamere esterne opportunamente posizionate. L'obiettivo di questo tirocinio è stato ricavare tali informazioni dai flussi video a disposizione massimizzando la precisione dei dati ottenuti.

Il primo compito è consistito nell'allenamento di un modello di Object Detection per il riconoscimento dei coni, del motociclo e del pilota; utilizzando vari strumenti di annotazione si è popolato un dataset con numerosi frame estratti dai video realizzati nel sito di prova, la pista della scuola guida Gerardo di Pontedera; tale dataset è stato continuamente migliorato aggiungendo immagini di casistiche specifiche, conducendo test a diverse risoluzioni e utilizzando numerosità crescenti dei modelli di partenza (come spiegato nei paragrafi 3.1.7 e 4.2.8). Raggiunta una soddisfacente prestazione del task di riconoscimento si è implementata la logica funzionale alla individuazione precisa della posizione della moto nello spazio e all'eventuale penalità connessa con lo spostamento di uno o più coni segnaletici (si veda paragrafo 4.3.1 e 4.3.2). Queste informazioni sono servite da input al sottosistema di fusione delle informazioni che non è compreso nel lavoro di questo tirocinio. Il modello custom di object detection ha conseguito ottimi risultati in termini di confidenza e precisione nel riconoscimento degli elementi in gioco ovvero coni, moto e pilota. La logica implementata per il rilevamento delle penalità commesse durante l'esame di guida è stata essenziale per il successo finale del sistema. Il lavoro svolto è stato presentato il giorno 14 giugno 2023 presso il sito di prova in un evento che ha previsto una dimostrazione del sistema dal vivo includendo vari scenari di guida: tutte le penalità principali sono state segnalate con elevata precisione spaziale e temporale.

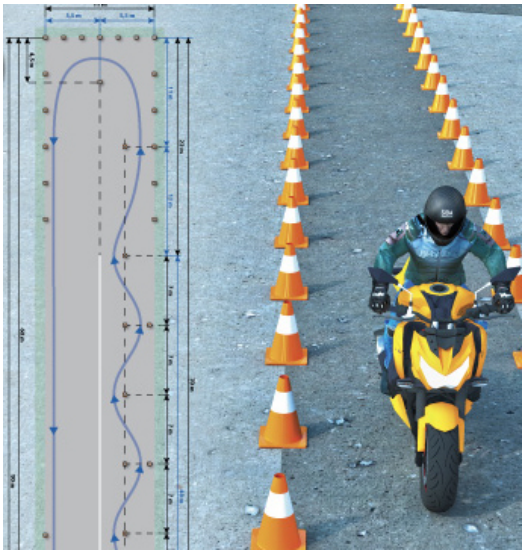

Figure 1: L'esame di guida su motocicli si svolge su percorsi predefiniti delimitati da coni stradali. L'allenamento di un modello di Object Detection per il riconoscimento dei coni, del motociclo e del pilota ha svolto un ruolo fondamentale per la buona riuscita del sistema (Fonte immagine: patente.it).

La relazione è suddivisa nei seguenti capitoli:

- **Basi di partenza del tirocinio** "fotografa" la situazione di partenza del tirocinio ovvero descrive la progettazione ad alto livello del sistema da implementare e quanto già realizzato in termini di raccolta dati.
- **Fondamenti teorici e tecnologie utilizzate** presenta le nozioni teoriche dei problemi affrontati con particolare attenzione per i sistemi di Object Detection basati su reti convoluzionali; sono inoltre descritte tutte le tecnologie di cui si è fatto uso all'interno del tirocinio
- **Lavoro svolto** mostra i sistemi creati e le loro soluzioni; in questo capitolo si descrivono in modo dettagliato le procedure di installazione, utilizzo e implementazione delle tecnologie descritte nel capitolo precedente.
- **Appendici** Raggruppa le informazioni più generiche, ma essenziali ai fini del progetto. Contiene una rappresentazione completa degli algoritmi utilizzati.

2 Basi di partenza del tirocinio

Questo tirocinio è iniziato a Gennaio 2023 presso il Laboratorio di Segnali e Immagini dell'Istituto di Scienza e Tecnologia dell'Informazione (ISTI) del CNR di Pisa. Il lavoro si è svolto nell'ambito del progetto di ricerca Artificial Intelligence driven RIding Distributed Eye (AI-RIDE)[1] che ha l'obiettivo di utilizzare tecniche di Intelligenza Artificiale per automatizzare il processo di valutazione dell'esame di guida per la patente A. Nei mesi precedenti l'inizio del tirocinio i ricercatori dello ISTI-CNR hanno analizzato le problematiche da risolvere, definito i requisiti delle telecamere effettuando vari test a risoluzioni e framerate differenti e installato la sensoristica necessaria nel sito di prova. Tale lavoro iniziale è stato presentato alla "16th International Conference on Signal-Image Technology & Internet-Based Systems (SITIS)" con l'articolo [1]. Nei seguenti paragrafi si riporta un riassunto delle parti essenziali di tale articolo, che rappresenta il punto di partenza di questo tirocinio.

2.1 Il progetto AI-RIDE

Lo stato dell'arte dei sensori Internet of Things (IoT) in ambito automotive si sta evolvendo in modo significativo negli ultimi anni, sospinto dall'ecosistema automobilistico e dalla guida autonoma di nuova generazione, che richiedono un'enorme quantità di dati online per prevedere, calcolare e mantenere traiettorie e comportamenti ottimali di auto e moto. Tuttavia, nell'ambito delle fasi di apprendimento umano e di addestramento alla guida di motociclette, tali strumenti non sono attualmente adeguati a fornire un feedback istruttivo a manovre anomale, pericolose o imprecise, in quanto progettati per interagire con un sistema informatico piuttosto che con l'uomo. Il progetto di ricerca AI-RIDE vuole supplire a questa mancanza: esso si prefigge lo studio e la realizzazione prototipale di un framework di Intelligenza Artificiale accelerato, online e integrato nel contesto della formazione dei motociclisti, mirando in particolare agli strumenti di verifica delle sessioni dei corsi pratici di guida e dell'esame della patente di guida per motocicli. Per inquadrare bene lo scenario e le problematiche connesse è bene partire dalla definizione precisa così come definito dalla legge italiana [2]. L'esame della patente di guida per motocicli si compone di tre fasi: le prime due sono finalizzate a

[1] AIRIDE è connesso al progetto europeo Very Efficient Deep Learning for Internet of Things (VEDLIoT) iniziato a Luglio 2022.

dimostrare le capacità di guida su un circuito "sicuro" chiuso al traffico, e, solo dopo averle superate con esisto positivo, è possibile effettuare la terza prova finale su strada pubblica. In particolare, le prime due prove si svolgono su percorsi predefiniti con misure standard delimitate da coni stradali come illustrato in figura 2: un percorso è più corto con passaggi ravvicinati e bassa velocità e l'altro è più lungo e richiede velocità più elevate. Su entrambe c'è uno slalom iniziale, poi una curva e infine un rettilineo. Alla fine della pista lunga, il veicolo deve essere fermato in uno spazio specifico. Tutto deve essere fatto rispettando dei vincoli di tempo predefiniti. Per non compromettere il buon esito dell'esame le prove devono essere svolte senza commettere le seguenti penalità:

- toccare uno o più coni;
- saltare un cono durante la fase di slalom
- uscire dal percorso predefinito;
- mettere un piede a terra durante la guida;
- impiegare meno di 15 secondi per completare il percorso breve;
- impiegare più di 25 secondi per completare il percorso lungo;
- (solo pista lunga) fermare la motocicletta con la ruota anteriore in un rettangolo predefinito di 50cm di lato
- effettuare una guida irregolare, dimostrando scarsa abilità;

Dal 2006 sono state definite procedure d'esame molto simili nei Paesi dell'Unione Europea (e nel mondo). Tuttavia, vi è una significativa mancanza di mezzi di verifica standard, misurabili e verificabili per gli esami pratici dei motocicli. Il progetto AI-RIDE persegue questo obiettivo: si focalizza sulla valutazione delle prime due prove, che si svolgono in uno scenario noto a priori, dove è possibile automatizzare la verifica raccogliendo dati con telecamere e sensori adatti alla valutazione automatica delle capacità di guida del candidato, fornendo dati affidabili e mezzi di valutazione rigorosi, garantendo trattamenti giusti ed equi.

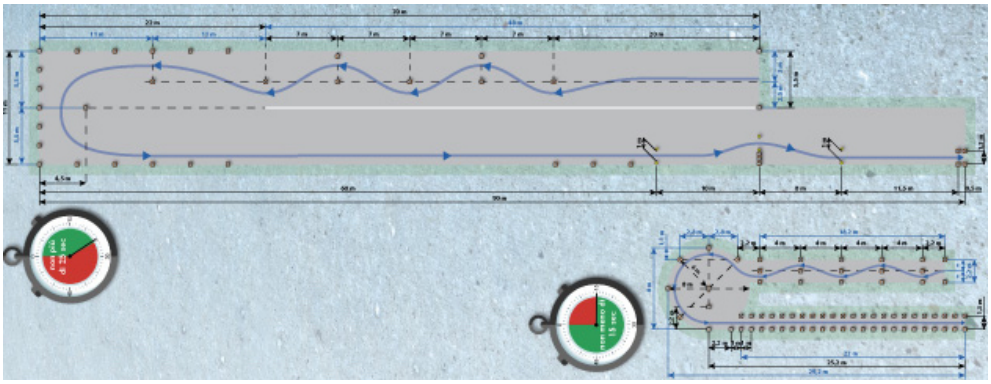

Figure 2: Le pista lunga e la pista corta (Fonte immagine: patente.it).

2.2 La pipeline di elaborazione

L'idea di base si fonda sul presupposto che la motocicletta sia l'unico elemento in movimento della scena. Ogni penalità avviene sempre nelle vicinanze della motocicletta (tocco o salto di un cono, piede del pilota a terra, mancato arresto nell'area designata). Con la sottrazione dello sfondo, possiamo facilmente identificare l'area di interesse da analizzare nel frame corrente. In questo modo non è necessario elaborare l'intera immagine ma solo una piccola parte di essa. L'area dovrebbe essere abbastanza grande da includere lo scenario rilevante attorno alla moto, come i coni vicini o i bordi del circuito. La figura 3 illustra la pipeline di elaborazione principale applicata al flusso video di ciascuna telecamera. I rettangoli blu sono le fasi di elaborazione delle immagini della pipeline, mentre gli ovali rossi indicano le sanzioni che stiamo cercando.

Non appena viene rilevata una singola penalità, l'esame termina con un fallimento. Solo la guida irregolare non è collegata a un evento specifico ma è decisa dall'esaminatore umano. La logica delle penalità è stata affrontata nel seguente modo:

1. **Cono toccato**: per l'individuazione di questa penalità bisogna considerare la posizione iniziale di ogni cono e confrontarla con quelle che si ottengono dai riconoscimenti nei frame successivi. A valle del riconoscimento degli oggetti bisogna sviluppare una logica che minimizzi l'errore dovuto ad eventuali occlusioni, falsi positivi e falsi negativi.

2. **Errore in fase di slalom o uscita di pista**: queste penalità sono

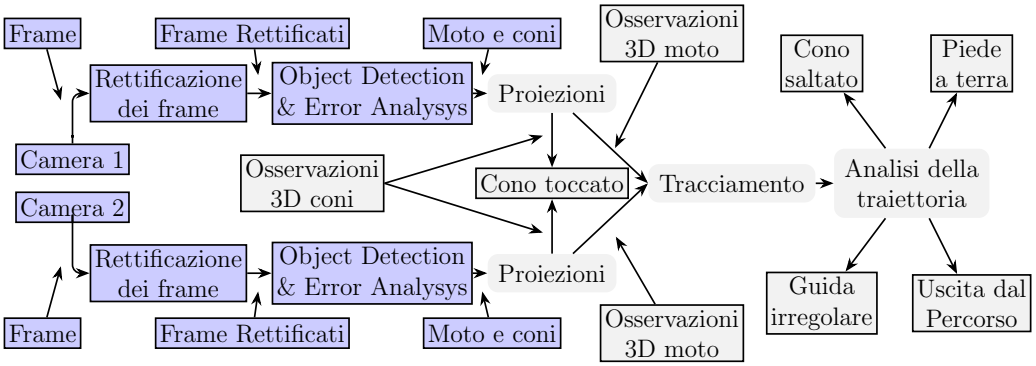

Figure 3: La pipeline di elaborazione del sistema (Fonte immagine: [1]).

segnalate dal modulo di analisi della traiettoria. La pista e l'area circostante sono suddivise in zone virtuali come mostrato in figura 4. I bordi della pista sono anch'essi delimitatori virtuali di zone proibite (l'area esterna ed il corridoio interno indicato in colore pieno in figura 4); se la motocicletta percorre in modo errato la sequenza di slalom oppure si trova in zona proibita la penalità viene segnalata.

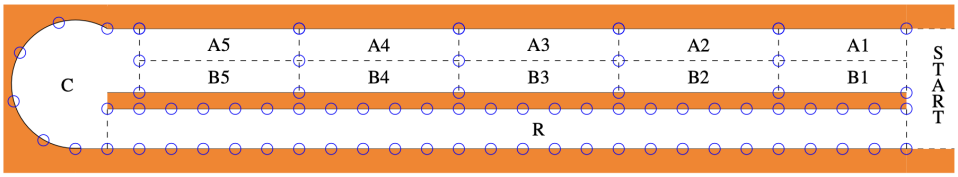

Figure 4: Le zone virtuali (Fonte immagine: [1]).

3. **Tempo di percorrenza**: se il pilota impiega più di 25 secondi per completare la pista lunga o meno di 15 secondi per quella corta la prova ha esito negativo. Questo task è relativamente semplice: una volta che si è calcolato il tempo di percorrenza della prova lo di confronta con tali vincoli temporali. Il segnale di partenza viene dato manualmente dall'esaminatore che preme il pulsante per avviare il cronometro ufficiale.

4. **Arresto al di fuori dell'area indicata**: nella prova su pista lunga è obbligatorio fermare la motocicletta con la ruota anteriore all'interno

di un'area ristretta di 50 cm delimitata da quattro coni. Se la ruota anteriore non ha superato il primo allineamento o se ha superato il secondo allineamento viene considerato come esame fallito. Una telecamera dedicata a basso costo e un solido metodo di sottrazione dello sfondo dovrebbero essere sufficienti per verificare la posizione della ruota nell'area predefinita.

5. **Tocco del piede a terra**: se durante la prova di guida c'è un arresto imprevisto e il pilota appoggia il piede a terra, la velocità della moto scende a zero e questo è un parametro che l'analisi della traiettoria può facilmente misurare. Sfortunatamente, c'è anche il caso in cui il pilota tocca leggermente il suolo senza fermare il veicolo, perché sta perdendo l'equilibrio. Questo è forse il compito più difficile da gestire solo con telecamere esterne. A causa della prospettiva e dell'elevata velocità del gesto, può essere molto difficile anche per l'occhio umano capire se il piede ha effettivamente toccato terra.

6. **Guida irregolare**: a differenza di tutte le altre penalità indicate in precedenza la guida irregolare non ha una definizione analitica che può essere tradotta in un output binario; si può cercare di assegnare un punteggio alla prova dell'esaminando confrontandola con una esecuzione di riferimento, ad esempio effettuata da un istruttore, di cui si registrano i parametri spaziali e temporali. Il parametro spaziale può essere funzione dell'area (integrale) tra le due traiettorie a confronto come illustrato in figura 5. Il paramentro temporale sarà influenzato dal confronto tra velocità e accelerazioni (massime, minime e medie)

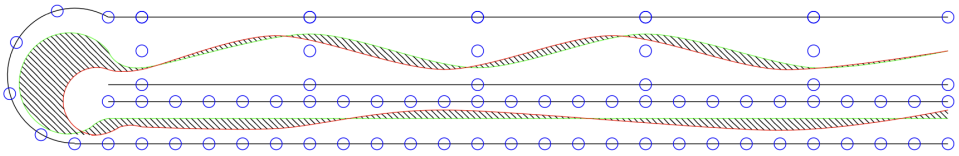

Figure 5: I simboli blu sono i coni, la linea verde è il percorso di riferimento (istruttore), la linea rossa è il percorso del test drive, le linee oblique rappresentano la differenza tra la traiettoria di riferimento e traiettoria dell'esaminando (Fonte immagine: [1]).

L'implementazione dell'intero progetto è di gran lunga superiore al tempo relativo ad un tirocinio di laurea triennale. Limitando lo scenario alla sola pista di dimensioni minori il lavoro è stato suddiviso in due tirocini curriculari:

Il primo (questo lavoro) con il compito di elaborare un singolo flusso video per l'estrazione delle informazioni e il secondo inerente la fusione di tali dati da più flussi.

In particolare il focus del lavoro svolto è stato il riconoscimento dei coni, della moto e del pilota, e della logica legata a tutte le informazioni ricavabili con tecniche di visione artificiale. Le funzionalità implementate sono quelle colorate che in figura 3. L'output di tale flusso sono informazioni numeriche di pochi byte per ogni frame video che indicano la posizione precisa della moto a livello terreno e la segnalazione di eventi legati alla penalità di cono toccato. Tale pipeline è replicabile su dispositivi di edge computing tipo Nvidia Jetson da associare ad ogni telecamera per ottenere un sistema scalabile in modo virtualmente infinito che non soffre problemi di banda in quanto non prevede alcun traffico video sulla rete.

3 Fondamenti teorici e tecnologie utilizzate

3.1 Object Detection

Quando gli esseri umani guardano immagini o video, possono riconoscere e localizzare oggetti di interesse in pochi istanti. Il riconoscimento degli oggetti (Object Detection) consiste nel replicare questa capacità utilizzando un computer. Più formalmente l'Object Detection è una tecnica di visione artificiale che ha lo scopo di individuare istanze di oggetti in immagini o video che vengono passati in input al sistema di Object Detection. Dopodiché generalmente vengono forniti in output, la classe dell'oggetto, il rettangolo di delimitazione e la confidenza. Gli algoritmi di rilevamento degli oggetti che ottengono i risultati più significativi utilizzano la recente tecnica di Intelligenza Artificiale chiamata Deep Learning (DL), che si basa sulle reti neurali convoluzionali [3].

3.1.1 Convolutional Neural Networks

Le reti neurali convoluzionali (CNN: Convolutional Neural Network) rappresentano un approccio altamente efficace per l'elaborazione delle immagini e la risoluzione di problemi legati alla computer vision, come descritto in dettaglio nei lavori di riferimento [3, 4]. Le CNN sono definite come una composizione di tre diversi tipi (esistono anche altre componenti) di strato: il livello convoluzionale, il livello di pooling e il livello completamente connesso; questi elaborano in modo sequenziale l'immagine per estrarre feature rilevanti al fine di affrontare il compito specifico, come ad esempio l'Object Detection. Nell'immagine 6, è possibile osservare una rappresentazione visiva di tali livelli che compongono una CNN. Il livello convoluzionale rappresenta la componente centrale di una CNN. Esso esegue un convoluzione tra due matrici: una matrice denominata kernel (K), composta da un insieme di parametri (pesi), e una matrice immagine (X). Questa operazione viene rappresentata dall'equazione sottostante:

$$y(m,n) = \sum_{i=1}^{K} \sum_{j=1}^{K} [i,j] X[m-1+i, n-1+j] \tag{1}$$

L'operazione di convoluzione svolta dall'omonimo livello permette di individuare le caratteristiche rilevanti nelle immagini, come bordi degli oggetti, texture o altre strutture significative. Attraverso l'apprendimento dei parametri

del kernel. L'idea alla base delle CNN è che l'utilizzo di più layer convoluzionali all'interno del contesto del DL permette di estrarre feature sempre più astratte. Ad esempio, gli strati più bassi possono estrarre i contorni di un oggetto, mentre negli strati più alti la rete può apprendere attivazioni che riconoscono parti di un oggetto e forme complesse, come un occhio o una bocca. Tipicamente il kernel ha una dimensione molto ridotta rispetto all'immagine. Tuttavia, quando l'immagine è composta da tre canali di colore (RGB), il kernel corrispondente sarà composto da tre livelli di profondità, corrispondenti ai tre canali. Durante il processo di avanzamento nella rete, il kernel scorre lungo l'altezza e la larghezza dell'immagine, producendo una rappresentazione della regione di input chiamata "regione ricettiva". Questo processo genera una mappa di attivazione bidimensionale, che rappresenta la risposta del kernel in ogni posizione spaziale dell'immagine. La mappa di attivazione cattura le caratteristiche salienti dell'immagine in base alla risposta del kernel nella regione corrispondente. La figura 7 fornisce un esempio visivo di come l'operazione di convoluzione produce la mappa di attivazione.

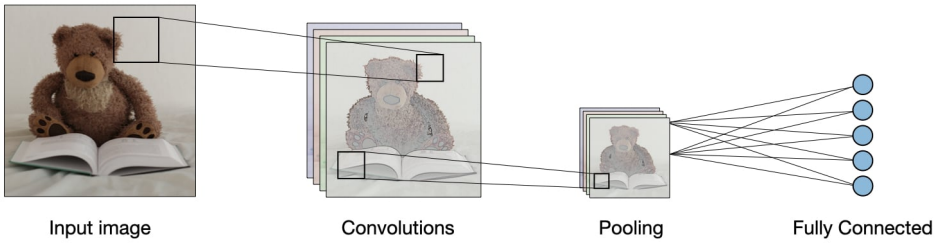

Figure 6: I diversi livelli di una CNN (Fonte immagine: [5]).

Come descritto nel datasheet della Stanford University, [5]. Per determinare la dimensione del volume di output (matrice di attivazione) si utilizza la seguente formula:

$$O = \frac{I - F + P_{start} + P_{end}}{S} + 1 \tag{2}$$

Dove I è la dimensione di input, F la dimensione del kernel (detto anche filtro) e P la dimensione dell'imbottitura. Ovvero le aree di imbottitura (si veda la figura 8).

Gli strati delle reti neurali tradizionali utilizzano la moltiplicazione di matrici per descrivere l'interazione tra le unità di input e le unità di output. Questo implica che ogni unità di output interagisce con tutte le unità di input.

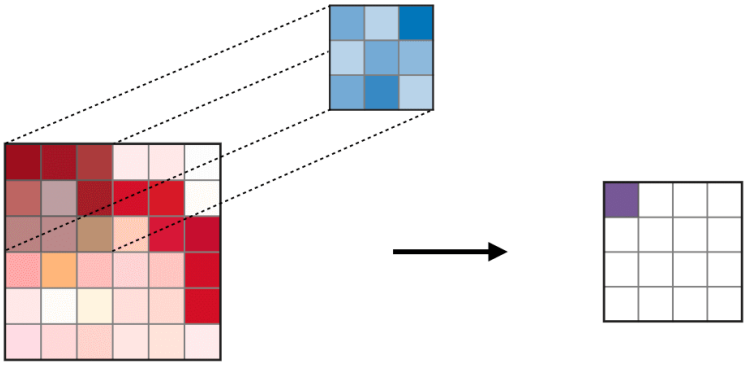

Figure 7: La produzione della mappa di attivazione (Fonte immagine: [5]).

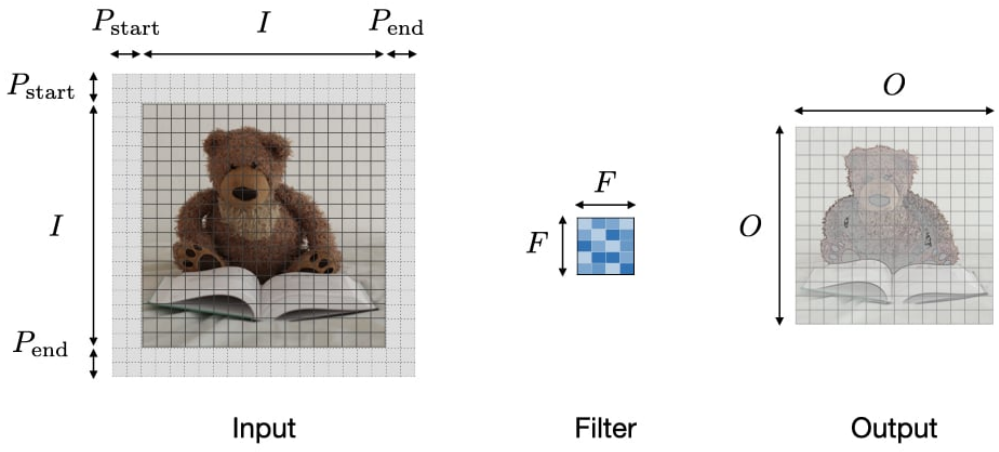

Figure 8: Parametri della produzione di O (Fonte immagine: [5]).

Tuttavia, le reti neurali convoluzionali (CNN) presentano un'interazione più limitata. Questo risultato è ottenuto utilizzando kernel più piccoli rispetto all'input. Ad esempio, un'immagine può avere migliaia o milioni di pixel, ma durante l'elaborazione mediante l'utilizzo di kernel possiamo catturare informazioni significative che riguardano solo decine o centinaia di pixel. Ciò significa che è necessario memorizzare meno parametri, riducendo il requisito di memoria del modello e migliorando l'efficienza statistica complessiva. (si veda la figura 10).

3.1.2 Matrice kernel e matrice di attivazione

Nel paragrafo 3.1.1 si è parlato di cosa avviene nello strato convoluzionale. La matrice kernel è necessaria per produrre la matrice di attivazione. Questa viene confrontata con quella immagine e viene effettuato il prodotto scalare tra le due (si veda l'esempio 10). La matrice kernel o anche detta filter è di dimensioni $F \times F$ dove dato un'input avente C canali la dimensione sarà $F \times F \times C$ (si veda la figura 9).

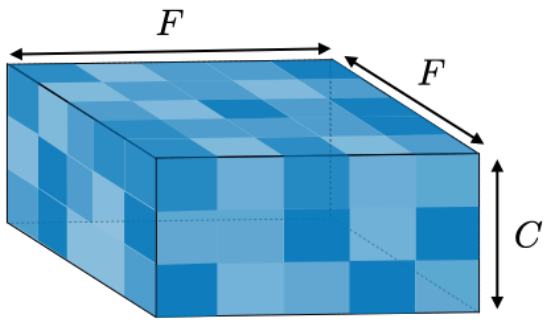

Figure 9: Dimensione di una matrice kernel (Fonte immagine: [5]).

La matrice di attivazione (output) si ottiene mediante il processo di Zero-Padding che va ad aggiungere un numero di P zeri intorno alla matrice di output così da mantenerne le dimensioni (si veda la figura 8).

3.1.3 Strato di pooling

Come viene descritto in [4]. Lo strato di pooling sostituisce l'output della rete in determinate posizioni derivando una statistica riassuntiva degli output vicini. Questo aiuta a ridurre la dimensione spaziale della rappresentazione,

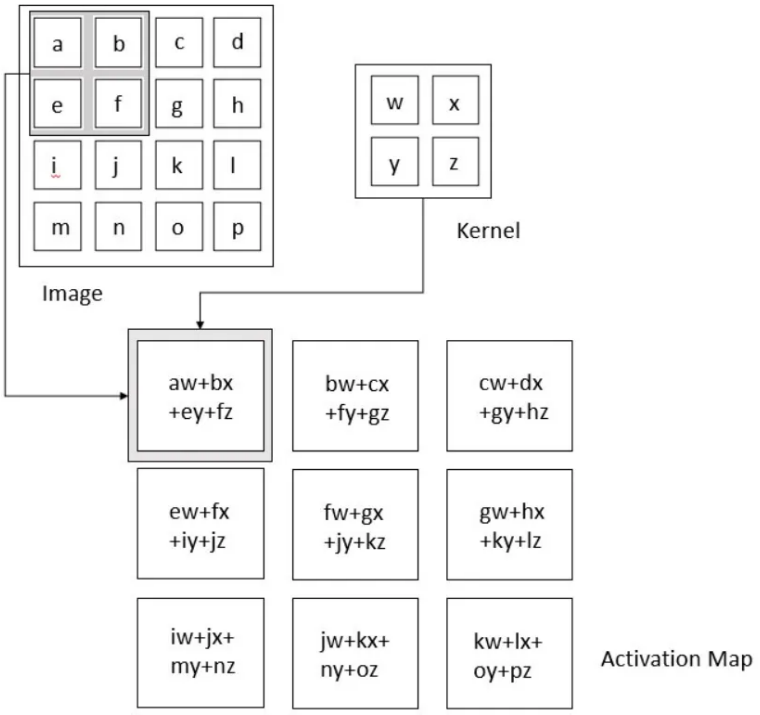

Figure 10: Prodotto scalare tra la prima matrice immagine e il kernel (Fonte immagine: [3]).

che diminuisce la quantità richiesta di calcolo e pesi. L'operazione di pooling viene elaborata individualmente su ogni strato della rappresentazione; questa operazione rende la rappresentazione robusta rispetto a piccole traslazioni o rotazioni.

Esistono diverse funzioni di pooling come la media dei valori vicini. Questo significa che per ogni regione di pooling, viene calcolata la media dei valori presenti nella regione e viene assegnato un valore medio alla corrispondente posizione nella mappa di pooling. Questo tipo di pooling aiuta a ridurre il rumore e l'effetto delle variazioni locali, fornendo una rappresentazione più generale delle caratteristiche rilevanti.

L'operazione più popolare è il max pooling, che riporta l'output massimo dei valori vicini (si veda la figura 11). Il pooling massimo è utile per individuare le caratteristiche più significative all'interno di una regione, contribuendo alla creazione di una rappresentazione più robusta delle caratteristiche distintive dell'immagine.

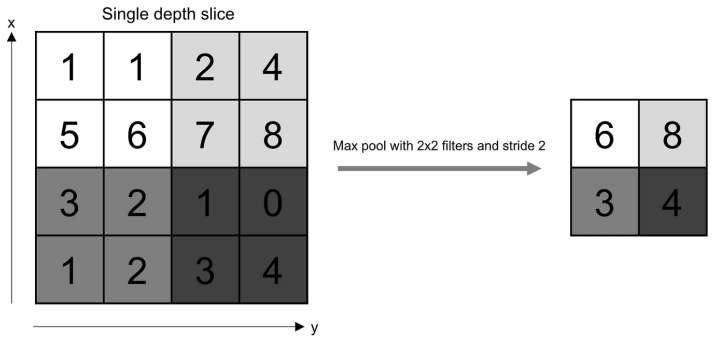

Figure 11: maxPooling (Fonte immagine: [3]).

3.1.4 Strato completamente connesso

A differenza degli strati convoluzionali e di pooling, gli strati completamente connessi non considerano la struttura spaziale delle caratteristiche di input. Invece, trattano le caratteristiche estratte come un vettore piatto o unidimensionale. Questi vengono spesso utilizzati per combinare le caratteristiche estratte dai precedenti strati convoluzionali e di pooling e per eseguire la classificazione o la regressione finale del problema (si veda la figura 12).

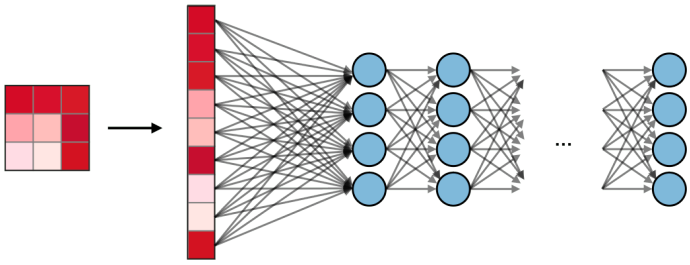

Figure 12: Rappresentazione dello strato completamente connesso (Fonte immagine: [5]).

3.1.5 Reti convoluzionali per Object Detection

Esistono due tipi di reti neurali utilizzate per l'Object Detetction: reti a due stadi e reti a uno stadio.

1. Reti a due stadi: nella prima fase identifica regioni specifiche o sottoinsiemi di immagini che potrebbero contenere un oggetto. La seconda fase classifica gli oggetti all'interno delle proposte regionali. Questo approccio è molto accurato ma è costoso in termini computazionali. Un esempio di rete a due stadi è R-CNN [6]

2. Reti a stadio singolo: le CNN producono previsioni di rete[2] per le regioni dell'intera immagine utilizzando un riquadro di ancoraggio[3] (anchor box) e le previsioni vengono decodificate per generare i riquadri di delimitazione finali per gli oggetti. Generalmente per via della struttura più semplice le reti a singolo stadio sono più veloci di quelle a due stadi, ma la precisione non è eccezionale soprattutto per un oggetto piccolo. Una rete a stadio singolo che ha riscosso molto successo è YOLO [7] (You Only Look Once) che negli ultimi anni si è evoluta in versioni sempre più performanti.

Attualmente YOLOv8[8, 9] è la versione più recente, mentre all'inizio del progetto lo era YOLOv7[10, 8, 11, 12]. Nonostante quest'ultimo fosse un

[2]Ottenute genericamente attraverso una combinazione di strati di convoluzione, pooling e strati completamente connessi.

[3]Il riquadro di ancoraggio è un meccanismo chiave utilizzato nelle reti neurali per l'object detection, utilizzato per guidare la generazione delle previsioni sugli oggetti all'interno delle immagini.

modello performante in termini di precisione e di inferenza, presentatava poco supporto da parte della community. Si è deciso di optare per YOLOv5[13, 8, 14] di cui se ne parla in modo approfondito nei paragrafi 3.1.6 e 3.1.7. Questo è un ottimo framework di Object detection che riesce ad ottenere ottime prestazioni anche su dispositivi poco performanti come quelli di edge-computing ed è costantemente supportato dalla community su GitHub[15]. Inoltre, supporta nativamente l'instance segmentation.

3.1.6 Architettura di YOLOv5

Nella fase iniziale l'immagine in input è analizzata dall'algoritmo e vengono generate delle "caratteristiche" che rappresentano gli oggetti presenti. Successivamente, queste caratteristiche vengono utilizzate per predire le posizioni degli oggetti e le relative classi. Il risultato finale è una serie di riquadri che delimitano gli oggetti individuati nell'immagine dette bounding box, insieme alle relative etichette di classificazione.

Una rete YOLOv5 è costituita principalmente da tre parti (come descritto in [8, 14, 16, 17]):

1. **Backbone** è la parte costituita da una rete neurale convoluzionale che aggrega e forma caratteristiche dell'immagine a diverse granularità. In YOLOv5 viene adottata la CSP-Darknet53[14] una versione modificata della rete Darkenet53[18].

2. **Neck** è responsabile di combinare le caratteristiche da diverse scale o livelli al fine di migliorare le prestazioni del rilevamento.

3. **Head** è responsabile della generazione delle predizioni finali per l'object detection, ovvero dei bounding box che delimitano gli oggetti individuati e delle relative classificazioni. Questa è simile a quelle addotate su YOLOv2[19] e YOLOv3[20].

YOLOv5 restituisce tre output: le classi degli oggetti rilevati, le bounding box e le confidenze.

3.1.7 I differenti tipi di modelli YOLOv5

YOLOv5 propone diversi modelli di partenza, ciascuno con un numero diverso di parametri e tempi di calcolo, come si può osservare nella figura 13.

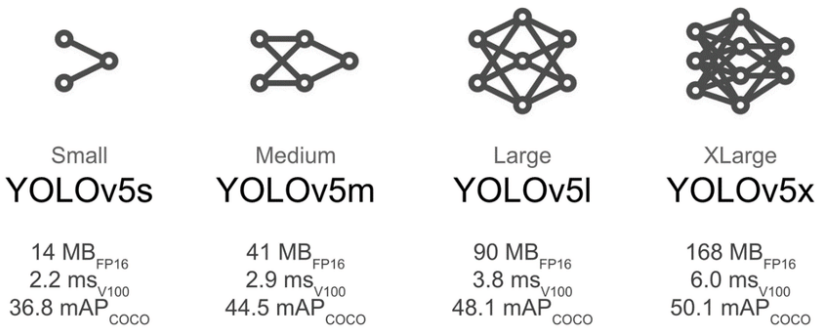

Figure 13: Confronto tra i diversi modelli YOLOv5 (Fonte immagine: [15]).

La scelta del modello giusto per un determinato progetto è di fondamentale importanza e richiede un'attenta valutazione delle caratteristiche del set di dati e dei requisiti specifici del problema. È cruciale trovare un modello che sia altamente compatibile con il set di dati disponibile, in modo da ottenere prestazioni ottimali e risultati accurati. Un aspetto importante da considerare è il numero di parametri del modello, che deve essere adeguato al problema specifico. Un modello con un numero eccessivo di parametri potrebbe portare ad avere tempi di inferenza[4] significativi, mentre un modello con pochi parametri potrebbe non essere in grado di catturare in modo accurato le complessità dei dati.

Nella tabella in figura 14 vengono confrontati i modelli in relazione alle due risoluzioni più utilizzate. Questo rappresenta mediamente le prestazioni dei diversi modelli, ciò torna utile per avere un'idea generale sull'efficienza.

3.2 Annotazione immagini e dataset

Durante il progetto, sono state utilizzate due tipologie di annotazione con Roboflow[21]. La prima annotazione è stata creata utilizzando il metodo della bounding box per marcare i coni principalmente. La seconda tipologia, chiamata instance segmentation, consente di selezionare i punti che definiscono la forma dell'oggetto da riconoscere (si veda i paragrafi 4.3.1 e 4.3.2). Parlando di reti neurali, è fondamentale discutere dataset che utilizziamo ai fini dell'apprendimento, come spiegato nei libri [22, 23]. La qualità dei

[4]Rappresenta il periodo di tempo necessario affinché il modello elabori i dati di input e produca le relative previsioni o output desiderati.

Model	size (pixels)	mAP^{val} 0.5:0.95	mAP^{val} 0.5	Speed CPU b1 (ms)	Speed V100 b1 (ms)	Speed V100 b32 (ms)	params (M)	FLOPs @640 (B)
YOLOv5n	640	28.0	45.7	**45**	**6.3**	**0.6**	**1.9**	**4.5**
YOLOv5s	640	37.4	56.8	98	6.4	0.9	7.2	16.5
YOLOv5m	640	45.4	64.1	224	8.2	1.7	21.2	49.0
YOLOv5l	640	49.0	67.3	430	10.1	2.7	46.5	109.1
YOLOv5x	640	50.7	68.9	766	12.1	4.8	86.7	205.7
YOLOv5n6	1280	36.0	54.4	153	8.1	2.1	3.2	4.6
YOLOv5s6	1280	44.8	63.7	385	8.2	3.6	12.6	16.8
YOLOv5m6	1280	51.3	69.3	887	11.1	6.8	35.7	50.0
YOLOv5l6	1280	53.7	71.3	1784	15.8	10.5	76.8	111.4
YOLOv5x6 + TTA	1280 1536	55.0 **55.8**	72.7 **72.7**	3136 -	26.2 -	19.4 -	140.7 -	209.8 -

Figure 14: Questa tabella mette a paragone i diversi modelli di partenza di yolov5 (Fonte immagine: [15]).

dati è influenzata dalla loro fonte. È importante assicurarsi che le immagini siano rappresentative del dominio in cui la rete neurale dovrà operare. In un modello che deve riconoscere moltissime classi differenti come persone, animali e oggetti, la numerosità del dataset e della singola classe deve essere appropriata, spesso nell'ordine delle decine di migliaia. In task di riconoscimento con poche classi e con oggetti che presentano una struttura regolare, con forma e colori predefiniti, possono essere sufficienti dataset di numerosità inferiore. Il dataset solitamente è composto da un insieme di addestramento (training set) e altri due insiemi di supporto (validation set e test set). Il set di addestramento contiene una quantità maggiore di dati rispetto agli altri set. Nel caso di progetti con un numero relativamente basso di immagini, nell'ordine delle migliaia, viene applicata la regola del 70%/30% (70% per il set di addestramento e il 30% per i set di validation e di test). Nei progetti molto consistenti, dove si utilizzano circa un milione o più immagini, non è necessario avere una percentuale di test eccessivamente grande: essa può essere compresa in un fascia dal 4% allo 0,5% perché il numero assoluto delle immagini è comunque molto elevato.

3.2.1 Training set

Il training set è il dataset principale utilizzato per addestrare il modello predittivo. È composto da esempi di dati etichettati, in cui le etichette rappresentano la variabile da predire (ad esempio, una classe o un valore numerico). Durante la fase di addestramento, la rete apprende i pattern e le relazioni presenti ottimizzando i parametri, al fine di creare un modello che possa generalizzare correttamente i dati non visti in precedenza.

3.2.2 Validation set

Il validation set è utilizzato per regolare gli iperparametri (Per esempio: learning rate, batch size) del modello e valutarne le prestazioni durante il processo di addestramento. È un set di dati indipendente dal training set, ma con la stessa distribuzione dei dati. Durante l'addestramento del modello, gli iperparametri possono essere regolati in base alle prestazioni osservate sul validation set, ad esempio modificando l'architettura del modello. Ciò aiuta a evitare l'overfitting, in cui il modello si adatta troppo ai dati di addestramento, ma non generalizza bene su nuovi dati. Il validation set non deve essere utilizzato per addestrare direttamente il modello, ma solo per valutare le sue prestazioni e trovare gli iperparametri ottimali; questo processo è detto "model selection".

3.2.3 Test set

Il test set è utilizzato per valutare le prestazioni finali del modello, dopo dopo essere stati effettuati la model selection e l'addestramento del modelli. Questo set di dati viene utilizzato per stimare "l'errore del modello" su nuovi dati che non sono stati utilizzati durante l'addestramento. Il test set deve essere indipendente sia dal training set che dal validation set, e deve essere rappresentativo della distribuzione dei dati reali che il modello incontrerà in fase di utilizzo.

3.3 Addestramento del modello

Il tempo di addestramento di un modello dipende dal numero di parametri di quest'ultimo, dalla risoluzione delle immagini e dalla numerosità del training dataset oltre che ovviamente dalla potenza di elaborazione dell'hardware

utilizzato. Quando si addestra una rete si impostano il numero del batch-size e delle epochs. Il parametro batch-size specifica il numero di campioni (immagini) che vengono processati in una singola iterazione. In generale, un valore batch-size più grande consente di sfruttare meglio la parallelizzazione e accelerare il processo di addestramento o inferenza, ma richiede più memoria. Il numero di epoch invece stabilisce quanti cicli di apprendimento debbano essere effettuati. Un numero maggiore non vuol dire un modello migliore, questo può portare il modello in overfitting, problematica che viene risolta come spiegato nel paragrafo 3.3.1.

3.3.1 Regolarizzazione

Come descritto in [24, 25] la regolarizzazione è una tecnica utilizzata per mitigare il problema dell'overfitting in un set di dati, introducendo termini aggiuntivi nell'obiettivo ottimizzato durante il training. Questi termini penalizzano soluzioni complesse, mitigando l'overfitting. Esistono due metodi comuni di regolarizzazione: la regolarizzazione L1 e la regolarizzazione L2.

La regolarizzazione L1 aggiunge una penalità alla funzione di costo, che è rappresentata come:

$$\sum_{i=1}^{M}(y_i - \sum_{j=0}^{n} Bj * X_{ij})^2 + \lambda \sum_{j=0}^{n} |B_j| \tag{3}$$

Dove $\sum_{i=1}^{M}(y_i - \sum_{j=0}^{n} Bj * X_{ij})^2$ è la componente della funzione di costo che misura l'adeguatezza del modello ai dati. Mentre $\lambda \sum_{j=0}^{n} |B_j|$ è la penalità introdotta dalla regolarizzazione L1, dove λ controlla l'intensità della penalità e B_j rappresenta i pesi dei parametri del modello.

La regolarizzazione L1 favorisce soluzioni sparse, in cui molti pesi dei parametri diventano zero. Ciò implica che solo un sottoinsieme dei parametri influisce sul modello finale, favorendo la selezione delle caratteristiche più rilevanti. Questa peculiarità rende il modello più interpretabile e riduce la sua complessità.

D'altra parte, la regolarizzazione L2 è definita come:

$$\sum_{i=1}^{M}(y_i - \sum_{j=0}^{n} Bj \cdot X_{ij})^2 + \lambda \sum_{j=0}^{n} B_j^2 \tag{4}$$

Questa introduce una penalità proporzionale alla somma dei quadrati dei

pesi dei parametri. A differenza della regolarizzazione L1, la regolarizzazione L2 non tende a rendere i pesi esattamente zero, ma li riduce in modo uniforme senza azzerarli. Questa penalizzazione aiuta a ridurre l'overfitting e migliora la capacità di generalizzazione del modello.

Quindi, la regolarizzazione L1 e L2 differiscono nell'effetto che hanno sui pesi dei parametri e nella capacità di produrre soluzioni sparse. Yolov5, in particolare, utilizza la regolarizzazione L2 come tecnica di regolarizzazione nel suo framework di Object Detection.

3.3.2 Confusion Matrix

Come viene descritto in [26, 27], La confusion matrix (matrice di confusione) è una rappresentazione tabellare utilizzata per valutare le prestazioni di un modello di classificazione nel campo dell'apprendimento automatico. La matrice di confusione mostra il numero di predizioni corrette e errate effettuate dal modello per ciascuna classe di destinazione. È organizzata in modo da avere le classi reali rappresentate sulle righe e le classi predette sulle colonne.

La matrice di confusione è composta da:

- **True positive (TP)** rappresenta i casi in cui il modello ha correttamente predetto una classe positiva.
- **True Negative (TN)** rappresenta i casi in cui il modello ha correttamente predetto una classe negativa.
- **False Positive (FP)** indica i casi in cui il modello ha erroneamente predetto una classe positiva quando in realtà era negativa.
- **False Negative (FN)** indica i casi in cui il modello ha erroneamente predetto una classe negativa quando in realtà era positiva.

La matrice di confusione fornisce una visione dettagliata delle performance del modello per ciascuna classe. A partire dalla matrice, è possibile calcolare diverse metriche di valutazione, come l'accuracy, la precision, la recall e F1 score che consentono di valutare le prestazioni complessive del modello in termini di classificazione corretta e gestione degli errori. Queste vengono calcolate nei seguenti modi:

1. **Accuracy** $= \frac{TP+TN}{TP+FP+TN+FN}$ questa fornisce la proporzione del numero totale di previsioni corrette;

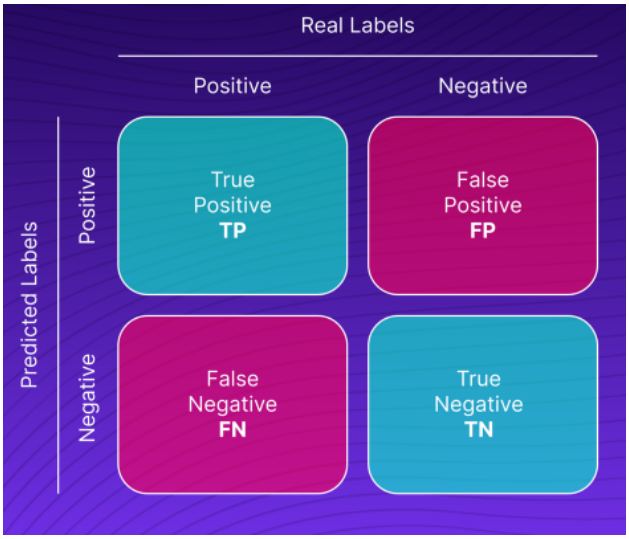

Figure 15: Matrice di confusione (Fonte immagine: [26]).

2. **Precision** = $\frac{TP}{TP+FP}$ è la frazione di valori positivi rispetto alle istanze positive previste totali;

3. **Recall** = $\frac{TP}{TP+FN}$ è la frazione di valori positivi rispetto alle istanze positive effettive totali;

4. **F1 score** = $2 \times \frac{Precision \times Recall}{Precision+Recall}$ è calcolato come la media armonica della precision e del recall;

3.3.3 La Mean Avarage Precision

La Mean Avarage Precision (mAP) fornisce una valutazione complessiva delle capacità del modello di Object Detection, combinando la precision e il recall su più classi o categorie di oggetti. Questo la rende una metrica più informativa rispetto ad altre misure, come l'accuracy, che potrebbe non tener conto della complessità del problema di rilevamento. La mAP viene calcolata considerando la precisione media per diverse soglie di confidenza. Per ogni classe di oggetti, si calcola l'Average Precision (AP), che rappresenta la media della precisione a diverse soglie di confidenza. La mAP viene quindi ottenuta calcolando la media delle AP per tutte le classi. In generale, un valore di mAP più alto indica prestazioni migliori del modello di Object Detection. Questo va da un valore compreso tra 0 e 1. Idealmente, una mAP di 1 indicherebbe

che il modello riesce a rilevare e localizzare correttamente tutti gli oggetti delle diverse classi presenti nel dataset di test. Tuttavia, nella pratica, ottenere una mAP con precisione del 100% è molto raro, specialmente in problemi complessi e dataset reali. In generale, una mAP superiore al 50% può essere considerata una buona performance, indicando che il modello riesce a rilevare correttamente la maggior parte degli oggetti delle diverse classi. Un valore di mAP intorno al 70% o superiore è spesso considerato molto buono, indicando una capacità di rilevamento degli oggetti molto accurata.

3.4 Hardware ad elevato parallelismo: la DGXA100

Per l'allenamento di modelli non complessi è possibile utilizzare normali computer progettati per l'uso domestico. Nel momento in cui la quantità di parametri del modello cresce questa soluzione non è più sufficiente. In tal caso esistono istanze hardware progettate appositamente per questo tipo di elaborazioni. Leader del settore è la Nvidia, che produce diversi tipi di schede per l'elaborazione di complessi modelli di reti neurali (le migliori attualmente sono la A100 e la H100). Inoltre tra i suoi prodotti di fascia alta si trovano infrastrutture atte a supportare cluster di queste schede. Un esempio è il sistema DGXA100 che permette di utilizzarne fino ad otto sulla stessa macchina (si veda la figura 16). La comunicazione tra le schede è resa efficiente tramite la tecnologia NVLink che permette di ottimizzare i carichi di lavoro in parallelo. Tale sistema è disponibile presso l'ISTI-CNR ed è stato utilizzato in questo tirocinio. In dettaglio la DGXA100 è composta da 8 schede A100 da 40GB (Esistono anche da 80GB) l'una fino ad arrivare ad un totale di 320GB di memoria video. Una nota importante va spesa sul tipo di memoria di cui queste tipi di schede dispongono, che a differenza della tipica GDDR6X da 384 bit comunemente utilizzata per le schede di fascia alta nel gaming, questa è una HBM2e da 5120 bit. Quindi una scheda chiaramente progetta per soluzioni molto più particolari. Questo tipo di hardware è tornato utile non solo per il carico imposto dal numero di parametri mediante modelli più grandi, ma anche per risolvere il problema della risoluzione delle immagini (si veda il paragrafo 4.2.8).

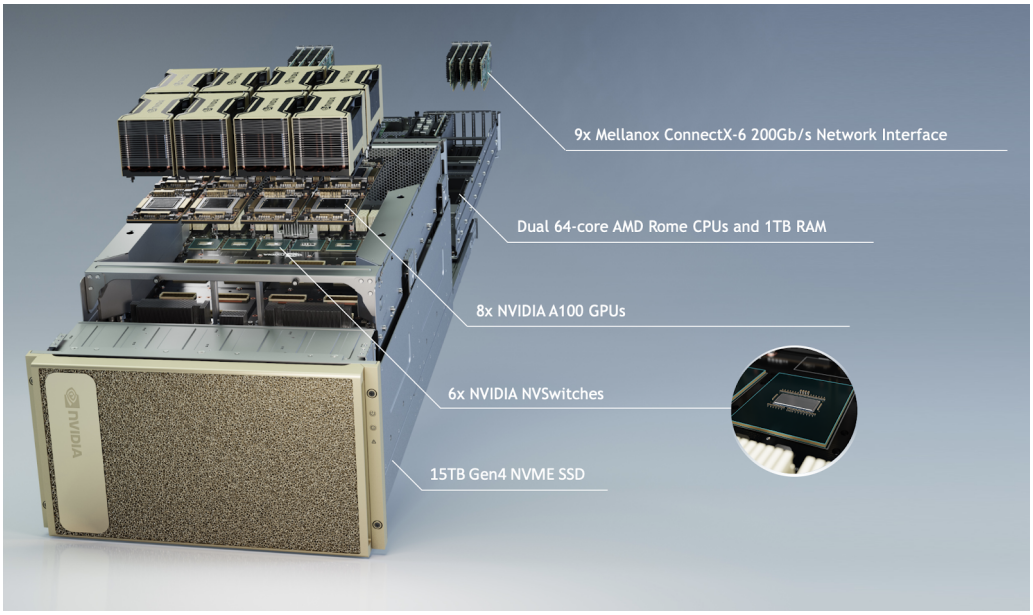

Figure 16: I principali componenti all'interno di sistema DGXA100

3.5 OpenCV

Come descritto in [28], OpenCV (Open Source Computer Vision Library) è una libreria di software di visione artificiale e machine learning open source. OpenCV è stato creato per fornire un'infrastruttura comune per le applicazioni di visione artificiale e per accelerare l'uso della percezione della macchina nei prodotti commerciali. Questa libreria contiene numerosi algoritmi ottimizzati per l'applicazione di tecniche di visione artificiale. La libreria è scritta in C++, ma questa si può interfacciare anche con Python, Java e MATLAB. OpenCV fornisce anche molte altre funzionalità utili per la manipolazione delle immagini, come la correzione del colore, il ritaglio e la rotazione, che possono essere utili per preparare i dati di input per YOLOv5 o per post-elaborare i risultati del rilevamento degli oggetti. Nel caso del tirocinio è stata adottata per stampare i punti che rappresentano la posizione reale della moto e dei coni. Inoltre è stata importante per integrare la correzione della distorsione dovuta dalla lente della telecamera (si veda il paragrago) e per la creazione della look up table (si veda il paragrafo 4.3.2.1).

3.6 Calibrazione della fotocamera

Come viene descritto [29, 30, 31, 32]. Alcune fotocamere stenopeiche introducono una distorsione significativa delle immagini. Due tipi principali di distorsione sono la distorsione radiale e la distorsione tangenziale.

- La **distorsione radiale** fa apparire curve le linee rette. Questa aumenta quanto più i punti sono lontani dal centro dell'immagine. Viene rappresentata nel seguente modo:

$$x_{distorted} = x \cdot (1 + k_1 \cdot r^2 + k_2 \cdot r^4 + k_3 \cdot r^6) \tag{5}$$

$$y_{distorted} = y \cdot (1 + k_1 \cdot r^2 + k_2 \cdot r^4 + k_3 \cdot r^6) \tag{6}$$

- La **distorsione tangenziale** si verifica perché la lente che scatta l'immagine non è allineata perfettamente in parallelo al piano dell'immagine. Pertanto, alcune aree dell'immagine potrebbero sembrare più vicine del previsto. È rappresentata come segue:

$$x_{distorted} = x + [2 \cdot p_1 \cdot x \cdot y + p_2 \cdot (r^2 + 2 \cdot x^2)] \tag{7}$$

$$y_{distorted} = y + [p_1 \cdot (r^2 + 2 \cdot x^2) + 2 \cdot p_2 \cdot x \cdot y] \tag{8}$$

Lo scopo è trovare i cinque parametri detti coefficienti di distorsione.

$$\text{Coefficienti di distorsione} = (k_1 k_2 p_1 p_2 k_3) \tag{9}$$

Sarà quindi necessario costruire la matrice della fotocamera. Questa è unica per ogni fotocamera ed una volta creata può essere riutilizzata per tutte le immagini effettuate dalla medesima telecamera. Viene rappresentata come una matrice 3x3:

$$\text{Matrice fotocamera} = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} \tag{10}$$

Per ottenere queste informazioni si adotta come punto di riferimento la scacchiera; di seguito viene mostrata un'immagine in cui due bordi di una

scacchiera sono contrassegnati da linee rosse. Si noti che il bordo della scacchiera non è una linea retta e non corrisponde alla linea rossa (si veda la figura Figure 17).

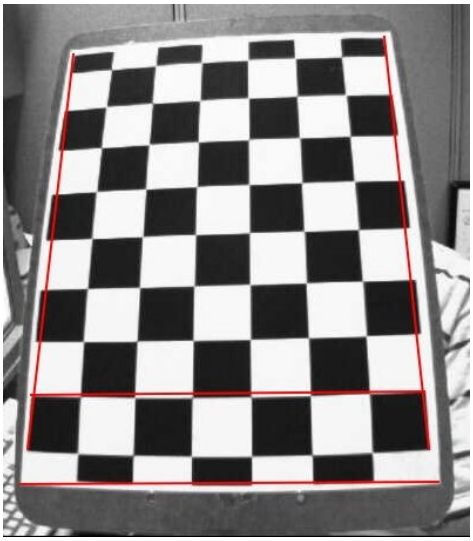

Figure 17: Si può notare la distorsione delle linee della scacchiera (Fonte immagine: [29]).

Per trovare questi parametri, dobbiamo fornire alcune immagini campione di un modello ben definito, in questo caso la scacchiera. Troviamo alcuni punti specifici di cui conosciamo già le relative posizioni (es. angoli quadrati nella scacchiera). Sono già note le coordinate di questi punti nello spazio del mondo reale e conosciamo le coordinate nell'immagine, quindi è possibile risolvere i coefficienti di distorsione. Per avere un risultato buono è richiesto un numero minimo di 15 immagini, la scacchiera deve essere posizionata in zone diverse e con angolazioni diverse. I dati di input importanti necessari per la calibrazione della fotocamera sono l'insieme di punti del mondo reale 3D e le corrispondenti coordinate 2D di questi punti nell'immagine (si veda la figura 18).

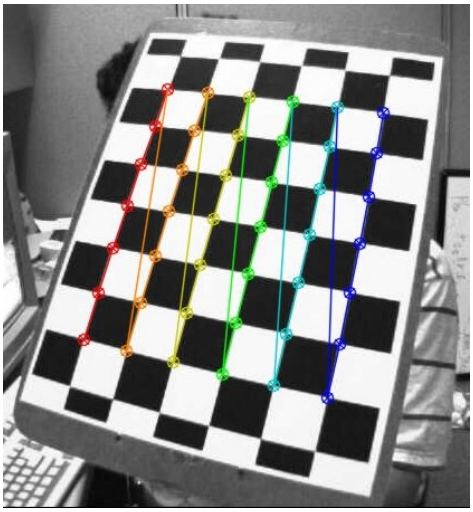

Figure 18: Nella figura vengono calcolati i punti immagine e i punti oggetto (Fonte immagine: [29]).

4 Lavoro svolto e obiettivi conseguiti

Il lavoro svolto in questo tirocinio è ben illustrato dalla figura 19 che mostra come il video della telecamera è elaborato mediante le routine del flusso di elaborazione dati per produrre in output la posizione della moto e quelle dei coni. In questo capitolo si illustrano in modo dettagliate le funzionalità realizzate. Nella sezione 4.1 si descrive il sistema di acquisizione dati, fornendo una descrizione dell'hardware utilizzato per la registrazione dei video e la rettificazione delle immagini per risolvere il problema della distorsione. Il secondo paragrafo illustra l'addestramento della rete, partendo dalla annotazione delle immagini e descrivendo le prestazioni dei modelli ottenuti in relazione ai parametri e alla risoluzione scelta. L'ultimo paragrafo illustra la logica di classificazione delle detection; in particolare come le detection della moto e dei coni vengano elaborate per avere una stima accurata e delle posizioni e per rilevare la penalità di cono toccato.

4.1 Il sistema di acquisizione dati

Il tirocinio si concentra sulla analisi delle prove di guida svolte nella pista di dimensione ridotte, il circuito Low Speed Balance (LSB). Su questa pista

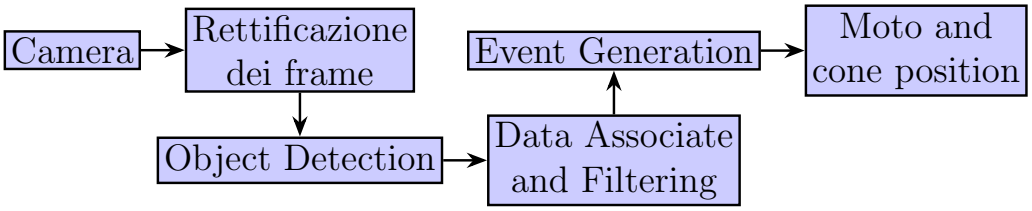

Figure 19: La pipeline di elaborazione del sottosistema sviluppato in questo tirocinio

· Sensore immagine CMOS progressivo 1/2.7" da 5 MP
· ROI, SMART H.264+/H.265+
· 20 fps@2592 × 1944, 50/60 fps@1080P(1920 × 1080)
· WDR (120 dB), Day/Night (ICR), 3D DNR, AWB, AGC, BLC
· Gestione da Web viewer, CMS (DSS/PSS) e DMSS
· Ottica motorizzata 2.7~13.5 mm
· 2/1 in/out allarmi, 1/1 in/out audio
· Portata LED IR 50 m
· Micro SD, IP67, IK10

Figure 20: La telecamera IP varifocale utilizzata per acquisizione video

sono state installate due telecamere IP[33] varifocali come quella illustrata in figura 20. Entrambe le telecamere permettono di registrare video a 50 fps con una risoluzione FullHD (1920x1080 pixel); la scelta di tale hardware è motivata da considerazioni di natura economica, ma soprattutto di natura prestazionale perché con una risoluzione maggiore si avrebbero tempi di elaborazione troppo elevati. Le telecamere sono posizionate ad un'altezza di 6 metri e sono state denominate L1 (frontale) e L2 (laterale) come si può osservare in figura 21. E' stato scelto di assegnare alla telecamera L2 la funzione di telecamera principale, perché dalla sua posizione è in grado di monitorare tutto il percorso in modo più dettagliato. La telecamera L1 apporta informazione solo quando la L2 non è in grado di fornire dati affidabili.

4.1.1 Rettificazione della immagini

Dopo aver impostato l'ambiente di lavoro e le componenti necessarie come descritto in appendice 6.1, è possibile registrare i flussi video delle telecamere per mezzo del software FFmpeg (per dettagli si veda appendice 6.3). Una volta ottenuta la registrazione è fondamentale correggere la distorsione della

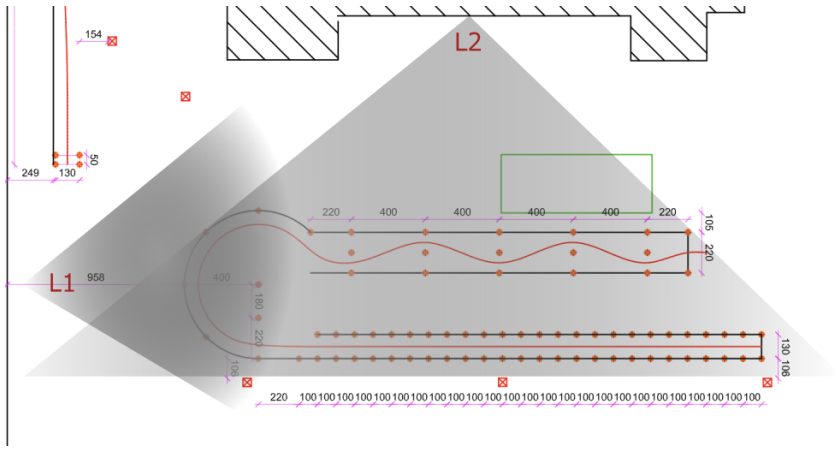

Figure 21: La pista di dimensioni ridotte è coperta da due telecamere, denominate L1 (frontale) e L2 (laterale), posizionate ad un'altezza di 6 metri.

telecamera (si veda il paragrafo 3.6), passaggio fondamentale per ottenere la posizione reale della moto e dei coni.

I parametri della telecamera sono stati calcolati mediante il metodo descritto nel paragrafo 3.6, questo viene facilmente implementato dalla libreria openCV (si veda il paragrafo 3.5). Sono state necessarie circa 15 immagini (si veda la figura 22) per ricavare i coefficienti di distorsione

I parametri necessari per effettuare la correzione vengono letti direttamente dal file di configurazione.

Per specificare quali usare in base alla telecamera è stato creato un parametro –camera. L'argomento del parametro può essere L1 o L2; entrambe le telecamere hanno parametri diversi. Ciò consente di configurare e adattare facilmente i parametri della telecamera senza apportare modifiche dirette al codice.

Listato 1: I coefficienti di distorsione salvati per ogni telecamera

```
1 [L1]
2 MTX = 2074.624122363291, 0, 993.0770846130686; 0, 2069.13907542046,
      ↪ 521.8304544414857; 0, 0, 1
3 DIST = -0.3616174796501423, 0.1890662167787267, 0.001367185736578999,
      ↪ -0.0008546669574081079, -0.1033342270625919
4
5 [L2]
6 MTX = 1353.050908825539, 0, 955.0104361890679; 0, 1336.64014501087,
      ↪ 549.0934086209091; 0, 0, 1
```

Figure 22: L'immagine insieme ad altre è stata usata per il processo di calibrazione.

```
7 DIST = -0.5106922712461706, 0.2988998555534707, 0.001994730491486648,
      ↪ 0.001113984144512712, -0.08882504474125062
```

Oltre a questo è stato aggiunto il parametro chiamato –rectify all'interno del file detect.py. Quando questo parametro è abilitato, viene eseguita la correzione della distorsione sul video fornito al processo di Object Detection.

Questo metodo prende in input un frame, i parametri della calibrazione della telecamera e la risoluzione dell'immagine. Esso applica la correzione utilizzando i parametri di calibrazione e restituisce l'immagine corretta senza distorsione. Questo approccio permette di garantire che l'immagine utilizzata per la successiva fase di rilevamento degli oggetti sia priva di distorsione, migliorando così l'accuratezza e l'affidabilità dei risultati ottenuti.

Listato 2: Metodo rectify

```
1 function rectify(frame, mtx, dist, w=1920, h=1080):
2     """
3     Corregge la distorsione dell'immagine utilizzando i parametri di calibrazione.
4 
5     Args:
6         frame: L'immagine di input.
7         mtx: La matrice di calibrazione della fotocamera.
8         dist: I coefficienti di distorsione ottenuti dalla calibrazione.
```

```
        w: Larghezza desiderata dell'immagine di output (predefinito: 1920).
        h: Altezza desiderata dell'immagine di output (predefinito: 1080).

    Returns:
        L'immagine corretta senza distorsione.
    """

    # Calcolo della matrice di proiezione della fotocamera ottimale
    newcameramtx, roi = cv2.getOptimalNewCameraMatrix(mtx, dist, (w, h), 0, (w, h)
        ↪ )

    # Metodo 1 per correggere la distorsione dell'immagine
    #dst = cv2.undistort(frame, mtx, dist, None, newcameramtx)

    # Metodo 2 per correggere la distorsione dell'immagine
    mapx, mapy = cv2.initUndistortRectifyMap(mtx, dist, None, newcameramtx, (w, h)
        ↪ , 5)

    # Applicazione delle mappe di correzione della distorsione all'immagine
    corrected_frame = cv2.remap(frame, mapx, mapy, cv2.INTER_LINEAR)

    # Salvataggio dell'immagine corretta su disco
    #cv2.imwrite("rect.jpg", corrected_frame)

    return corrected_frame
```

Figure 23: Con effetto di distorsione.

Figure 24: Senza effetto di distorsione.

4.2 Addestramento della rete

In questa sezione vengono descritte le procedure per l'allenamento di un rete mediante il framework YOLOv5. Inoltre vengono descritti alcuni esperimenti effettuati tra modelli e risoluzioni alternative. È presente una descrizione dei modelli principali e del loro allenamento.

4.2.1 Annotazione immagini

All'interno della piattaforma Roboflow, è disponibile una sezione dedicata alle annotazioni, attraverso la quale gli utenti possono effettuare il caricamento delle nuove immagini da sottoporre al processo di annotazione. Questa operazione risulta piuttosto semplice da eseguire. Per estrarre i frame dai video, è possibile avvalersi del software ffmpeg (consultare appendice 6.3), oppure si può optare per il caricamento diretto dei video sulla piattaforma, permettendo di selezionare quanti frame estrarre in media da ciascun secondo. L'annotazione delle immagini rappresenta una procedura relativamente agevole, ma richiede un elevato grado di concentrazione.

All'interno del progetto sono stati pensati diversi modi per poter annotare le immagini. Le possibili scelte sono state la creazione delle bouding box e quello dell'instance segmentation. A parte il sistema di rilevamento della moto che viene trattato nel paragrafo 4.3.1. La instance segmentation in ambito di annotazione dei coni ha dato ottimi risultati, a discapito del tempo di inferenza e della complessità delle annotazioni. Questa ha di base un tempo di inferenza maggiore rispetto alla bouding box ed inoltre bisogna considerare la difficoltà maggiore nell'annotare un grande numero di coni. Va infatti

ricordato che essa tiene conto della sagoma dell'oggetto in questione (si veda la figura 25), e ciò rende molto più complesso il processo di annotazione rispetto allo standard bouding box (si veda la figura 26). In termini di complessità la bouding box ha dato ottimi risultati tanto da non giustificare l'ausilio di un metodo alternativo e più dispendioso.

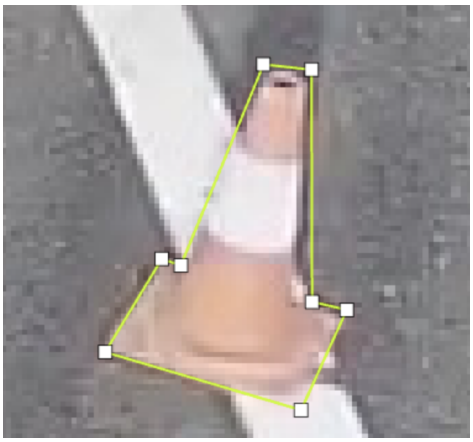

Figure 25: Instance Segmentation del cono

Figure 26: Bouding box del cono

È importante sottolineare che un'annotazione errata delle immagini può generare risultati di scarsa qualità sia durante la fase di addestramento del

modello che nel suo successivo utilizzo. Una volta completata l'annotazione di un numero sufficiente di immagini, si dovrà procedere alla suddivisione delle stesse tra il set di addestramento (training set), il set di test (test set) e il set di validazione (validation set) (si veda il paragrafo 3.2). Successivamente, la piattaforma offrirà la possibilità di generare il dataset completo, includendo tutte le immagini annotate. La piattaforma mette a disposizione diverse funzionalità, sebbene alcune di queste siano accessibili esclusivamente tramite l'abbonamento premium. Tra queste, vi è la possibilità di generare un dataset aggiuntivo contenente immagini simili a quelle originali, ma con l'aggiunta di effetti di distorsione e altri accorgimenti. Tale funzionalità risulta particolarmente utile poiché contribuisce ad aumentare la varianza del dataset, introducendo elementi di rumore e incrementando così la capacità di adattamento del modello. È importante specificare che tutte le immagini utilizzate per l'annotazione sono state rettificate per risolvere il problema della distorsione (si veda il paragrafo 3.6).

4.2.1.1 Le classi annotate In totale tra tutte le classe all'interno del dataset, sono contenute oltre 7000 annotazioni. Più nello specifico una media di 13.9 annotazioni per immagine. Per l'annotazione sono state utilizzate sia immagini che rappresentano situazioni "corrette" che casi particolari.

Le struttura delle classi annotate all'interno del dataset è la seguente:

1. **cone** la classe con più annotazioni all'interno del dataset; conta oltre 5000 annotazioni.

2. **motorcycle** rappresenta la classe della moto.ì; questa contiene oltre 450 annotazioni.

3. **Xcone** rappresenta la classe dei coni caduti; conta oltre 550 coni caduti annotati.

4. **pilot** la classe meno incisiva ai fini del progetto, ma fondamentale solo nel caso in cui si voglia analizzare nel dettaglio le movenze del pilota; anche questa conta oltre 450 annotazioni.

Nonostante il numero di annotazioni non sia bilanciato i risultati della detection sin dal primo momento sono stati superiori alle nostre aspettative. Il motivo per il quale bilanciare i numeri di annotazione tra le classi non è stato semplice è dipeso dal fatto che in una sola immagine si poteva arrivare

ad annotare oltre 80 coni, ma con solo una moto ed un pilota annotati (si veda la figura 27). Questo problema si evidenzia anche con i coni caduti, che sono generalmente un numero inferiore rispetto a quelli in piedi.

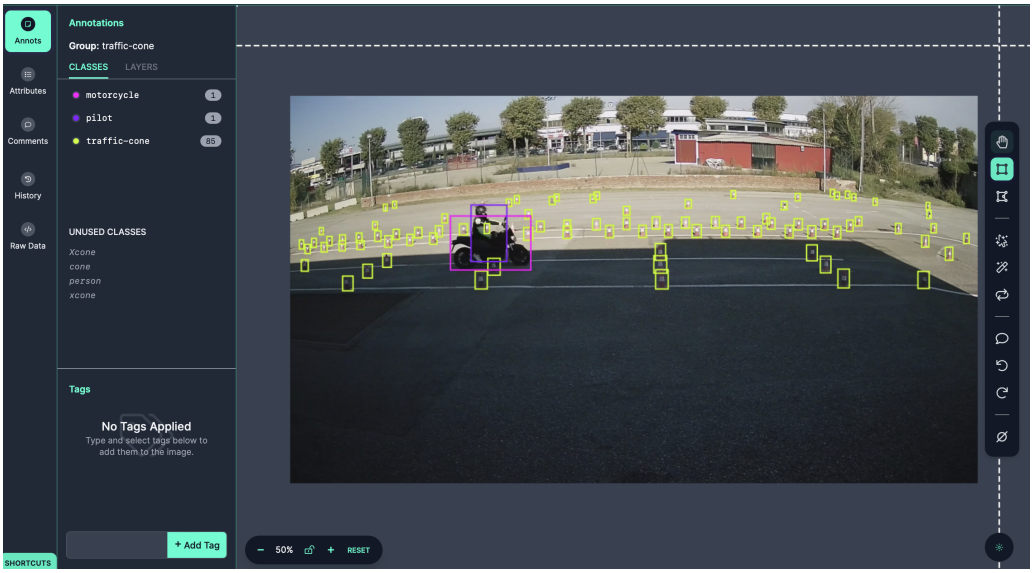

Figure 27: Nella figura si possono vedere i coni, la moto e il pilota annotati e il loro numero sulla sinistra

Inizialmente i coni caduti e quelli in piedi venivano rappresentati in un'unica classe. Successivamente la distinzione in due classi separate ha portato numerosi benefici, tra cui la possibilità di semplificare l'algoritmo di rilevazione degli errori sui coni (descritto nel paragrafo 4.3.2). Rilevare direttamente un cono caduto come tale, anziché dedurlo attraverso ragionamenti logici complessi, si è rivelata una soluzione più semplice.

4.2.2 Procedura di addestramento della rete

Roboflow offre la comodità di scaricare il dataset appena generato direttamente dal browser o tramite un comando da eseguire da terminale. Una volta ottenuto il dataset compatibile con YOLOv5, è sufficiente posizionare le cartelle "train", "valid" e "test" nella sua directory principale e configurare il file "data.yaml" nel proprio ambiente di lavoro.

Per installare YOLOv5 è necessario scaricare il repository[15] da GitHub e per farlo si mandano in esecuzione i comandi sottostanti:

```
!git clone https://github.com/ultralytics/yolov5
!pip3 install -U -r yolov5/requirements.txt

%cd yolov5
```

Il file "data.yaml" contiene i nomi delle classi e le posizioni dei dataset. È importante configurare correttamente questo file, altrimenti il sistema non sarà in grado di individuare le immagini e caricarle per l'addestramento. Questa soluzione permette una gestione agevole del dataset generato da Roboflow, facilitando il processo di preparazione dei dati per l'addestramento del modello YOLOv5.

4.2.3 Allenamento di un modello mediante bouding box

Il sistema di bounding box rappresenta un approccio efficace per affrontare la fase della detection, consentendo di osservare i risultati in modo immediato. Le bounding box sono facili da utilizzare e richiedono l'impiego di uno dei modelli standard di partenza forniti da YOLOv5 (come descritto nel paragrafo 13). L'approccio iniziale è lo stesso per tutti e prevede l'utilizzo del comando "python3" per avviare lo script "train.py" specifico per le bounding box. Questo script, come altri, presenta diversi parametri configurabili. Di seguito viene fornito un esempio:

```
python3 train.py --weights yolov5s.pt --batch -1 --epochs
    ↪ 300 --data dataset/data.yaml --cfg models/yolov5s.
    ↪ yaml --cache disk --img 1920
```

Nello specifico:

1. **weights** indica il modello di partenza di YOLOv5 (si veda il paragrafo 3.1.7).

2. **batch** indica il numero di batch per ogni epochs, se impostato a -1 calcolerà la dimensione massima per il proprio sistema (si veda il paragrafo 3.3).

3. **epochs** indica il numero di epochs per allenamento (si veda il paragrafo 3.3).

4. **data** indica il path da cui prendere il file data.yaml, questo fa riferimento ai diversi path dei dataset train, valid e test.

5. **cfg** indica il file da cui prendere i parametri del modello.

6. **cache** indica dove immagazzinare le immagini durante l'addestramento, si può optare per il disco oppure per la ram.

7. **img** indica la risoluzione di ridimensionamento delle immagini durante l'allenamento. (si veda il paragrafo 4.2.8).

Questi sono i parametri comunemente più utilizzati, molti possono essere approfonditi tramite lo studio del file **train.py** (disponibile sul repository di YOLOv5[13]).

4.2.4 Allenamento di un modello con instance segmentation

Quando viene sfruttata l'instance segmentation all'interno del sistema di identificazione della moto (si veda il paragrafo 4.3.1) il modello viene allenato con un comando simile alle bounding box (come indicato nel paragrafo 4.2.3 che affronta il training su bounding box), ma con parametri leggermente diversi. La prima differenza è il modello di partenza che invece di quello standard viene indicato come yolov5x-seg.pt (corrispettivo di yolov5x.pt) o per esempio yolov5m-seg.pt (corrispettivo di yolov5m.pt). Il numero di parametri è simile, ma quello che varia con questi modelli è il tempo di inferenza, maggiore rispetto ai modelli standard.

Per eseguire il train viene normalmente usato il seguente comando:

```
1  python3 segment/train.py --batch -1 --epochs 300 --data
       ↪ dataset/data.yaml --cfg models/yolov5m-seg.yaml --
       ↪ cache disk --img 1920
```

4.2.5 Allenare il modello con più GPU

Come descritto nel paragrafo 3.4 sull'introduzione all'ambiente della dgxa100 è necessario l'ausilio di allenamenti mediante l'elaborazione di più gpu in parallelo. Non possono essere usati i comandi sopra indicati poiché questi non risulterebbero ottimizzati allo scopo. Per risolvere il problema è necessario aggiungere al comando già usato in precedenza la stringa:

```
-m torch.distributed.run --nproc_per_node Nnodes
```

e quindi:

```
python3 -m torch.distributed.run --nproc_per_node 2 train.
    ↪ py --batch -1 --epochs 300 --data dataset/data.yaml
    ↪ --cfg models/yolov5m-seg.yaml --cache disk --img 1920
    ↪  --device 0,1
```

Le differenze principali sono:

1. **nproc_per_node** indica il numero di nodi utilizzati, in questo caso con due schede abbiamo due nodi.

2. **device** indica quali schede utilizzare durante l'apprendimento. Per esempio, su un sistema con tre schede numerate da 0 a 2, scrivendo device 0,1 andremo ad utilizzare le prime due.

4.2.6 Utilizzare il modello allenato

Per effettuare la detection basta semplicemente lanciare il comando python3 con lo script detect.py (nel caso il modello fosse allenato con bouding box) o predict.py (nel caso di un modello con instance segmentation). Questi file sono molto importanti perché sono stati modificati per effettuare la detection della posizione della moto e per il rilevamento degli errori dei coni.

Ecco alcuni esempi:

```
python3 detect.py --source ../Video/video.mp4 --weigths ../
    ↪ models/mymodels.pt --conf-thres 0.5 --img 1920
```

```
python3 segment/predict.py --source ../Video/video.mp4 --
    ↪ weigths ../models/mymodels.pt --conf-thres 0.5 --img
    ↪ 1920
```

Inoltre sono stati creati due nuovi parametri per rendere il sistema più configurabile al problema della distorsione:

1. **rectify** se abilitato applica la correzione della distorsione sull'immagine o video a cui viene applicata la detection (si veda il paragrafo 3.6).

2. **camera** serve per selezionare dal file ".ini" i coefficienti di distorsione in base alla telecamere utilizzata, L1 o L2 (si veda il paragrafo 3.6).

4.2.7 Analisi delle prestazioni dei modelli iniziali generati

Il primo allenamento è consistito nel testare una prima versione del dataset. Questo dataset era costituito da circa 100 immagini, ottenute dalle prime registrazioni effettuate sul campo e quindi contenenti pochi casi particolari.

```
1   python3 train.py --batch -1 --epochs 250 --data
        ↪ coneDetectionV6/data.yaml --project testRoboflowV3 --
        ↪ weights yolov5s.pt --cache disk
```

Nel primo comando eseguito, sono state utilizzate le configurazioni minimali. È stato selezionato il modello yolov5s come modello di partenza e il batch size è stato impostato a -1. L'allenamento è stato eseguito con un numero ridotto di epochs (250). Successivamente, è stato utilizzato il seguente comando per eseguire la fase di detection:

```
1   python3 detect.py --weights testRoboflowV3/exp2/weights/
        ↪ best.pt --source ../03\ dahua\ 14\ Novembre\
        ↪ 2022/1668419601-14.novembre.2022-10.53.mp4 --conf-
        ↪ thres 0.1
```

Come mostrato nella figura 14 nel paragrafo 3.1.7, il modello yolov5s è caratterizzato da circa 7 milioni di parametri, un numero relativamente ridotto rispetto ad altri modelli. Inoltre, durante i test iniziali, il modello ha richiesto circa 14,5 GB di memoria video. È importante notare che il consumo di memoria dipende dalla batch-size e dalla risoluzione utilizzate. Nei primi esperimenti, è stata adottata la risoluzione standard di 640x387 pixel.

Come si vede dal grafico di figura 28, nelle prime 100-120 epochs l'apprendimento è molto altalenante fino a diventare più lineare. Nell'immagini sottostanti abbiamo l'accuratezza delle bouding box sulla stessa immagine corrispondenti ad epoch diverse. La prima si riferisce alla epoch 201 (si veda la figura 29), mentre la seconda alla 300 (si veda la figura 30).

È evidente che la confidenza è aumentata dal 76,9% nel primo cono coperto al 79,2% nella seconda immagine. Questo risultato è molto interessante in quanto dimostra che l'aggiunta di 100 epochs su un dataset così limitato può portare a miglioramenti significativi delle prestazioni. È stata quindi condotta un'altra prova utilizzando lo stesso dataset, ma addestrando il modello YOLOv5l. Rispetto al modello precedente, questo è caratterizzato da una complessità maggiore. La stabilità maggiore viene evidenziata dai risultati

Figure 28: I valori riportati nel grafico sono la precision, la recall, la mAP_0.5 e la mAP_0.5:0.95

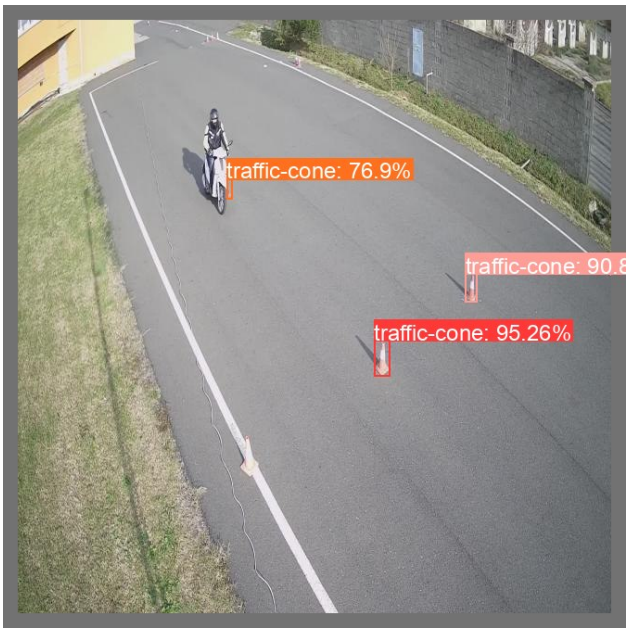

Figure 29: Immagine con 201 epochs

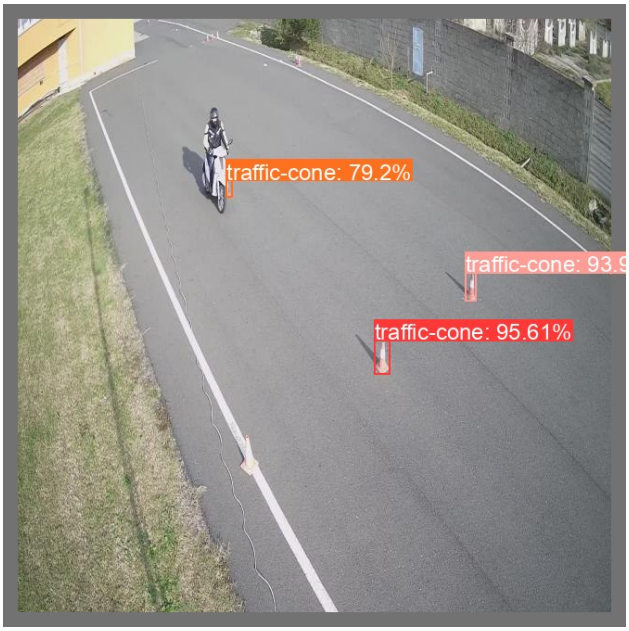

Figure 30: Immagine con 300 epochs

ottenuti dopo 100 epochs (si veda il grafico di figura 31). È importante sottolineare che a causa delle limitate risorse di memoria video disponibili (24GB), non è stato possibile utilizzare un batch-size pari al numero di immagini del dataset. I risultati ottenuti mettono in luce l'importanza del modello utilizzato e delle risorse hardware a disposizione nel determinare le prestazioni complessive del sistema di rilevazione. L'utilizzo di modelli più complessi può portare a miglioramenti delle prestazioni, ma richiede anche una maggiore capacità di calcolo. Pertanto, è cruciale trovare un equilibrio tra la complessità del modello e le risorse disponibili al fine di ottenere risultati ottimali.

```
1    python3 train.py --batch -1 --epochs 300 --data
        ↪ coneDetection100withyolov5l/data.yaml --project
        ↪ coneDetection100ModelWithyolov5l --cfg models/yolov5l
        ↪ .yaml --weights yolov5l.pt --cache disk
```

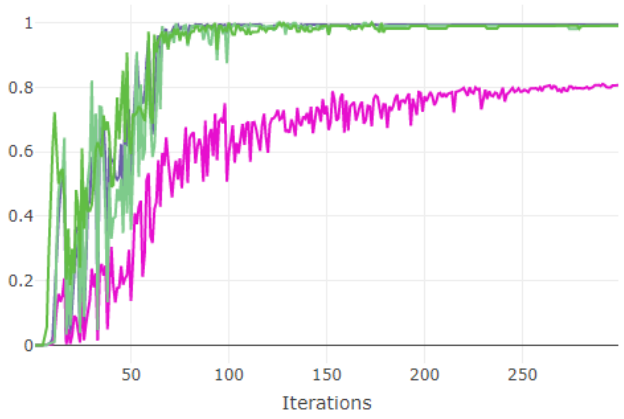

Figure 31: Grafico dell'allenamento con il modello yolov5l

4.2.8 Confronto delle risoluzioni

Con il sistema di Object Detection a risoluzioni diverse corrispondono risultati differenti. I video sono stati girati a 1920 x 1080, una risoluzione che viene considerata accessibile a tutti a rispetto a una risoluzione 3840×2160 (4k). Il ridimensionamento a partire da (640 x 384)[5], mostra come ciò influisce sulle prestazioni di elaborazione, per l'addestramento dei modelli e il rilevamento di oggetti. In questo caso con una risoluzione di 640x384, l'hardware che è stato utilizzato è quello indicato nel paragrafo 6.1. Il tempo di inferenza calcolato è di circa 4-5 ms per ogni fotogramma[6]. Una risoluzione più alta aumenta il tempo di inferenza del modello. Utilizzare una risoluzione maggiore richiederebbe hardware più potente (si veda il paragrafo 3.4). Per capire quale sia la scelta migliore è importante valutare costi e benefici che una maggiore risoluzione pur con un tempo di inferenza maggiore ha nei confronti di una risoluzione inferiore, ma con un tempo di inferenza minore. Sullo stesso video con risoluzioni diverse (in questo caso 640 x 384 e 1280 x 736 ovvero 720p), possiamo vedere come lo sforzo di avere dettagli in più determini differenze importanti in termini di precisione (si vedano le figura a confronto 32 e 33).

[5]Se non specificata la risoluzione delle immagini o dei video su YOLOv5 viene riadattata alla dimensione standard di 640 x 384

[6]Le prestazioni vengono anche influenzate dal numero di oggetti da identificare nell'immagine. Per esempio: più coni ci sono, maggiore sarà il tempo di inferenza

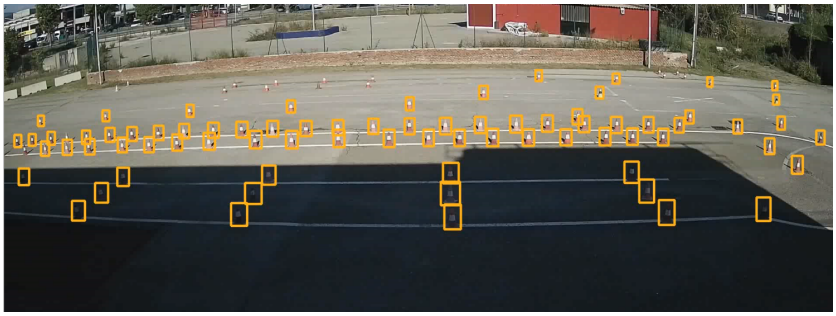

Figure 32: Immagine primo frame con 640 x 384

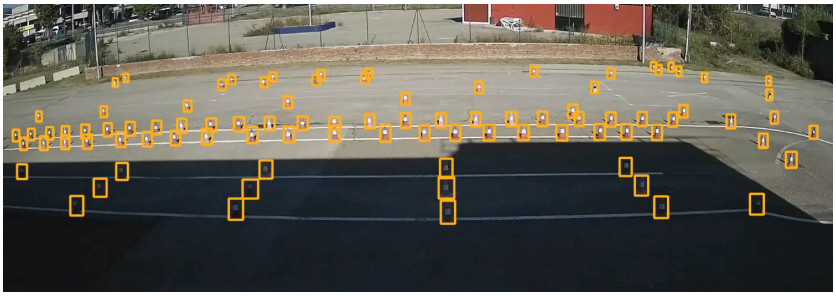

Figure 33: Immagine primo frame con 1280 x 736

Nella prima immagine non vengono riconosciuti tutti i coni stradali (si noti bene i coni in fondo), mentre nella seconda questi vengono riconosciuti dal primo all'ultimo. Tutte le informazioni aggiuntive raccolte attraverso una risoluzione migliore non fanno che aumentare la precisione del modello. Un' altro dettaglio che è possibile osservare è il cono caduto (si vedano le figure 34 e 35).

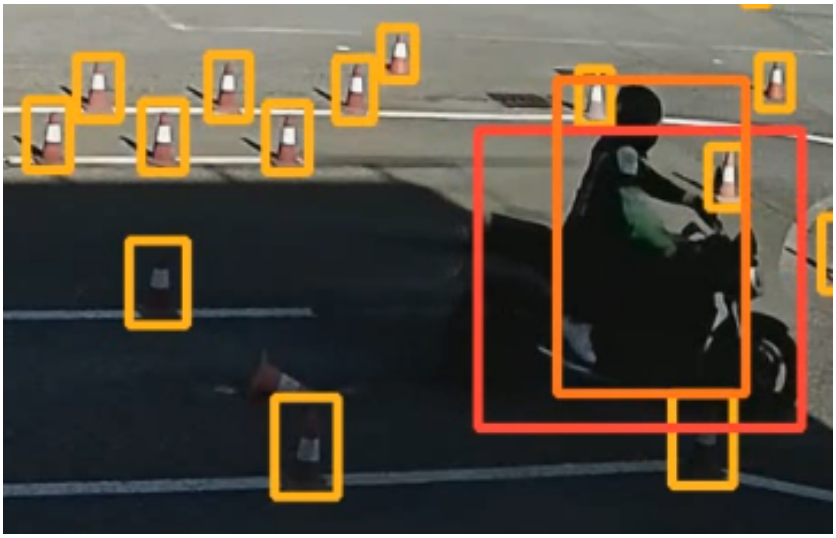

Figure 34: Riconoscimento cono caduto in 640 x 384

Questa differenza è cruciale affinché il sistema funzioni correttamente; il riconoscimento di un cono caduto può portare a numerosi benefici (si veda il paragrafo 4.2.1.1). Ai fini di un corretto funzionamento del sistema di rilevamento degli errori sui coni, acquisire correttamente un cono come tale è cruciale. Ciò dimostra come la risoluzione più alta abbia un impatto reale sulle performance del sistema a discapito di un tempo di inferenza maggiore.

4.2.9 Sviluppo del modello di rilevamento (Large)

Il primo modello utilizzato all'interno del sistema di rilevamento è stato allenato adoperando yolov5x come base. yolov5x è il modello più grande e complesso disponibile nel framework YOLOv5. La scelta di utilizzare questo modello è stata motivata principalmente dalla necessità di garantire un buon rilevamento anche dei coni più lontani. Il modello con il maggior numero di parametri disponibili si è rivelato la migliore opzione per questo scopo.

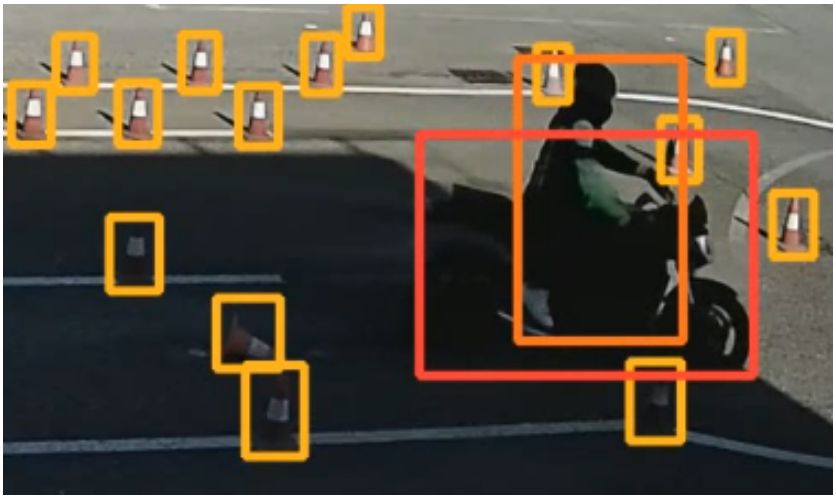

Figure 35: Riconoscimento cono caduto in 1280 x 736

L'allenamento del modello è stato eseguito utilizzando il sistema DGXA100 (descritto nel paragrafo 3.4). Questo sistema è in grado di gestire il carico di lavoro richiesto dai numerosi parametri del modello e dalla risoluzione delle immagini impostata a 1920x1080. Il dataset utilizzato per l'allenamento conteneva oltre 7000 annotazioni (come descritto nel paragrafo 4.2.1.1). Il modello ha ottenuto ottimi risultati e ha richiesto un numero di epochs relativamente basso prima di manifestare segni di overfitting (si veda la figura 36) inoltre è possibile osservare la matrice di confusione ottenuta durante il processo di apprendimento (si veda la figura 37).

Nonostante le buone prestazioni, il modello è stato successivamente abbandonato a causa del tempo di elaborazione richiesto. Infatti, il modello richiedeva in media 80 ms per frame su un'immagine contenente circa 80 coni. Successivamente, l'allenamento di nuovi modelli ha dimostrato che non era strettamente necessario adottarne di così complessi per il riconoscimento dei coni.

4.2.10 Sviluppo del modello di rilevamento (Medium)

Osservata la difficoltà sul tempo di elaborazione richiesto dal modello yolov5x, è stato preferito optare su un modello che avesse un numero di parametri inferiore, ma che comunque riuscisse a mantenere una buona precisione. Questo modello è stato allenato con lo stesso dataset di quello precedente (si veda

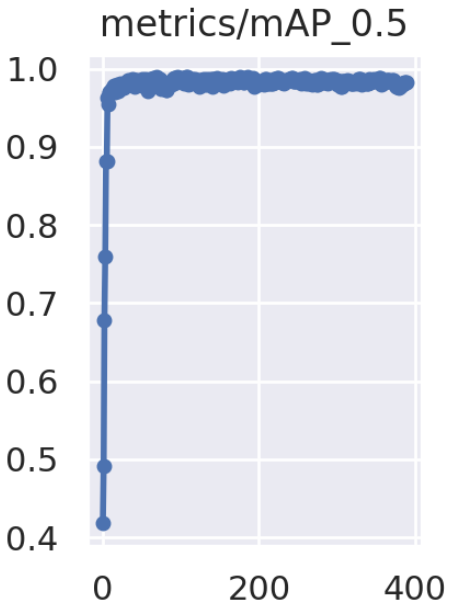

Figure 36: Il grafico mostra la precisione calcolata durante la fase di apprendimento

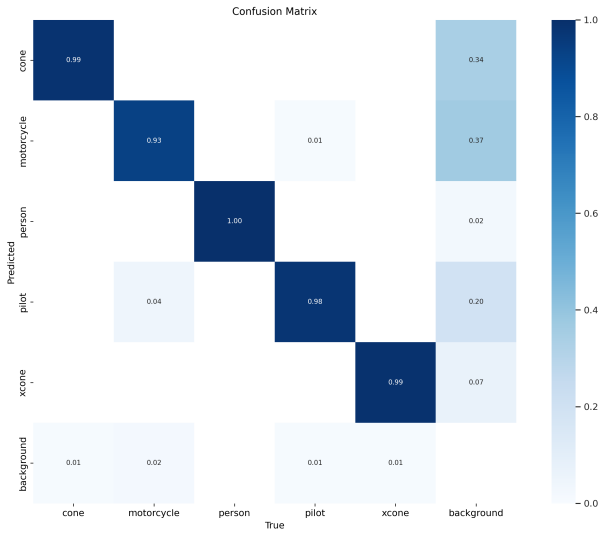

Figure 37: La matrice di confusione del modello con yolov5x

il paragrafo 4.2.9); ha impiegato un numero maggiore di cicli di apprendimento (epochs) prima di raggiungere l'overfitting (si veda la figura 38), questo perché meno complesso è richiedeva più tempo vista la precisione inferiore data dal numero minore di parametri. Il modello è stato un successo, nonostante si fosse passati da un modello più complesso ad uno più semplice il rendimento non è stato per niente intaccato; dimostrando che nel nostro sistema si poteva adottare una soluzione realmente efficiente senza intaccarne la precisione. Si è passati quindi ad elaborare un singolo frame in 30ms contro gli 80ms di yolov5x (4.2.9), portando così benefici enormi, ed avvicinando il progetto ad una possibile soluzione real time. Il modello è stato ufficialmente usato il giorno della demo a Pontedera.

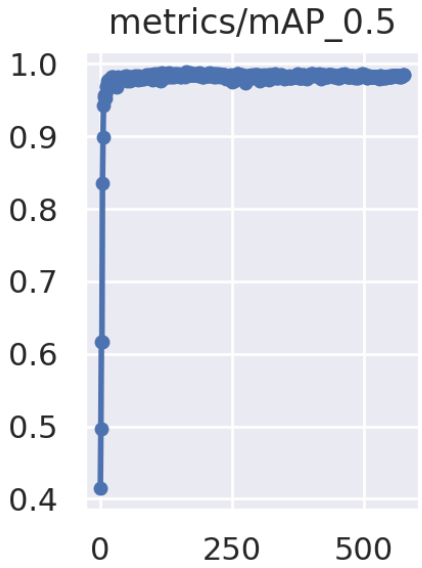

Figure 38: Il grafico mostra la precisione ottenuta durante l'apprendimento

Il modello è stato testato (si veda la figura 40) su un sistema Jetson[34], un dispositivo progettato da Nvidia per l'edge-computing. Il modello è stato ottimizzato con il software TensorRT di Nvidia impostando a 16 bit la precisone ed impostando la modalità di alimentazione sul modulo Jetson per massimizzare la potenza di calcolo, il modulo Jetson può eseguire l'algoritmo di rilevamento degli oggetti a 30-32 frame al secondo(fps).

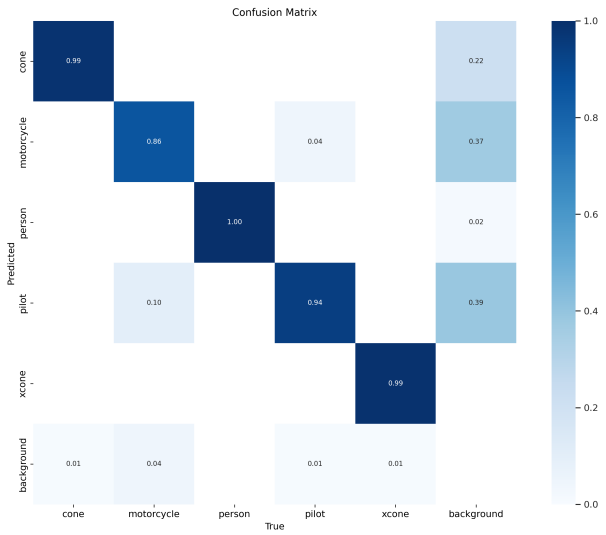

Figure 39: La matrice di confusione del modello con yolov5m

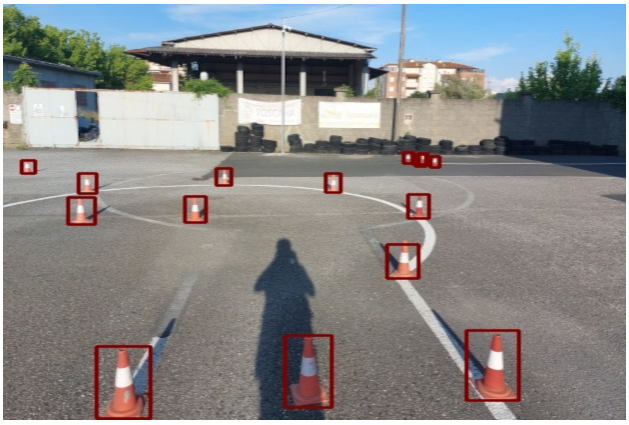

Figure 40: Il sistema può fornire l'output al tablet dell'istruttore di guida in tempo reale, consentendo un feedback e una valutazione immediati.

4.3 Logica di processamento delle detection

Nel paragrafo vengono esposte la progettazione e le soluzioni dei problemi inerenti i sistemi di rilevamento della posizione della moto e di riconoscimento degli errori sui coni. Inoltre viene brevemente spiegato l'output dei sistemi implementati e del post-elaborazione video.

4.3.1 Rilevamento della posizione della moto

All'interno del progetto, il rilevamento della posizione della moto riveste un'importanza fondamentale. L'obiettivo è stato progettare un sistema affidabile che fosse in grado di fornire una stima precisa della posizione della moto nello spazio, tenendo conto della prospettiva e degli effetti di distorsione (si veda il paragrafo 3.6). L'utilizzo della posizione della moto è essenziale per il sistema, poiché consente di tracciarne il percorso e di avere una scala temporale degli eventi che la coinvolgono. Questa è importante cosi da poter valutare gli errori dovuti al percorso che il motociclista esegue. Questo aspetto non viene gestito in questo tirocinio, ma rientra negli incarichi principali fornire la posizione della moto, affinché questa possa venire utilizzata da un sistema a parte. Nella figura 41 si può osservare un esempio di come il percorso della moto venga ricostruito dal sistema.

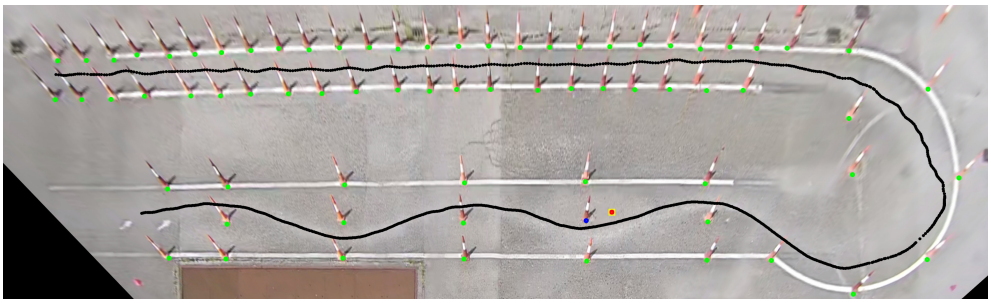

Figure 41: Il percorso della moto ricostruito

4.3.1.1 La posizione della moto ottenuta tramite bounding box

Nella fase iniziale del progetto, è stato adottato un approccio elementare basato sull'utilizzo della bounding box come metodo di annotazione per identificare la moto. Mediante questo sistema di detection sono state osservate

diverse soluzioni al fine di ottenere una buona precisione. Una volta individuata la moto è stato necessario ottenerne la posizione, ma questo non è semplice visto l'effetto della prospettiva. Le telecamere si trovano in posizioni diverse, alcune soluzioni possono solo diminuire l'errore su una, ma aumentarlo su un'altra. Il primo approccio al problema è stato quello di calcolare il baricentro della moto come punto di riferimento per la sua posizione. Tuttavia, questo ha fatto emergere una problematica nello specifico; il punto del baricentro è risultato essere troppo elevato rispetto ai coni circostanti a causa delle distorsioni prospettiche. È stato evidente che tracciare il percorso della moto in base al baricentro avrebbe dato l'idea che questa fosse molto più in alto rispetto alla realtà. Un altro approccio è stato quello di utilizzare il punto di terra, ovvero il punto inferiore al centro del rettangolo della detection della moto. Anche se questa soluzione ha contribuito a ridurre il problema della prospettiva, non è stato ancora sufficiente per ottenere una posizione da considerarsi accurata.

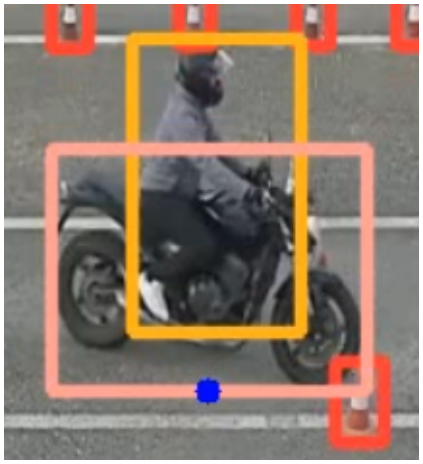

Figure 42: Il punto blu coincide con il punto di terra del cono

Nella figura 43 si può notare che il punto blu (punto di terra) è troppo vicino al punto di terra del cono, la distanza non rispecchia in alcun modo la realtà.

L'approccio finale adottato durante la demo è stato quello di calcolare un punto intermedio che si sposta tra il baricentro della moto e il punto inferiore al centro del rettangolo di rilevamento della moto. Questo approccio ha prodotto risultati significativamente più precisi rispetto a tutte le altre

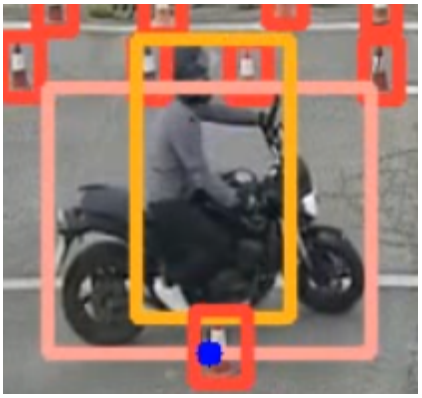

Figure 43: Immagine che illustra come il punto blu non rispecchi una posizione affidabile

soluzioni precedentemente menzionate. Per implementare questa soluzione, è stato utilizzato il seguente codice sorgente:

Listato 3: Rilevamento della posizione della moto

```
for det in reversed(det):
    xyxy, conf, cls = det

    if save_txt: # Write to file
        TLx, TLy, BRx, BRy = xyxy

        if cls == 1: # CLASSE MOTO
            Tx = (TLx + BRx)
            Ty = BRy
            Bx = (TLx + BRx)
            By = (TLy + BRy)
            W = BRx - TLx
            H = BRy - TLy
            ratio = W / H
            Rx = Bx
            alfa = ratio - OFFSET
            if alfa <= 0:
                alfa = 0
            if alfa >= 1.0:
                alfa = 1
            R0 = (Ty + By)
            Deltay = Ty - R0
            Ry = R0 + alfa * Deltay

            if FirstFrame:
                MPx = Rx
```

```
27                MPy = Ry
28            else:
29                if MPx == 0 and MPy == 0:
30                    MPx = Rx
31                    MPy = Ry
32                else:
33                    dist1 = distanceFromPoints(PrevMotoPx, PrevMotoPy, MPx, MPy)
34                    dist2 = distanceFromPoints(PrevMotoPx, PrevMotoPy, Rx, Ry)
35                    if dist1 > dist2:
36                        MPx = Rx
37                        MPy = Ry
```

Per ottenere la posizione della moto, vengono considerati i punti che definiscono la bounding box del rettangolo, ovvero TLx, TLy (Top-Left) e BRx, BRy (Bottom-Right). Questi punti sono calcolati automaticamente da YOLOv5 durante il ciclo di rilevamento che avviene per ogni frame (riferimento alla riga 1 del codice). Inizialmente viene verificata la classe di riferimento della detection (indicata dalla classe 1). Se la classe corrisponde a una moto, vengono calcolati il punto di baricentro e il punto di terra associati alla moto rilevata. Successivamente viene calcolata la ratio del rettangolo di rilevamento, che rappresenta il rapporto tra la larghezza e l'altezza della bounding box. A causa della prospettiva, questa ratio può variare nel tempo, tendendo a diventare gradualmente un quadrato. Il punto calcolato viene quindi spostato linearmente verso l'alto e verso il basso in base alla variazione della ratio, con il punto di terra come valore massimo e il baricentro come valore minimo. Al fine di gestire eventuali duplicati di detection che potrebbero causare uno sfasamento nella posizione reale della moto, viene effettuato un controllo aggiuntivo. Tra i punti calcolati nel frame corrente viene utilizzato il punto più vicino a quello del frame precedente. Sotto si possono osservare le figure 44 e 45 che mostrano come il punto verde (la posizione calcolata della moto) si sposti in base alla alla ratio della bounding box.

4.3.1.2 La posizione della moto ottenuta tramite istance segmentation

L'instance segmentation con la sagoma dell'oggetto rilevato, insieme alle bounding box disegnate, ha permesso di ottenere informazioni più precise sulla posizione della moto. In particolare, è stato preso in considerazione il punto di terra più basso e centrale all'interno della sagoma della moto, come evidenziato nella figura 46. Questo punto di terra rispetto a quello ottenuto tramite la bounding box ha dimostrato di fornire una precisione leggermente

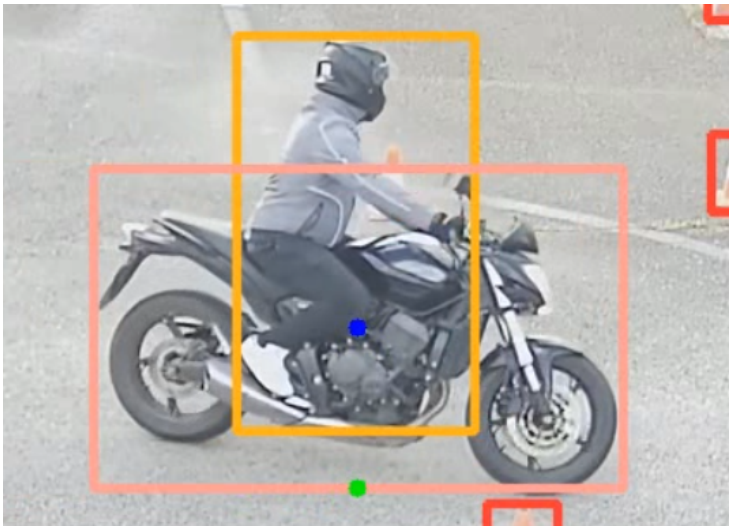

Figure 44: Il punto verde si sovrappone al punto di terra

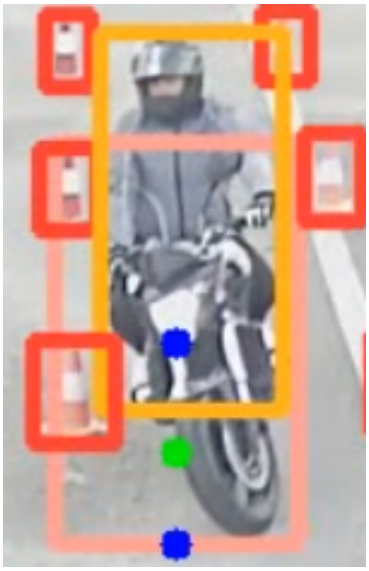

Figure 45: Il punto verde si trova tra il baricentro ed il punto di terra

maggiore. Sfruttando la sagoma dell'oggetto, è stato possibile ottenere una posizione più accurata, basata sulla forma effettiva della moto, a differenza della bounding box in cui la posizione variava in base alla detection e non alla forma reale della moto. Questa sistema non è stato utilizzato per motivi legati alla difficoltà delle annotazioni.

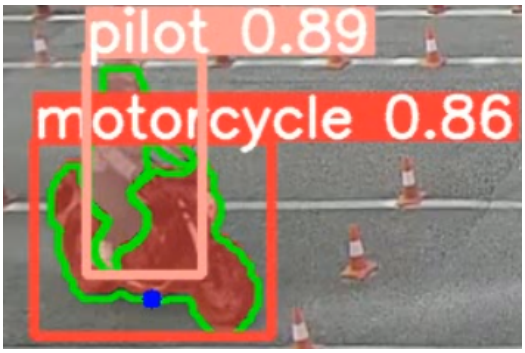

Figure 46: Il punto blu indica la posizione della moto

Il secondo approccio proposto, sebbene non sia stato testato approfonditamente, si basava sull'utilizzo dell'inviluppo convesso dei punti che compongono la sagoma della moto. Una volta calcolato si sarebbe proceduto al calcolo di una retta che congiunge i due punti di terra delle ruote. Successivamente, si sarebbe preso il punto medio di questa retta come posizione finale della moto.

Alla fine è stato deciso di utilizzare il sistema di rilevamento mendiante bouding box per tre ragioni:

1. **Numero di modelli**: A differenza della bounding box, l'instance segmentation richiede l'uso di un modello separato per l'elaborazione, in quanto utilizza approcci di apprendimento diversi. Ciò comporta la necessità di addestrare e gestire due modelli distinti: uno per la rilevazione dei coni e uno per la rilevazione della moto. Questo avrebbe aggiunto complessità al sistema rispetto all'utilizzo di un unico modello.

2. **Tempi di elaborazione**: Durante la demo, era fondamentale ottenere un'elaborazione rapida. Avere due modelli distinti, uno per i coni e uno per la moto, avrebbe comportato un aumento significativo dei tempi di elaborazione rispetto all'utilizzo di un unico modello in grado di

gestire entrambi i compiti. La scelta di utilizzare la bounding box ha consentito di ottimizzare i tempi di esecuzione senza compromettere le prestazioni complessive.

3. **Annotazione delle immagini**: Rispetto alla semplice annotazione della bounding box che richiede solo di tracciare un rettangolo intorno alla moto, l'instance segmentation richiede l'annotazione dell'intera sagoma della moto. Ciò comporta un notevole aumento del tempo richiesto per l'annotazione delle immagini. Scegliere l'approccio della bounding box ha permesso di ridurre il carico di lavoro per l'annotazione, semplificando il processo complessivo.

4.3.2 Logica di controllo delle detection dei coni

Uno degli elementi più importanti del lavoro svolto riguarda il riconoscimento delle penalità relative ai coni. L'idea di base è quella di monitorare la posizione di default che ogni cono dovrebbe avere all'inizio della prova. Tuttavia, alcune delle soluzioni considerate inizialmente non risolvevano efficacemente la complessità del problema. L'approccio per verificare se un cono è in regola è stato quello di mappare le posizioni iniziali dei coni. Queste posizioni sono fondamentali poiché è essenziale sapere fin da subito se tutti i coni sono presenti e se si trovano nella posizione prestabilita. Per affrontare questa problematica, è stato adottato il seguente approccio: sono state acquisite le posizioni iniziali dei coni ed è stata creata un'immagine di riferimento, completamente nera, in cui tutti i pixel sono impostati a zero. Questa mappa di riferimento viene chiamata "Look up table". Successivamente sono state raccolte sei informazioni fondamentali per ogni cono: l'ID, il baricentro, le coordinate X1, Y1, X2, Y2. Le ultime quattro informazioni vengono utilizzate per calcolare l'area della bounding box.

4.3.2.1 Costruzione della look up table La parte principale del sistema è la look up table, che viene utilizzata per mappare le posizioni iniziali dei coni e usarle come riferimento per le nuove rilevazioni. Prima di costruire la look up table, viene effettuata la fase di calibrazione. All'interno della cartella "airide", è presente una sotto-cartella chiamata "csvdefault", che contiene file CSV contenenti le posizioni di default dei vari coni. Ogni telecamera ha il suo file specifico, poiché il cambio di prospettiva fa apparire lo stesso cono in due posizioni completamente diverse nell'immagine. Inoltre

è necessario effettuare la calibrazione per correggere eventuali spostamenti leggeri dei coni. La calibrazione avviene per i primi K frame, durante i quali vengono apportate correzioni alla posizione dei coni. Questo processo permette di allineare correttamente le posizioni rilevate con le posizioni di default predefinite della look up table.

Ecco lo pseudo codice della calibrazione:

Listato 4: Pseudocodice della calibrazione

```
function calibrationCamera(cones, conesInF):
    for i in cones
        # Controllo quante detection sovrapposte ci sono
        dim = len(conesInF[i+1])
        if dim >= 2:
            mindist
            #Se sono presenti piu' detection sulla stessa area
            #Ciclo le detection e prendo quella piu' attendibile
            #Rimuovo tutte le altre e lascio solo quella attendibile
        if dim == 0:
            print("Non ho trovato il cono")
        elif dim == 1:
        # Se la dim e' 1
        # Calcolo la media dei valori precedenti con quello corrente
        newTLx = mediaTLx + currentConeTLx
        newTLy = mediaTLy + currentConeTLy
        newBRx = mediaBRx + currentConeBRx
        newBRy = mediaBRy + currentConeBRy
        newCordinateConesIndex = (newTLx,newTLy,newBRx,newBRy)
```

Per il codice completo in Python si vada in appendice 6.4. Le coordinate calcolate a fine calibrazione vengono salvate in nuovo file csv. Una volta completata la calibrazione viene creata la look up table. Per farlo vengono sfruttate le coordinate corrette mediante calibrazione. Anche un solo pixel di differenza può dare risultati non allineati con la realtà. Durante il processo di costruzione della look-up table, si sono verificate delle sovrapposizioni con i coni più distanti. Al fine di risolvere questo problema, si è adottato un approccio diverso rispetto all'utilizzo iniziale di un solo livello di colore. Invece, si utilizzano tutti e tre i livelli di colore (rosso, verde e blu) al fine di tener conto delle sovrapposizioni che non sono dovute ad errori.

Lo pseudo codice per costruire la look up table:

Listato 5: Pseudocodice della lookuptable

```
def buildHashMap(blackHashMap, cones):
    print("Building hashmap image")
    #blackHashMap = getHashMap(name)
```

```
4     for c in cones:
5       if c.valid:
6         count = 0
7         count2 = 0
8         count3 = 0
9         c1, c2, c3 = 0, 0, 0
10        for x in [topleft.x - bottomright.x]:
11            for y in [topleft.y - bottomright.y]:
12                count += 1
13                val0 = blackHashMap[y,x,0]
14                val1 = blackHashMap[y,x,1]
15                val2 = blackHashMap[y,x,2]
16
17                #Vengono verificate val0,val1,val2
18                #Se il primo e' diverso da 0, allora e' occupato
19                #Allora passo al secondo livello
20                #Se il secondo e' diverso da 0, allora e' occupato
21                #Allora passo al terzo livello
22                #Se il terzo e' diverso da 0, allora e' occupato
23                #Allora genero errore per troppe sovrapposizione
```

Per esempio se un cono che possiamo considerare valido va a toccare un area già segnata da pixel diversi da zero può significare una sovrapposizione. In questo caso andremo ad aggiungere l'indice del cono al secondo livello o addirittura al terzo. Completata la mappatura della look up table questa verrà utilizzata nei successivi frame per riconoscere i possibili errori.

4.3.2.2 Data association and Filtering Il sistema valuta in base alla posizione dei coni presenti nella look up table gli eventuali errori che questi generano. Questo entra in funzione ad ogni singolo frame. La prima cosa effettuata dal sistema è la classificazione delle detection, mediante l'algoritmo di classificazione. Tutte le detection presenti nel frame vengono indicizzate in base al confronto tra la posizione del centro della detection e la look up table. Se per esempio il centro della detection dovesse cadere in un area scritta da tutti 30, allora questa molto probabilmente (vengono considerati altri aspetti come la sovrapposizione) sarebbe la detection corrispondente al cono con id 30. Dopodiché questa viene inserita nella struttura chiamata conesInFrame, che rappresenta il numero di detection su ciascun id cono.

Lo pseudo codice della classificazione è:

Listato 6: Pseudocodice della classificazione delle detection

```
1  function classifyDetection(lookUpTable, detection, conesInFrame):
2    #conesInFrame raggruppa le detection che cadono sullo stesso indice
```

```
3   #Calcolo le cordinate x,y del centro dell'area della detection
4    detectionCenterX, detectionCenterY = detection.center()
5    #Avendo a disposizione il centro ottengo il valore dell'indice a quelle
         ↪ cordinate sulla lookUpTable
6    index = lookUpTable[detectionCenterY, detectionCenterX,0]
7    #Prendo il puntatore alla lista di detection
8    #Il ragionamento non e' lontano da un hash table
9    # Al c corrisponde una lista di detection che vanno da uno o piu'
10   c = conesInFrame[index]
11   c.append(detection)
12   #Se l'indice sulla lookUpTable e' 0 non faccio niente
13   #Bisogna tenere conto che possa essere un cono spostato
14   #Su indice 0 non c'era nulla secondo la lookUpTable
15   if index == 0:
16     #STAMPO EVENTO
17     #Altrimenti la detection cade su qualcosa
18   else:
19     #Prendo l'indice del secondo livello
20     index1 = lookUpTable[detectionCenterY, detectionCenterX,1]
21     #Se questo e' maggiore di 0 (C'e' una sovrapposizione)
22     if index1>0:
23       #Salvo la detection sull'indice che rappresenta
24       c = conesInFrame[index1]
25       c.append(detection)
26       index2 = lookUpTable[detectionCenterY, detectionCenterX,2]
27       #Faccio la stessa cosa per il terzo livello se necessario
28     if index2>0:
29       #STAMPO EVENTO
30       c = conesInFrame[index2]
31       c.append(detection)
32     else:
33       #STAMPO EVENTO
```

4.3.2.3 Generazione degli eventi

Completata la classificazione si passa al sistema incaricato a riconoscere gli errori.
Questo viene svolto dal metodo processDetectList il cui funzionamento generale si suddivide:

- La funzione prende diversi parametri in ingresso, tra cui il frame corrente, la lista dei coni rilevati, le detection di ciascun cono nel frame corrente, le detection che sono cadute sullo 0, la detection della moto, e la lista degli errori identificati eventsList.

Listato 7: Parametri della processDetectList

```
1  function processDetecList(frame, cones, conesInFrame, moto, newPos,
       ↪ eventsList):
2      #frame e' il numero del frame
3      #cones e' la lista dei coni
4      #conesInFrame le detection che compaiono su ogni indice
5      #newPos detection che cadono sullo 0
6      #moto rappresenta la detection della moto
7      #EventList e' la struttura che contiene tutti gli errori che vengono
           ↪ identificati
```

- La funzione inizia controllando se ci sono delle detection cadute sullo 0. Se questo è il caso, popola una lista chiamata newPos con le detection che si trovano in posizioni nuove rispetto al frame precedente. Successivamente queste detection in newPos vengono processate per cercare eventuali errori e aggiungerli alla lista eventsList.

Listato 8: Caso coni caduti su indice 0 nella processDetectList

```
1  numdet = len(conesInFrame[0])
2    #Se sono maggiori di 0, allora ci sono delle detection fuori posto
3    if numdet>0:
4      #STAMPO EVENTO
5      #Controllo tutte le detection su indice 0
6      for d in conesInFrame[0]:
7        #Verifico che la moto sia nella vicinanze
8        if moto == None:
9          #Non ho la moto nelle vicinanze
10         nearmoto = False
11       else:
12         #Attraverso questo metodo posso controllare se la detection si trova
               ↪ nell'area di perturbazione della moto
13         nearmoto = pointNearBbox(d.Cx,d.Cy,moto.TLx,moto.TLy,moto.BRx,moto.
               ↪ BRy,MINDIST)
14         #Se con una moto vicino lo segnalo, ma non tengo conto dell'evento
15       if nearmoto:
16         #STAMPO EVENTO
17       else:
18         #Popoliamo la newPos list con le detection che si trovano in
               ↪ posizioni nuove
19         processZeroIndexDetection(d,newPos,debugLevel)
20         #Attraverso il metodo vengono verificate le detection su newPos
21         processNewPositionList(newPos,eventsList,debugLevel,StatusError,frame
               ↪ )
22         #Se non sono state gia' segnalate le detection qui vengono segnalate
               ↪ per la prima volta
23         #Viene opportunamente verificato che queste detection non siano al di
               ↪  fuori di una certa soglia
```

- Dopo aver identificato le detection cadute sullo 0, il metodo procede a controllare ogni cono rilevato nella lista cones e le relative detection nel frame corrente. Il comportamento per ciascun cono dipende dal numero di detection associate a esso:

  (a) Se ci sono due detection associate a un cono, il metodo verifica se queste detection si sovrappongono. Se le classi delle detection corrispondono, viene presa quella con la confidence maggiore. Se le classi non corrispondono, viene verificato lo stato.

  (b) Se ci sono più di due detection associate a un cono (caso molto raro), seleziona la detection più vicina al cono e la considera come la detection corretta. Altre detection associate al cono vengono eliminate.

  (c) Se c'è una sola detection associata a un cono, il metodo controlla la distanza tra la detection e il cono. Se la distanza supera una soglia minima, il cono potrebbe essere stato spostato e l'errore viene aggiunto alla lista eventsList. Se la distanza è entro la soglia, la classe e la confidence della detection vengono salvate nel cono.

Listato 9: Caso di una sola detection sul cono corrispondente

```
1  if numdet == 0:
2    if c.Status != STATUSRESET:
3      #STAMPO EVENTO
4      if nearmoto:
5        #STAMPO EVENTO
6        #Resetto la timewindow
7        if c.TimeWindow>0:
8          c.TimeWindow = 0
9          #STAMPO EVENTO
10         c.Occluded +=1
11       else:
12         if c.Status != STATUSLOST:
13           if c.TimeWindow == 0:
14             c.NextStatus = STATUSLOST
15             c.TimeWindow += 1
16           if c.TimeWindow >= FRAMEWIN:
17             delta = frame - c.LastSeen - c.Occluded #puo' diventare
                   ↪ negativo?
18             if delta<0: # prudenza
19               delta=1
20             rate = c.TimeWindow / delta
21             if rate > RATEDETECTION:
22               c.Status = STATUSLOST #Setto lo stato su perso
```

```
23                c.NextStatus = STATUSRESET #Resetto il next status
24                c.TimeWindow = 0 #Resetto timeWindow
25                eventObj = EventClass.Event(c.LastSeen,c.Id,c.Status,c.Cx,
                      ↪ c.BRy,c.Conf)
26                eventsList.append(eventObj)
27              else: #status = LOST
28                findNearPos(c,newPos,debugLevel)
```

Il metodo effettua vari controlli sullo stato del cono e lo gestisce. Gli stati in questione possono essere STATUSINIT, STATUSMOVED e STATUSRESET. Se viene rilevato un errore durante le verifiche allora viene aggiunto l'evento in questione alla eventsList. Per il codice si veda in appendici 16.

4.3.3 Gli output del sistema

Entrambi i sistemi come già detto, devono tornare la posizione della moto e lo stato dei coni a fine prova. Per farlo vengono stampati due file "moto.txt" e "eventi.txt" all'interno della cartella di output dell'elaborazione.

- Il file "moto.txt" contiene per ogni riga il frame e la coppia (x,y) della posizione calcolata della moto.
- Il file "eventi.txt" contiene per ogni riga il frame, l'id del cono, lo stato, le coordinate x e y e la confidenza della detection.

Listato 10: Esempio del file di output moto.txt

```
1 122 (100, 571)
2 123 (101, 570)
3 124 (103, 569)
4 125 (105, 568)
5 126 (107, 568)
```

Listato 11: Esempio del file di output eventi.txt

```
1 25,1,1,270,541,0.91967
2 25,2,1,163,621,0.91246
3 25,3,1,347,577,0.90672
```

All'interno del video vengono anche disegnate le posizioni dei coni(punti di terra) e gli id dei coni, inoltre il rettangolo blu intorno alla moto indica l'area di effetto del "motoisnear" (si veda il paragrafo 4.3.2).

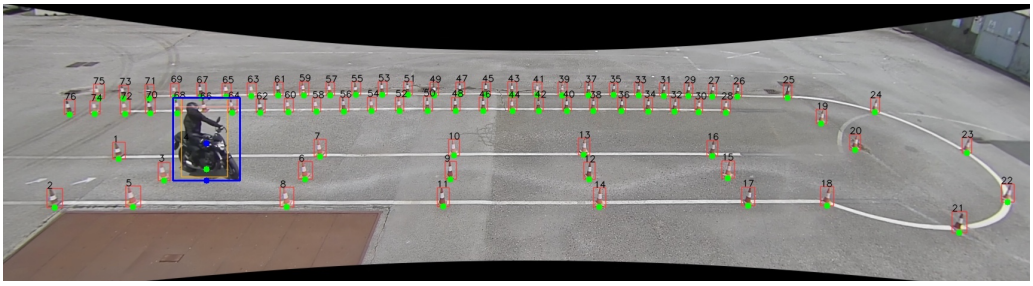

Figure 47: La figura mostra una frame del video dopo l'elaborazione

5 Conclusioni

Il presente tirocinio si è svolto da Gennaio a Giugno 2023 presso l'Istituto di Scienza e Tecnologie dell'Informazione del Consiglio Nazionale delle Ricerche (ISTI-CNR). L'ambito di lavoro è stato il progetto di ricerca Artificial Intelligence driven RIding Distributed Eye (AI-RIDE) che ha come obiettivo lo studio e la realizzazione prototipale di un framework di Intelligenza Artificiale integrato nel contesto della formazione dei motociclisti, mirando in particolare ad una verifica standard, misurabile e imparziale dell'esame della patente di guida per motocicli. Il lavoro di tirocinio si è concentrato in primis sullo sviluppo di un sistema di rilevamento degli elementi caratterizzanti dello scenario: la moto, il pilota e i coni stradali segnaletici. Questa attività ha portato allo studio dei sistemi di object detection tra cui quelli a singolo stadio denominati YOLO (You Look Only Once). Si è scelto di usare la versione YOLOv5 perché, oltre alle ottime performance, ha permesso di superare le difficoltà iniziali grazie all'interazione costante con la community. Si sono presi in considerazione diverse risoluzioni video e vari modelli di partenza con numerosità di parametri variabile tra 2 milioni e 80 milioni. E' stato possibile allenare modelli molto complessi grazie all'utilizzo della Universal System for AI Infrastructure, NVidia DGXA100 presente presso ISTI-CNR, un hardware ad elevato parallelismo appositamente progettato per sistemi di Deep Learming. Si è scelto di utilizzare il modello Medium (YOLOv5m) per un giusto compromesso tra precisione del riconoscimento e velocità di elaborazione. Con tale modello custom si è raggiunta una accuratezza del 99% per il riconoscimento dei coni stradali, del 94% per il pilota e del 84% per il motociclo. Quest'ultimo valore può essere migliorato incrementando il numero e i modelli di moto presenti nel dataset.

Successivamente alla fase di object detection è stato sviluppato il sistema per il calcolo della posizione della moto e per la segnalazione di penalità di cono toccato. Per la stima della posizione della moto si è partito dalla detection del riconoscitore e, tenuto conto della prospettiva, la soluzione adottata ha fornito la proiezione del baricentro sul piano di terra. Per il rilevamento di errori di guida relativi ai coni, si è sviluppata una logica di associazione dati basata su lookup table e finestre temporali di conferma per filtrare i falsi positivi e le occlusioni temporanee. Il sistema è stato testato il 14 giugno 2023 alla presentazione in live a Pontedera presso l'autoscuola Gerardo, dove se ne è dimostrata l'affidabilità anche in casistiche particolari.

Figure 48: Presentazione del lavoro svolto nella demo evento il giorno 14 giugno 2023 a Pontedera presso il sito di test della scuola guida Gerardo.

Lo sviluppo di nuove soluzioni YOLO in futuro potrebbe portare a scelte implementative diverse, anche se il modello teorico adottato rimane comunque valido. Potranno essere adottati i modelli basati su YOLOv7 e YOLOv8; questi hanno precisione maggiore, ma sono molto recenti e necessitano di supporto in più dalla community. L'utilizzo della instance-segmentation potrebbe migliorare la stima della posizione della moto mediante inviluppo convesso del profilo ed inoltre potrebbe essere usata anche nel sistema di rilevamento degli errori per capire l'orientamento del cono in casistiche particolari. Come funzionalità aggiuntive si potranno adottare soluzioni per il sistema di rilevamento tramite "pose estimation", che permetteranno di raccogliere informazioni riguardo la posizione del pilota durante la guida. Si

Figure 49: Test live durante la demo del 14 giugno 2023 a Pontedera.

potranno sfruttare framework come CVAT, che consentono l'annotazione in tempo reale attraverso la rilevazione effettuata da un modello preliminare, al fine di aumentare il numero di annotazioni disponibili. Inoltre, attraverso uno studio più accurato dei sistemi Jetson e delle tecnologie TensorRT, i modelli potranno essere ottimizzati per ottenere l'elaborazione del video in tempo reale così da sfruttare l'edge-computing. In futuro verrà pubblicato un articolo dall'ISTI-CNR sul lavoro svolto con il titolo "Computer Vision applied to Motorbike licence exam" (da definire).
Il progetto di tirocinio mi ha dato la possibilità di migliorarmi attraverso il lavoro di squadra e la gestione delle tempistiche, ma soprattutto mi ha dato l'occasione di conoscere meglio delle applicazioni di intelligenza artificiale e computer vision: è un ambito di cui conoscevo solo le basi che sicuramente approfondirò nei miei studi futuri.

6 Appendice

6.1 Ambiente di lavoro

Si è reso necessario installare ed impostare gli strumenti usati ai fini del progetto. Il sistema operativo utilizzato è Ubuntu nella versione 20.04. Dopo averlo installato è importante verificare la versione dei software installati e aggiornarli col seguente comando:

```
1    sudo apt update && sudo apt upgrade
```

Dopodiché è stato necessario passare alla fase di installazione dei driver compatibili con la scheda utilizzata, una RTX 3090 Ti [7](si veda il paragrafo 3.4). Come già spiegato nel paragrafo 3.4, il progetto richiede un'elevata capacità di calcolo. Per installarli è necessario eseguire la sequenza di comandi sotto indicata [35]:

Listato 12: Install CUDA drivers

```
1  wget https://developer.download.nvidia.com/compute/cuda/repos/
       ↪ wsl-ubuntu/x86_64/cuda-wsl-ubuntu.pin
2  sudo mv cuda-wsl-ubuntu.pin /etc/apt/preferences.d/cuda-
       ↪ repository-pin-600
3  wget https://developer.download.nvidia.com/compute/cuda/12.0.0/
       ↪ local_installers/cuda-repo-wsl-ubuntu-12-0-local_12
       ↪ .0.0-1_amd64.deb
4  sudo dpkg -i cuda-repo-wsl-ubuntu-12-0-local_12.0.0-1_amd64.deb
5  sudo cp /var/cuda-repo-wsl-ubuntu-12-0-local/cuda-*-keyring.gpg
       ↪  /usr/share/keyrings/
6  sudo apt-get update
7  sudo apt-get -y install cuda
```

Prima di passare all'installazione di YOLOv5 (si veda il paragrafo 3.1.6) è necessario aver installato il pacchetto python3. Ciò da accesso ai comandi pip3 e python3 , che vengono rispettivamente usati per eseguire lo script in python ed installare moduli essenziali per garantirne l'esecuzione. È stato necessario installare le versioni più recenti per evitare eventuali conflitti. Per farlo si lancia il comando:

```
1    sudo apt install python3.8
```

[7]CUDA core 10752, 24564MB

L'ultimo strumento installato è stato clearML[8][36]. Questi strumenti sono integrati direttamente in YOLOv5, quindi basta installarli e automaticamente partiranno in background durante la sessione di allenamento per ricavarne maggiori informazioni.

Per installarlo è necessario mandare il seguente comando:

```
1  pip3 install clearml
```

Dopodiché viene configurato lanciando:

```
1  clearml-init
```

È necessario inserire le proprie credenziali, così da poter visualizzare gli eventi sulla piattaforma online di clearML.

[8]Il software non è stato utilizzato sulla DGXA100 a causa delle limitazioni dell'ambiente di lavoro

6.2 Ambiente di lavoro di una DGXA100

Per utilizzare la DGX è stato necessario impostare l'ambiente[9] di lavoro generando una chiave pubblica da fornire al tecnico ISTI per la creazione dei requisiti di accesso. Una volta creato l'account è stato possibile accedervi tramite il comando ssh verso l'indirizzo dedicato della macchina. Un passo importante nella configurazione del proprio ambiente riguarda l'uso di Docker[10][37]; ciò necessita di un'"immagine" di partenza che contiene già pacchetti pre-installati in base al caso d'uso [11]. Verrà generata in base alla personalizzazione un nuova immagine mediante Dockerfile. Nel caso di questo progetto è stata adottata la seguente impostazione:

Listato 13: Dockerfile

```
1 FROM nvcr.io/nvidia/pytorch:23.02-py3
2 WORKDIR /raid/my-project
3 RUN apt-get update
4 RUN apt-get install sudo -y
5 RUN apt-get -y install python3.8-venv
6 RUN python3 -m venv yolov5-env
7 RUN source yolov5-env/bin/activate
8 RUN apt-get install libgl1 -y
```

Generata l'immagine, è stato poi necessario, secondo le istruzioni fornite dall'ISTI-CNR (come descritto sulla guida [38]), creare uno script eseguibile "drun.sh" e mandarlo in esecuzione secondo i seguenti criteri:

Listato 14: Dockerfile

```
1 #!/bin/bash
2
3 docker run \
4     --interactive --tty \
5     --rm \
6     --user $(id -u):$(id -g) \
7     --cpus 32 \
```

[9] Va osservato che i passaggi possono cambiare in base alla configurazione della macchina

[10] Docker è una piattaforma open-source che permette di creare, distribuire e eseguire applicazioni in ambienti isolati chiamati container, fornendo un'infrastruttura leggera e portatile per lo sviluppo e la gestione delle applicazioni.

[11] La maggior parte sono rilasciati direttamente sul sito Nvidia

```
8     --gpus '"device=0,2"' \
9     --volume $PWD:$PWD \
10    --workdir $PWD \
11    <username>/my-project:latest \
12    $@
```

Successivamente, mandando in esecuzione "drun.sh" viene avviata un istanza del container usando l'immagine costruita precedentemente.

6.3 FFmpeg

FFmpeg è un progetto open-source che consiste in librerie e programmi per la gestione di video, audio e flussi multimediali. E' molto usato per la trascodifica dei formati, editing di base, effetti video di post produzione nel rispetto degli standard.

Per l'installazione su Ubuntu Linux si utilizza il comando:

```
1  sudo apt install ffmpeg
```

La verifica della corretta installazione si può effettuare con il comando:

```
1  ffmpeg -version
```

Il flusso della telecamera IP può essere registrato in real-time collegandosi al dispositivo con i parametri adeguati di utente, password e indirizzo IP. L'opzione "copy" non decodifica lo stream ma lo copia direttamente su disco, ottenendo dei tempi di occupazione della CPU molto bassi. Il file ottenuto dipende dalla codifica effettuata dalla telecamera. Nel nostro caso il formato utilizzato è stato H265.

```
1  ffmpeg -rtsp_transport tcp -i "rtsp://admin:passwd@198
   ↪ .175.207.61:554/live" -c copy -map 0 video.mp4
```

Il seguente comando permette di estrapolare ogni singolo fotogramma da un video. Questa operazione ha permesso la creazione delle immagini da annotare per mezzo della piattaforma Roboflow (si veda il paragrafo 4.2.1).

```
1  ffmpeg -i inputvideo.mp4 -q:v 1 frame%05d.jpg
```

6.4 Sorgente - Progetto

In questo tirocinio tutto il codice è stato sviluppato utilizzando il linguaggio Python.

Listato 15: La funzione di calibrazione della posizione iniziale dei coni

```
1  def calibrationCamera(frame, cones, conesInF,debugLevel):
2      errorInFrame(conesInF)
3      for i, c, in enumerate(cones):
4          dim = len(conesInF[i+1])
5          if dim >= 2:
6              mindist = 10000
7              jmin = 10000
8              for j, det in enumerate(conesInF[i+1]):
9                  d2 = det.dist2(c)
10                 if debugLevel >= DEBUGLEVELHIG:
11                     print("\t\t"+str(det)+"␣at␣d2:␣"+str(d2))
12                 if d2<mindist:
13                     mindist = d2
14                     jmin = j
15             if debugLevel >= DEBUGLEVELHIG:
16                 print("\t\tMin␣distance:␣"+str(mindist)+"␣by␣Detect:␣"+str(conesInF[
                       ↪ i+1][jmin]))
17             if mindist < THESHOLD:
18                 mindet = conesInF[i+1].pop(jmin)
19                 conesInF[i+1] = []
20                 conesInF[i+1].append(mindet)
21                 dim = len(conesInF[i+1])
22             else:
23                 if debugLevel >= DEBUGLEVELMED:
24                     print(str(frame) + "\tCE\t␣Nessuna␣detection␣abbastanza␣vicina")
25             if debugLevel >= DEBUGLEVELHIG:
26                 print(str(frame)+"\t"+str(dim)+"\tCone␣" + str(c.Id) +"␣now␣has␣" +
                       ↪ str(dim)+"␣det")
27         if dim == 0:
28             print("Non␣ho␣trovato␣il␣cono␣" + str(c.Id))
29         elif dim == 1:
30             newDet = conesInF[i+1][0]
31             c.nextTLx *= c.countingCalib
32             c.nextTLy *= c.countingCalib
33             c.nextBRx *= c.countingCalib
34             c.nextBRy *= c.countingCalib
35
36             c.nextTLx += newDet.TLx
37             c.nextTLy += newDet.TLy
38             c.nextBRx += newDet.BRx
39             c.nextBRy += newDet.BRy
40             c.countingCalib += 1
41
```

```
42            c.nextTLx /= c.countingCalib
43            c.nextTLy /= c.countingCalib
44            c.nextBRx /= c.countingCalib
45            c.nextBRy /= c.countingCalib
```

Listato 16: La funzione per il rilevamento della penalità di cono toccato

```
1  function processDetecList(frame, cones, conesInFrame, moto, newPos, eventsList):
2              #frame e' il numero del frame
3              #cones e' la lista dei coni
4              #conesInFrame le detection che compaiono su ogni indice
5              #newPos detection che cadono sullo 0
6              #moto rappresenta la detection della moto
7              #EventList e' la struttura che contiene tutti gli errori che vengono
                   ↪ identificati
8
9              #Per prima cosa controlliamo il numero di detection che sono cadute
                   ↪ sullo 0
10             numdet = len(conesInFrame[0])
11             #Se sono maggiori di 0, allora ci sono delle detection fuori posto
12             if numdet>0:
13                    #STAMPO EVENTO
14                    #Controllo tutte le detection su indice 0
15                for d in conesInFrame[0]:
16                     #Verifico che la moto sia nella vicinanze
17                    if moto == None:
18                        #Non ho la moto nelle vicinanze
19                        nearmoto = False
20                    else:
21                        #Attraverso questo metodo posso controllare se la
                              ↪ detection si trova nell'area di perturbazione
                              ↪ della moto
22                        nearmoto = pointNearBbox(d.Cx,d.Cy,moto.TLx,moto.TLy,moto
                              ↪ .BRx,moto.BRy,MINDIST)
23                     #Se con una moto vicino lo segnalo, ma non tengo conto dell'
                           ↪ evento
24                    if nearmoto:
25                        #STAMPO EVENTO
26                    else:
27                        #Popoliamo la newPos list con le detection che si trovano
                              ↪  in posizioni nuove
28                        processZeroIndexDetection(d,newPos,debugLevel)
29
30             #Attraverso il metodo vengono verificate le detection su newPos
31              processNewPositionList(newPos,eventsList,debugLevel,StatusError,frame)
32              #Se non sono state gia' segnalate le detection qui vengono segnalate
                    ↪ per la prima volta
33              #Viene opportunamente verificato che queste detection non siano al di
                    ↪ fuori di una certa soglia
```

```
34
35              for i, c in enumerate(cones):
36
37                   nearmoto=False
38                   #Cicliamo la lista dei coni
39                   #Controlliamo se la moto si trova vicino
40                   if moto == None:
41                        nearmoto = False
42                   else:
43                        nearmoto = pointNearBbox(c.Cx,c.Cy,moto.TLx,moto.TLy,moto.BRx
                              ↪ ,moto.BRy,MINDIST)
44                   #Prendiamo il numero di detection che occorrono su quell'indice
45                   numdet = len(conesInFrame[i+1]) #num detection per cone
46
47
48                   #In questo primo caso affrontiamo il problema di due detection che
                         ↪  si sovrappongono
49                   #very often one is the same cone detected up and down - take the
                         ↪ one with bigger conf
50                   if numdet == 2:
51                     #Prendo la prima detection (A) e la seconda (B)
52                        detA = conesInFrame[i+1][0]
53                        detB = conesInFrame[i+1][1]
54
55                     #Calcoliamo la distanza
56                        d2a = detA.dist2(c) # d2a e' la distanza tra la detection A
                              ↪ ed il cono
57                        d2b = detB.dist2(c) # d2b e' la distanza tra la detection B
                              ↪ ed il cono
58                        d2ab = detA.dist2(detB) # d2ab e' la distanza tra la
                              ↪ detection A e la B
59                        #STAMPO EVENTO
60
61                         #Verifichiamo la distanza, se questa e' minore di una certa
                               ↪ soglia
62                         #Allora abbiamo due detection sovrapposte
63                        if d2ab <= MINDIST:
64                              #STAMPO EVENTO
65                              #Nel caso di sovrapposizione con la stessa classe
                                    ↪ andiamo a prendere la confidence maggiore
66                              if detA.Class == detB.Class:
67                                 #STAMPO EVENTO
68                                 if detA.Conf > detB.Conf:
69                                      conesInFrame[i+1].pop(1)
70                                 else:
71                                      conesInFrame[i+1].pop(0)
72                                #In base alla detection scelta rimuoviamo dalla
                                    ↪ conesInFrame quella scartata mediante pop
73                                #Questo avviene anche nelle casistiche sottostanti
```

```
74                          else: #classe diversa - la scelta dipende dallo stato (
                                ↪ dall'osservazione sperimentale)
75                              #Qui analizziamo il caso di un cono fermo fin dall'
                                    ↪ inizio.
76                               if c.Status == STATUSINIT and c.NextStatus ==
                                     ↪ STATUSRESET:
77                                  #STAMPO EVENTO
78                                  #Se la detection A e' in stato init(cono in
                                        ↪ piedi), scartiamo la detection B
79                                  if detA.Class == 0:
80                                      conesInFrame[i+1].pop(1)
81                                  else:
82                                      conesInFrame[i+1].pop(0)
83
84                              #In questo caso se il cono rappresenta lo stato ini
                                    ↪ , ma si trova su moved come nextstatus
85                              #Allora dobbiamo tenere conto che il cono
                                    ↪ probabilmente e' stato mosso
86                               if c.Status == STATUSINIT and c.NextStatus ==
                                     ↪ STATUSMOVED:
87                                  #STAMPO EVENTO
88                                  #Se la detection A e' classe 4 (cono caduto)
                                        ↪ allora, scarto la detection B
89                                  if detA.Class == 4:
90                                      conesInFrame[i+1].pop(1)
91                                  else:
92                                      conesInFrame[i+1].pop(0)
93
94                      else: #Altrimenti non abbiamo sovrappisizione - quindi
                            ↪ teniamo la detection con la distanza minore
95                          #STAMPO EVENTO
96                          # Se la detection A e' piu' vicina
97                          if d2a < d2b:
98                              #Se la detectio A rispetta la soglia, scarto la B
99                               if d2a < THESHOLD:
100                                  conesInFrame[i+1].pop(1)
101                         #Se la detection B e' piu' vicina
102                         else:
103                             #Se la detectio B rispetta la soglia, scarto la A
104                              if d2b < THESHOLD:
105                                  conesInFrame[i+1].pop(0)
106
107                     numdet = len(conesInFrame[i+1]) #Ricalcolo il numero delle
                            ↪ detection dopo aver eventualmente rimosso una della
                            ↪ due
108                     #STAMPO EVENTO
109
110
111                 #Piu' di due detection (Caso molto raro), scegliamo la detection
```

```
                    ↪ piu' vicina se viene rispettata una certa soglia"
112                 if numdet>2:
113                     #STAMPO EVENTO
114                     #Troviamo la distanza minima
115                     mindist = 10000
116                     jmin = 10000
117                     for j, det in enumerate(conesInFrame[i+1]):
118                        d2 = det.dist2(c)
119                        #STAMPO EVENTO
120                        if d2<mindist:
121                            mindist = d2
122                            jmin = j
123                     #STAMPO EVENTO
124                      #Se la distanza minima trovata rispetta la massima soglia
                            ↪ consentita
125                     if mindist < THESHOLD:
126                         mindet = conesInFrame[i+1].pop(jmin) #Allora prendiamo la
                                ↪  detection che corrisponde jmin
127                         conesInFrame[i+1] = [] #Svuotiamo la lista di detection
                                ↪ che si trova all'indice i+1
128                         conesInFrame[i+1].append(mindet) #Salviamo solamente
                                ↪ sulla lista la detection mindet
129                         numdet = len(conesInFrame[i+1]) #Ricalcoliamo il numero
                                ↪ di detection all'indice i+1
130                     #STAMPO EVENTO
131
132
133                 # Nel caso di una sola detection all'indice i+1
134                 if numdet == 1:
135                     #Ci salviamo il frame in cui il cono e' stato visto l'ultima
                            ↪ volta
136                     c.LastSeen = frame
137                     #Prendiamo la detection
138                     det = conesInFrame[i+1][0]
139                     #Calcoliamo la distanza della detection dal cono
140                     d2 = det.dist2(c) # distance^2 from init pos
141                     #dist=conesInFrame[i+1][0].similar(c)
142                     #Seganlo se la classe e' cono caduto
143                     if det.Class == 4:
144                         #STAMPO EVENTO
145
146                     #Controlliamo la distanza, se questa e' maggiore della soglia
                            ↪  consentita allora il cono molto probabilemente e'
                            ↪ spostato
147                     if d2>THESHOLD:
148                         #STAMPO EVENTO
149                         if det.Class == 4:
150                             #Se il cono e' classe cono caduto
151                             #STAMPO EVENTO
```

```
152
153
154                         #Controlliamo se e' vicina la moto
155                         if nearmoto and det.Class == 0:
156                              #STAMPO EVENTO
157                              #Se la detection e' classe cono in piedi
158                              #Resettiamo la timewindow
159                              c.FrameChange = frame
160                              if c.TimeWindow > 0:
161                                   c.TimeWindow = 0
162                                   #STAMPO EVENTO
163
164                         else:
165                             #Se la moto si trova lontano
166                             #Nel caso in cui la detection ha classe cono caduto e
                                    ↪  il cono classe not con caduto
167                              if det.Class == 4 and c.Class!=4:
168                                   c.FrameChange = frame #Salvo il frame in cui e'
                                          ↪ avvenuto questo cambiamento sul cono
169                                   c.Class = det.Class #Salvo la classe della
                                          ↪ detection sul cono
170                                   c.Conf = det.Conf #Salvo la confidence
171
172                              #Se la classe e' la stessa
173                              if det.Class == c.Class:
174                                   if c.Conf < det.Conf: #Prendiamo la conf
                                          ↪ maggiore
175                                        c.Conf = det.Conf
176
177                              #Se lo stato del cono e' diverso da cono spostato
178                              if c.Status != STATUSMOVED:
179                                   #Se timewindow e' ancora a 0, allora non e'
                                          ↪ avvenuto prima di adesso alcun
                                          ↪ cambiamento sul cono
180                                   #Questo e' il primo caso
181                                   #Quindi settiamo il frameChange (il frame in cui
                                          ↪  e' iniziato il cambiamento)
182                                   #E settiamo il prossimo stato come moved
183                                   #Il cono continua ad avere lo stato precedente
184                                   if c.TimeWindow == 0:
185                                        c.FrameChange = frame
186                                        c.NextStatus = STATUSMOVED
187                                   #Portiamo avanti il counter del timewindow
188                                   c.TimeWindow += 1
189                                   #STAMPO EVENTO
190                                   #Controlliamo il counter del timeWindow
191                                   if c.TimeWindow >= FRAMEWIN:
192                                        #Se questo ha superato il framewin(soglia)
                                               ↪ allora settiamo il nuovo stato al
```

```
                                    ↪ cono
193                                 #Inoltre segnaliamo l'evento
194                                 #Questo p il caso in cui si presenta un
                                        ↪ errore
195                                 delta = frame - c.FrameChange #Ricaviamo il
                                        ↪  frame in cui l'evento e' iniziato
196                                 rate = c.TimeWindow / delta
197                                 #STAMPO EVENTO
198                                 #Controlliamo il rate di detection
199                                 if rate > RATEDETECTION:
200                                     #Superata una certa soglia, allora
                                            ↪ segnaliamo l'evento del cono
                                            ↪ spostato
201                                     #STAMPO EVENTO
202                                     c.Status = STATUSMOVED # Il nuovo stato
                                            ↪  diventa spostato
203                                     c.NextStatus = STATUSRESET # Il
                                            ↪ prossimo stato viene settato sul
                                            ↪  reset
204                                     c.TimeWindow = 0 # La timeWindows viene
                                            ↪  resettata
205                                     #Viene creato un oggetto evento, che
                                            ↪ contiene tutte le informazioni
                                            ↪ essenziali che riguardano l'
                                            ↪ errore
206                                     eventObj = EventClass.Event(c.
                                            ↪ FrameChange,c.Id,c.Status,det.
                                            ↪ z0x,det.z0y,c.Conf)
207                                     #Salvo l'evento nella eventList
208                                     eventsList.append(eventObj)
209                                     #STAMPO EVENTO
210                             else: #status = MOVED
211                                 #L'errore e' gia' stato aggiunto alla eventList
212                                 #STAMPO EVENTO
213
214
215                         else: #Altrimenti la detection e' entro la soglia consentita
216                             #Salviamo la classe e la conf al cono corrispondente
217                             c.Class = det.Class
218                             c.Conf = det.Conf
219                             #STAMPO EVENTO
220                             #Se la distanza tra la detection e il cono equivale ad un
                                    ↪  pixel
221                             #Allora salviamo le cordinate della detection come nuove
                                    ↪ cordinate del cono
222                             #Cosi da corregere piccole imperferzioni
223                             if d2==1:
224                                 c.TLx = det.TLx
225                                 c.TLy = det.TLy
```

```
226                         c.BRx = det.BRx
227                         c.BRy = det.BRy
228                         c.center()
229
230                     #Se lo stato del cono e' diverso da init(in piedi)
231                     if c.Status != STATUSINIT:
232                         #STAMPO EVENTO
233                         #Salvo il frame come frameChange
234                         c.FrameChange = frame
235                         c.Status = STATUSINIT #Faccio tornare lo stato del
                                ↪ cono ad init
236                         c.NextStatus = STATUSRESET #Il next status viene
                                ↪ resettato
237                         c.TimeWindow = 0 #Resetto la timeWindow
238                         #Viene creato un oggetto evento, che contiene tutte
                                ↪ le informazioni essenziali che riguardano l'
                                ↪ errore
239                         eventObj = EventClass.Event(c.FrameChange,c.Id,c.
                                ↪ Status,det.z0x,det.z0y,c.Conf)
240                         #Salvo l'evento nella eventList
241                         eventsList.append(eventObj)
242
243                     else:
244                         #Con status init
245                         #Se la distanza e' minore della soglia
246                         if d2<=MINDIST: #Resetto lo stato e la timewindow
247                             c.NextStatus = STATUSRESET
248                             c.TimeWindow = 0
249
250                         else: #Altrimenti se la detection e' tra la mindist
                                ↪ e la framwin
251                             #Abbasso il counter pian piano
252                             if c.TimeWindow > 0:
253                                 c.TimeWindow -=1
254
255                 #Se non ci sono detection sull'indice i+1 in conesInFrame
256                 if numdet == 0:
257                     #Se lo stato non e' stato INIT forse manca effettivamente il
                            ↪ cono nel file conelist
258                     #Dobbiamo tenere conto che il cono sia scomparso
259                     if c.Status != STATUSRESET:
260                         #STAMPO EVENTO
261                         #Controllo se c'e' vicina la moto
262                         if nearmoto:
263                             #STAMPO EVENTO
264                             #Resetto la timewindow
265                             if c.TimeWindow>0:
266                                 c.TimeWindow = 0
267                                 print("\tR\tResetting␣Timewindow")
```

```
268                             #Setto a +1 il counter occluded
269                             c.Occluded +=1
270                         else:
271                             #Se la moto non e' in mezzo
272                             #Con stato diverso da stato perso
273                             if c.Status != STATUSLOST:
274                                 #Settiamo il nuovo stato se siamo alla prima
                                        ↪ iterazione di timeWindow
275                                 if c.TimeWindow == 0:
276                                     c.NextStatus = STATUSLOST
277                                 #Setto a +1 il counter timeWindow
278                                 c.TimeWindow += 1
279                                 #STAMPO EVENTO
280                                 #Se la soglia frameWin e' stata raggiunta o
                                        ↪ superata
281                                 if c.TimeWindow >= FRAMEWIN:
282                                     #Risaliamo al frame in cui l'evento e'
                                            ↪ inziato
283                                     #Considerando anche il numero di frame in
                                            ↪ cui il cono era occluso dalla moto
284                                     delta = frame - c.LastSeen - c.Occluded #
                                            ↪ puo' diventare negativo?
285                                     if delta<0: # prudenza
286                                         delta=1
287                                     rate = c.TimeWindow / delta
288                                     #STAMPO EVENTO
289                                     #Se il tasso di errori sulla detection e'
                                            ↪ oltra la soglia
290                                     if rate > RATEDETECTION:
291
292                                         #STAMPO EVENTO
293                                         c.Status = STATUSLOST #Setto lo stato
                                                ↪ su perso
294                                         c.NextStatus = STATUSRESET #Resetto il
                                                ↪ next status
295                                         c.TimeWindow = 0 #Resetto timeWindow
296                                         #Viene creato un oggetto evento, che
                                                ↪ contiene tutte le informazioni
                                                ↪ essenziali che riguardano l'
                                                ↪ errore
297                                         eventObj = EventClass.Event(c.LastSeen,
                                                ↪ c.Id,c.Status,c.Cx,c.BRy,c.Conf)
298                                         #Salvo l'evento nella eventList
299                                         eventsList.append(eventObj)
300                                         #STAMPO EVENTO
301                             else: #status = LOST
302                                 #Il cono e' stato gia' segnalato su perso
303                                 #STAMPO EVENTO
304                                 #Trovo possibili coni nuovi vicino a quello
```

```
                                                    ↪ perso
305                                         findNearPos(c,newPos,debugLevel)
```

Bibliografia

[1] G. R. Leone, M. Righi, D. Moroni, and F. Paolucci, "Towards multi-camera system for the evaluation of motorcycle driving test," in *2022 16th International Conference on Signal-Image Technology and Internet-Based Systems (SITIS)*, pp. 561–568, 2022.

[2] Ministero dei Trasporti, "Decreto 26/09/2018 - Nuova disciplina delle prove di valutazione delle capacità e dei comportamenti per il conseguimento delle patenti di guida delle categorie A1, A2 e A." https://www.gazzettaufficiale.it/eli/id/2018/10/12/18A06493/sg. Last Access: 7-7-2023.

[3] Mayank Mishra, "Convolutional Neural Networks, Explained." https://towardsdatascience.com/convolutional-neural-networks-explained-9cc5188c4939. Last Access: 10-7-2023.

[4] J. Raitoharju, "Chapter 3 - convolutional neural networks," in *Deep Learning for Robot Perception and Cognition* (A. Iosifidis and A. Tefas, eds.), pp. 35–69, Academic Press, 2022.

[5] Afshine Amidi and Shervine Amidi, "Convolutional neural networks cheatsheet." https://stanford.edu/~shervine/teaching/cs-230/cheatsheet-convolutional-neural-networks. Lasta Access: 10-7-2023.

[6] MathWorks, "Getting started with r-cnn." https://it.mathworks.com/help/vision/ug/getting-started-with-r-cnn-fast-r-cnn-and-faster-r-cnn.html. Last Access: 10-7-2023.

[7] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You Only Look Once: Unified, Real-Time Object Detection," 2016.

[8] J. Terven and D. Cordova-Esparza, "A Comprehensive Review of YOLO: From YOLOv1 and Beyond," 2023.

[9] Ultralytics, "YOLOv8 GitHub Repository." https://github.com/ultralytics/ultralytics. Last Access: 10-7-2023.

[10] "YOLOv7 repositry GitHub." https://github.com/WongKinYiu/yolov7. Implementation of paper - YOLOv7: Trainable bag-of-freebies sets new state-of-the-art for real-time object detectors, Last Access: 10-7-2023.

[11] C.-Y. Wang, H.-Y. M. Liao, and I.-H. Yeh, "Designing Network Design Strategies Through Gradient Path Analysis," *arXiv preprint arXiv:2211.04800*, 2022.

[12] C.-Y. Wang, A. Bochkovskiy, and H.-Y. M. Liao, "YOLOv7: Trainable bag-of-freebies sets new state-of-the-art for real-time object detectors," *arXiv preprint arXiv:2207.02696*, 2022.

[13] Glenn Jocher and Sergiu Waxmann, "Comprehensive guide to ultralytics yolov5." https://docs.ultralytics.com/yolov5/. Created: 29-3-2023, Updated: 4-7-2023, Last Access: 10-7-2023.

[14] G. Jocher and S. Waxmann, "Ultralytics YOLOv5 Architecture." https://docs.ultralytics.com/yolov5/tutorials/architecture_description/. Created: 21-04-2023, Updated: 15-06-2023, Last Access: 10-7-2023.

[15] Ultralytics, "Yolov5 github repository." https://github.com/ultralytics/yolov5. Last Access: 10-7-2023.

[16] OpenGenus, "Yolo v5 model architecture." https://iq.opengenus.org/yolov5/. Last Access: 10-7-2023.

[17] Jacob Solawetz, "Yolov5 roboflow guide." https://blog.roboflow.com/yolov5-improvements-and-evaluation/. Last Access: 10-7-2023.

[18] H. Wang, F. Zhang, and L. Wang, "Fruit Classification Model Based on Improved Darknet53 Convolutional Neural Network," in *2020 International Conference on Intelligent Transportation, Big Data & Smart City (ICITBS)*, pp. 881–884, 2020.

[19] J. Redmon and A. Farhadi, "YOLO9000: Better, Faster, Stronger," 2016.

[20] J. Redmon and A. Farhadi, "YOLOv3: An Incremental Improvement," 2018.

[21] Roboflow Inc., "Roboflow annotation framework." http://www.roboflow.com/. Last Access: 10-7-2023.

[22] A. Ng, *Machine Learning Yearning*. Online Draft, 2017.

[23] P. A. Flach, *Machine learning: The art and science of algorithms that make sense of data*. Cambridge University Press, 2017.

[24] Javatpoint, "Regularization in machine learning." https://www.javatpoint.com/regularization-in-machine-learning. Last Access: 10-7-2023.

[25] Prashant Gupta, "Regularization in machine learning." https://towardsdatascience.com/regularization-in-machine-learning-76441ddcf99a. Created: 15-11-2017, Last Access: 10-7-2023.

[26] Jacob Solawetz, "What is mean average precision (map) in object detection?." https://blog.roboflow.com/mean-average-precision/. Created: 6-5-2020, Last Access: 10-7-2023.

[27] D. K. Sharma, M. Chatterjee, G. Kaur, and S. Vavilala, "3 - deep learning applications for disease diagnosis," in *Deep Learning for Medical Applications with Unique Data* (D. Gupta, U. Kose, A. Khanna, and V. E. Balas, eds.), pp. 31–51, Academic Press, 2022.

[28] OpenCV, "Opencv about." https://opencv.org/about/. Last Access: 10-7-2023.

[29] OpenCV Documentation, "Opencv camera calibration." https://docs.opencv.org/4.x/dc/dbb/tutorial_py_calibration.html. Last Access: 10-7-2023.

[30] Y.-J. Zhang, *Camera Calibration*, pp. 37–65. Singapore: Springer Nature Singapore, 2023.

[31] Y. M. Wang, Y. Li, and J. B. Zheng, "A camera calibration technique based on OpenCV," in *The 3rd International Conference on Information Sciences and Interaction Sciences*, pp. 403–406, 2010.

[32] Z. Zhang, "A flexible new technique for camera calibration," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 22, pp. 1330–1334, Nov 2000.

[33] DahuaSecurity, "Bullet camera ipc-hfw5541e-ze." https://www.dahuasecurity.com/it/products/All-Products/Discontinued-Products/Network-Cameras/WizMind-Series/IPC-HFW5541E-ZE. Last Access: 11-7-2023.

[34] Nvidia, "Nvidia integrated systems with Jetson." https://www.nvidia.com/it-it/autonomous-machines/embedded-systems/. Last Access: 10-7-2023.

[35] Nvidia, "CUDA on WSL User Guide." https://docs.nvidia.com/cuda/wsl-user-guide/index.html#getting-started-with-cuda-on-wsl-2. Last Access: 10-7-2023.

[36] ClearML, "ClearML Official Site." https://app.clear.ml/. Last Access: 10-7-2023.

[37] Docker, "Docker guide - get started." https://docs.docker.com/get-started/. Last Access: 10-7-2023.

[38] ISTI-CNR, "Nvidia dgx a100." https://mediawiki-s2i2s.isti.cnr.it/wiki/AIE:Guida_Utenti#Accesso_e_Utilizzo_NVIDIA_DGX_A100. Last Access: 10-7-2023.