Università degli Studi di Padova

Dipartimento di Matematica "Tullio Levi-Civita"

Corso di Laurea Magistrale in Informatica

Tracciamento e predizione di traiettorie umane su dati di profondità

Tesi di laurea magistrale

Relatore
Prof. Lamberto Ballan
Co-relatori
Dott. Pasquale Coscia
Dott. Luigi Filippo Chiara

Laureando
Marco Costantino
1234239

Anno Accademico 2021-2022

La filosofia è scritta in questo grandissimo libro che continuamente ci sta aperto innanzi a gli occhi (io dico l'universo), ma non si può intendere se prima non s'imparara a intender la lingua, e conoscer i caratteri, ne' quali è scritto. Egli è scritto in lingua matematica, e i caratteri son triangoli, cerchi, ed altre figure geometriche, senza i quali mezi è impossibile a intenderne umanamente parola; senza questi è aggirarsi vanamente per un oscuro laberinto.

— Galileo Galilei, Il Saggiatore

Ringraziamenti

Innanzitutto, vorrei ringraziare la mia famiglia per avermi dato la possibilità di frequentare l'università e per avermi sostenuto durante questi anni di studio.

Desidero poi ringraziare il Prof. Lamberto Ballan, Luigi Filippo Chiara e Pasquale Coscia del gruppo di ricerca VIMP per avermi dato l'opportunità di lavorare a questa tesi dalla quale ho imparato molto.

Padova, Aprile 2022 Marco Costantino

Abstract

Riconoscere, tracciare e predire il movimento di persone in ambienti indoor ed analizzare la loro interazione con lo spazio circostante rappresentano task fondamentali della visione artificiale. Se da un lato tali ambienti presentano caratteristiche favorevoli per l'applicazione di tecniche di riconoscimento e predizione, come illuminazione e copertura totale degli spazi, dall'altro la grande quantità di oggetti con cui le persone interagiscono e l'ambiguità dei loro movimenti rappresentano sfide ancora non completamente risolte. Diverse tecnologie basate sull'apprendimento automatico hanno permesso di identificare le persone monitorate e capire come interagiscono con la scena, spesso con l'aiuto di esperti, sfruttando principalmente camere di profondità (RGB-D) e monitorando la scena dall'alto. Con la grande disponibilità di dati, tuttavia, questo paradigma si e' evoluto utilizzando approcci basati sul deep learning, che hanno consentito di trattare efficacemente la complessità delle dinamiche delle persone. Questa tesi ha l'obiettivo di dimostrare le potenzialità di tali tecnologie in un ambito applicativo reale, quale quello offerto dai magazzini della catena di supermercati Alí, partendo dalla raccolta dei dati, per poi passare all'impiego di tecnologie allo stato dell'arte con lo scopo di tracciare il movimento e predire le traiettorie di operatori durante lo svolgimento delle loro mansioni. Nella prima parte verrà presentato il setting sperimentale messo a punto per la raccolta dei dati; successivamente, vengono proposte reti neurali per il tracciamento di persone e carrelli, che rappresentano i principali elementi coinvolti nella scena. Infine vengono comparate reti ricorrenti e basate su attenzione per prevedere il loro movimento futuro. I risultati ottenuti dimostrano che l'impiego di una pipeline basata su reti convoluzionali ed i recenti meccanismi di attenzione permettono di soddisfare i requisiti richiesti dall'ambito applicativo considerato.

Indice

Elenco delle figure

Elenco delle tabelle

Capitolo 1

Introduzione

1.1 Il contesto

Nell'ambito della *computer vision*, la predizione delle traiettorie di soggetti in movimento (trajectory prediction) in flussi video è importante per innumerevoli applicazioni. Un esempio applicativo è l'utilizzo in automobili a guida autonoma che devono considerare la traiettoria degli altri soggetti nella scena per evitare collisioni. In generale, la trajectory prediction è necessaria per lo sviluppo di agenti intelligenti autonomi, quali ad esempio robot e droni, che hanno il compito di spostarsi in ambienti dinamici in cui la capacità di prevedere la traiettoria di altri soggetti in movimento è necessaria per evitare collisioni. Le applicazioni tuttavia non sono limitate all'ambito degli agenti autonomi: la predizione della traiettoria può essere usata per studiare il flusso di soggetti in una scena.

Prerequisito alla predizione delle traiettorie è il tracciamento dei soggetti nel video (object tracking). Esiste una multitudine di approcci al tracciamento di multipli soggetti in una scena. L'approccio tracking-by-detection può essere compreso come un processo che si compie in 2 fasi: la prima è quella di riconoscimento dei soggetti nella scena (object detection). Più nello specifico, si tratta di capire quali porzioni dell'immagine rappresentano i soggetti che intendiamo tracciare. La seconda è quella di associazione di un identità ai soggetti individuati: dati 2 frame successivi presi da un flusso video è necessario infatti determinare quali parti dei 2 frame rappresentano lo stesso soggetto e quindi associare un identificativo univoco ad ogni soggetto in modo da poter considerare la differenza in posizione, quindi anche velocità e direzione, tra un frame e l'altro. Il tracciamento dei soggetti nella scena è necessario per la predizione delle traiettorie perché questa consiste nel, data una traccia parziale, restituire delle possibili traiettorie, in termini di punti nello spazio, che i soggetti possono verosimilmente seguire.

Da circa un decennio il mondo della computer vision è stato rivoluzionato dai modelli basati sulle reti neurali. Lo stato dell'arte per quanto riguarda il tracciamento e la predizione di traiettorie non fa eccezione: metodi come YOLO, R-CNN vengono usati per fare object detection mentre metodi come Track-RCNN, ROLO sono usati per fare object tracking e infine modelli come Ynet, SocialLSTM vengono usati per fare trajectory prediction. Tutti questi modelli si basano sulle reti neurali e

sono solo alcuni del gran numero di modelli sviluppati e in competizione su *dataset* usati per fare benchmarking come ad esempio COCO per l'object detection, i dataset della MOTChallenge per l'object tracking e TrajNet++ per la trajectory prediction.

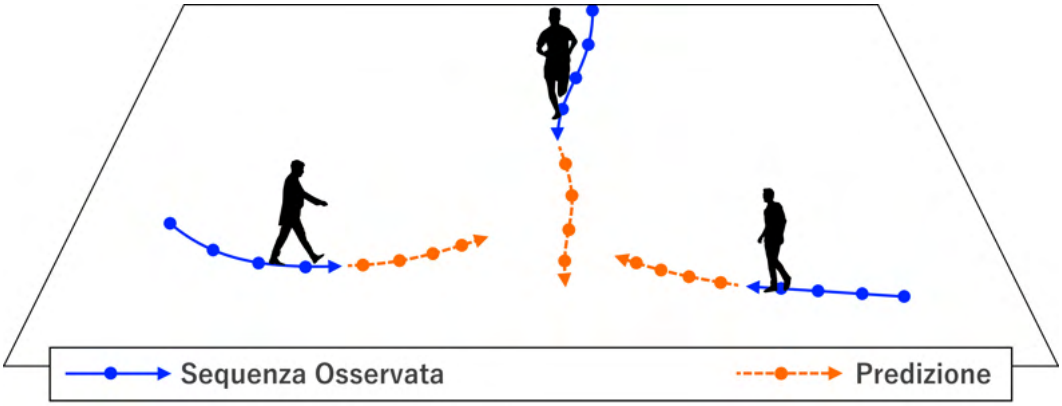

Figura 1.1: Diagramma che schematizza la predizione di traiettorie: la sequenza osservata è il risultato dell'object tracking, la predizione è il risultato della trajectory prediction.

1.2 Il progetto

Il progetto specifico al quale questa tesi contribuisce è in collaborazione con Alí Supermercati e si pone l'obiettivo di dimostrare le potenzialità di queste tecnologie applicandole a video raccolti all'interno di uno dei magazzini di Alí. In particolare, la tesi affronta tre fasi distinte del progetto:

1. Raccolta dati;
2. Object Tracking;
3. Trajectory Prediction.

I dati raccolti consistono in video ripresi dall'alto degli operatori di Alí mentre svolgono il loro lavoro, raccogliendo merce da degli scaffali e caricandola su dei carrelli. L'estensione della scena ripresa e il tipo di dati raccolti sono sufficienti per un *proof of concept* delle tecnologie menzionate. Il mio lavoro in particolare si concentra sulle prime due fasi del progetto, ovvero dalla raccolta e analisi dei dati al tracciamento dei target, persone e carrelli, nei dati raccolti. Infine, per quanto riguarda l'ultima fase, quella di predizione delle traiettorie, produco dei risultati preliminari utili a determinare un punto di partenza per lavori futuri.

1.2.1 Obiettivi

Ad ogni fase sono associati degli obiettivi che sono utili e necessari alle fasi successive:

- Per la prima fase, quella di raccolta dati, l'obiettivo principale è la produzione di video utilizzabili ai fini di tracciare i target presenti e successivamente predirne le traiettorie;
- Per la seconda fase l'obiettivo principale è quello di produrre le tracce dei target nei video, ovvero produrre i dati sui quali i modelli di trajectory prediction possono essere applicati;
- L'obiettivo della fase di predizione delle traiettorie è quello di produrre dei risultati preliminari utili come punto di partenza per analisi future.

1.2.2 Vincoli

Mentre applicazioni di queste tecnologie in contesti menzionati in precedenza, come ad esempio in agenti autonomi, hanno come principale vincolo quello dell'elaborazione delle informazioni in real-time, per permettere all'agente autonomo di determinare la propria traiettoria, nel caso di questo progetto l'obiettivo è quello di dare una dimostrazione delle potenzialità di questi metodi, quindi non si ha alcun vincolo temporale.

Il diritto alla privacy

Il vincolo principale affrontato in questo progetto riguarda le normative sulla privacy. Il GDPR, una serie di normative a livello europeo, ha il compito di proteggere i dati personali dell'individuo tra cui immagini e video in cui l'individuo è identificabile[6]. Questo significa che, nell'ambito del progetto, raccogliere video in cui gli operatori sono riconoscibili implicherebbe una serie di passaggi e vincoli: dalla raccolta del consenso all'uso dei dati personali a limiti e vincoli sull'uso e il trattamento dei dati.

Per rispettare il diritto alla privacy dei lavoratori ed aderire al GDPR, la soluzione scelta e concordata con Alí Supermercati è quella di raccogliere video in cui non sia possibile identificare le persone presenti. Perciò i video vengono raccolti usando una telecamera di profondità che produce immagini che codificano, a bassa risoluzione, la distanza dei punti nella scena dalla telecamera. L'immagine prodotta non permette di identificare le persone presenti garantendo il diritto alla privacy degli operatori.

Modalità di trattamento dei dati raccolti

In accordo con Alí Supermercati, i dati vengono trattati ai soli fini della ricerca scientifica, pertanto saranno conservati per il tempo necessario alla ricerca e non oltre i 12 mesi dalla conclusione del progetto. Infine, è possibile che analisi, esperimenti e risultati ottenuti a partire dai dati siano pubblicati in articoli scientifici.

1.2.3 Tecnologie e metodologie impiegate

Linguaggio di programmazione e ambiente di sviluppo

Poiché il progetto tratta temi di computer vision, il linguaggio di programmazione scelto è Python che è uno dei linguaggi più usati dalla ricerca e dall'industria nell'ambito del machine learning grazie all'ampio ecosistema di librerie, framework e implementazioni pubbliche che offre. Tra le librerie più importanti che ho usato ci sono:

- OpenCV: una libreria pensata per la computer vision che permette in generale di trattare immagini e video;
- DarkNet: una libreria che permette in generale di lavorare con reti neurali, io in particolare ho usato l'implementazione di YOLOv4 offerta;
- PyTorch, una libreria che offre implementazioni di modelli di machine learning, io in particolare ho usato TorchVision e in particolare l'implementazione di YOLOv5 offerta.

L'ambiente di sviluppo che ho usato per la maggior parte del lavoro è stato PyCharm Community Edition. L'IDE offre svariati strumenti tra cui tool per il *linting* e il *debugging*, rendendolo degli IDE più usati per Python.

Per manipolare i dataset sul disco ho usato principalmente script powershell.

Ulteriori strumenti e hardware

Il training di modelli basati sulle reti neurali consiste nel risolvere un problema di ottimizzazione: in generale, si cercano i *parametri* della rete tali che l'errore presente tra l'output dato dalla rete e quello atteso è minimizzato. In generale, per trovare i valori ottimali per questi parametri si impiega: un metodo di *discesa del gradiente* e, per aggiornare i pesi, l'algoritmo di *backpropagation*. Quest'ultimo fa moltiplicazioni di matrici per calcolare il gradiente della loss function che verrà poi usato per aggiornare i pesi. La moltiplicazione di matrici è un operazione fondamentale nella computer grafica, infatti le moderne *GPU* sono specializzate nell'esecuzione di questa operazione, il che le rende la piattaforma hardware più efficiente sulla quale fare il training di modelli basati sulle reti neurali.

Al fine di allenare i modelli ho usato: *Google Colab*, in una breve fase di test, e le macchine dedicate ai tesisti del dipartimento di matematica, presso Torre Archimede, alle quali ho potuto accedere semplicemente tramite protocollo *SSH*.

Documentazione

Per documentare il codice ho deciso di usare un metodo che già conoscevo dalla mia precedente esperienza di stage; ho usato docstring opportunamente formattate per l'integrazione con il linter per l'analisi statica del codice, e per l'integrazione con Sphinx: un tool, originariamente usato per la documentazione ufficiale di Python, che

produce automaticamente documentazione in formato html a partire dalle docstring nel codice. Un esempio di docstring per documentare un metodo:

```
"""
Brief method description

In depth method description

Args:
    arg1 (type):
        argument description

Returns:
    type:
        return description

Raises:
    list of raised exceptions
"""
```

Considerata la durata e l'estensione del progetto ritengo che lo sforzo di documentazione del codice sia stato utile e che le docstring prodotte siano adeguate a descrivere il funzionamento del codice.

Versioning

Per il versionamento del codice ho usato git e github, tecnologie che già conoscevo e che sono uno standard dell'industria.

Principi di programmazione

Il codice che ho prodotto non implementa una particolare architettura, si tratta per lo più di script che fanno uso di metodi ricorrenti definiti da me o importati da librerie per elaborare i video. Nonostante ciò vi sono comunque dei principi di buona programmazione da seguire per garantire codice di qualità:

- Single Responsability Principle: sebbene questo principio sia comunemente applicato alle classi, il senso del principio è più generale e può essere applicato a funzioni e metodi. Funzioni con un unica responsabilità, ovvero un unico scopo e risultato aiutano a produrre codice riutilizzabile e modulare, inoltre questo aiuta a limitare la dimensione delle funzioni rendendo il codice più comprensibile;

- KISS, keep it simple, stupid: sebbene formalmente il principio fa riferimento alla minimizzazione dell'accoppiamento tra moduli in favore della coesione, anche questo principio ha un carattere generale che è utile applicare: evitare di scrivere codice complesso per risparmiare righe di codice o generalizzare dove non necessario. Codice più semplice aiuta a fare debugging ed è più comprensibile. Quando non è possibile evitare la complessità è importante fornire commenti che aiutino la lettura del codice;

- DRY, don't repeat yourself: l'idea di questo principio è quella di evitare di avere ripetizioni nel codice. Per ottenere ciò è necessario codificare come funzioni o metodi i comportamenti ricorrenti. L'aderenza a questo principio facilita il debugging, la modifica e la verifica del codice;

Metodologia

L'analisi di dati di tipo visuale come immagini e video implica che i risultati abbiano un aspetto visuale. Sebbene sia facile verificare la correttezza di questi risultati solo osservando le visualizzazioni, la verità è che queste non danno il totale dell'informazione necessaria a verificarne la correttezza oltre a non fornire i mezzi necessari per paragonare metodi in competizione. Per assicurarmi quindi che gli obiettivi di ogni fase del progetto vengano raggiunti, faccio uso di 2 approcci che ritengo essere complementari: uno qualitativo, in cui, dove possibile, l'output viene visualizzato e semplicemente osservato per darne un giudizio, l'altro quantitativo in cui calcolo delle metriche numeriche che danno oggettività al giudizio dei risultati. Il primo approccio è utile specialmente nelle prime fasi dei lavori, in cui osservare le visualizzazioni da modo di capire intuitivamente il livello dell'output e possibili ragioni per cui è o non è adeguato. Il secondo approccio invece permette di migliorare e comparare obiettivamente risultati e metodi in competizione.

Capitolo 2

La raccolta dei dati

2.1 Obiettivi

Gli obiettivi principali della fase di raccolta dei dati sono:

1. Raccogliere dati su cui poter applicare modelli per il tracciamento e la predizione delle traiettorie;
2. Raccogliere abbastanza dati da poter allenare i modelli;
3. Rispettare le normative sulla privacy, quindi raccogliere dati dai quali non è possibile identificare le persone presenti.

2.2 Raccolta dei dati e tecnologie

Per mantenere l'anonimato degli operatori di Alí Supermercati, facciamo uso di una telecamera di profondità. Questo tipo di telecamere restituisce immagini in cui ogni pixel ha un valore legato alla distanza tra il punto rappresentato e la telecamera stessa. Il dispositivo specifico in uso è una telecamera Intel RealSense D455.

Figura 2.1: Intel RealSense D455, la telecamera usata per raccogliere i dati

Da sinistra a destra, i sensori integrati nella telecamera sono:

1. Primo sensore stereoscopico IR;
2. Sensore RGB in luce visibile;
3. Proiettore IR;
4. Secondo sensore stereoscopico IR.

I sensori stereoscopici IR utilizzano luce negli infrarossi e tecniche di stereoscopia per determinare la distanza dei punti nell'immagine dal sensore. Il proiettore IR proietta un pattern negli infrarossi sulla scena. Il pattern non è visibile all'occhio umano ma viene utilizzato internamente dalla telecamera a supporto del calcolo delle distanze. Sebbene il proiettore non sia necessario per questi calcoli, ne facciamo uso in quanto migliora la qualità dei dati, specialmente nei punti meno illuminati dell'immagine, infatti, il proiettore è pensato per permettere di usare la telecamera in condizioni di scarsa illuminazione.

La telecamera è posizionata a circa 7 m dal suolo e punta direttamente verso il basso. Inquadra un'area di circa 25 m^2 che consiste di una corsia tra 2 scaffali adoperata dagli operatori di Alí. Il supporto su cui è montata serve al cablaggio del magazzino ed è usato per fissare anche i cavi dati e di alimentazione della telecamera. Questi cavi scendono a livello del suolo su una corsia non utilizzata dagli operatori, nella quale è possibile il collegamento alla telecamera, tramite USB, e la raccolta dei dati.

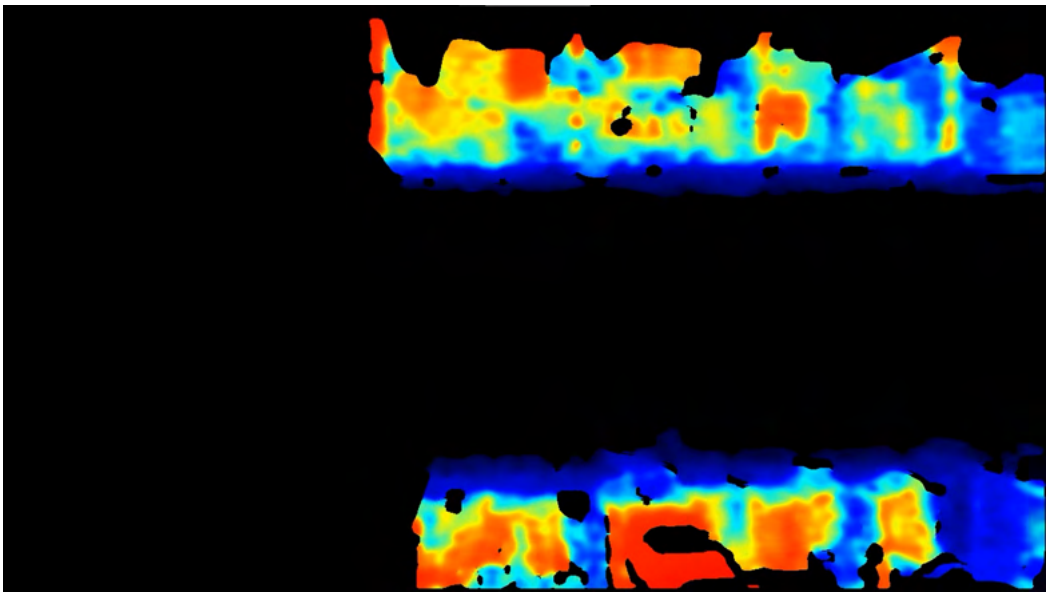

Figura 2.2: Immagine catturata dalla telecamera. La scena è vuota, si possono vedere solo gli scaffali ad ambo i lati della corsia in cui gli operatori passano. Lo spettro dei colori va dal rosso al blu al crescere della distanza dalla telecamera.

I prodotti della linea RealSense di Intel mettono a disposizione un SDK open source per facilitare lo sviluppo di applicativi. Per la raccolta dei dati, Intel mette a disposizione il software Intel RealSense che offre una *GUI* per calibrare la telecamera e raccogliere i dati. La telecamera è in grado di produrre dei file in formato *bag* la cui dimensione è di circa 100 MB per secondo di registrazione. Questi file contengono i dati di profondità grezzi dai quali, tra le altre cose, è possibile ricostruire una visualizzazione 3d della scena. Questi file sono utili per fare post-processing successivamente alla sessione di raccolta dati, in particolare, per calibrare parametri quali distanze minime e massime di visualizzazione, e per ottenere file in formato video *mp*4 che risultano di dimensioni più contenute e più semplici da trattare con librerie per la computer vision.

Per rendere più facile l'analisi successiva dei dati, convertiamo i *bag* in *mp*4 e non consideriamo distanze superiori a circa 7 m per non visualizzare il pavimento nei video finali.

2.3 Il dataset

Il dataset è stato raccolto in 4 sessioni separate nel mese di dicembre 2021. Con l'intenzione di osservare scene con target multipli, le sessioni di raccolta dei dati si sono concentrate nelle giornate di giovedì e venerdì a partire approssimativamente dalle 10 del mattino per circa 2 ore per sessione. La prima sessione, è servita a calibrare la telecamera e fare alcuni test. La raccolta dei dati è avvenuta in loco poiché, per ragioni concernenti il diritto alla privacy, era necessaria la nostra presenza per rispondere ad eventuali domande degli operatori di Alí.

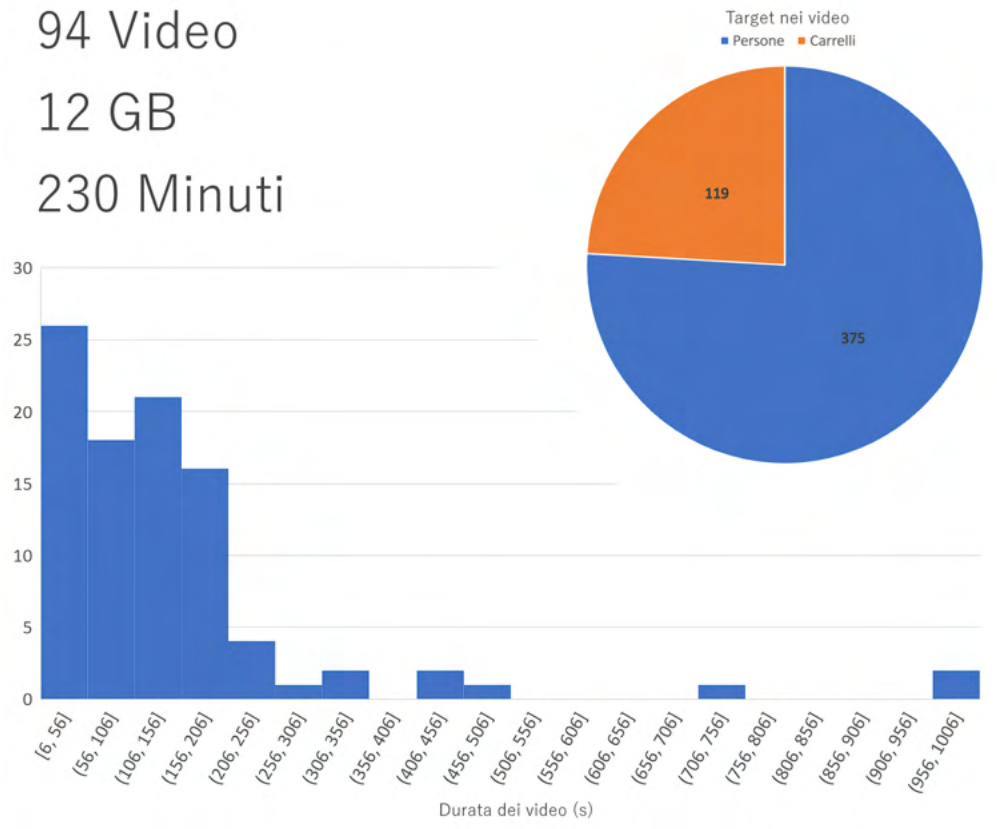

Figura 2.3: Alcune statistiche sul dataset, riassunte graficamente

In totale abbiamo raccolto 94 video per un totale di 230 minuti circa di girato e circa 12 GB di file in formato *mp*4. La dimensione dell'immagine è di 1280×720 px e il framerate è di 30 fps. La durata media dei video è di 2.4 minuti mentre la durata massima è di oltre 16 minuti. Per avere un senso delle dimensioni del dataset, e per avere dei valori affidabili con i quali confrontare gli algoritmi di tracking, faccio un conteggio manuale di persone e carrelli presenti nei video. Per poter comparare i risultati per ogni video, salvo i risultati di questi conteggi in dei file *csv* che associano

ad ogni video il corrispettivo numero di persone e carrelli. Nei video si possono contare 375 persone e 119 carrelli per un totale di 494 target. Dai dati raccolti è possibile fare alcune osservazioni.

Riguardo alle interazioni tra target, i dati catturano interazioni tra:

- Operatori e carrelli: queste interazioni sono di natura attrattiva, ovvero, gli operatori interagiscono con i carrelli trascinandoli o spingendoli ma anche caricando la merce, tuttavia questo tipo di interazione avviene tra l'operatore e il proprio carrello, carrelli di altri operatori sono semplicemente evitati in quanto ostacoli;
- Operatori e operatori: queste interazioni sono di natura repulsiva, ovvero, gli operatori cercano di evitarsi per non ostacolarsi;
- Operatori e scaffali: un altra interazione di tipo attrattivo, gli operatori entrano in contatto con gli scaffali per raccogliere la merce.

Riguardo ai carrelli, può essere utile osservare che si muovono soltanto quando in contatto con un operatore, con l'eccezione del caso in cui l'operatore, in movimento con il carrello, lasci andare il carrello, che per inerzia si muove leggermente da solo.

Un altro aspetto qualitativo dei dati importante da sottolineare è che, nella pressoché totalità dei casi, gli operatori con carrello, entrano nella scena da destra, e al bivio svoltano alla loro sinistra. Questo comportamento è dettato dall'organizzazione specifica del magazzino. Operatori senza carrello invece non hanno particolari restrizioni di movimento.

2.3.1 Visualizzazioni

Alcuni esempi di visualizzazioni. Da questi pochi esempi è possibile notare la diversità dell'aspetto visivo che possono assumere persone e carrelli, inoltre è facile osservare come l'identificazione delle persone non sia possibile.

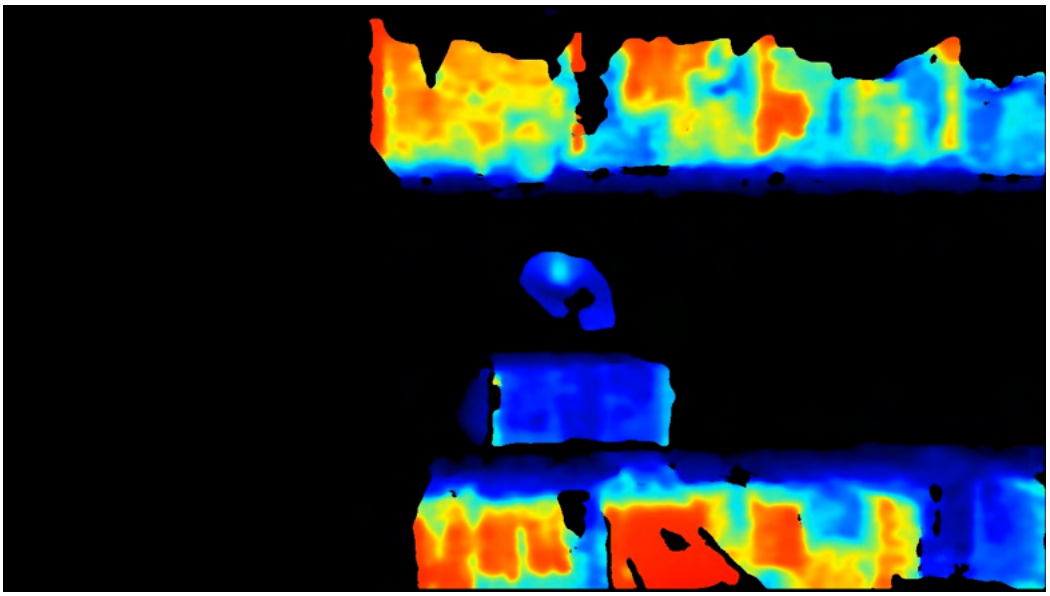

Figura 2.4: La scena ideale: la sagoma al centro dell'immagine è un operatore, è possibile osservare il colore più chiaro della testa che essendo più in alto è più vicina alla telecamera. L'operatore sembra avere della merce nella mano sinistra. La sagoma sotto è il carrello nel quale la merce sarà caricata.

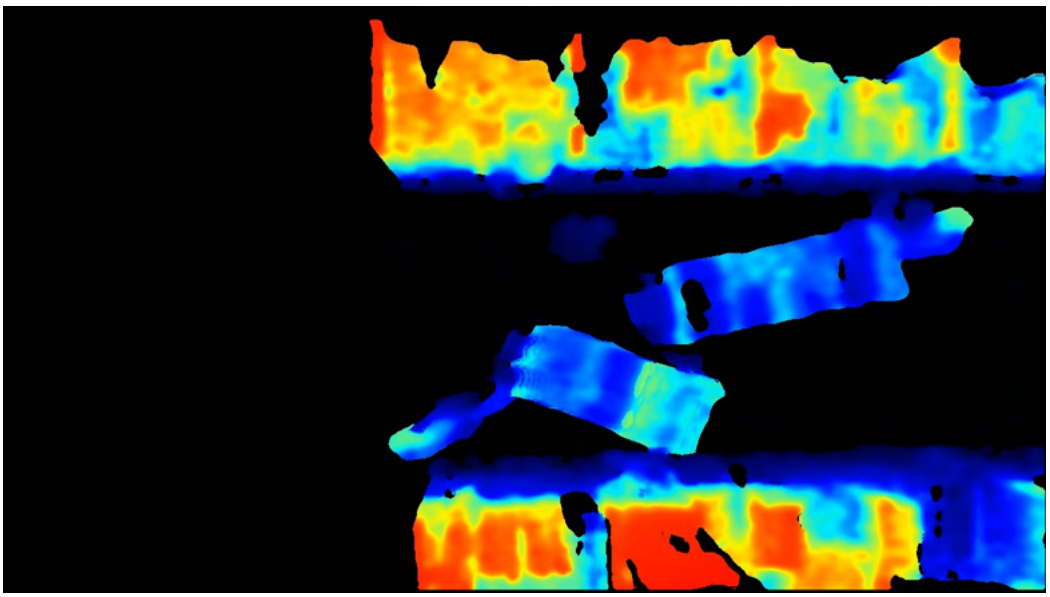

Figura 2.5: Una scena con multipli operatori e carrelli. Un terzo carrello è presente appena fuori dalla scena, sulla destra. L'operatore responsabile di questo carrello è entrato nella scena per prendere della merce ed è chinato vicino lo scaffale in alto, al centro dell'immagine.

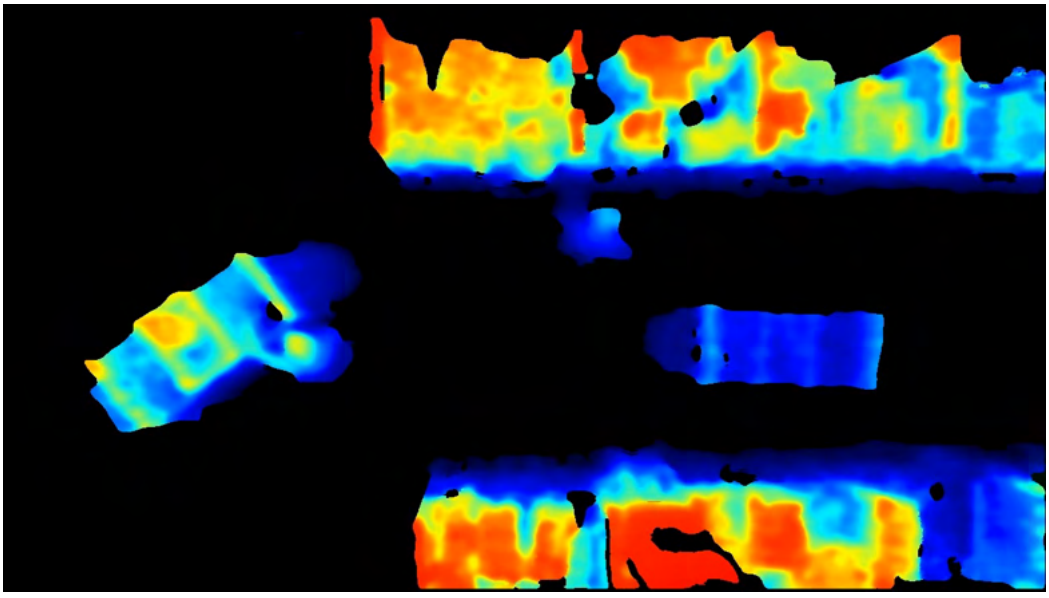

Figura 2.6: Un'altra immagine con target multipli. In questa immagine è interessante notare come la prospettiva giochi un ruolo. Plausibilmente la telecamera è stata leggermente inclinata durante l'installazione, quindi soggetti a sinistra dell'immagine risultano più vicini, notiamo i colori del carrello a sinistra. Inoltre il carrello risulta più grande.

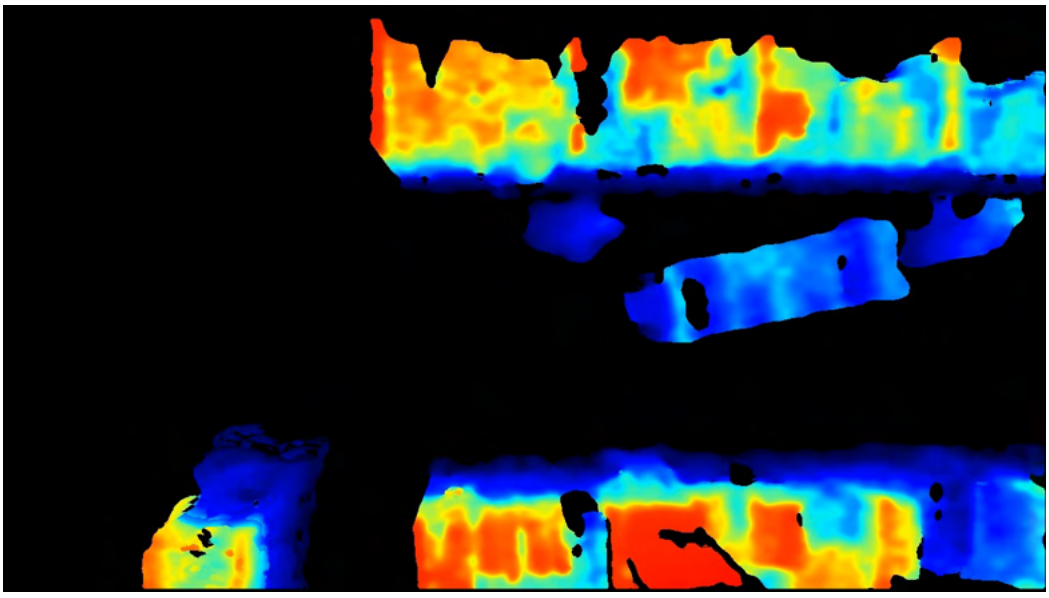

Figura 2.7: Un'altra immagine interessante, è possibile notare come il carrello in uscita dall'immagine, in basso a sinistra risulti più vicino alla telecamera, inoltre notiamo come l'aspetto degli operatori sia molto diverso rispetto alle precedenti immagini in cui sono direttamente sotto alla telecamera.

Capitolo 3

Il tracciamento

3.1 Obiettivi

Gli obiettivi principali della fase di tracciamento sono:

1. Individuare e distinguere persone e carrelli nei video raccolti;
2. Generare per ogni frame dei bounding boxes attorno ad ogni soggetto individuato;
3. Associare un identificativo univoco affidabile ad ogni soggetto;
4. Produrre dei file contenenti le informazioni necessarie agli algoritmi di Trajectory Prediction.

3.2 Object Detection

3.2.1 Definizione del problema

Nell'ambito della computer vision, fare Object Detection significa individuare e classificare istanze di oggetti appartenenti a determinate classi in immagini o video. Il risultato desiderato quando si fa Object Detection è un bounding box, un rettangolo che circoscrive l'oggetto nell'immagine. In termini formali un bounding box è spesso rappresentato con la tupla: x, y, w, h dove x e y sono le coordinate dell'angolo in alto a sinistra del rettangolo, e w e h rappresentano base e altezza del rettangolo. Un altra rappresentazione comune di bounding box è la tupla $x1, y1, x2, y2$ dove $x1$ e $y1$ rappresentano, come prima, le coordinate dell'angolo in alto a sinistra del rettangolo, e $x2, y2$ rappresentano le coordinate dell'angolo in basso a destra del rettangolo. Generalmente, dato un bounding box, è possibile passare da una rappresentazione all'altra e ricavare ulteriori informazioni come ad esempio il centro del bounding box. Nell'ambito del progetto, l'obiettivo è identificare persone e carrelli in ogni frame dei video, quindi le classi sono 2: persone, carrelli. Per quanto riguarda la rappresentazione dei bounding boxes non ci sono particolari vincoli, è importante tuttavia poter ricavare il centro del bounding box perché quella è l'informazione principale necessaria ai metodi di trajectory prediction.

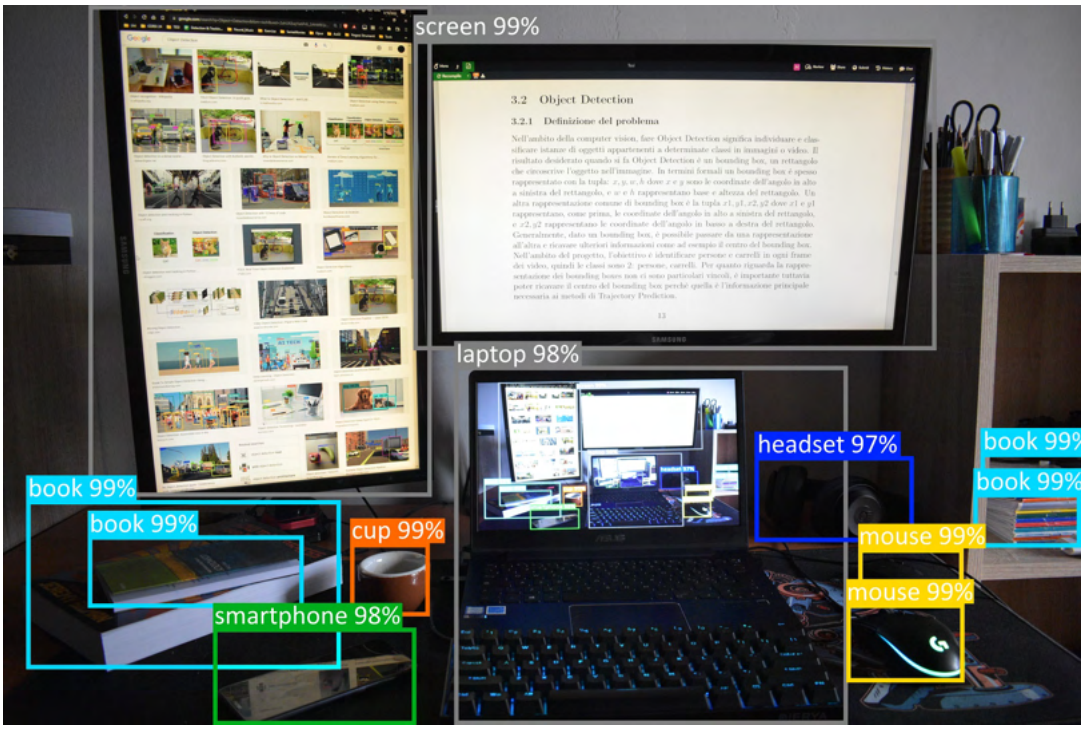

Figura 3.1: Esempio di object detection

3.2.2 Stato dell'arte

Negli ultimi dieci anni, il potere di calcolo delle GPU è aumentato notevolmente, il costo della memoria di massa è sceso, gli smartphone dotati di telecamera sono diventati ubiquiti e i social network hanno spostato il loro focus dalla condivisione di testo alla condivisione di immagini e video. In questo contesto si sono venute a creare le condizioni che hanno permesso alle reti neurali, in particolare reti neurali convoluzionali (CNN), di diventare l'approccio standard per applicazioni di computer vision, infatti, i modelli basati su reti neurali hanno bisogno di molto potere di calcolo e di grandi moli di dati. Gli object detector moderni non fanno eccezione, si basano sulle reti neurali e adottano l'approccio di apprendimento supervisionato: un dataset di immagini associate a delle annotazioni, compilate da un essere umano e rappresentanti i risultati attesi dalla rete, vengono dati al detector affinché possa avvenire l'apprendimento. Per quanto riguarda l'object detection, esistono 2 tipi di detector:

- Two-Stage detectors;
- One-Stage detectors;

Il primo tipo, opera object detection in 2 passaggi: individua prima delle regioni di interesse nell'immagine, chiamate object proposal, e poi, per ognuna di queste regioni di interesse, effettua la classificazione e la regressione del bounding box. La famiglia di metodi basata su R-CNN fa uso di questo approccio.

R-CNN

R-CNN individua, con un apposito algoritmo, le regioni di interesse, che servono ad individuare vagamente la possibile posizione di un oggetto. Ne individua circa 2000. Queste regioni di interesse, anche chiamate object proposal, possono avere dimensioni e aspect ratio diversi e inoltre possono sovrapporsi. Ad ognuna di queste il modello applica una CNN per estrarre i feature vectors. Le feature estratte vengono usate poi per fare regressione dei bounding boxes e per fare la classificazione dell'oggetto[8].

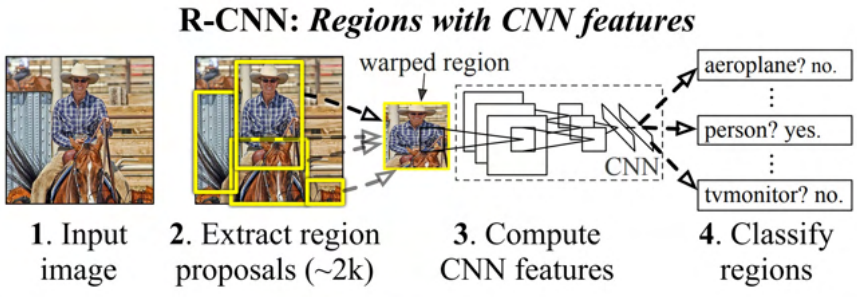

Figura 3.2: Schema riassuntivo di R-CNN[8]

Fast R-CNN utilizza lo stesso approccio, ma invece di applicare una CNN su ogni regione di interesse estratta, utilizza una sola CNN a cui viene data in input l'intera immagine. Dalla mappa delle feature prodotta dalla CNN vengono poi estratte le feature relative alle regioni di interesse che vengono poi usate come in R-CNN[7]. Faster R-CNN migliora ulteriormente le prestazioni di Fast R-CNN utilizzando una rete neurale per individuare le regioni di interesse invece di usare l'algoritmo sviluppato inizialmente per R-CNN[19]. Mask R-CNN è un modello sviluppato per fare *semantic segmentation* ovvero per classificare ogni pixel e produrre delle maschere che delineano precisamente gli oggetti individuati. Mask R-CNN estende Faster R-CNN aggiungendo una rete neurale fully connected di piccole dimensioni per fare la classificazione dei singoli pixel producendo K maschere binarie dove K è il numero di classi[9].

Generalmente, i two-stage detectors sono molto accurati ma hanno lunghi tempi di inferenza. Per le applicazioni real-time, sono stati sviluppati gli one-stage detector. L'approccio consiste nell'evitare il primo step dei two-stage detector, ovvero evitare di produrre degli object proposal, e quindi data un immagine in input, produrre direttamente bounding boxes e classificazioni.

YOLO

YOLO (You Only Look Once) è un one-stage detector che funziona dividendo l'immagine in una griglia $S \times S$ e per ogni cella della griglia, effettua classificazione e regressione del bounding box. Ogni cella della griglia quindi è responsabile della detection degli oggetti il cui centro ricade all'interno della cella. Per ogni cella della griglia, oltre a predire le coordinate del bounding box, il modello produce una misura di quanto è confidente che ci sia il centro di un oggetto nella cella. Questa misura sarà tendente a 0 se non ci sono oggetti nella cella. Il meccanismo della griglia

impone dei limiti sul numero di oggetti che possono essere individuati, in particolare per quanto riguarda gruppi di oggetti piccoli e vicini tra loro[18].

Diverse versioni di YOLO sono state sviluppate negli anni con l'obiettivo di aumentare l'accuratezza del modello. L'ultima versione, YOLOv5, implementa il modello usando il framework Pytorch.

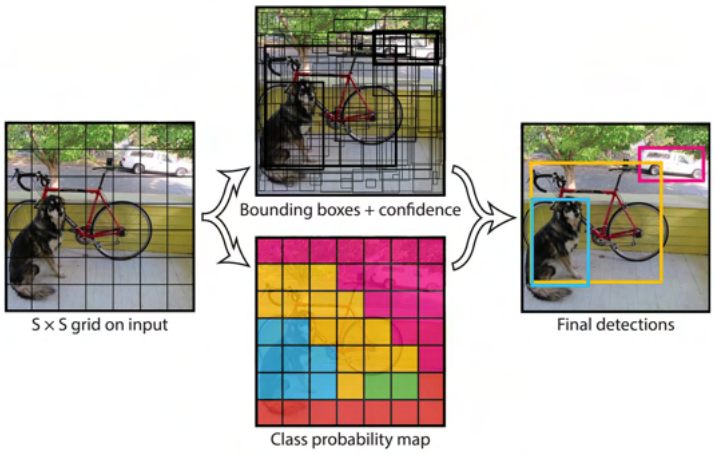

Figura 3.3: Schema riassuntivo di YOLO[18]

Uno dei dataset più usati per fare il benchmarking di modelli di object detection è COCO. COCO, che sta per Common Objects in Context, contiene oltre $300K$ immagini e circa $1.5M$ di istanze di oggetti e può essere usato per diversi task, come ad esempio: image classification, object detection e semantic segmentation[11]. I risultati su COCO per i task di object detection e real-time object detection sono rispettivamente:

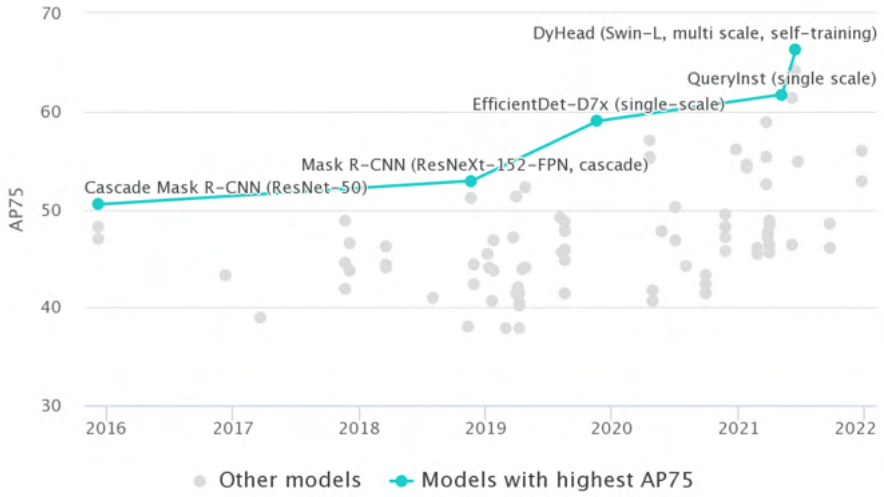

Figura 3.4: Risultati su COCO per il task di object detection. Per diversi anni i modelli basati su R-CNN si sono dimostrati i migliori in termini di AP (average precision). Recenti sviluppi nei modelli, nel caso di DyHead nel modulo di classificazione e regressione, hanno permesso di avanzare lo stato dell'arte[5].

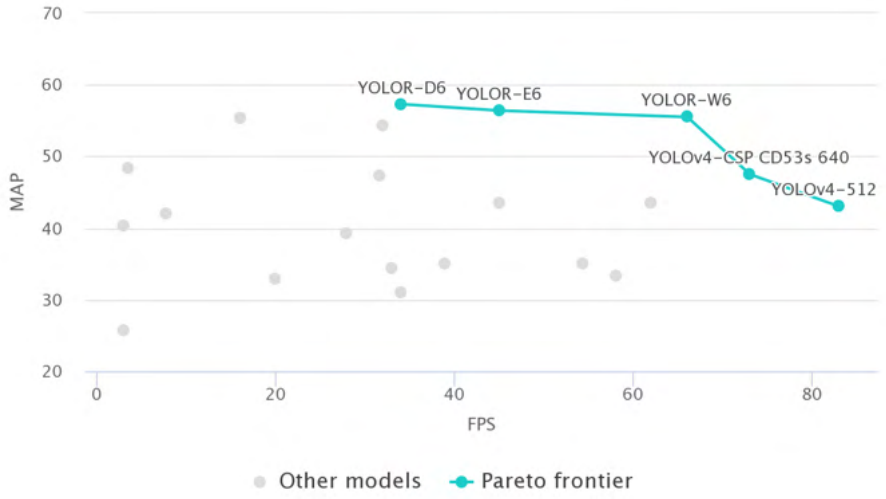

Figura 3.5: Risultati su COCO per il task di real-time object detection. I modelli basati su YOLO ottengono i migliori risultati in termini di mAP (mean average precision) e FPS (frames per second)[4].

3.2.3 Basic Object Detector

Image Pre-processing

Le caratteristiche delle immagini raccolte nel dataset permettono di fare delle assunzioni che semplificano notevolmente il problema dell'object detection:

- La videocamera è ferma, quindi gli scaffali appaiono sempre nella stessa posizione;
- Il background è nero, qualsiasi forma nelle immagini, che non siano gli scaffali, rappresenta o un operatore o un carrello;

Sfruttando queste assunzioni ho costruito un detector ad hoc molto semplice, utile a raccogliere dei risultati di partenza. Il detector fa uso di un meccanismo che si ispira alla background subtraction. La background subtraction effettua la sottrazione tra il frame corrente e un modello di background per ottenere la foreground mask, ovvero una maschera che applicata al frame corrente restituisce gli oggetti in primo piano.

La sottrazione tra il frame e il modello di background avviene semplicemente per ogni pixel. Il risultato della sottrazione dev'essere filtrato per non introdurre rumore nella foreground mask. Per fare ciò è necessario individuare un valore appropriato di threshold: per ogni pixel, se il risultato della sottrazione supera il threshold, il corrispondente pixel nella foreground mask avrà valore 1, 0 altrimenti. Formalmente:

$$FM_{i,j} = \begin{cases} 1 & se\ |CF_{i,j} - BM_{i,j}| > t \\ 0 & altrimenti \end{cases} \tag{3.1}$$

Dove $FM_{i,j}$ rappresenta il valore del pixel della foreground mask a coordinate i, j, $CF_{i,j}$ è il pixel analogo nel frame corrente e $BM_{i,j}$ è il pixel analogo nella background mask, infine t rappresenta il threshold.

Il modello di background può essere costruito in diversi modi, un metodo semplice è quello di prendere il valore mediano per ogni pixel per gli n frame passati. In questo modo il modello di background si aggiorna allo scorrere dei frame e se un oggetto entra nella scena e si ferma, viene incluso nel background. Utilizzare un modello del background che si aggiorna nel tempo non è l'ideale per il dataset di Alí, infatti gli operatori entrano nella scena trascinando il carrello, si fermano per raccogliere della merce e in quel lasso di tempo la foreground mask si degrada mentre operatore e carrello entrano a fare parte del background.

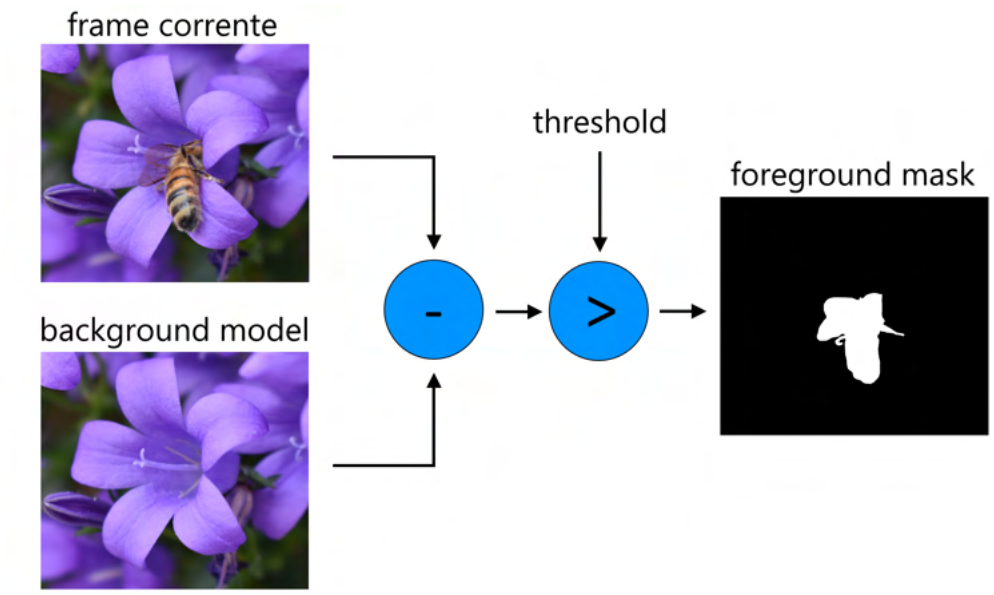

Figura 3.6: Schema riassuntivo della background subtraction.

Il caso ideale in cui applicare background modelling è quello in cui non è possibile ottenere un frame rappresentante il background vuoto, e in cui gli oggetti non si fermano, ad esempio un autostrada.

Nel caso del dataset di Alí, il background è semplice da determinare e definire: è uno sfondo nero con gli scaffali. Quindi è possibile effettuare background subtraction utilizzando un modello del background definito a priori e senza aggiornarlo. Tuttavia, la natura delle riprese fatte implica che l'aspetto degli scaffali non sia perfettamente costante, e che un threshold alto abbastanza da azzerare queste variazioni nell'aspetto degli scaffali, azzererebbe anche i target di interesse.

La soluzione quindi è quella di usare come background noto a priori uno sfondo completamente nero, e sfruttare la posizione fissa della telecamera per applicare al frame corrente una maschera definita a priori che azzera le regioni dell'immagine corrispondenti agli scaffali.

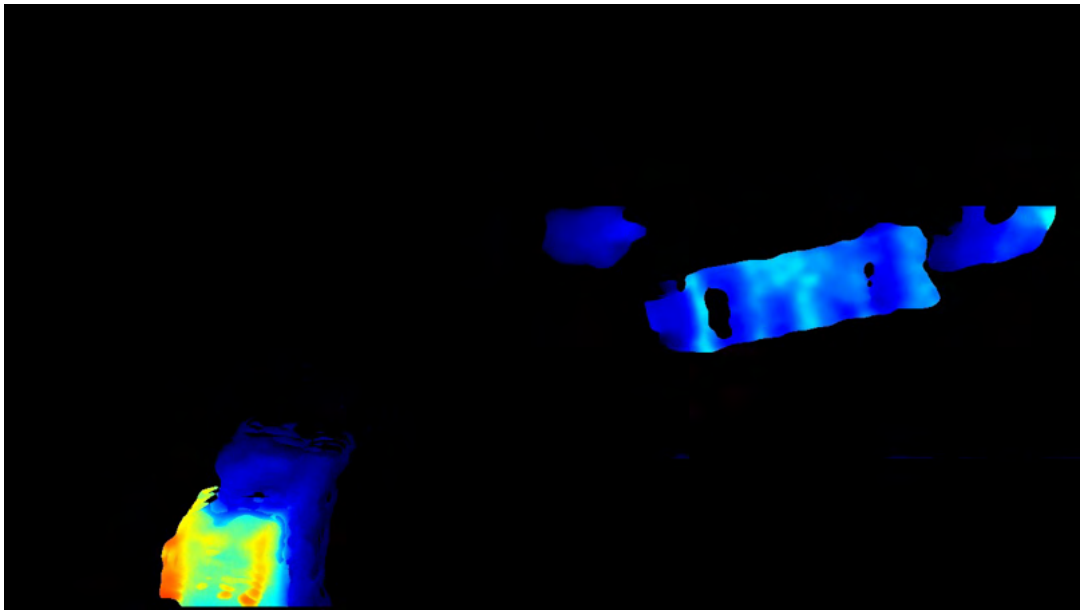

Figura 3.7: Un frame al quale è stata applicata la foreground mask ottenuta dalla background subtraction.

La sottrazione da un frame di uno sfondo nero corrisponde al non effettuare alcuna operazione, infatti il colore nero è rappresentato dal valore 0 in greyscale e dalla tupla $(0, 0, 0)$ in RGB. Questo permette di saltare il passaggio della sottrazione e di passare direttamente a filtrare l'immagine per eliminare il rumore presente. Il risultato ottenuto è un frame in cui le uniche forme presenti rappresentano target che devono essere individuati e dei quali si vuole ottenere un bounding box.

Generare i bounding boxes

Per ottenere un bounding box a partire da un immagine simile è possibile usare dei metodi standard della libreria OpenCV, in particolare, `cv2.findContours` e `cv2.boundingRect`.

`cv2.findContours` utilizza un algoritmo di border following per, data un immagine binaria, ritornare un vettore di contours. Ogni contour rappresenta una forma chiusa nell'immagine ed è rappresentato come un vettore di punti (coordinate x, y) che descrive la forma individuata.

`cv2.boundingRect` dato un contour, ritorna un bounding box che circoscrive la forma data in input. Il metodo rappresenta i bounding boxes nel formato (x, y, w, h).

Classificazione

Per classificare gli oggetti individuati ho scelto di usare un approccio molto semplice ma relativamente efficace data la natura delle riprese: il metodo `cv2.contourArea` dato un contour in input, calcola e ritorna l'area occupata dal contour. Stabilendo dei threshold per i valori dell'area è possibile identificare l'oggetto individuato, infatti i carrelli hanno un area maggiore nell'immagine rispetto agli operatori. Quindi i

contour con area più grande saranno classificati come carrelli e quelli con area minore come operatori.

Il seguente pseudocodice descrive brevemente il classificatore:

```
if area > noise_threshold and area < person_threshold:
    classification = 'person'
elif area > person_threshold and area < cart_threshold:
    classification = 'cart'
elif area > cart_threshold:
    classification = 'person and cart'
```

Pregi e limitazioni

I vantaggi del detector sviluppato stanno nella semplicità dell'approccio e nel fatto che non richiede allenamento. Il detector può risultare valido per applicazioni semplici quali ad esempio la conta del numero di persone che attraversa l'inquadratura. Le limitazioni sono diverse e derivano tutte dalla semplicità del metodo:

1. Il detector non riesce a separare persone e carrelli: quando l'operatore è in contatto con il carrello, l'algoritmo di border following identifica un unica forma e non ha maniera di distinguere l'operatore e il carrello;

2. Quando operatori e carrelli entrano o escono dall'inquadratura, per alcuni frame sono visibili solo parzialmente, questo implica che, ad esempio quando un carrello entra nell'immagine, l'area occupata nell'immagine inizialmente è piccola, inferiore al threshold usato per identificare i carrelli, questo comporta errori di classificazione;

3. Com'è possibile notare dalla Figura 2.6, la prospettiva gioca un ruolo nelle immagini: una lieve inclinazione della telecamera fa sì che gli oggetti risultino più piccoli nella parte destra dell'immagine e più grandi nella parte sinistra, questo comporta che i threshold necessari alla classificazione devono cambiare in base alla posizione degli oggetti nell'immagine;

4. L'approccio è molto specifico e quindi poco generale. Sebbene sia possibile mitigare i problemi appena illustrati, tali aggiustamenti valgono solo per questa specifica scena. Un altra telecamera avrebbe bisogno di ulteriori aggiustamenti. L'introduzione di carrelli di dimensione diversa, com'è successo nell'ultimo giorno di riprese, interferisce con l'algoritmo di classificazione. In definitiva, il metodo è poco robusto a variazioni nei parametri dell'inquadratura.

Alcune delle limitazioni appena elencate possono essere affrontate senza cambiare radicalmente approccio: per quanto riguarda il limite numero 2, è possibile riservare la classificazione per gli oggetti che non sono in contatto con il bordo dell'immagine. Per quanto riguarda il limite numero 3, è possibile dividere l'immagine in zone diverse in cui valgono threshold per la classificazione diversi. Aumentare la robustezza del detector e separare carrelli e persone risulta difficile con il metodo usato, quindi ho deciso di utilizzare un object detector allo stato dell'arte.

3.2.4 Object Detection con YOLO

Modello

YOLO, You Only Look Once, è un one-stage object detector: funziona dividendo l'immagine in una griglia $S \times S$, ad ogni cella della griglia è associato l'oggetto il cui centro giace in tale cella. Per ogni cella vengono prodotti B bounding boxes e per ognuno una misura di confidenza, inoltre alla cella viene associata una vettore di probabilità, ognuna delle quali esprime la probabilità che l'oggetto centrato in quella cella sia di una data classe. La misura di confidenza è un valore che descrive quanto il modello sia sicuro della classificazione che ha fatto e della correttezza del bounding box associato. Se una cella non contiene alcun oggetto, le misure di confidenza associate saranno basse, idealmente tendenti a 0[18].

YOLO è considerato un one-stage detector perché a fare l'object detection è una singola rete neurale che produce coordinate del bounding box, classificazione e score di confidenza.

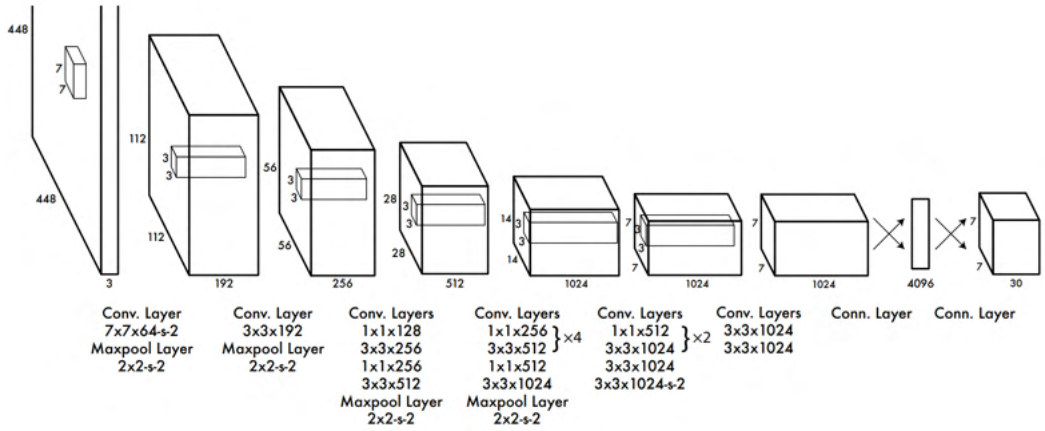

Figura 3.8: La rete neurale convoluzionale impiegata da YOLO[18]. Diverse modifiche sono state apportate alla rete nelle successive versioni di YOLO, tuttavia l'approccio generale rimane invariato.

Gli strati convoluzionali della rete estraggono le feature dall'immagine in input, ovvero elaborano l'immagine e ne creano una rappresentazione vettoriale utile agli strati fully connected per predire bounding boxes, classificazione e score di confidenza.

YOLOv2 migliora alcuni aspetti di YOLO, tra le differenze più importanti[16]:

- Effettua il pre-training della rete con immagini a maggior risoluzione;
- Introduce anchor boxes definiti a priori in base ad aspect ratio comuni, 5 anchor boxes sono definiti per ogni cella, la rete poi produce offset dagli anchor boxes invece di coordinate del bounding box;
- Introduce batch normalization nei layer convoluzionali.

YOLOv3 introduce dei cambiamenti a livello di architettura della rete infatti utilizza DarkNet-53, che aumenta il numero di layer della rete dai 19 di YOLOv2 a 53.

L'obiettivo principale di YOLOv3 è quello di aumentare l'accuracy del modello. I vecchi modelli infatti avevano difficoltà a individuare oggetti piccoli, particolarmente se molti e vicini tra loro. YOLOv3 quindi fa detection a 3 scale diverse per sopperire a questi problemi[17].

YOLOv4 introduce degli ulteriori miglioramenti, l'architettura è composta da 4 blocchi:

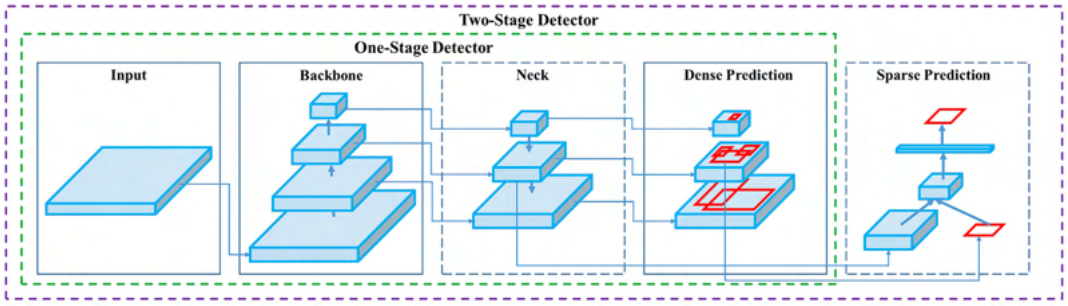

Figura 3.9: L'architettura generale di YOLOv4 è rappresentata dai blocchi parte del one-stage detector[3].

Per la backbone del modello, YOLOv4 usa CSPDarknet53. Questa rete applica i principi introdotti da CSPNet a Darknet 53. Il meccanismo introdotto da CSPNet consiste nel separare la mappa di attivazione del layer base di una rete convoluzionale densa in due parti, una delle due verrà data in input ad un layer convoluzionale, l'altra sarà aggregata all'output di tale layer. Questo approccio porta miglioramenti in termini di complessità di calcolo e di accuratezza del modello[22].

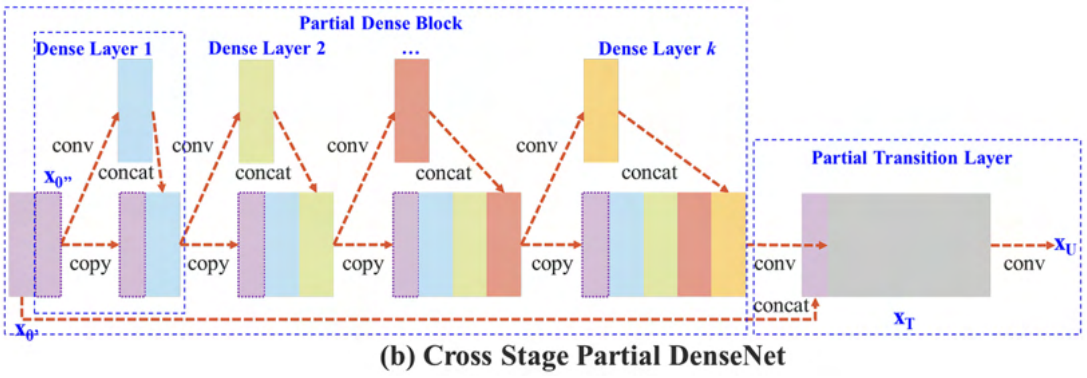

Figura 3.10: L'architettura generale di CSPNet[22].

Per quanto riguarda il neck, YOLOv4 impiega dei meccanismi attentivi per aggregare le informazioni prodotte dalla backbone del modello. Questo favorisce un aumento dell'accuratteza del modello. Infine per quanto riguarda l'head, la parte che produce effettivamente le predizioni di bounding boxes e classificazioni, non ci sono modifiche rispetto a YOLOv3.

YOLOv5 ha un architettura molto simile a YOLOv4[23] e la principale differenza tra i due modelli, pubblicati ad appena un mese di distanza, è l'implementazione in Pytorch che ne semplifica l'uso rispetto all'implementazione in Darknet di YOLOv4.

Dopo aver testato sia YOLOv4 che YOLOv5, per qualità della documentazione, facilità di utilizzo e integrazione con Pytorch[20], scelgo di usare YOLOv5.

Dataset

Allenare un Object Detector richiede un dataset di immagini annotate. Data un immagine, le annotazioni per l'immagine consistono in una lista di bounding boxes e classi di appartenenza degli oggetti nell'immagine stessa. In questo contesto, le annotazioni sono le *ground truth* per il modello. I detector YOLO richiedono uno specifico formato per le annotazioni:

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

Dove:

- `object-class` è un intero tra 0 e $n-1$ con n il numero di classi di oggetti;
- `x y` sono le coordinate del centro del bounding box, normalizzate rispetto alla larghezza e all'altezza dell'immagine, ovvero: $x = absolute\_x/image\_w$ e $y = absolute\_y/image\_h$. Quindi entrambi i valori sono reali e appartengono all'intervallo $(0.0, 1.0]$;
- `w h` sono rispettivamente larghezza e altezza del bounding box, anch'esse normalizzate rispetto a larghezza e altezza dell'immagine.

Ad esempio, al seguente frame:

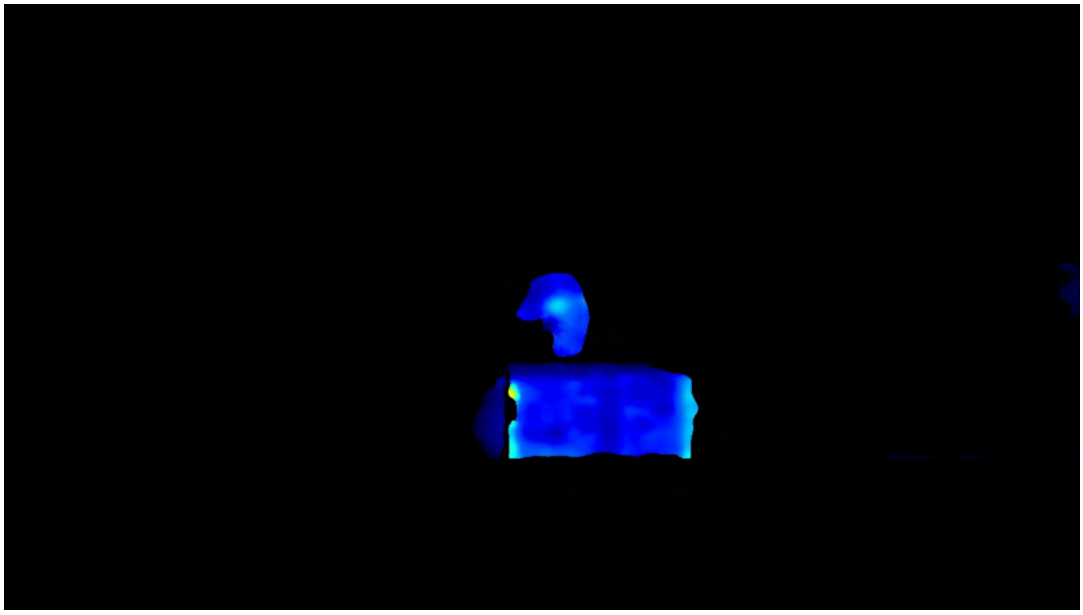

Figura 3.11: Frame tratto dal dataset creato per allenare un modello YOLOv5.

corrisponde l'annotazione:

```
0 0.510938 0.513194 0.065625 0.134722
1 0.544531 0.669444 0.201563 0.161111
```

Dove la prima riga identifica l'operatore, la seconda il carrello.

Per creare il dataset da impiegare per l'allenamento del modello YOLOv5, utilizzo un tool apposito: *labelImg* un tool grafico che permette di selezionare la cartella sorgente dalla quale vengono aperte le immagini da annotare, la cartella destinazione in cui salvare i file con le annotazioni, e soprattutto permette di tracciare manualmente i bounding boxes attorno agli oggetti target presenti nelle immagini.

Dalla collezione originale di video, estraggo e annoto 1971 frame da 11 video (circa il 13% del totale dei video) selezionati in quanto particolarmente interessanti dal punto di vista del numero di persone e carrelli presenti e dal punto di vista delle interazioni tra di essi. I frame che estraggo sono frame non vuoti, e tra un frame e il successivo intercorre almeno un secondo, questo per evitare di avere frame troppo simili, problema che sorge vista la frequenza di raccolta immagini di circa 30 fps. Il numero di istanze di persone contenute nel dataset è di 1565 mentre per i carrelli è di 1518. La simmetria tra il numero di istanze di oggetti di classi diverse è importante per evitare di introdurre bias nel modello nei confronti di una delle classi. In totale la dimensione del dataset, annotazioni incluse, è di circa 31.5 MB.

Infine, divido il dataset in training, validation e test set. In generale, più esempi nel training set implicano che il modello ha più dati dai quali apprendere; più esempi nel validation set implicano che la validazione del modello è più accurata e meno soggetta a rumore; più esempi nel test set danno un idea più accurata di come il modello generalizzi. Dato e osservato il totale di 1971 frame annotati, trovo ragionevole che circa 200 frame rappresentino bene i diversi aspetti che persone e carrelli possono assumere, quindi scelgo di usare uno split $75 - 12 - 12$ per avere circa 200 frame nei validation e test sets. Per operare lo split uso un semplice script powershell:

```
ls -n *.txt | random -c 256 | ForEach-Object { move $_ labels/test;
move $_.replace('.txt', '.jpg') images/test }
```

Con le opportune modifiche allo script produco train, validation e test set. Per il funzionamento del modello creo un file di configurazione: `depth_dataset.yaml` che contiene la seguente semplice descrizione del dataset:

```
path:  # root del dataset
train: # cartella contente le immagini del training set
val:   # cartella contente le immagini del validation set
test:  # cartella contente le immagini del test set
nc: 2  # numero di classi
names: # lista di nomi delle classi
```

Il risultato finale è la seguente struttura di cartelle:

```
depth_dataset
├── images
│   ├── train
│   ├── val
│   └── test
├── labels
│   ├── train
│   ├── val
│   └── test
└── depth_dataset.yaml
```

Training

YOLOv5, sviluppato e mantenuto da Ultralytics, mette a disposizione modelli pre-allenati di dimensioni diverse. Scelgo di allenare un modello YOLOv5m, che ottiene una $mAP\_0.5:0.95$ di 45.4 su COCO avendo 21.2 milioni di parametri per circa 41 MB di dimensioni.

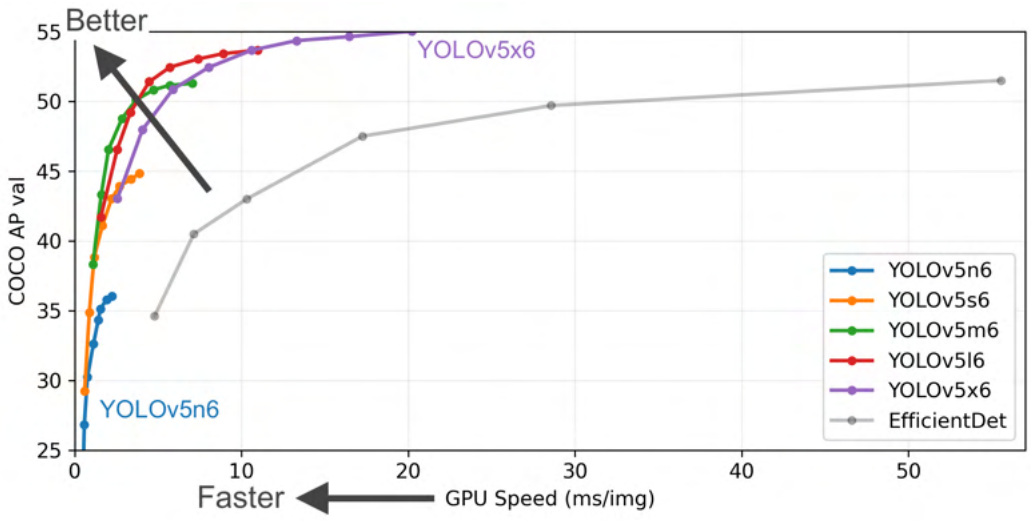

Figura 3.12: Confronto tra modelli YOLOv5 di dimensioni diverse. EfficientDet è un altro one-stage object detector sviluppato da Google[20].

Per l'allenamento del modello uso una macchina messa a disposizione dei tesisti dal dipartimento di matematica, collegandomi tramite SSH e SFTP. La macchina è dotata di una GPU Quadro M4000. La frequenza di clock della GPU è 773 MHz, la frequenza della memoria, che è di 8 GB GDDR5, è di 1502 MHz.

Per il logging delle metriche del training utilizzo Weights & Biases (W&B), un servizio usato per storicizzare metriche, produrre grafici, costruire reports, dashboards etc. In particolare utilizzo W&B per il logging degli output del training e per i grafici delle metriche che si aggiornano in tempo reale.

Per la *data augmentation*, il framework YOLOv5 integra Albumentations, una libreria che permette di definire un oggetto, solitamente chiamato `transform` che applica online i metodi di data augmentation specificati ai dati. La data augmentation permette di incrementare la mole di dati a disposizione del modello semplicemente applicando delle trasformazioni ai dati. Questo può essere fatto:

- Offline: prima della fase di training, vengono applicate le trasformazioni ai dati, vengono salvati i risultati e aggiunti al training set;
- Online: durante il training, prima di fornire i dati al modello, vengono applicate con una certa probabilità le trasformazioni i cui parametri spesso sono randomizzati;

La differenza nei due metodi sta nel fatto che con l'offline data augmentation il modello viene allenato sulle stesse trasformazioni ad ogni epoca, mentre con l'approccio online le trasformazioni applicate cambiano.

Utilizzando albumentations, applico le seguenti trasformazioni:

- `RandomCrop` effettua un crop randomico;
- `HorizontalFlip` effettua un flip orizzontale dell'immagine;
- `VerticalFlip` effettua un flip verticale dell'immagine;
- `RandomRotate90` effettua una rotazione di 90 gradi;
- `RandomBrightnessContrast` cambia luminosità e contrasto dell'immagine randomicamente;
- `Blur` applica del blur all'immagine;
- `MedianBlur` applica del blur usando un filtro mediano all'immagine;
- `ToGray` trasforma l'immagine in greyscale.

Applico ognuna di queste trasformazioni con probabilità 0.25, quindi, il numero di trasformazioni atteso è di 2 per ogni immagine. Durante il training il framework userà sia l'immagine originale, sia l'immagine a cui sono applicate le trasformazioni.

Avviare il training è molto semplice, si tratta di lanciare l'esecuzione dell'opportuno script del framework:

```
python train.py --batch 16 --epochs 250 --data depth_dataset.yaml
--weights yolov5m.pt
```

- `--batch 16` indica la dimensione dei batches, ovvero il numero di esempi su cui il modello è allenato tra un update dei parametri e il successivo. In generale è consigliato partire da batch sizes piccoli e andare crescendo. Il valore 16 è consigliato nella documentazione di YOLOv5;

- `--epochs 250` indica quante volte il modello andrà allenato sull'intero training set. Il valore scelto non ha particolare importanza in quanto i parametri vengono salvati ed è possibile riprendere l'allenamento una volta terminate le 250 epoche. Ovviamente questo valore non può essere troppo piccolo perché la rete non riuscirebbe ad apprendere dai dati;

- `--data depth_dataset.yaml` indica al modello il file di configurazione definito per il dataset;

- `--weights yolov5m.pt` indica al modello quale configurazione della rete usare.

Metriche

Per valutare gli object detector, una delle metriche più comunemente usate in letteratura è la mAP (mean average precision). In generale mAP è definita come la AP media sulle classi di oggetti nel dataset oppure come la AP media su threshold diversi per l'IoU. AP è definita come l'area sotto la curva precision-recall. Generalmente, precision e recall sono definite come segue:

$$Precision = \frac{TP}{TP + FP} \tag{3.2}$$

$$Recall = \frac{TP}{TP + FN} \tag{3.3}$$

Intuitivamente, la precision da una misura di quanti positivi identificati dal modello sono effettivi positivi, la recall indica quale proporzione di effettivi positivi è stata correttamente individuata.

Per poter applicare queste metriche al dominio dell'object detection è necessario definire cos'è un vero positivo. Idealmente, un vero positivo nell'object detection ha un bounding box molto vicino alla ground truth, ovvero al bounding box fornito dalle annotazioni delle immagini, e classifica correttamente l'oggetto. La classificazione corretta dell'oggetto è semplice da verificare, è necessario quindi definire una metrica che dia una misura di somiglianza tra bounding boxes.

La metrica usata in letteratura è la IoU (intersection over union). Formalmente IoU è definita come:

$$IoU = \frac{Intersection\ Area}{Union\ Area} \tag{3.4}$$

Intuitivamente, dati due bounding boxes, IoU da una misura di somiglianza tra bounding boxes. IoU tendente ad uno indica che l'area di intersezione tra i bounding boxes è vicina all'area totale occupata dai due bounding boxes, il che indica che sono molto simili.

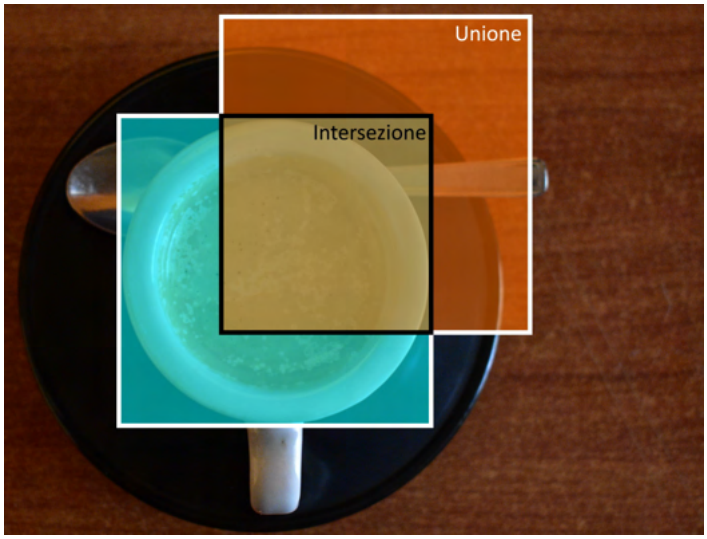

Figura 3.13: Visualizzazione delle aree di intersezione e unione. Più simili sono i bounding boxes, più simili sono i valori per queste due aree, più IoU si avvicina ad uno. Due bounding boxes disgiunti avranno IoU uguale a zero.

Data la metrica IoU è necessario scegliere un valore di threshold oltre il quale una predizione è considerata come vero positivo. Un threshold comunemente usato è 0.5, in altri casi si considera un insieme di thresholds che va da 0.5 a 0.95 con incrementi di 0.05.

3.2.5 Risultati

Il training di 250 epoche sul dataset di Alí è durato circa 16 ore, per una media di circa 4 minuti per epoca. Al termine del training, i risultati ottenuti sul test set sono i seguenti:

Classi	mAP@.5	mAP@.5:.95
Tutte	0.975	0.972
Persone	0.975	0.974
Carrelli	0.975	0.97

Tabella 3.1: Metriche calcolate sul test set dal framework YOLOv5.

Inoltre, l'IoU medio dei veri positivi è 0.92, il tempo medio di inferenza 32.2 ms.

I risultati ottenuti sono adeguati alle necessità degli algoritmi di tracking e prediction. Rispetto al basic detector introdotto in §3.2.3, i risultati ottenuti permettono di sopperire alle limitazioni elencate, in particolare, YOLOv5 ovviamente separa carrelli e persone, inoltre, la prospettiva non crea problemi al detector che avendo esempi di persone e carrelli in ogni posizione dell'immagine ha appreso i vari aspetti che possono avere, inclusi carrelli di dimensioni diversi. Una limitazione presente nel detector semplice si ripresenta, per ragioni diverse, con YOLO. All'ingresso di persone e carrelli nell'inquadratura la classificazione è meno robusta.

Visualizzazioni

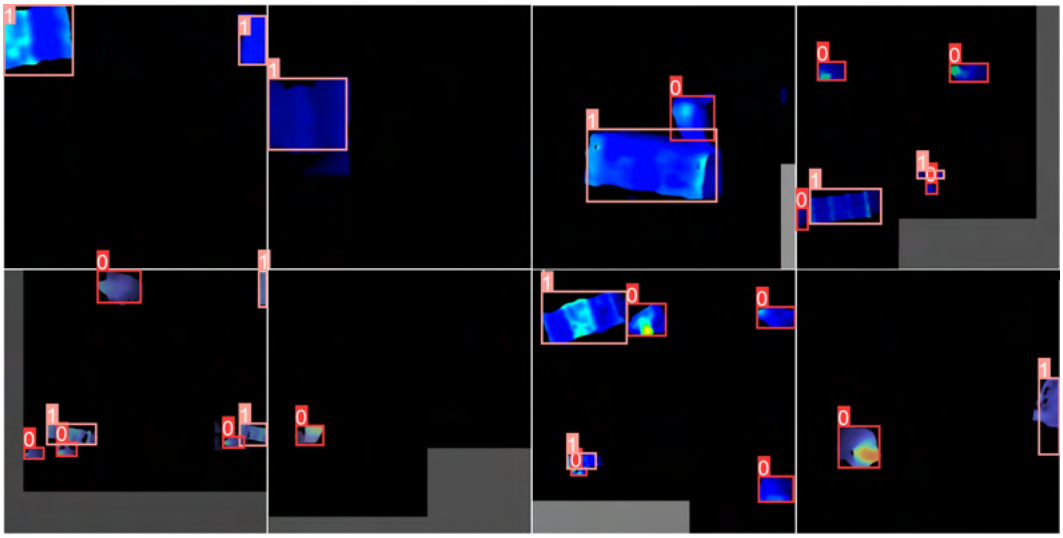

Figura 3.14: Mosaico prodotto da Weights & Biases durante il training. Si possono notare le classificazioni dei bounding boxes: 0 - persona, 1 - carrello. Inoltre è possibile notare l'applicazione di alcune trasformazioni per la data augmentation.

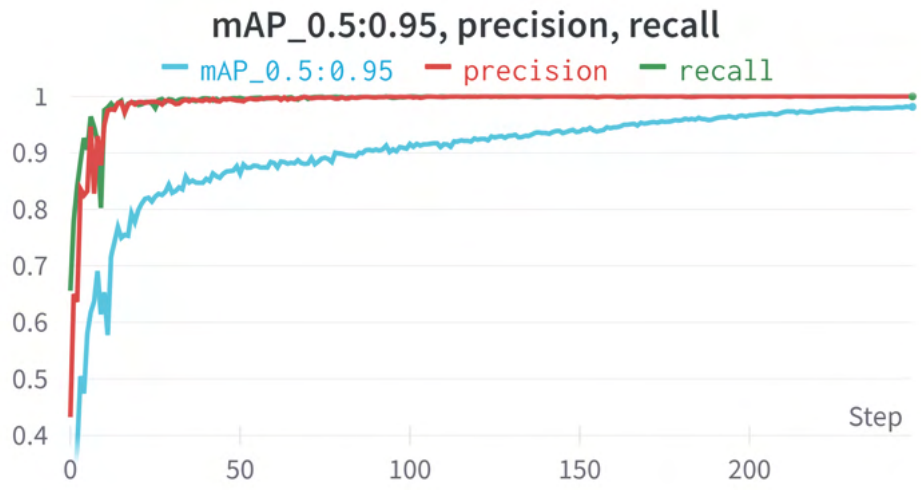

Figura 3.15: Plot di mAP.5:.95, precision e recall. Precision e recall sono molto alte, vicine entrambe ad uno, questo implica che anche l'area sotto la curva precision recall sia vicino ad uno come mostrato dalla mAP. La mAP mostrata è la AP media sui threshold da 0.5 a 0.95 con incremento di 0.05. Più alto è il threshold sull'IoU più preciso dovrà essere il modello con la previsione del bounding box per ottenere un vero positivo.

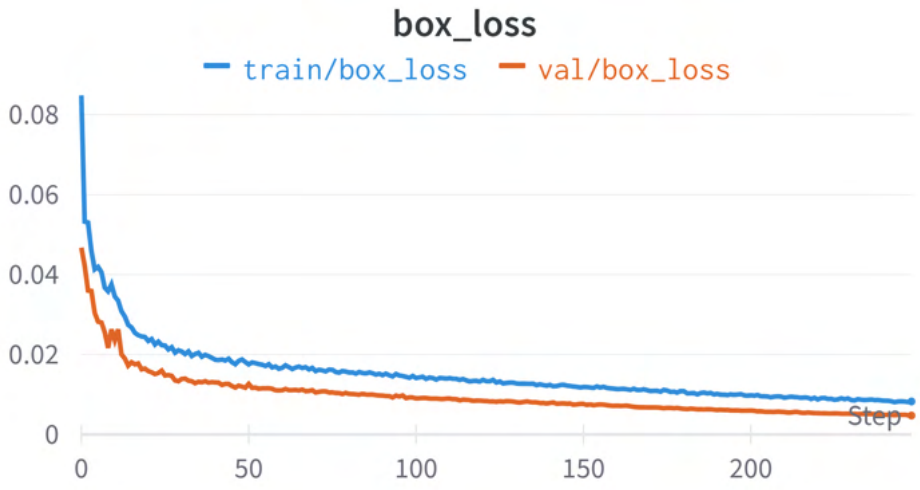

Figura 3.16: Plot di box loss, una delle tre loss function usate da YOLO, calcolata sul training set e sul validation set. Questa loss function è basata su *IoU* con delle modifiche sul caso in cui i bounding boxes siano disgiunti per evitare di avere gradiente pari a zero.

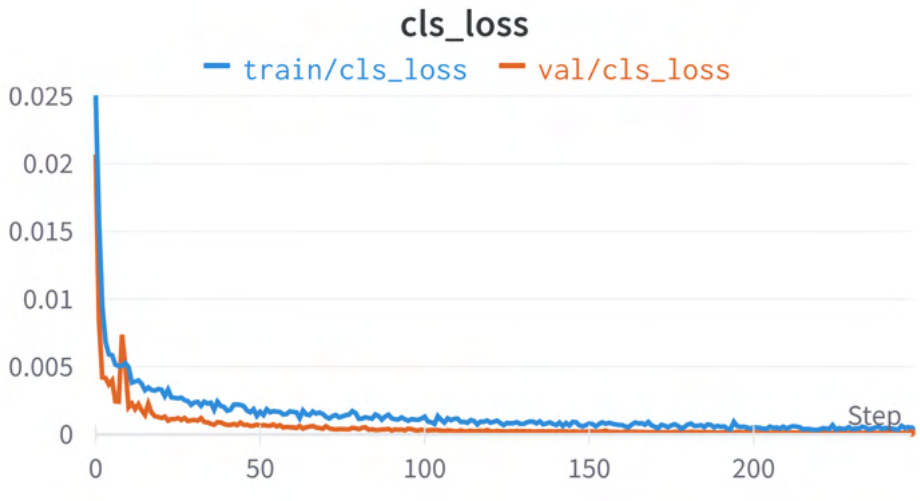

Figura 3.17: Plot di class loss, loss function usata da YOLO, calcolata sul training set e sul validation set. Questa è una semplice classification loss.

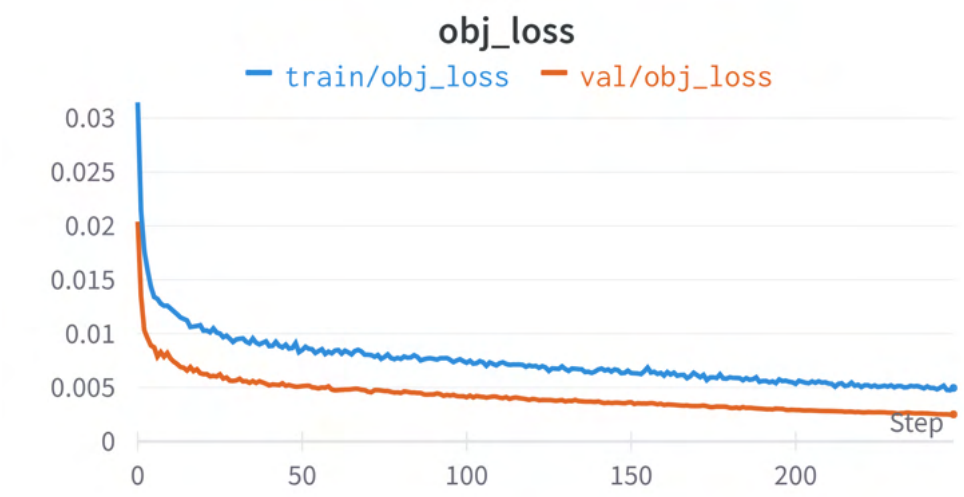

Figura 3.18: Plot di objectness loss, loss function usata da YOLO, calcolata sul training set e sul validation set. Questa è la loss function inerente all'objectness score di YOLO, una misura di confidenza sulla presenza di un oggetto.

I grafici prodotti mostrano le capacità del modello, che è in grado di generalizzare su esempi non presenti nel training set. In particolare, i grafici delle loss function mostrano che il modello non sta facendo overfitting: il modello non ha "memorizzato" gli esempi dati nel training set, ha appreso dei pattern generali che gli permettono di classificare i target su cui è stato allenato. La loss function ha valori più bassi sul validation set che sul training set, questo comportamento è inusuale ma può essere spiegato dalla data augmentation che è applicata solo al training set e lo rende più "difficile" del validation set.

3.3 Object Tracking

3.3.1 Definizione del problema

Dato un video, fare object tracking significa identificare i target al frame n, assegnare loro un identificativo, e, al frame $n+1$ tenere traccia dei target e assegnare loro lo stesso identificativo se rappresentano lo stesso oggetto presente nel frame precedente. Un possibile punto di partenza per fare object tracking è l'object detection. Una volta individuati i target nell'immagine ci sono diversi approcci che si possono usare per determinare a quali oggetti corrispondono nei frame adiacenti, ad esempio tramite misura di similarità, o tramite misura di correlazione. Nell'ambito del progetto, il risultato desiderato è, per ogni video, una lista di tuple del tipo $id, class\_id, x, y, w, h$, dove:

- id rappresenta l'id univoco del target;
- $class\_id$ rappresenta la classe del target (0 - persona, 1 - carrello);
- x, y, w, h rappresentano le coordinate e le dimensioni del bounding box come ritornato da YOLO.

L'obiettivo è quello di tracciare separatamente persone e carrelli e, usare un sistema di assegnamento di id tale che l'id massimo dia una stima del numero totale di target apparsi nel video. I dati così prodotti sono necessari e sufficienti agli algoritmi di trajectory prediction.

3.3.2 Stato dell'arte

Nella letteratura si fanno diverse distinzioni nel dominio dell'object tracking. In particolare si distingue tra:

- SOT e MOT: Single Object Tracking e Multiple Object Tracking, in base al numero di oggetti che si vuole tracciare;
- Offline e Online tracking: si parla di offline tracking quando il modello ha a disposizione un insieme di frame già salvato. In questa situazione, supponendo che il modello sia arrivato a fare il tracciamento fino al frame n, può fare uso sia di frame precedenti che di frame successivi per fare il tracking. Nell'online tracking il modello ha a disposizione solo frame precedenti ad n in quanto il modello deve poter funzionare in real-time;
- Tracking by detection e detection free tracking: i modelli che fanno tracking by detection effettuano il matching tra oggetti individuati al frame n e oggetti individuati al frame $n+1$. I modelli detection free utilizzano object detection solo per inizializzare la traccia e poi localizzano l'oggetto nel frame successivi senza fare detection.

GOTURN

GOTURN è uno dei primi modelli, facenti uso di reti neurali, sviluppati per l'object tracking. GOTURN fa single object tracking: dato un frame, un crop del frame con

l'oggetto da tracciare al centro e una search region nel frame successivo, GOTURN impiega un modello basato su una CNN per fare regressione del bounding box dell'oggetto da tracciare. L'architettura di GOTURN è piuttosto semplice: vengono impiegate due CNN per produrre i feature vectors dei crop dati in input al modello. I feature vectors vengono poi dati a dei layer fully connected per essere confrontati e produrre un bounding box che circoscrive l'oggetto nel frame corrente[10].

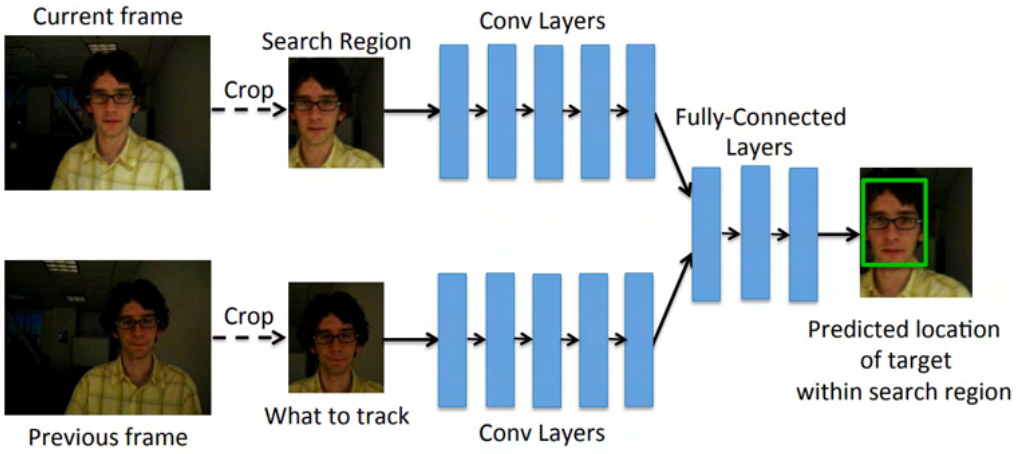

Figura 3.19: Schema che riassume l'architettura di GOTURN [10].

MDNet

MDNet (Multi-Domain Net) è un online tracker che fa single object tracking. L'architettura del modello è la seguente: una CNN viene usata come feature extractor, questa non viene allenata online, alla CNN, basata su VGG-M, sono aggiunti K rami ognuno dei quali domain-specific: questi sono classificatori binari responsabili di distinguere lo sfondo dall'oggetto che si vuole tracciare in uno specifico dominio (sequenza di frame). Per fare il tracking su un specifico video si usa uno di questi classificatori, quello allenato sul video o su un video della stessa scena. Se la scena è nuova è necessario allenare un nuovo classificatore[13].

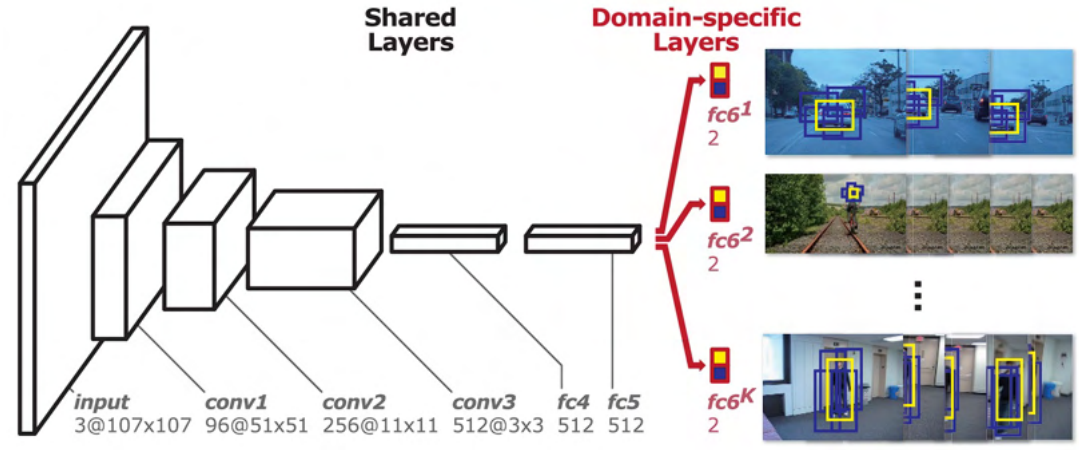

Figura 3.20: Schema che riassume l'architettura di MDNet [13].

ROLO

ROLO (Recurrent YOLO) è un online single object tracker che usa YOLO e lo estende per l'utilizzo in video per il tracciamento di oggetti utilizzando LSTM. Il modello da in input a YOLO i frame del video, YOLO restituisce i feature vectors del frame e i bounding boxes individuati, questi output sono concatenati e sono dati in input ad una rete LSTM che restituisce la posizione dell'oggetto tracciato[14].

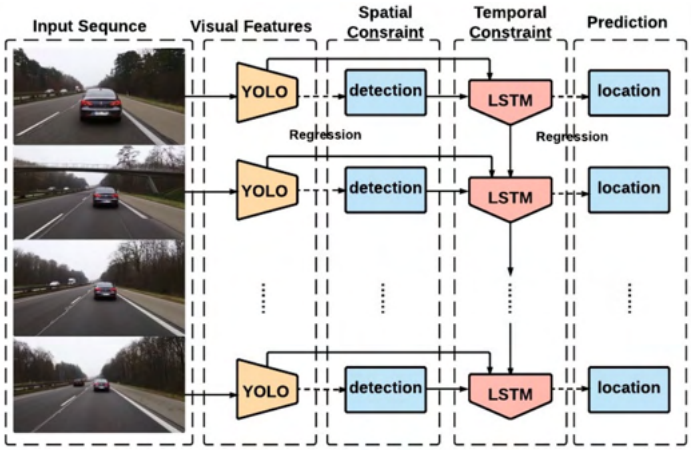

Figura 3.21: Schema che riassume il funzionamento di ROLO [14].

SORT e DeepSORT

SORT (Simple Online Real-time Tracking)[2] è un tracker che utilizza i bounding boxes individuati da un object detector ad ogni frame per fare il tracking di multipli oggetti. L'approccio usato è quello di interpretare il problema dell'object tracking come un problema di data association, ovvero il problema di associare detections (bounding boxes) tra un frame e il successivo. Per fare questo vengono usati due metodi classici che non fanno uso di reti neurali: i *Kalman Filters* per stimare la posizione dell'oggetto tracciato nel prossimo frame e *l'algoritmo ungherese* per associare detections a oggetti tracciati. SORT impiega quattro componenti principali:

- Un Object Detector: fornisce i bounding boxes ad ogni frame;
- Un Estimation Model: stima la posizione nel prossimo frame del target tracciato assumendo velocità costante e lineare. Lo stato di ogni target è rappresentato dalla tupla: $x = [u, v, s, r, u', v', s']^T$ dove:
  - u e v rappresentano le coordinate del centro dell'oggetto;
  - s e r rappresentano le dimensioni del bounding box;
  - u', v', s' rappresentano la velocità dell'oggetto.

  Quando una detection è associata ad un target (una nuova detection è considerata essere lo stesso oggetto già incontrato in frame precedenti) viene aggiornato lo stato del target, in particolare le componenti della velocità vengono aggiornate usando Kalman filter. Se non viene associata alcuna detection al target, lo stato viene aggiornato assumendo velocità lineare e costante;

- Un sistema per la data association: l'associazione di una detection ad un target avviene in 3 passaggi:
  - Si usa lo stato dei target per predirne le posizioni e i bounding boxes nel frame corrente;
  - Si computa una assignment cost matrix popolata con gli IoU calcolati tra ogni detection e ogni bounding box predetto nel passo precedente;
  - Si usa l'algoritmo ungherese per trovare un assegnamento detection-bounding box ottimale.

  Inoltre, per evitare associazioni spurie, si impone un threshold di IoU minimo da superare per poter esserci associazione.
- Un sistema di gestione del ciclo di vita dei target: ad ogni target è associata una traccia. Quando un target entra nella scena una nuova traccia dev'essere inizializzata e quando un target esce la corrispondente traccia dev'essere eliminata.

SORT ottiene buoni risultati sia in termini di precisione che di performance. Il fattore limitante dell'approccio è la qualità dell'object detection e la quantità di occlusioni di lunga durata. Per risolvere il problema delle occlusioni di lunga durata, o dell'ingresso ripetuto in scena dello stesso target, gli autori di SORT propongono DeepSORT.

DeepSORT, SORT with Deep association metric, associa ad ogni target un deep appearance descriptor, ovvero un feature vector che contiene informazioni sull'aspetto visivo del target ottenuto usando una CNN. Questo viene usato per l'associazione delle detection alle tracce già esistenti e aiuta a risolvere il problema della re-identificazione dei target.

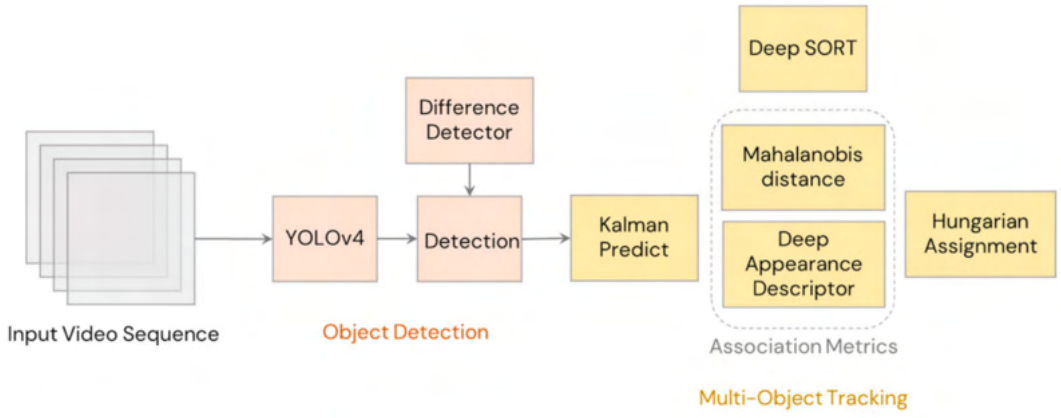

Figura 3.22: Schema che riassume il funzionamento di DeepSORT [15].

3.3.3 Object tracking con SORT

Il metodo di raccolta dei dati è stato scelto con l'intenzione di impedire l'identificazione delle persone quindi è ragionevole assumere che il meccanismo di associazione tramite appearance descriptor di DeepSORT non darebbe buoni risultati sul dataset di Alí. Per questa ragione, e per il fatto che i video, essendo raccolti dall'alto, non hanno particolari istanze di occlusioni, decido di usare SORT per il tracking dei target nei video, in particolare utilizzo l'implementazione di SORT pubblicata dagli autori dell'articolo originale.

SORT non richiede allenamento quindi non è necessario alcun processo di annotazione di dati e training. Per utilizzare SORT è sufficiente, per ogni frame: effettuare la detection, formattare i risultati, fornirli a SORT. Sviluppo quindi la seguente pipeline che descrivo in pseudocodice:

```
source = frames_from_video(video_ali)
for frame in source:
    detections = sort_format(yolo_model(masked_frame))
    person_tracks = sort(person_detections)
    cart_tracks = sort(cart_detections)
    write_to_file(person_tracks, cart_tracks)
```

Gli oggetti della classe SORT hanno tre parametri che vanno aggiustati per ottimizzare il tracciamento:

- `max_age` indica per quanti frame tenere viva una traccia alla quale non vengono associate nuove detection. Questo parametro limita le dimensioni in termini di numero di frame delle possibili occlusioni: se una nuova detection del target non viene fornita al tracker entro `max_age` frame, la traccia associata viene eliminata, se una nuova detection del target è fornita dopo `max_age` una nuova traccia viene inizializzata e l'id assegnato al target sarà diverso da quello precedente;

- `min_hits` indica il numero minimo di detection consecutive associate alla traccia prima di essere considerata completamente inizializzata. Una traccia prima di essere considerata inizializzata passa per una fase di pre-inizializzazione che dura `min_hits` detection e quindi almeno `min_hits` frame. Questa fase serve a non inizializzare tracce per detection erronee che avvengono per un numero piccolo di frame successivi;

- `iou_thresh` threshold minimo di IoU da superare per poter associare una detection ad una traccia usando l'algoritmo ungherese.

Prima di operare il fine-tuning di questi parametri devo considerare alcune limitazioni dell'implementazione di SORT che uso, infatti questa implementazione non mi permette di:

1. Instanziare due tracker diversi, uno per persone ed uno per carrelli: infatti l'id associato alle tracce è salvato in una variabile globale che risulta condivisa da ogni oggetto della classe SORT;

2. Usare l'ultimo id (id massimo) per stimare il numero di target apparsi nella scena: infatti l'id, che non è altro che un contatore, viene incrementato durante la pre-inizializzazione delle tracce, non una volta che le tracce superano `min_hits` detection. Quindi l'id dell'ultimo target apparso, ovvero l'id massimo, è sempre di molto maggiore al numero di target effettivamente apparsi nella scena.

Per superare queste limitazioni decido di effettuare delle opportune modifiche all'implementazione originale, in particolare, per risolvere i rispettivi problemi:

1. Sostituisco alla variabile globale che funge da id una lista di variabili globali. Alla dichiarazione di un oggetto della classe SORT, il costruttore si aspetterà un parametro aggiuntivo ai tre menzionati in precedenza: un indice che indica all'oggetto quale degli elementi della lista adoperare come id. In questo modo posso dichiarare un oggetto tracker per persone ed uno per carrelli;

2. Non modifico il comportamento di pre-inizializzazione degli id, perché necessario al funzionamento di SORT, invece, considero gli id già usati da SORT come delle variabili interne, usate dall'algoritmo per funzionare, e aggiungo un altra lista di id, che fungerà da variabili esterne, che vengono incrementate solo dopo che una traccia ha superato `min_hits` e che vengono ritornati al posto degli id interni.

Con queste modifiche ottengo il comportamento che desidero da SORT e passo al fine-tuning sperimentale dei parametri. In una prima fase tento un approccio grid-search per trovare i valori ottimali dei parametri entro un range predefinito con incrementi predefiniti. Questo approccio richiede troppo tempo quindi passo ad un testing di valori scelti ragionando sul senso dei parametri.

Non avendo a disposizione ground-truth sulle sequenze per poter calcolare metriche usate in letteratura, valuto la bontà dei risultati confrontando l'id massimo dei target e il numero di target apparsi nel video che ho manualmente prodotto per mezzo di un semplice conteggio nella fase di costruzione del dataset. Inoltre valuto il tracker visionando dei video campione sui quali applico l'object tracker e visualizzo i risultati. I valori migliori che ho trovato per i parametri di SORT sono i seguenti:

- `max_age = 40` con un frame rate di circa 30 fps, questo valore indica che un target può non essere associato ad una detection per circa 1.3 secondi prima che la sua traccia venga eliminata. Questo valore così alto è giustificato dal fatto che quando gli operatori si chinano per prendere la merce dalla porzione inferiore degli scaffali, la detection diventa difficile in quanto la distanza dalla telecamera si avvicina al limite definito per non rilevare il pavimento;

- `min_hits = 30` questo valore rende il tracker robusto a classificazioni errate da parte dell'object detector. Particolarmente nei momenti di entrata e uscita dalla scena, l'object detector è prono ad errori di classificazione e poiché gli operatori stanno raccogliendo merce dagli scaffali, capita che si fermino, specialmente in entrata nella scena sul bordo del frame ed entrino ed escano ripetutamente dalla scena, questo valore aiuta a non inizializzare tracce che durano pochi frame per queste ragioni;

- `iou_thresh = 0.1` il valore di questo parametro è stato determinato tramite trial-and-error. Valori più alti anche di pochi decimali risultano nell'inizializzazione di un numero maggiore di tracce. La mia ipotesi è che l'assunzione di velocità costante e lineare che viene fatta per predire la posizione del nuovo bounding box non sia completamente corretta e che perciò ci sia bisogno di un alta tolleranza nei confronti di questi errori per permettere il tracciamento dei target.

3.3.4 Risultati

Per verificare il miglioramento apportato dalle modifiche fatte a SORT calcolo, per ogni video, l'errore assoluto tra numero di persone conteggiato manualmente e id massimo restituito dal tracker, infine per ottenere il MAE (mean absolute error) sommo gli errori e divido per il numero di video.

Classi	MAE SORT (orig.)	MAE SORT (mod.)	Num. Target medio
Persone	3.86	0.93	3.97
Carrelli	1.77	0.19	1.31

Tabella 3.2: MAE calcolati tra numero di target apparsi e id massimo restituito dai tracker.

Con:

$$MAE_{alg} = \frac{1}{n}\sum_{i=0}^{n} |GT_{num\_target}(i) - alg_{max\_id}(i)| \tag{3.5}$$

Dove n è il numero di video nel dataset, $GT_{num\_target}(i)$ è il numero di target (persona o carrello) conteggiato manualmente apparsi nel video i, $alg_{max\_id}(i)$ è l'id massimo restituito dal tracker per lo stesso target di GT sul video i.

Osservando le visualizzazioni è apparente che gli errori avvengono principalmente in due occasioni:

- Quando il target si sofferma all'ingresso o all'uscita dalla scena: in diversi casi gli operatori si fermano con il carrello in prossimità del bordo della scena, in questi casi si possono avere errori nella classificazione degli oggetti e nell'inizializzazione delle tracce a causa della sequenza di ingressi e uscite dalla scena;

- Quando gli operatori si chinano per raccogliere la merce nella porzione inferiore degli scaffali: infatti in questa situazione spesso la sagoma dell'operatore scompare e la detection non è possibile. In alcuni di questi casi, il meccanismo che mantiene le tracce anche per max_age frame dopo l'ultima detection del target previene l'errore, in altri l'operatore rimane chinato per più di max_age frame e quando si rialza gli viene assegnato un nuovo id.

Parte della differenza tra l'errore sul numero di persone e sul numero di carrelli è da ricondurre al fatto che le persone si muovono nella scena in maniera più erratica, entrano ed escono dal frame più frequentemente dei carrelli che invece si muovono

in maniera più lineare. Questo comporta che per le persone il tracker perda e re-inizializzi più tracce.

Per quanto riguarda la successiva fase di trajectory prediction, questo genere di errori è gestibile: è sufficiente filtrare la tracce andando ad selezionare quelle lunghe abbastanza per permettere la prediction, un altra possibilità è quella di utilizzare la tracce raccolte nella porzione più centrale della scena.

Per quanto dovesse riguardare una successiva applicazione effettivamente operativa all'interno del magazzino, ad esempio a supporto di sistemi automatici di stoccaggio e raccolta della merce, verrebbero utilizzate diverse telecamere i cui campi visivi si possono sovrapporre e dai quali sarebbe possibile ricostruire una macro scena rappresentante l'intero magazzino. In un contesto simile errori di entrata e uscita dalla scena verrebbero meno. Per quanto riguarda errori legati all'occlusione degli operatori quando si chinano, questi sono plausibilmente correggibili effettuando una calibrazione e/o una installazione diversa delle telecamere che permetta di separare il pavimento da oggetti anche a poche decine di centimetri di altezza.

Infine, un esempio del contenuto dei file prodotti per l'utilizzo con modelli di trajectory prediction:

```
30,0,1,1184,339,1119,283,130,113
30,1,1,950,335,780,261,340,149
31,0,1,1184,344,1120,288,129,113
31,1,1,949,335,780,261,339,149
32,0,1,1179,349,1114,293,131,113
32,1,1,949,335,781,261,337,148
33,0,1,1175,355,1109,299,133,112
33,1,1,948,335,780,261,336,148
```

Il formato usato è: $frame\_id$, $class\_id$, $target\_id$, $x1$, $y1$, $x2$, $y2$, w, h.

In questo caso abbiamo il tracciamento di due target, un carrello e una persona, i primi a comparire nella scena, per tre frame.

Visualizzazioni

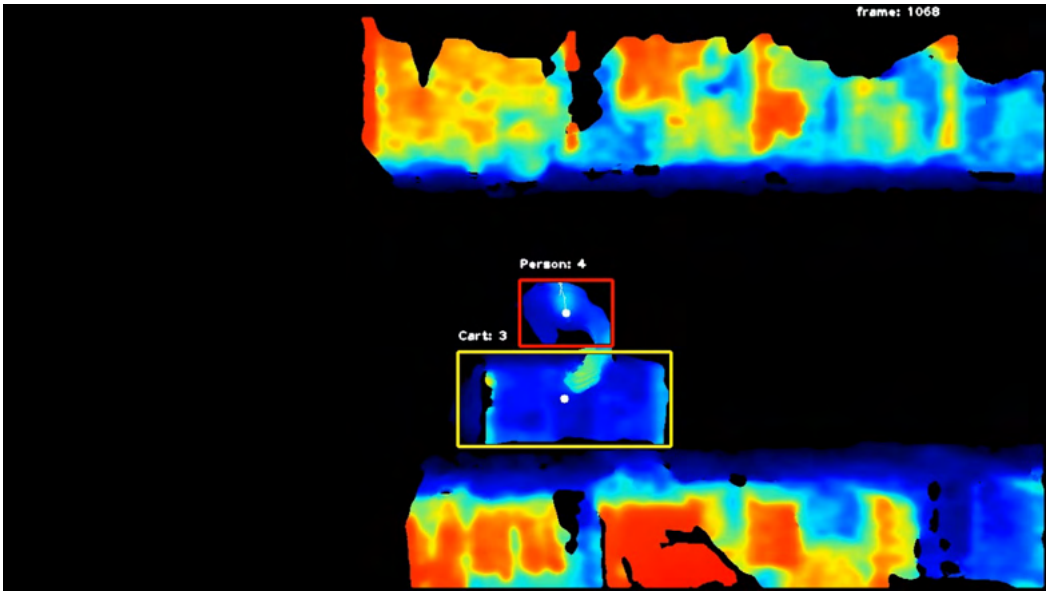

Figura 3.23: Visualizzazione dei risultati dell'object tracking. Associo colori diversi a bounding boxes di target di classi diverse: rosso per le persone, giallo per i carrelli. L'uso di colori diversi aiuta a comprendere i risultati quando le label non sono visibili. Le label indicano la classe del target e il suo id. Il centro dei bounding boxes è indicato da un cerchio bianco seguito da una scia che rappresenta le posizioni nei precedenti 60 frame del centro del bouning box corrispondente.

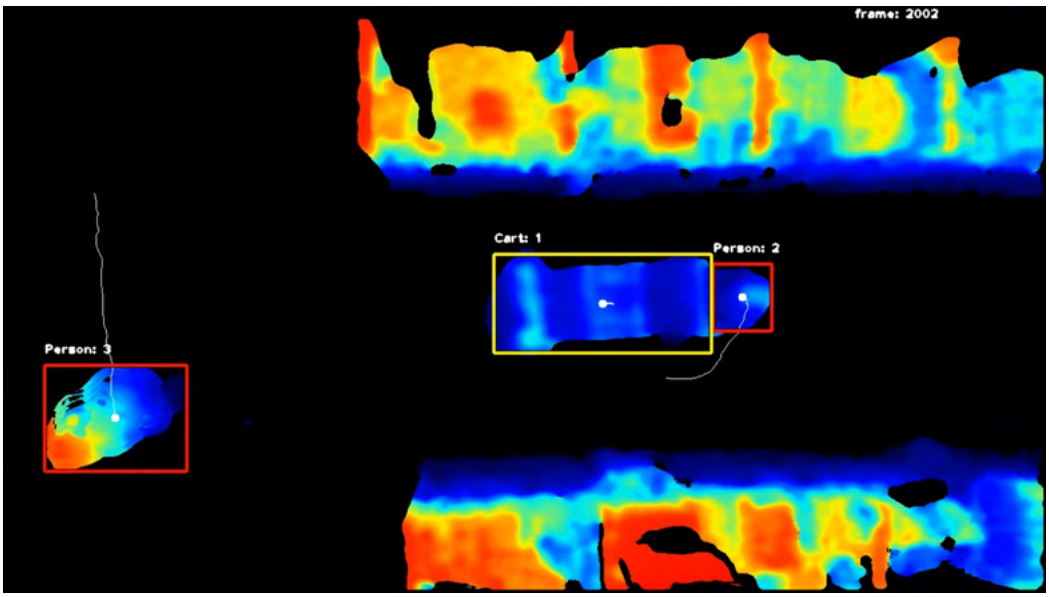

Figura 3.24: Un altra visualizzazione, è possibile notare le differenze in traiettoria passata tra l'operatore e il carrello. Questa immagine da un idea della dinamica della scena: generalmente il carrello segue una traiettoria più lineare: viene portato in scena, eventualmente accostato su un lato, l'operatore invece segue traiettorie più articolate per raccogliere la merce.

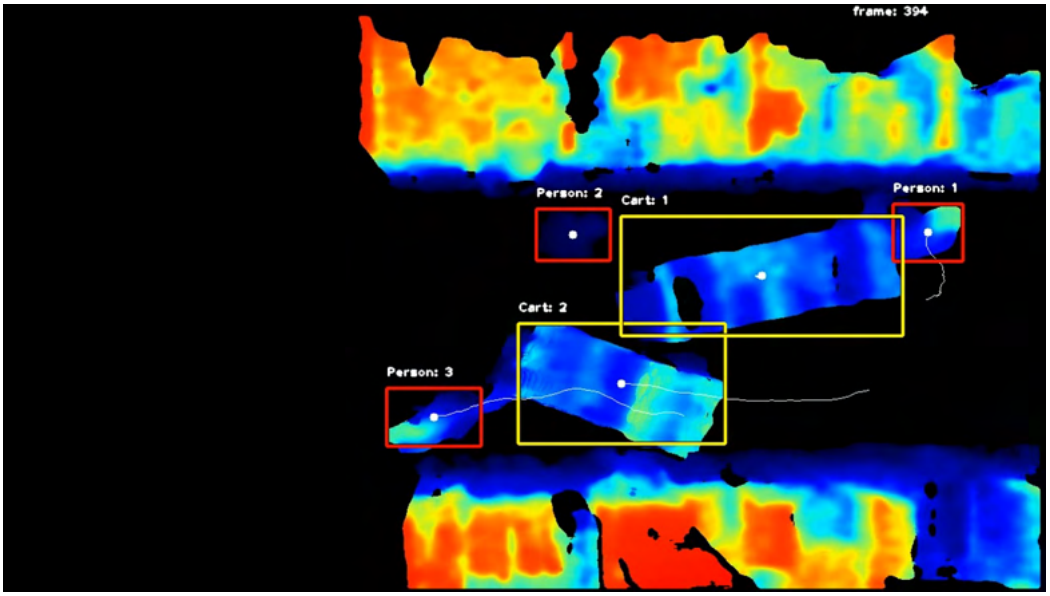

Figura 3.25: La stessa scena mostrata in figura 2.6

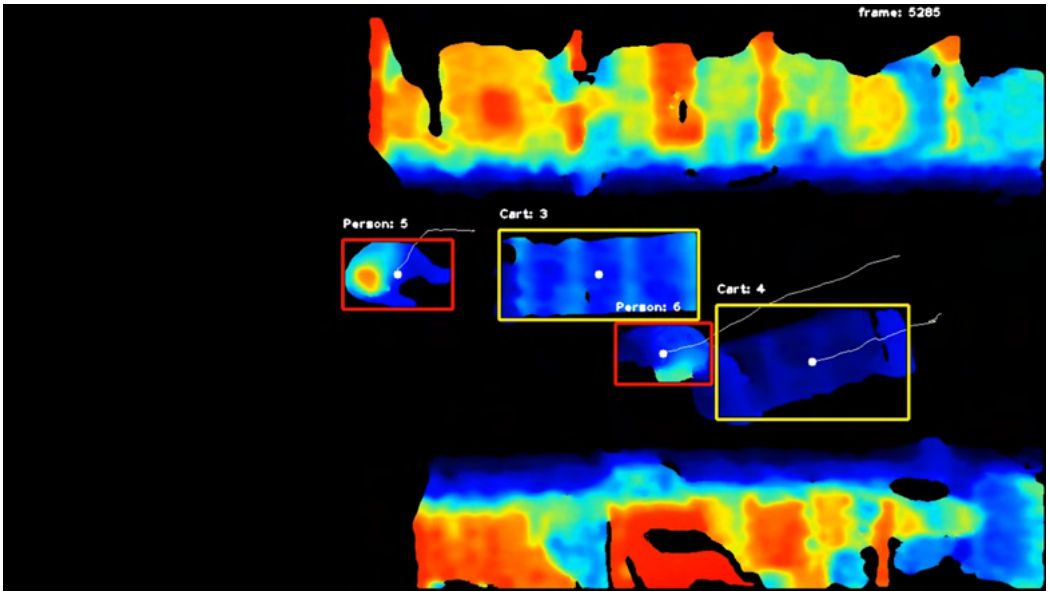

Figura 3.26: Un altra visualizzazione di una scena con soggetti multipli. Notare gli id crescenti dei target appena entrati nella scena dal lato destro dell'immagine.

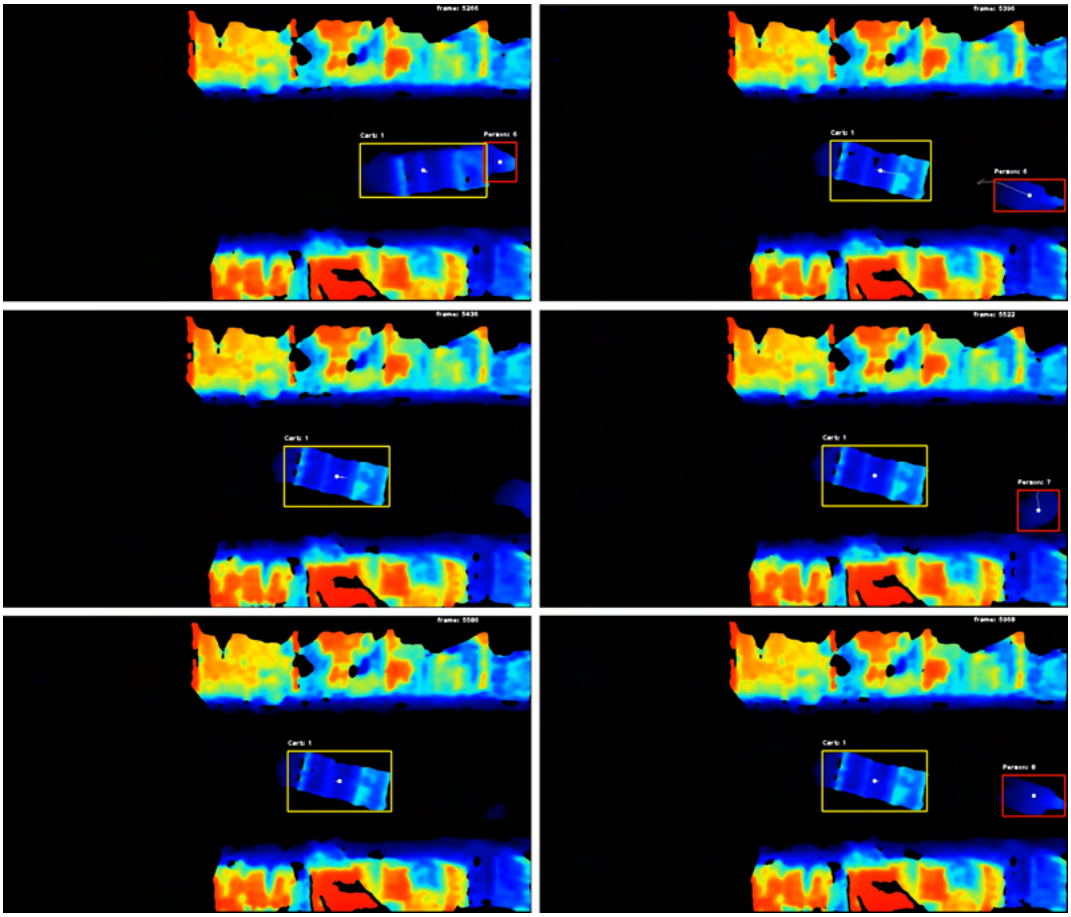

Figura 3.27: Una sequenza di frame (non direttamente successivi, notare il numero di frame in alto a destra) che esemplifica un fallimento del tracker. Nei primi due frame vediamo il tracciamento corretto di carrello e operatore. Nel terzo frame vediamo che l'operatore si posiziona sul bordo della scena, il tracker perde la traccia dell'operatore. L'operatore rientra nella scena nel quarto frame e gli viene assegnato un nuovo id. Nel quinto frame l'operatore si è abbassato per raccogliere della merce. Nel sesto frame l'operatore si è rialzato, gli viene assegnato un nuovo id perchè la traccia è stata per persa per più di max_age frame.

Capitolo 4

La predizione

4.1 Obiettivi

Gli obiettivi principali della fase di predizione sono:

1. Adattare i dati per l'input ai modelli di predizione delle traiettorie;
2. Fare predizione short-term: date otto posizioni consecutive predire le prossime dodici;
3. Provare alcuni semplici modelli per determinare un punto di partenza in termini di metriche.

4.1.1 Definizione del problema

La predizione di traiettorie consiste nel, data una traiettoria parziale fare previsione a corto o a lungo termine delle future posizioni dell'oggetto tracciato. Nella sua forma più semplice viene considerato un oggetto alla volta e l'unico input dato al modello responsabile della predizione è una traiettoria parziale di un oggetto. Il modello quindi non ha a disposizione informazioni sulla presenza di altri soggetti in movimento nella scena, ne informazioni semantiche sulla natura della scena (presenza di ostacoli, luoghi che attraggono l'interesse, etc.). I modelli che fanno questa forma semplicistica di predizione della traiettoria stanno effettivamente facendo predizione di serie temporali. Nello stato dell'arte si vanno a considerare aspetti ulteriori per produrre previsioni più affidabili, appunto: informazioni semantiche sulla scena e interazione tra soggetti in movimento. Nell'ambito di questo progetto faccio predizione delle traiettorie usando solo dati relativi a traiettorie passate e ignorando questi aspetti più avanzati. L'obiettivo è quello di determinare un minimo di prestazioni a livello di metriche con cui poter confrontare modelli più complessi.

4.1.2 Stato dell'arte

La letteratura sulla predizione delle traiettorie è estensiva. In base agli obiettivi della ricerca si sono sviluppati modelli diversi: predire la traiettoria di un veicolo date informazioni sulla strada è più semplice di predire le traiettorie di persone in un area aperta come ad esempio un parco. In questa sezione considero il compito specifico di

predire la traiettoria di esseri umani e in particolare vado a considerare due modelli: SocialLSTM e Ynet.

SocialLSTM

SocialLSTM è un modello basato sulle LSTM che ha come obiettivo quello di apprendere le regole sociali a cui obbediamo inconsciamente quando ci muoviamo in spazi affollati, come ad esempio rispettare lo spazio personale altrui, o non passare tra due persone che si stanno parlando. L'apprendimento del comportamento umano durante le interazioni migliora la predizione delle traiettorie ed è in contrapposizione a modelli precedenti in cui queste regole sociali sono codificate manualmente dai ricercatori.

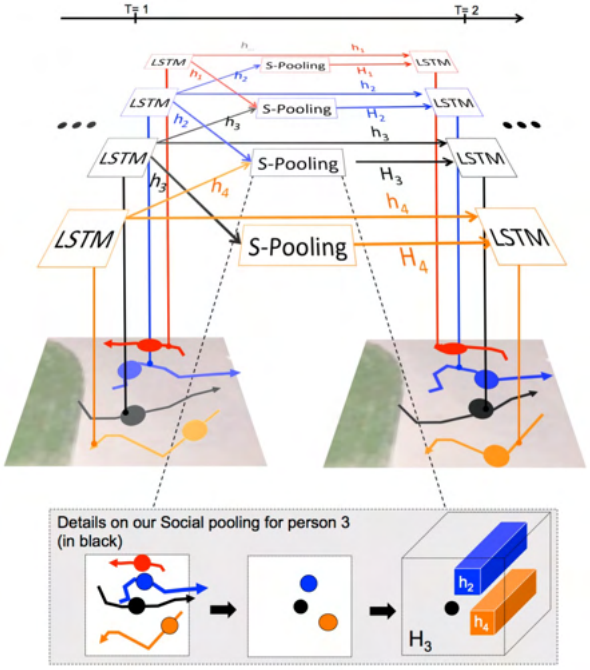

Figura 4.1: Schema che riassume il funzionamento di SocialLSTM[1].

Per modellare ogni traiettoria della scena viene usata una LSTM diversa, questo perché persone diverse hanno andature, velocità e tendenze diverse. LSTM, un tipo di rete neurale ricorrente, è ideale per apprendere le caratteristiche di sequenze di dati in cui c'è correlazione tra dato precedente e successivo. Affinchè le LSTM possano prendere in considerazione la presenza degli altri soggetti e le loro rispetti traiettorie, le LSTM sono collegate tramite un social pooling layer, questo in particolare avviene per traiettorie adiacenti e modella la presa in considerazione degli altri pedoni da parte di ogni pedone. Il social pooling layer permette alle LSTM di condividere il loro stato interno e quindi elaborare in maniera condivisa le traiettorie per prevedere le future posizioni[1].

Ynet

L'approccio di Ynet alla predizione di traiettorie è multimodale: invece di produrre una singola predizione, ne vengono prodotte multiple. In particolare, la multimodalità

dell'approccio è relativa a:

- Le destinazioni finali dei pedoni;
- I possibili percorsi per giungere ad ognuna delle possibili destinazioni.

L'approccio usato da Ynet è quello di predire prima K_e possibili destinazioni, e in base a queste, predire K_a possibili percorsi per ognuna. Inoltre, Ynet tiene in considerazione la semantica della scena analizzata e fa predizioni fino ad un minuto nel futuro [12].

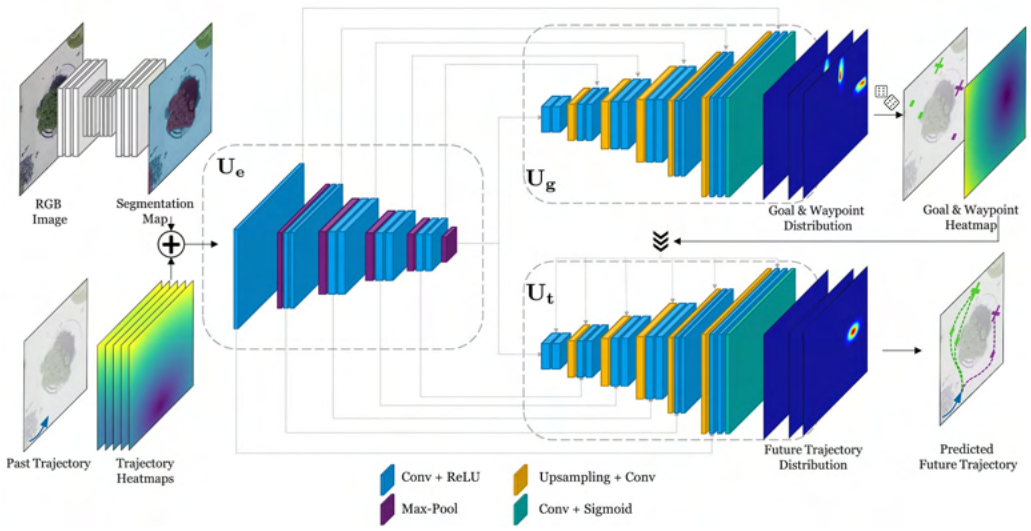

Figura 4.2: Architettura di Ynet[12].

4.1.3 Trajectory Prediction

Per ottenere dei risultati preliminari sul dataset di Alí ho usato 4 modelli diversi:

- Zero_Vel: un modello che produce una serie di traiettorie casuali;
- Const_Vel: un modello che assume velocità costante e produce una serie traiettorie lineari in direzioni leggermente diverse.
- LSTM: un semplice modello basato su LSTM.
- Transformer: un semplice modello basato su Transformer.

LSTM (Long Short-Term Memory) e Transformer sono modelli per sequenze, pensati per applicazioni in cui gli input sono sequenze ordinate (testo, video, audio, etc.) e l'ordine contiene informazione. Per lavorare con sequenze un modello ha bisogno di una sorta di memoria che gli permetta di usare in input informazioni relative a k elementi precedenti al corrente della sequenza.

LSTM

Le RNN (Recurrent Neural Network) sono tra i primi modelli neurali sviluppati per le sequenze. Una RNN è caratterizzata da un *hidden state* che viene aggiornato per ogni elemento della sequenza e dipende dall'input corrente e dall'hidden state precedente. L'output della rete dipende dall'input corrente e dall'hidden state. LSTM espande il modello RNN con degli elementi chiamati *gates* che permettono al modello di regolare quali informazioni ricordare/dimenticare.

Transformer

Transformer è un modello che processa sequenze usando un meccanismo attentivo che permette al modello di usare qualsiasi sezione della sequenza di input e di pesarne l'importanza per il calcolo dell'output. Mentre inizialmente i meccanismi attentivi erano usati in congiunzione con la ricorrenza tipica di RNN e LSTM, [21] ha introdotto transformer: un modello basato sull'architettura encoder-decoder e mostrato che il solo meccanismo attentivo era in grado di ottenere prestazioni superiori ai vecchi modelli.

Dataset

Facendo object tracking ho prodotto, a partire da ogni video, un file che descrive le traiettorie come illustrato in §3.3.4. Per facilitare l'uso dei modelli e confrontare i risultati, distinguo chiaramente tra traiettorie di persone e carrelli. Il dataset creato per fare il tracciamento delle traiettorie è costituito da:

	Persone	Carrelli
Numero di scene	72	59
Numero di punti	182144	133521
Numero di target	388	125
Numero di frame	1565565	125882

Tabella 4.1: Statistiche del dataset per la trajectory prediction.

Rispetto al numero totale di 94 video, le scene contenti persone o carrelli sono di meno, perché non ho elaborato otto video raccolti il primo giorno per calibrare la telecamera e perché alcuni dei video raccolti sono vuoti. Rispetto al numero di istanze di persone: 375, e di carrelli: 119, i numeri di target sono maggiori per le ragioni di perdita e re-acquisizione della traccie spiegate in §3.3.4, inoltre, i numeri sono diversi perché nel conteggio includo solo tracce con abbastanza punti per fare short-term prediction.

LSTM e Transformer sono modelli di apprendimento automatico e richiedono training. Per produrre dei modelli specializzati e per rendere il compito più facile ai modelli, alleno i modelli separatamente su traiettorie di persone e di carrelli. Per quanto riguarda Zero_Vel e Const_Vel sono modelli molto semplici che non richiedono training e li uso per determinare a livello di metriche delle prestazioni minime per verificare che i modelli più complessi stiano funzionando.

Il protocollo di predizione delle traiettorie short-term prevede che al modello vengano fornite otto posizioni del target e che il modello predica le dodici posizioni successive. Se fornissi ai modelli le tracce prodotte direttamente dall'object tracker, le tracce osservate e quelle prodotte rappresenterebbero spostamenti nello spazio molto ridotti perché la raccolta dati è avvenuta a 30 fps. Quindi è necessario fare del downsampling, ovvero fornire al modello una posizione ogni n frame, dove scelgo di usare 5 come valore per il downsampling rate n.

Training

Per l'allenamento di LSTM e Transformer uso la stessa macchina tesisti usata per allenare YOLOv5m. Allo stesso modo faccio il logging delle metriche del training usano Weights & Biases.

Per quanto riguarda la data augmentation, sempre usando albumentations applico le seguenti trasformazioni:

- ShiftScaleRotate: randomicamente trasla, cambia la scala e ruota i dati, applico questo metodo eliminando la possibilità di cambiare scala e di ruotare;
- Rotate: ruota randomicamente i dati;
- Affine: applica la trasformazione specificata, in particolare uso lo shear una trasformazione che cambia la prospettiva.

Per il training uso batch_size pari a 128 ed effettuo l'allenamento per 300 epoche.

Metriche

Per valutare le performance di un modello di trajectory prediction, due semplici metriche utilizzate anche in letteratura sono:

ADE (Average Displacement Error) ovvero la distanza $L2$ media, calcolata tra ogni punto di traiettoria predetta e traiettoria effettiva, formalmente:

$$ADE = \frac{1}{n}\sum_{1}^{n}\sqrt{(x_i - \hat{x}_i)^2 + (y_i - \hat{y}_i)^2} \tag{4.1}$$

FDE (Final Displacement Error) ovvero la distanza $L2$, calcolata tra l'ultimo punto della traiettoria predetta e l'ultimo punto della traiettoria effettiva, formalmente:

$$FDE = \sqrt{(x_n - \hat{x}_n)^2 + (y_n - \hat{y}_n)^2} \tag{4.2}$$

Dove n è il numero di punti predetti (nel protocollo short-term sono 12), x_i, y_i sono le coordinate ground-truth dell'i-esimo punto, $\hat{x}_i$, $\hat{y}_i$ sono le coordinate dell'i-esimo punto della traiettoria predetta dal modello.

4.1.4 Risultati

L'applicazione dei quattro modelli per la predizione delle traiettorie da i seguenti risultati in termine di metriche:

Modello	ADE (persone)	ADE (carrelli)	FDE (persone)	FDE (carrelli)
Zero_Vel	72.95	33.09	100.41	39.67
Const_Vel	89.43	15.87	182.35	31.34
LSTM	52.24	9.42	74.64	17.47
Transformer	60.42	9.17	94.75	14.02

Tabella 4.2: Risultati dei quattro metodi di trajectory prediction utilizzati.

Le distanze calcolate sono in termine di pixel. Quindi, ad esempio, l'ultimo punto della traiettoria predetta da transformer per i carrelli è in media a 14.02 pixel di distanza dal punto effettivo. Per avere un intuizione di quanto grande sia questo errore, considero che nelle immagini raccolte, un carrello è assimilabile ad un rettangolo di circa 300×100 pixel, una persona invece ad una circonferenza di 90 pixel in raggio.

I risultati mostrano che, in generale, predire la traiettoria dei carrelli è più facile di predire la traiettoria delle persone, infatti, i carrelli seguono delle traiettorie più lineari, mentre gli operatori, dovendo raccogliere la merce, seguono traiettorie difficili da prevedere utilizzando solo le posizioni passate.

Il fatto che per le persone, il modello che assume velocità costante, $Const\_Vel$ risulti in un errore maggiore rispetto al modello che produce traiettorie casuali, $Zero\_Vel$, suggerisce che l'assunzione di velocità costante e lineare non sia particolarmente corretta e che la traiettoria delle persone è più complessa.

I modelli neurali, LSTM e Transformer hanno risultati simili tra loro e non sono particolarmente buoni, in particolare per le persone.

I risultati ottenuti presentano un grande margine di miglioramento: modelli più complessi che sfruttano ulteriori informazioni sulla scena possono sicuramente ottenere risultati migliori. In particolare, i risultati sono migliorabili sfruttando:

- Informazioni semantiche sulla scena: introdurre nei modelli la nozione di ostacolo statico, in questo caso gli scaffali;

- Informazioni sociali: introdurre nei modelli le traiettorie degli altri target presenti nella scena: in molte delle traiettorie l'operatore deve fare il giro del carrello per raccogliere merce o per spingerlo o trascinarlo, in altre operatori interagiscono o si evitano. I modelli che ho utilizzato non hanno alcuna nozione di queste informazioni e pertanto non possono sfruttarle per prevedere dei palesi cambi di traiettoria;

- Informazioni su possibili zone di interesse: gli scaffali sono delle zone di interesse per gli operatori e possono essere interpretati come delle possibili destinazioni per le persone;
- Informazioni sulla posa di persone e carrelli: informazioni sulla direzione in cui guardano carrelli e specialmente le persone possono aiutare nella predizione della loro traiettoria, infatti capita che gli operatori si fermino e si girino sul posto, un modello che tiene in considerazione la posa sa che plausibilmente la traiettoria riprenderà nella nuova direzione in cui guarda la persona.

I modelli che ho utilizzato non tengono conto di nessuna di queste informazioni, è ragionevole pensare che modelli che lo fanno possano ottenere risultati molto migliori.

In conclusione, i risultati ottenuti sono dei buoni risultati di partenza, utili per il confronto con modelli più complessi che hanno il potenziale di produrre delle predizioni accurate.

Visualizzazioni

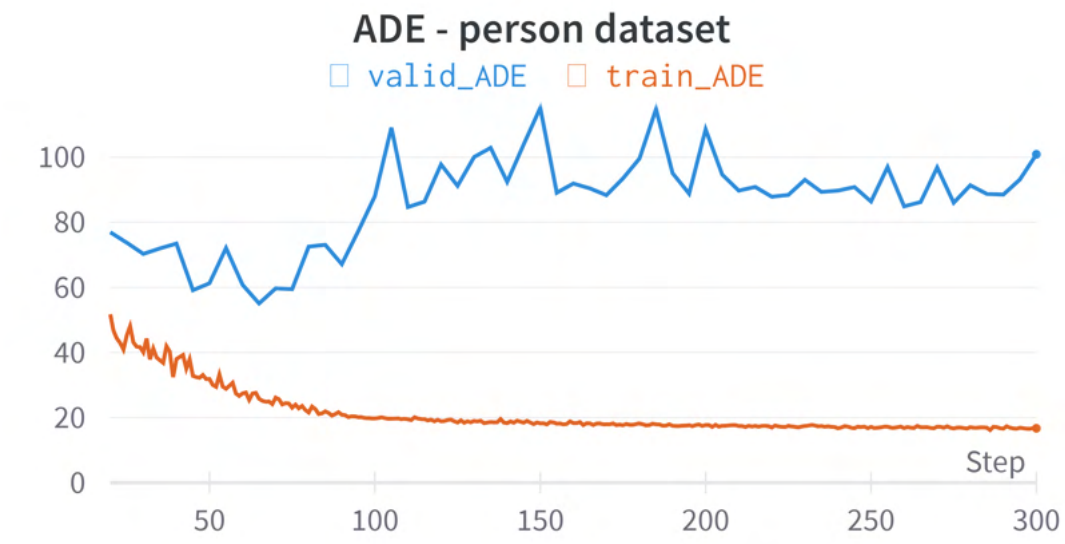

Figura 4.3: Plot di ADE (Average Displacement Error) durante il training di $Transformer$ su traiettorie di persone. Si può notare che l'errore sui dati di validazione aumenta nonostante scenda invece sui dati di training. Questo è un indicatore di overfitting: il modello non riesce ad apprendere dai dati e invece "memorizza" gli esempi del training set. Lo stesso plot per LSTM è analogo.

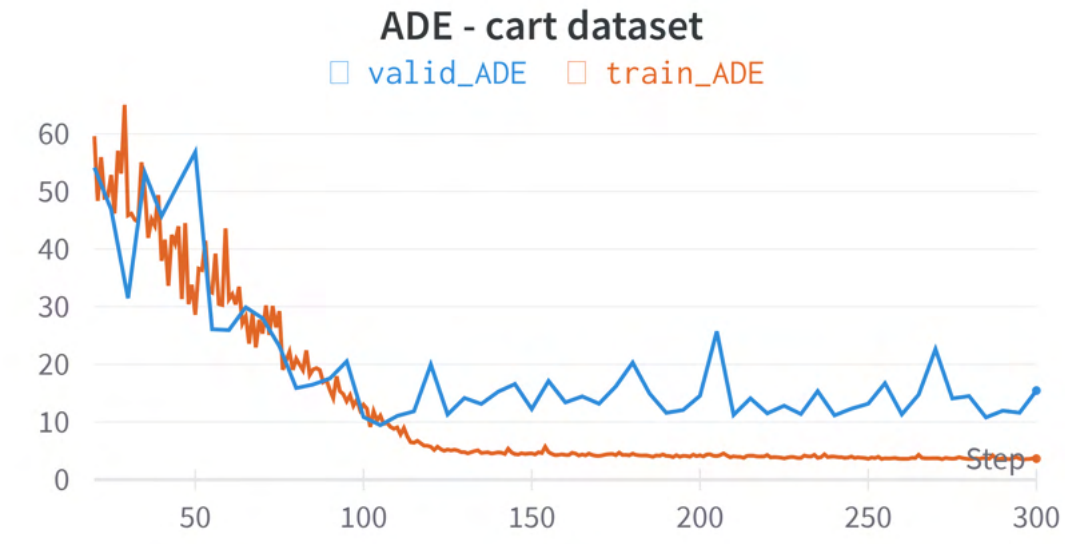

Figura 4.4: Plot di ADE (Average Displacement Error) durante il training di transformer su traiettorie di carrelli. A differenza del precedente plot, l'errore sui dati di validazione, nonostante sia poco stabile, non aumenta.

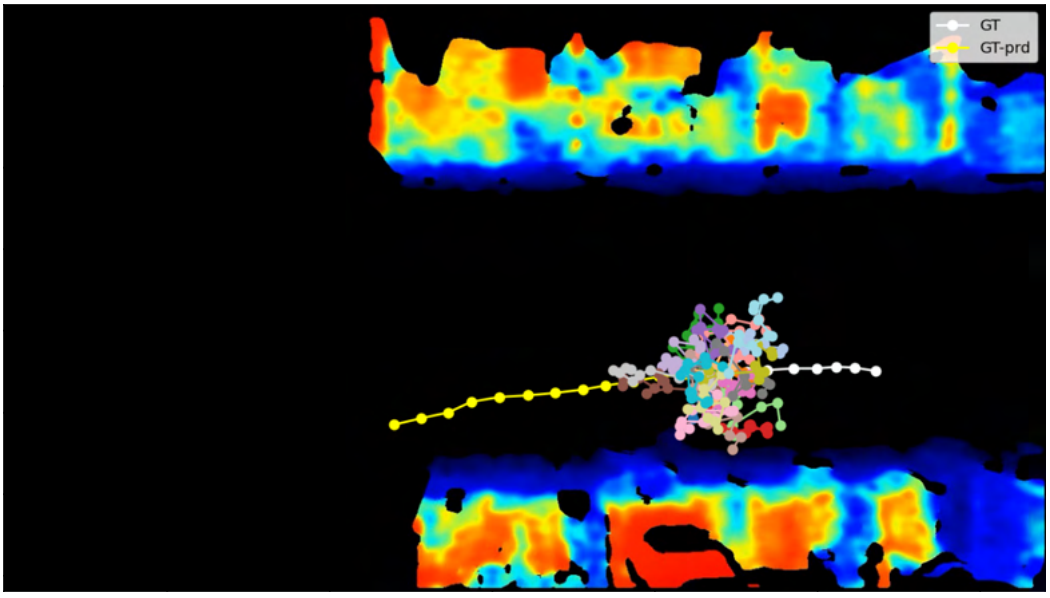

Figura 4.5: Una visualizzazione delle traiettorie predette dal modello $Zero\_Vel$. In bianco la porzione di traiettoria data in input al modello per predire la porzione gialla. In questo caso è possibile vedere che la traiettoria è lineare e la previsione casuale non è affatto accurata.

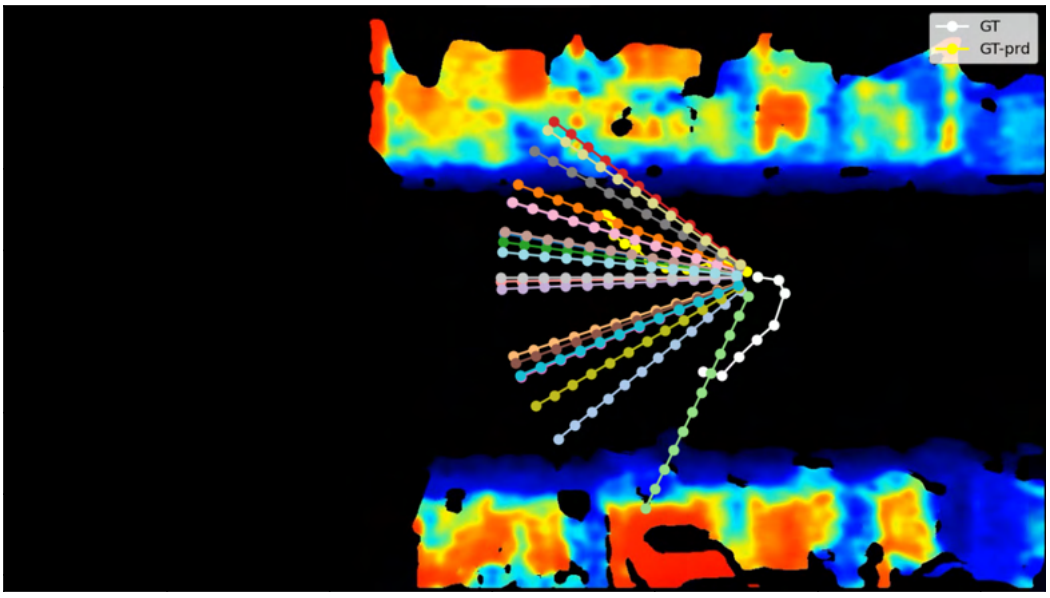

Figura 4.6: Una visualizzazione delle traiettorie predette dal modello $Const\_Vel$. In questo esempio è possibile notare quanto repentino possa essere il cambio di traiettoria di una persona.

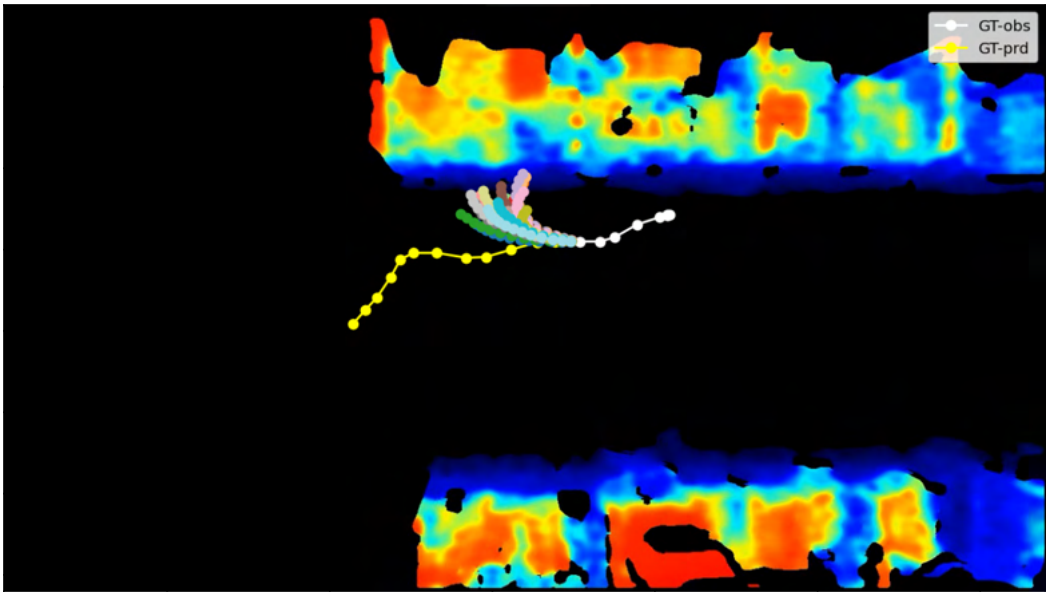

Figura 4.7: Una visualizzazione delle traiettorie predette dal modello *Transformer* su una traiettoria di una persona. Si può notare che il modello ha appreso che gli scaffali sono di interesse per gli operatori e che il modello abbia assunto che l'operatore si dirigesse lì.

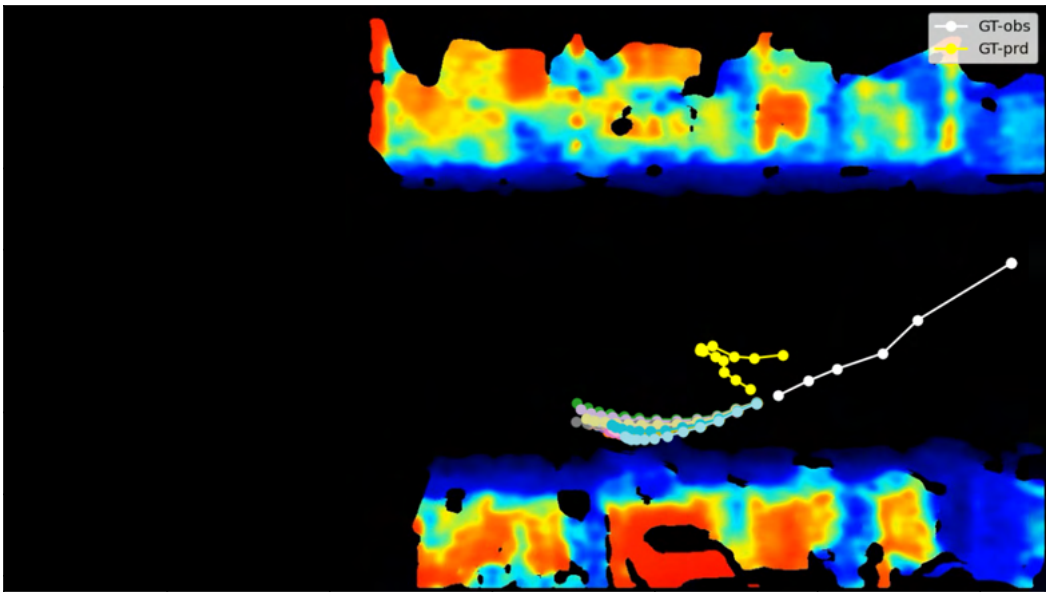

Figura 4.8: Un altra visualizzazione delle traiettorie predette dal modello *Transformer* su una traiettoria di una persona. Questa è una traiettoria che da un esempio chiaro delle ragioni che portano alla difficoltà nella predizione.

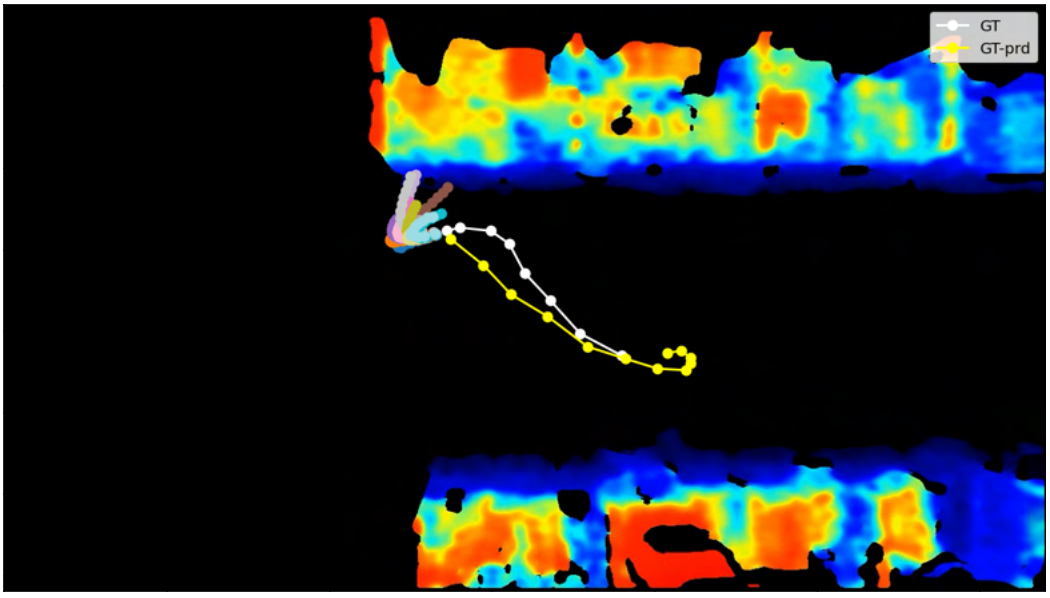

Figura 4.9: Una visualizzazione delle traiettorie predette dal modello *LSTM* su una traiettoria di una persona. Questa traiettoria mostra che il modello basato su *LSTM* ha appreso l'importanza degli scaffali e inoltre mostra che le traiettorie delle persone senza contesto (dove si trova il carrello?, l'operatore ha già raccolto la merce?) sono difficili da prevedere.

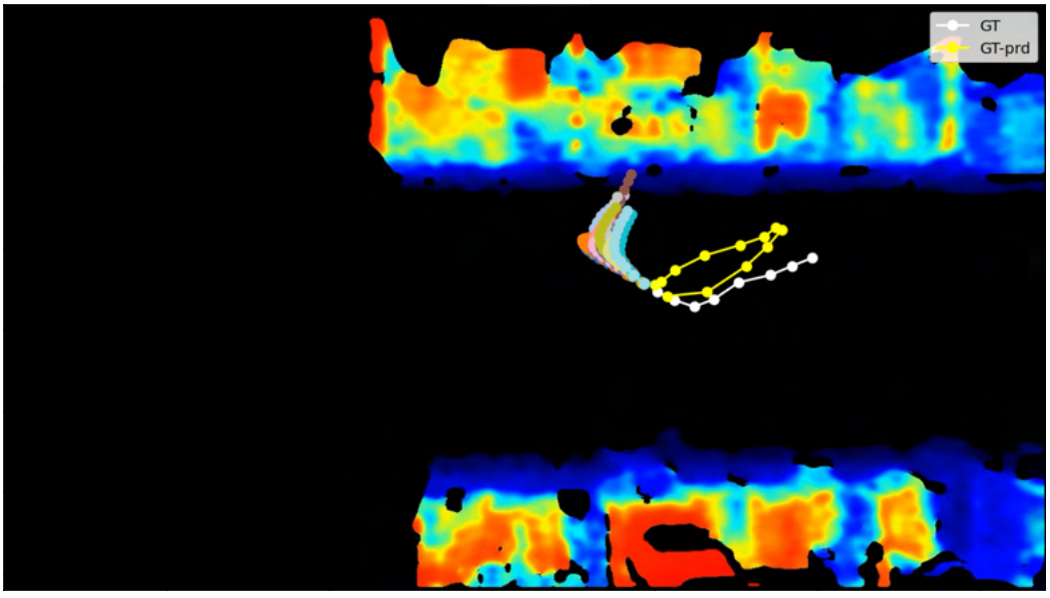

Figura 4.10: Un altra visualizzazione delle traiettorie predette dal modello *LSTM* su una traiettoria di una persona.

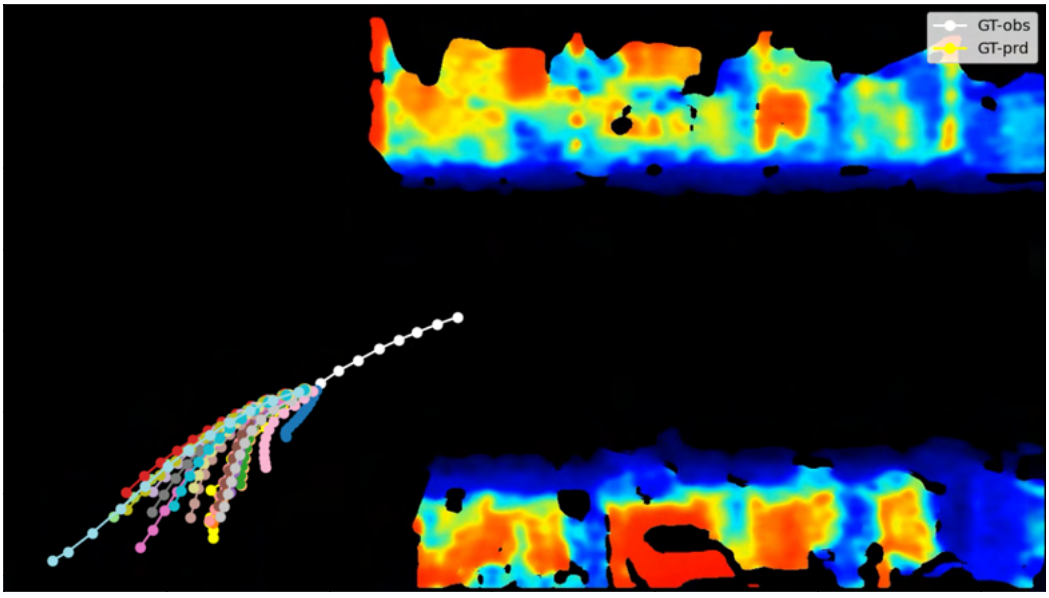

Figura 4.11: Una visualizzazione delle traiettorie predette dal modello $Transformer$ su una traiettoria di un carrello. Si può notare che questa traiettoria è molto più lineare di quelle illustrate in precedenza.

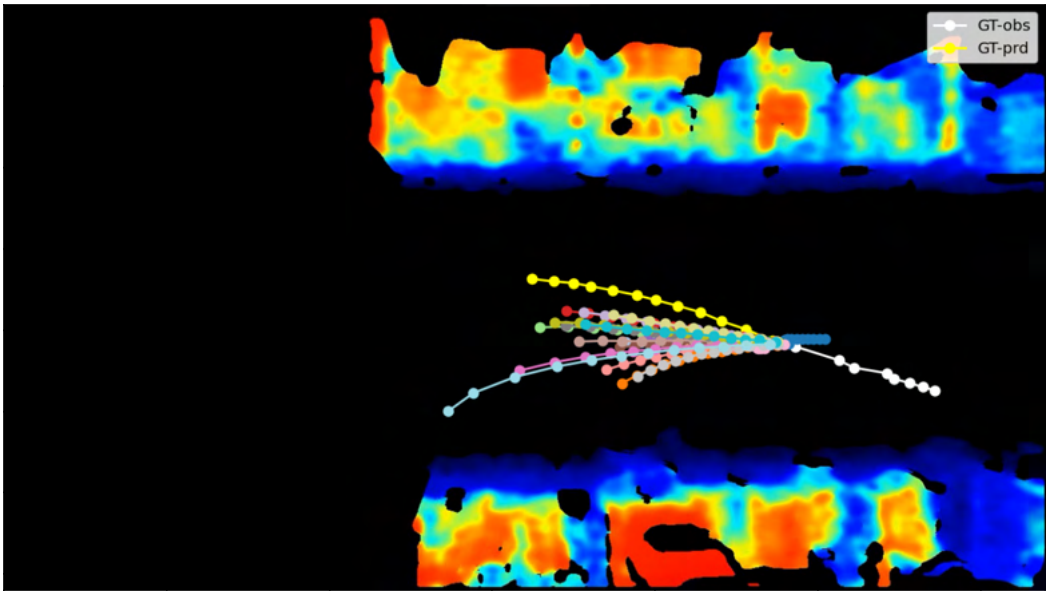

Figura 4.12: Un altra visualizzazione delle traiettorie predette dal modello $Transformer$ su una traiettoria di un carrello. In alcuni casi anche su traiettorie apparentemente "facili" i modelli restituiscono risultati particolari. La mia ipotesi è che i cambiamenti di velocità, che si possono notare nelle diverse distanze che intercorrono tra i punti bianchi, ingannino il modello.

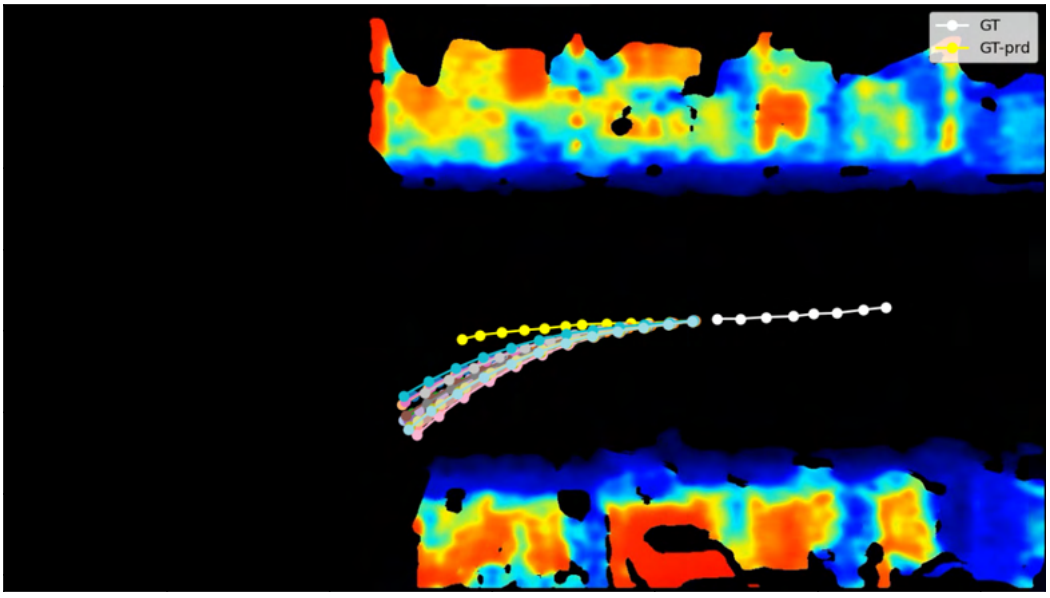

Figura 4.13: Una visualizzazione delle traiettorie predette dal modello *LSTM* su una traiettoria di un carrello. In questo esempio è possibile vedere che *LSTM* produce previsioni in gruppi più stretti. Le previsioni date sono ragionevoli per un carrello che deve uscire dalla scena, tuttavia possiamo notare che si discostano abbastanza dalla traiettoria corretta.

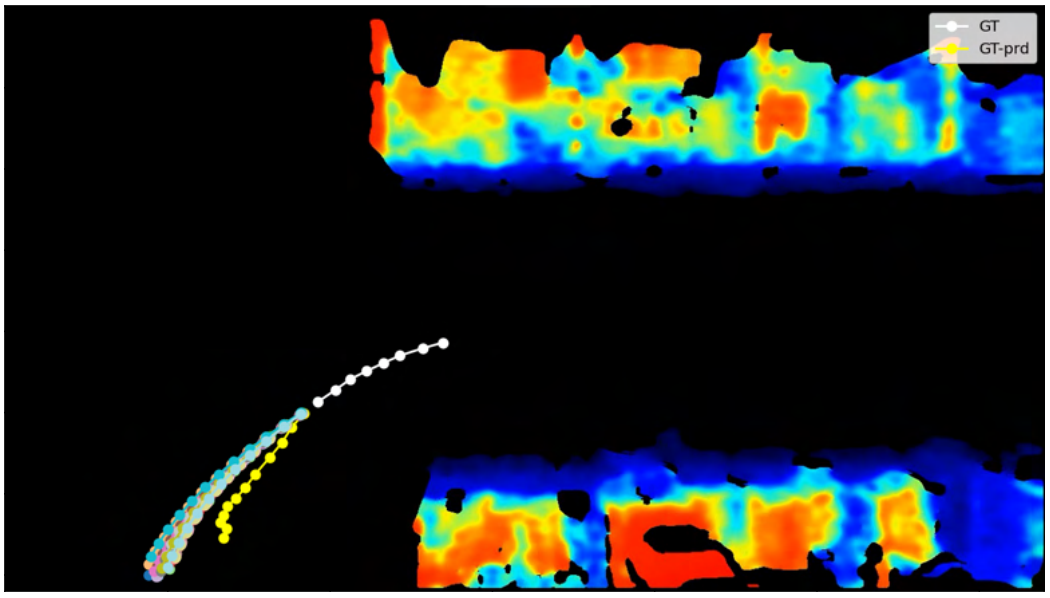

Figura 4.14: Un altra visualizzazione delle traiettorie predette dal modello *LSTM* su una traiettoria di un carrello. Ancora una volta le previsioni si discostano sensibilmente dalla traiettoria corretta anche se sono ragionevoli e verosimili.

Capitolo 5

Conclusioni

5.1 Obiettivi raggiunti

L'obiettivo principale del progetto era quello di dare una dimostrazione di ciò che le tecnologie nell'ambito della computer vision sono in grado di fare, in particolare per quanto riguarda il tracciare e predire traiettorie in quello che è un contesto applicativo.

Specialmente per quanto riguarda il tracciamento delle traiettorie, i risultati che ho ottenuto dimostrano le capacità di tecnologie quali l'object detector YOLOv5 e l'object tracker SORT.

Per quanto riguarda la predizione delle traiettorie, i risultati preliminari che ho ottenuto sono promettenti, dimostrano che dei modelli molto semplici sono in grado di stimare delle possibili traiettorie e le metriche ottenute settano una baseline con la quale confrontare modelli più complessi. Lo stato dell'arte indica la direzione per ottenere risultati migliori: usare modelli consapevoli della scena e degli attori presenti in essa.

5.2 Conoscenze acquisite

Durante lo svolgimento del progetto ho acquisito conoscenze nell'ambito della computer vision specialmente dal punto di vista applicativo. Oltre a fare esperienza con le tecnologie citate in §1.2.3, delle quali voglio menzionare OpenCV e Pytorch per importanza, ho avuto modo di studiare, anche se brevemente e parzialmente, la letteratura riguardante Object Tracking e Trajectory Prediction e al di là della specificità degli ambiti studiati, ho apprezzato l'esperienza fatta e sono sicuro potrà tornarmi utile. In generale, posso dire di aver apprezzato molto il carattere applicativo di questo lavoro di tesi, che mi ha visto coinvolto in tutte le fasi di un progetto, dalla raccolta dei dati al loro uso, che ritengo avere del merito e delle applicazioni pratiche e di valore.

5.3 Considerazioni finali

L'esperienza fatta durante il corso di questo progetto mi lascia la netta sensazione che dal punto di vista applicativo il potenziale non ancora sfruttato delle nuove tecnologie nell'ambito della computer vision, ma anche dell'intelligenza artificiale più in generale, è ancora ampio e che ci risulti persino difficile immaginare come saranno utilizzate. Come agli albori di internet, quando le pagine web erano semplice html e forse del css, era difficile immaginare un applicazione di internet quale i social network, ritengo che queste nuove tecnologie, alimentate dalla rivoluzione delle reti neurali, abbiano applicazioni che ancora fatichiamo ad immaginare ma che cambieranno il mondo.

Per concludere questo documento ci tengo a ringraziare nuovamente il relatore di questa tesi Lamberto Ballan e i co-relatori Pasquale Coscia e Luigi Filippo Chiara per avermi seguito e aiutato durante lo svolgimento dei lavori e la stesura della tesi.

Appendice A

Glossario

Backpropagation

Algoritmo utilizzato nelle reti neurali per aggiornare i parametri della rete durante l'allenamento della stessa.

CNN

Convolutional Neural Network: tipologia di rete neurale usata principalmente per elaborare immagini, ma più in generale sequenze di dati in cui dati prossimi tra loro sono correlati.

Computer Vision

Disciplina nell'informatica che si occupa di estrarre informazioni automaticamente da immagini digitali.

Data Augmentation

Nel machine learning, una pratica che serve ad aumentare le dimensioni del training set (insieme di dati usati per l'allenamento del modello) applicando trasformazioni automatiche ai dati già presenti.

Dataset

Una collezione di dati.

Debugging

In informatica, pratica di ricerca delle cause di un errore al fine di risolverlo.

Discesa del gradiente

Metodo di ottimizzazione dei parametri di una funzione, in particolare per trovare i parametri che minimizzano una funzione chiamata di costo tramite il calcolo del gradiente (derivata generalizzata a funzioni con parametri multipli).

GDPR

Regolamento a livello europeo sul trattamento dei dati personali e la privacy.

GPU

Graphics Processing Unit: componente hardware in un computer con il compito di effettuare le operazioni necessarie all'output a schermo.

Ground Truth

Nel contesto del machine learning, si tratta di dati raccolti in maniera affidabile (ad esempio da un essere umano) con i quali allenare un modello o confrontare un modello per stabilirne l'accuratezza.

Google Colab

Servizio offerto da Google che permette di scrivere ed eseguire codice online e che supporta l'allenamento di modelli di machine learning, incluse reti neurali.

GUI

Interfaccia grafica.

IR

Infrarosso: porzione dello spettro della radiazione elettromagnetica (luce) con lunghezza d'onda tra 1 mm e 700 nm. Questa porzione dello spettro non è visibile all'occhio umano.

Linting

In informatica, pratica di pulizia del codice affinché aderisca a delle regole stilistiche che mirano a migliorarne la qualità.

SSH

Protocollo sicuro per operare su una shell (linea di comando) da remoto.

Parametri (reti neurali)

Parametri interni ad una rete neurale dai quali dipendono direttamente gli output della rete stessa.

Proof of concept

Dimostrazione di fattibilità di un progetto o metodo fatta realizzandone una versione parziale.

Bibliografia

[1] Alexandre Alahi et al. «Social LSTM: Human Trajectory Prediction in Crowded Spaces». In: (2016), pp. 961–971. DOI: 10.1109/CVPR.2016.110.

[2] Alex Bewley et al. «Simple Online and Realtime Tracking». In: *CoRR* abs/1602.00763 (2016). arXiv: 1602.00763. URL: http://arxiv.org/abs/1602.00763.

[3] Alexey Bochkovskiy, Chien-Yao Wang e Hong-Yuan Mark Liao. «YOLOv4: Optimal Speed and Accuracy of Object Detection». In: *CoRR* abs/2004.10934 (2020). arXiv: 2004.10934. URL: https://arxiv.org/abs/2004.10934.

[4] *COCO Benchmark (Real-Time Object Detection)*. Accessed: 2022-03-30. URL: https://paperswithcode.com/sota/real-time-object-detection-on-coco.

[5] *COCO minival Benchmark (Object Detection)*. Accessed: 2022-03-30. URL: https://paperswithcode.com/sota/object-detection-on-coco-minival?metric=AP75.

[6] *General Data Protection Regulation (GDPR)*. Accessed: 2022-04-11. URL: https://gdpr.eu/what-is-gdpr/.

[7] Ross B. Girshick. «Fast R-CNN». In: *CoRR* abs/1504.08083 (2015). arXiv: 1504.08083. URL: http://arxiv.org/abs/1504.08083.

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

[9] Kaiming He et al. «Mask R-CNN». In: *CoRR* abs/1703.06870 (2017). arXiv: 1703.06870. URL: http://arxiv.org/abs/1703.06870.

[10] David Held, Sebastian Thrun e Silvio Savarese. «Learning to Track at 100 FPS with Deep Regression Networks». In: *CoRR* abs/1604.01802 (2016). arXiv: 1604.01802. URL: http://arxiv.org/abs/1604.01802.

[11] Tsung-Yi Lin et al. «Microsoft COCO: Common Objects in Context». In: *CoRR* abs/1405.0312 (2014). arXiv: 1405.0312. URL: http://arxiv.org/abs/1405.0312.

[12] Karttikeya Mangalam et al. «From Goals, Waypoints & Paths To Long Term Human Trajectory Forecasting». In: *CoRR* abs/2012.01526 (2020). arXiv: 2012.01526. URL: https://arxiv.org/abs/2012.01526.

[13] Hyeonseob Nam e Bohyung Han. «Learning Multi-Domain Convolutional Neural Networks for Visual Tracking». In: *CoRR* abs/1510.07945 (2015). arXiv: 1510.07945. URL: http://arxiv.org/abs/1510.07945.

[14] Guanghan Ning et al. «Spatially Supervised Recurrent Convolutional Neural Networks for Visual Object Tracking». In: *CoRR* abs/1607.05781 (2016). arXiv: 1607.05781. URL: http://arxiv.org/abs/1607.05781.

[15] Addie Ira Borja Parico e Tofael Ahamed. «Real Time Pear Fruit Detection and Counting Using YOLOv4 Models and Deep SORT». In: *Sensors* 21.14 (2021). ISSN: 1424-8220. DOI: 10.3390/s21144803. URL: https://www.mdpi.com/1424-8220/21/14/4803.

[16] Joseph Redmon e Ali Farhadi. «YOLO9000: Better, Faster, Stronger». In: *CoRR* abs/1612.08242 (2016). arXiv: 1612.08242. URL: http://arxiv.org/abs/1612.08242.

[17] Joseph Redmon e Ali Farhadi. «YOLOv3: An Incremental Improvement». In: *CoRR* abs/1804.02767 (2018). arXiv: 1804.02767. URL: http://arxiv.org/abs/1804.02767.

[18] Joseph Redmon et al. «You Only Look Once: Unified, Real-Time Object Detection». In: *CoRR* abs/1506.02640 (2015). arXiv: 1506.02640. URL: http://arxiv.org/abs/1506.02640.

[19] Shaoqing Ren et al. «Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks». In: *CoRR* abs/1506.01497 (2015). arXiv: 1506.01497. URL: http://arxiv.org/abs/1506.01497.

[20] *ultralytics/yolov5: YOLOv5 in PyTorch*. Accessed: 2022-03-30. URL: https://github.com/ultralytics/yolov5.

[21] Ashish Vaswani et al. «Attention Is All You Need». In: *CoRR* abs/1706.03762 (2017). arXiv: 1706.03762. URL: http://arxiv.org/abs/1706.03762.

[22] Chien-Yao Wang et al. «CSPNet: A New Backbone that can Enhance Learning Capability of CNN». In: *CoRR* abs/1911.11929 (2019). arXiv: 1911.11929. URL: http://arxiv.org/abs/1911.11929.

[23] Xingkui Zhu et al. «TPH-YOLOv5: Improved YOLOv5 Based on Transformer Prediction Head for Object Detection on Drone-captured Scenarios». In: *CoRR* abs/2108.11539 (2021). arXiv: 2108.11539. URL: https://arxiv.org/abs/2108.11539.