End-to-End Object Detection Pipeline: Zero-Shot Data Curation, Model Optimization, and DeepStream Implementation

Davide Bulotta[1,*] and Giuseppe Riccardo Leone[1,*]

[1]Institute of Information Science and Technologies "A. Faedo",
National Research Council of Italy, Via G. Moruzzi, 1, 56124 Pisa, Italy

Abstract

This is the report of the activities carried out by Davide Bulotta during the internship at the Institute of Information Science and Technologies (ISTI-CNR) in the period between April 2024 and March 2025. The study addresses the theoretical and implementation challenges associated with deploying computer vision systems in domains characterized by high data scarcity and strict hardware constraints. The core of the research proposes a semi-supervised methodology defined as "Zero-shot Annotation Coaching," which formalizes a knowledge distillation pipeline. In this framework, a Vision Transformer-based open-vocabulary detector, specifically OWLv2, functions as a "teacher" model, leveraging its semantic generalization capabilities to generate pseudo-labels from unannotated video streams. This process effectively transfers the semantic understanding of the transformer architecture into a structured dataset, enabling the supervised training of a "student" model, YOLOv8, which is architecturally optimized for low-latency inference rather than open-world semantic breadth.

From an architectural perspective, the research moves beyond standard detection paradigms by exploring composite neural structures. While the YOLOv8 backbone is retained for its efficiency in feature localization, the study theoretically analyzes a decoupled classification head. This involves extracting bounding box crops to feed a secondary Convolutional Neural Network (CNN) for embedding generation, followed by a Fully Connected Neural Network (FCNN) for the final classification. Furthermore, the integration of autoencoder architectures is examined to process segmentation masks, extracting latent vector representations (embeddings) that can be concatenated with visual features to enhance discrimination in complex scenarios.

The transition from theoretical architecture to operational deployment is addressed through the optimization of the computational graph for specific hardware accelerators. The pipeline necessitates the conversion of the model into an intermediate representation (ONNX) and subsequently into a TensorRT engine. This phase highlights critical graph-level challenges, such as the handling of unsupported layers (e.g., IShuffleLayer) and the necessity of INT8 quantization calibration to map floating-point weights to 8-bit integers without degrading the model's feature extraction capabilities. The result is a

highly optimized inference engine capable of executing within the DeepStream analytical pipeline, ensuring that the theoretical advantages of the distilled model are preserved in a real-time, edge-computing environment.

Contents

1 Introduzione

Questo documento riporta il resoconto delle attività dello studente Davide Bulotta durante il tirocinio extracurriculare svolto presso lo Istituto di Scienza e Tecnologia dell'Informazione nel periodo Aprile 2024 - Marzo 2025. Il tirocinio è connesso con il progetto europeo F.A.I.T.H. (Fostering Artificial Intelligence Trust for Humans)[1], il cui obiettivo è quello di definire ed implementare degli strumenti ad hoc (FAITH Risk Management Framework) per misurare e migliorare l'affidabilità dei sistemi che utilizzano modelli di Intelligenza Artificiale (IA). Tale Framework è implementato, dimostrato e perfezionato tramite una selezione rappresentativa di sette progetti pilota su larga scala in ambiti critici, tra cui la mobilità nel trasporto pubblico. Lo ISTC-CNR si occupa di quest'ultimo dominio ed in particolare dell'utilizzo di sistemi di visione artificiale a bordo di treni regionali del vettore nazionale Trenitalia. Si opera in un ambiente in cui non è possibile ingegnerizzare al meglio la scena e vi è il divieto di effettuare registrazioni per rispetto del GDPR (ovvero le registrazioni effettuate sono visionabili solo da autorità di pubblica sicurezza).

Lo studio effettuato affronta le sfide teoriche e implementative associate all'implementazione di sistemi di visione artificiale in domini caratterizzati da elevata scarsità di dati e rigidi vincoli hardware. Il fulcro della ricerca propone una metodologia semi-supervisionata definita "Zero-shot Annotation Coaching", che formalizza una pipeline di distillazione della conoscenza. In questo contesto, un rilevatore di vocabolario aperto basato su Vision Transformer, in particolare OWLv2, funge da modello "insegnante", sfruttando le sue capacità di generalizzazione semantica per generare pseudo-etichette da flussi video non annotati. Questo processo trasferisce efficacemente la comprensione semantica dell'architettura del trasformatore in un set di dati strutturato, consentendo l'addestramento supervisionato di un modello "studente", YOLOv8, ottimizzato architettonicamente per l'inferenza a bassa latenza piuttosto che per l'ampiezza semantica open-world.

1.1 L'Intelligenza Artificiale nel trasporto pubblico

Negli odierni paesaggi urbani in rapida evoluzione, la domanda di servizi di mobilità migliorati è diventata un'esigenza critica. Con l'espansione delle città e la crescita della popolazione, i sistemi di trasporto tradizionali faticano a tenere il passo con la crescente complessità dei modelli di pendolarismo e le diverse esigenze dei residenti. La congestione del traffico, le preoccupazioni ambientali e la distribuzione iniqua delle risorse di trasporto evidenziano l'urgenza di soluzioni innovative. I servizi di mobilità migliorati non riguardano solo il miglioramento dell'efficienza; svolgono un ruolo fondamentale nel promuovere l'equità sociale, la crescita economica e la sostenibilità ambientale. Adottando un approccio olistico alla mobilità, che incorpori il trasporto pubblico, il car sharing, le camminate, le biciclette e le tecnologie emergenti, le società possono creare sistemi interconnessi che promuovono l'accessibilità, riducono l'impronta di carbonio e migliorano la qualità della vita di tutti i cittadini.

L'IA ha un potenziale enorme per rivoluzionare i servizi di mobilità. Ci sono diversi ruoli chiave che l'IA può svolgere nel settore del trasporto pubblico:

- **Ottimizzazione delle reti di trasporto**: analizzare i dati in tempo reale sui modelli di traffico e sulla domanda degli utenti, ottimizzando percorsi e orari si possono ridurre i tempi di attesa, ridurre al minimo la congestione e migliorare l'efficienza dei sistemi di trasporto pubblico.

- **Soluzioni di mobilità personalizzate**: offrire opzioni di trasporto personalizzate in base alle preferenze, alle esigenze e ai comportamenti dell'utente, come una maggiore accessibilità per le persone con disabilità.
- **Trasporto su richiesta**: facilitare i servizi di trasporto su richiesta, adattando percorsi e disponibilità in base alla domanda in tempo reale, con vantaggi particolarmente significativi per le aree meno servite.
- **Analisi predittiva**: prevedere le tendenze e le richieste future, sfruttando dati e modelli storici, aiutando i responsabili delle città a prendere decisioni informate sugli investimenti infrastrutturali.
- **Riduzione delle barriere all'accesso**: aiutare a identificare e affrontare le barriere che diversi gruppi demografici incontrano nell'accesso alle risorse di mobilità.
- **Miglioramento della sicurezza**: migliorare la sicurezza sui sistemi di trasporto pubblico e sulle strade analizzando i modelli di incidenti o identificando le condizioni di pericolo.
- **Sostenibilità ambientale**: ottimizzare l'uso dell'energia nei sistemi di trasporto, dando priorità ai percorsi ecosostenibili e incoraggiando l'uso di opzioni di mobilità condivisa, incrementando l'uso di veicoli elettrici.
- **Comfort dei viaggiatori**: offrire servizi automatici di segnalazione dei vagoni meno affollati o di supporto del personale di bordo come il riconoscimento di spazzatura da rimuovere migliorando in tal modo l'esperienza di viaggio dei passeggeri.

Proprio dall'ultima voce dell'elenco precedente è stata scelta la funzionalità di test oggetto del presente lavoro: il riconoscimento automatico di spazzatura (*dirty detection*) a bordo di un vagone ferroviario per favorire l'intervento del personale addetto e di conseguenza mantenere un livello di igiene del vagone sempre molto elevato. Per la realizzazione di tale compito ci si avvale del flusso video proveniente dal sistema CCTV presente a bordo del treno. In questi video sono presenti non solo gli oggetti di interesse ma soprattutto le persone che viaggiano nel treno. Per ragioni di privacy (regolamento GDPR) non è possibile memorizzare ed utilizzare questi video senza prima definire dettagliatamente tutte le procedure connesse all'uso e alla conservazione di questi dati. I sistemi di IA che elaborano queste immagini sensibili devono essere affidabili o meglio ancora "trustworthy" come si usa dire recentemente. Il rispetto della privacy è solo una delle proprietà che un sistema di IA "trustworthy" deve avere; le altre caratteristiche vengono descritte nel prossimo paragrafo.

1.2 L'Intelligenza Artificiale "trustworthy"

Lo standard ISO/IEC TS 5723:2022 [2] definisce la "trustworthiness" come "capacità di soddisfare le aspettative degli stakeholder in modo verificabile". Analogamente, l'ente americano NIST (National Institute of Standards and Technology) considera l'affidabilità come un attributo oggettivo di un sistema, affermando che "l'affidabilità di un sistema si basa sul concetto di garanzia" [3]. Le caratteristiche principali che un sistema di IA "trustworthy" deve avere sono:

- **Trasparenza e comprensibilità -** Trasparenza e comprensibilità implicano la documentazione completa dell'intero ciclo di vita di un sistema di IA, per consentire all'utente di sapere come vengono prese le decisioni, inclusi i dati e gli algoritmi utilizzati. Aiutano anche a creare fiducia consentendo alle parti interessate di vedere la logica alla base degli output dell'intelligenza artificiale.

- **Equità -** La IA dovrebbe essere progettata per ridurre al minimo i pregiudizi e garantire un trattamento equo tra i diversi gruppi [4]. Ciò richiede un'attenta considerazione dei dati di addestramento e degli algoritmi per evitare di rafforzare i pregiudizi esistenti o creare risultati ingiusti.

- **Responsabilità -** Devono esserci chiare linee di responsabilità per le decisioni dell'intelligenza artificiale. Le parti interessate dovrebbero sapere chi è responsabile delle azioni di un sistema di intelligenza artificiale, soprattutto in caso di danni o conseguenze indesiderate.

- **Affidabilità -** I sistemi di IA affidabili devono funzionare in modo coerente in condizioni variabili. Ciò include la robustezza agli attacchi avversari e la resilienza agli input inaspettati, assicurando che l'IA funzioni correttamente in una gamma di scenari.

- **Privacy -** La protezione dei dati degli utenti è fondamentale per la fiducia. Come qualsiasi sistema che utilizza dati sensibili (ad esempio, dati visivi), i sistemi di IA dovrebbero incorporare forti misure di protezione dei dati, assicurando che le informazioni personali siano gestite in modo etico e sicuro.

- **Considerazioni etiche -** Lo sviluppo e l'implementazione dell'IA dovrebbero essere in linea con le linee guida etiche. Ciò include la considerazione degli impatti sociali più ampi e la garanzia che l'IA avvantaggi l'umanità nel suo insieme[5].

- **Co-creazione e coinvolgimento degli utenti -** Coinvolgere gli utenti nella progettazione e nella valutazione dei sistemi di IA può aumentare la fiducia. Ciò significa sollecitare feedback, affrontare le preoccupazioni e coinvolgere diverse prospettive nel processo di sviluppo.

- **Conformità e governance -** Aderire alle normative legali e agli standard del settore è essenziale per mantenere l'affidabilità. Le organizzazioni dovrebbero stabilire quadri di governance per supervisionare l'implementazione della IA e garantire la conformità alle leggi pertinenti.

L'affidabilità deve quindi essere considerata in più dimensioni di analisi e sistematicamente stabilita e affrontata durante l'intero ciclo di vita di un sistema di IA. Per avere una visione olistica dell'affidabilità in tutte le fasi, è fondamentale riconoscere i rischi distinti che l'IA pone rispetto ai sistemi ICT tradizionali. Un primo passo fondamentale è la documentazione dettagliata dei dataset e dei modelli utilizzati nel sistema di IA. Uno strumento atto a svolgere tale compito, nato nell'ambito del progetto FAITH, si chiama AIPassport ed è descritto nell'appendice A. Tale strumento verrà utilizzato nella nostra applicazione per garantire che i dataset e i modelli utilzzati possano essere certificati come affidabili.

2 Dataset e modelli IA in regime di scarsità di dati

I dati rappresentano uno degli elementi fondamentali per la costruzione di modelli di apprendimento automatico accurati e affidabili. In contesti reali, la disponibilità di dataset di grandi dimensioni e di alta qualità è spesso un problema critico. In questo capitolo verranno descritti i diversi approcci presi in considerazione per la realizzazione dei dataset in scenari in cui la scarsità dei dati è la problematica principale da affrontare. Le soluzioni prese in considerazione in prima battuta sono tre: l'uso di dataset sintetici, l'integrazione di dataset preesistenti e l'utilizzo di modelli di auto-annotazione dei dati. Con tali soluzioni sono stati allenati e messi a confronto modelli per svolgere il task sionject detection relativo ad oggetti classificati come spazzatura. Nella seconda parte del capitolo si presenta *Zero-shot Annotation Coaching* un approccio basato sulla terza soluzione ma molto più evoluto, che seppur in fase embrionale, promette molto bene come sviluppo futuro.

2.1 Dataset Sintetici

La generazione di dataset sintetici è un tema di grande attualità. In un contesto in cui i dati scarseggiano, molte aziende e istituti di ricerca stanno adottando questa strategia. L'uso di dati sintetici è particolarmente vantaggioso in due scenari principali:

- **Dati rari o difficili da reperire**: Alcuni eventi estremi, come disastri naturali o situazioni rare in ambito medico, non producono abbastanza dati reali per l'addestramento di un modello efficace.
- **Privacy e conformità alle normative**: Specialmente in Europa, con regolamenti come il GDPR, l'accesso a dati sensibili è limitato. L'uso di dati sintetici permette di evitare problemi di privacy senza compromettere la qualità del training dei modelli.

Due strumenti significativi per la generazione di dataset sintetici sono:

Kubric Kubric[6] è una pipeline open-source per la generazione di dataset sintetici basati su simulazioni fisiche e rendering. Questo framework, sviluppato da Google Research, consente di creare scene complesse con illuminazione realistica, ombre, riflessi e interazioni tra oggetti. Kubric utilizza Blender[7] e il motore di fisica Bullet per simulare dinamiche realistiche degli oggetti. Grazie alla sua modularità, è possibile generare dataset personalizzati per compiti di visione artificiale come la segmentazione, il riconoscimento e il tracciamento degli oggetti.

Meta-Sim Meta-Sim[8] è un altro framework avanzato per la generazione di dati sintetici, che permette di creare dataset in modo controllato e mirato. Una delle caratteristiche principali di Meta-Sim è la sua capacità di adattare il dataset generato in base alla distribuzione dei dati reali, riducendo i problemi sul modello finale. Questo strumento utilizza un approccio basato su metadati e simulazioni per garantire che i dati sintetici siano quanto più simili possibile a quelli reali, migliorando così la generalizzazione dei modelli di deep learning.

2.2 Unione di Dataset Preesistenti

Un'alternativa alla generazione di dati sintetici è la fusione di diversi dataset già esistenti. Questa soluzione offre il vantaggio di utilizzare dati reali, evitando la necessità di simulazioni avanzate. Tuttavia, l'integrazione di dataset differenti presenta diverse sfide:

- **Differenze nella risoluzione delle immagini**: I dataset possono avere immagini con risoluzioni diverse, il che può portare a una rappresentazione distorta delle classi, influenzando negativamente le prestazioni del modello.
- **Incoerenza delle annotazioni**: Le etichette possono variare tra dataset differenti. Ad esempio, se un dataset contiene immagini di borse annotate senza le persone circostanti, mentre un altro dataset include persone ma ignora le borse, l'unione di questi due insiemi potrebbe creare incongruenze nelle etichette, confondendo il modello durante l'addestramento.
- **Qualità e accuratezza delle annotazioni**: Alcuni dataset possono contenere annotazioni errate o mancanti, rendendo necessario un lavoro manuale di revisione e correzione prima di poter essere utilizzati in modo efficace.

Piattaforme come Roboflow[9] offrono una vasta gamma di dataset preetichettati e strumenti per combinare, trasformare e aumentare i dati. Tuttavia, l'uso di queste piattaforme richiede una selezione attenta dei dataset più adatti allo specifico task di apprendimento automatico.

2.3 Auto-Annotazione delle immagini

Una terza alternativa è rappresentata dai modelli di auto-annotazione, come il SAM2[10]. Questi modelli e strumenti, basati su reti neurali avanzate, sono in grado di segmentare automaticamente oggetti nelle immagini senza la necessità di annotazioni manuali, facilitando la costruzione di dataset di grandi dimensioni.tuo

SAM2 L'uso di **Segment Anything Model v2 (SAM2)**[10] risulta particolarmente vantaggioso quando l'annotazione manuale è troppo onerosa e quando la segmentazione degli oggetti segue pattern ricorrenti e ben definiti. Tuttavia, l'efficacia di questi strumenti dipende fortemente dalla natura dei dati. Nel caso di oggetti con contorni ben definiti, come quelli presenti in molte applicazioni di visione artificiale, questi modelli possono fornire risultati eccellenti. Quando invece si tratta di segmentare elementi con forme irregolari e caratteristiche poco definite, come le macchie di sporco, la loro affidabilità si riduce sensibilmente.

Un ulteriore ostacolo è rappresentato dai costi computazionali. SAM2 richiede risorse hardware ingenti su dataset di grandi dimensioni, rendendone l'adozione proibitiva in ambienti con capacità di calcolo limitate.

Il compito specifico di identificare lo sporco presenta caratteristiche particolari che rendono l'uso di SAM2 meno efficace poiché il modello non è capace di riconoscere elementi come macchie di sporco.

CVAT **Computer Vision Annotation Tool** [11] è uno strumento open source sviluppato per agevolare il processo di annotazione di immagini e video, fondamentale per l'addestramento di modelli di machine learning che richiedono una grande quantità di

dati. Grazie alla sua interfaccia intuitiva e alla capacità di integrarsi con algoritmi di auto annotazione, CVAT permette di generare proposte di etichettatura che possono essere successivamente verificate e corrette manualmente. Questa sinergia tra annotazione automatica e supervisione umana consente di ridurre il tempo e il costo nella creazione di dataset accurati, garantendo al contempo la flessibilità necessaria per gestire diversi formati e tipologie di dati. Permette di integrarsi con la maggior parte dei modelli di *object detection*.

OWLv2 Il modello **OWLv2**[12] rappresenta un significativo avanzamento nel campo delle tecnologie di riconoscimento degli oggetti, in particolare nell'ambito dell'open vocabulary[13], ovvero la capacità di identificare categorie non predefinite. Basato su architetture innovative, come i transformers[14], OWLv2 è progettato per apprendere rappresentazioni visive robuste e adattabili, permettendo così di riconoscere e localizzare oggetti in contesti complessi e variabili. OWLv2 può contribuire a generare annotazioni di alta qualità in maniera automatizzata. Questo approccio riduce ulteriormente l'intervento manuale, accelerando la preparazione dei dataset e migliorando la precisione complessiva dell'annotazione. L'integrazione di OWLv2 nelle pipeline di annotazione consente di affrontare sfide legate alla variabilità e alla complessità degli oggetti presenti nelle immagini, offrendo un supporto prezioso per le applicazioni di computer vision che contano su un numero di dati basso.

2.4 Un approccio evoluto per la classificazione di oggetti

Si presenta in questo paragrafo un metodo di classificazione differente da quelli convenzionali che consiste nello sfruttare le *bounding box* (bbox) estratte da YOLO, in questo caso YOLOv8, per riconoscere l'oggetto ritagliato mediante una successiva classificazione basata su *Fully Connected Neural Network*[15] (FCNN). In realtà, l'approccio può essere ulteriormente ampliato ed esistono almeno tre strategie differenti che partono dal medesimo principio, ossia ricavare le porzioni significative di un'immagine tramite un *object detector* e, in seguito, utilizzare una rete neurale per la classificazione vera e propria. Il dataset selezionato per questi esperimenti è un insieme di quattro *dataset* monoclasse, indicati come D_1, D_2, D_3, D_4, che formano complessivamente $D_c = \{D_1, D_2, D_3, D_4\}$.

La base del modello Nel dettaglio, si parte dalle bbox ottenute dalla classificazione di YOLOv8 (provenienti da qualsiasi partizione di addestramento, validazione o test) e si applica a ciascuna di esse un bordo aggiuntivo di circa 15–20 pixel, così da prevenire tagli indesiderati in fasi di *data augmentation* come rotazioni, zoom o aggiunta di *noise*. Quando i margini di ritaglio risultano troppo stretti, infatti, la figura principale potrebbe non rientrare completamente nel riquadro dopo la trasformazione. L'obiettivo del *data augmentation* è generare immagini aggiuntive, aumentare la variabilità e migliorare la capacità di generalizzazione del modello.

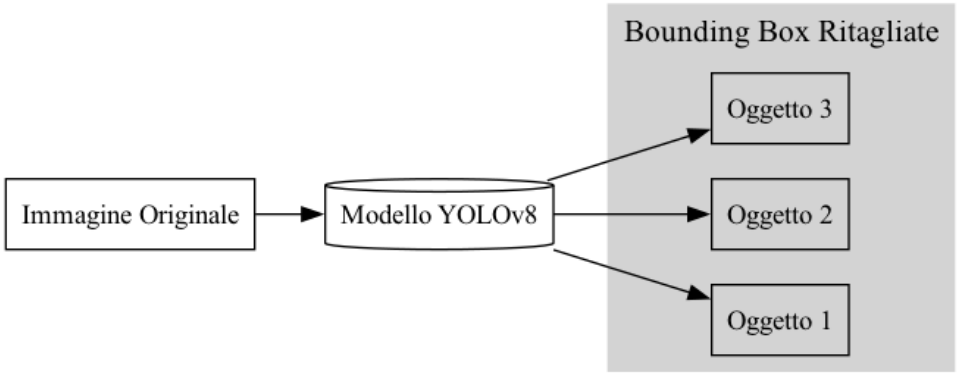

Figure 1: Esempio di come vengono raccolte e ritagliate le bounding box da YOLOv8.

Le bbox, uniformate a una dimensione comune con lati l_1 e l_2, vengono poi inoltrate a una rete convoluzionale (CNN) in grado di estrarre un *embedding* descrittivo dell'oggetto. La rete è organizzata in diversi blocchi convoluzionali contenenti come funzione di attivazione ReLU e, se necessario, *max pooling*[16] per ridurre la dimensionalità, nel primo layer della rete è presente la *batch normalization*[17]. Inoltre, viene inserito il *dropout*[18] per mitigare l'overfitting. Al termine della CNN, invece di sfruttare una classificazione diretta, si preleva il vettore di *embedding* $\mathbf{v}$, la cui dimensione (in intervallo 32 o 64 negli esperimenti) dipende dalla profondità e dalle scelte progettuali della rete. Formalmente, se $\mathbf{x}$ rappresenta la bbox ritagliata e $f_{\text{CNN}}(\cdot)$ indica la trasformazione operata dalla rete convoluzionale, l'*embedding* è:

$$\mathbf{v} = f_{\text{CNN}}(\mathbf{x}), \quad \text{dove } \mathbf{v} \in \mathbb{R}^n \text{ con } 32 \leq n \leq 64.$$

Tale vettore costituisce una rappresentazione compatta delle caratteristiche più rilevanti dell'oggetto stesso.

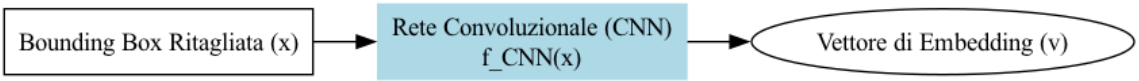

Figure 2: Schema del passaggio delle bounding box attraverso la rete CNN, con output finale il vettore di embedding.

Come ultimo passaggio, si impiega una rete *FCNN* per la classificazione vera e propria. Tale rete riceve in ingresso il vettore di embedding $\mathbf{v}$ e, attraverso strati densi dotati di funzioni di attivazione ReLU, produce in uscita quattro valori che rappresentano le probabilità di appartenenza alle rispettive classi. Se indichiamo con W e $\mathbf{b}$ i pesi e il bias dell'ultimo strato, la *softmax* $\sigma(\cdot)$ fornisce l'output:

$$\hat{\mathbf{y}} = \sigma\big(W\,\mathbf{v} + \mathbf{b}\big),$$

dove $\hat{\mathbf{y}} = (\hat{y}_1, \hat{y}_2, \hat{y}_3, \hat{y}_4)$ dove tutti gli elementi sono una probabilità sommata a 1. L'addestramento si basa su una *cross-entropy*, che, dato un vettore di etichette $\mathbf{t} = (t_1, t_2, t_3, t_4)$, può essere definita come:

$$\mathcal{L}_{\text{CE}}(\hat{\mathbf{y}}, \mathbf{t}) = -\sum_{i=1}^{4} \Big[t_i \log\big(\hat{y}_i\big) + (1 - t_i) \log\big(1 - \hat{y}_i\big) \Big].$$

Se le classi sono mutuamente esclusive, si usa una codifica *one-shot*, in cui ogni oggetto è associato a una sola classe attiva (1) mentre le altre restano a 0. Nel caso di un problema multi-label, invece, si utilizza una codifica binaria, in cui ogni classe è indicata indipendentemente come presente (1) o assente (0).

Figure 3: Esempio di rete *FCNN* che riceve in input il vettore di embedding e produce la classificazione a 4 uscite.

In questo modo, si realizza una pipeline in cui YOLOv8 fornisce il ritaglio dell'oggetto e una CNN estrae un *embedding* ridotto ma informativo, demandando al *perceptron* finale il compito di discriminare l'oggetto tra quattro categorie diverse. Questo schema si presta a varianti più sofisticate, mantenendo però la flessibilità di poter sostituire o aggiornare singolarmente ciascun modulo, a seconda delle esigenze o di nuove strategie di addestramento.

Aggiungere informazioni extra Al modello base, una possibile estensione è l'aggiunta di informazioni extra (quali dimensioni originali, distanza dall'oggetto, orientamento, parametri geometrici, ecc.) che risultano utili per migliorare la capacità di discriminazione. L'idea è che la rete *FCNN* non riceva come ingresso solo il vettore di embedding $\mathbf{v}$ (che possiamo indicare come $\mathbf{x}_1$), ma anche un secondo insieme di *feature* ausiliarie $\mathbf{e}$ (indicato come $\mathbf{x}_2$). In tal modo, l'input complessivo della rete diventa:

$$\mathbf{z} = [\mathbf{x}_1;\, \mathbf{x}_2],$$

dove $[\cdot;\, \cdot]$ denota la concatenazione. Graficamente, si può rappresentare come due flussi distinti di ingresso che, prima di raggiungere gli strati densi, vengono fusi in un unico vettore.

Figure 4: Rappresentazione schematica dell'aggiunta di un secondo input $\mathbf{x}_2$ al perceptron.

Tramite questa integrazione, la rete può utilizzare sia l'informazione visiva estratta dalla CNN (contenuta in $\mathbf{x}_1$) sia i dati numerici forniti a corredo (raccolti in $\mathbf{x}_2$), migliorando potenzialmente il processo di classificazione.

Aggiungere le maschere degli oggetti Come avviene per le informazioni extra, è possibile integrare nella pipeline ulteriori dati per migliorare la classificazione di un oggetto. In particolare, YOLO, oltre a generare le bounding box, può fornire anche maschere di segmentazione. Tali maschere sono rappresentazioni matriciali troppo complesse per essere inserite direttamente nel processo di classificazione: diventa quindi una soluzione adottare un'architettura di tipo *Autoencoder*; quindi una struttura composta

da un encoder e da un decoder, in cui si disponga sia della maschera di input che di quella di output.

Data la maschera $\mathbf{m} \in \mathbb{R}^{W \times H}$,
l'encoder $f_{\text{enc}}(\cdot) : \mathbb{R}^{W \times H} \to \mathbb{R}^n$ estrae un embedding di dimensione n,
il decoder $f_{\text{dec}}(\cdot) : \mathbb{R}^n \to \mathbb{R}^{W \times H}$ ricostruisce la maschera $\hat{\mathbf{m}}$.

Concettualmente, si ottiene:

$$\hat{\mathbf{m}} = f_{\text{dec}}\big(f_{\text{enc}}(\mathbf{m})\big).$$

L'obiettivo dell'addestramento di questa porzione del sistema è minimizzare la differenza tra la maschera generata $\hat{\mathbf{m}}$ e la maschera reale $\mathbf{m}$. Tale differenza può essere misurata tramite una *loss function* su base *pixel-wise* tramite metriche di similarità come il *DICE coefficient*[19]. Se si indica con $\mathcal{L}_{\text{dice}}(\mathbf{m}, \hat{\mathbf{m}})$ la DICE loss; essa si costruisce a partire dalla definizione di DICE score (DS) fra due insiemi (o maschere) A e B:

$$DS(A, B) = \frac{2 \times |A \cap B|}{|A| + |B|}.$$

In termini di immagini binarie, $|A|$ e $|B|$ rappresentano la somma (o integrazione) dei pixel attivi in ciascuna maschera. La *DICE loss* può essere definita come

$$\mathcal{L}_{\text{dice}}(\mathbf{m}, \hat{\mathbf{m}}) = 1 - DS\big(\mathbf{m}, \hat{\mathbf{m}}\big).$$

Raggiunta una DICE loss sufficientemente bassa; quindi un DICE score sufficientemente alto (indicativo di un'elevata corrispondenza fra input e output), è possibile rimuovere il decoder e *freezare* l'encoder, poiché l'obiettivo finale non è la ricostruzione della maschera, ma l'estrazione del corrispondente embedding.

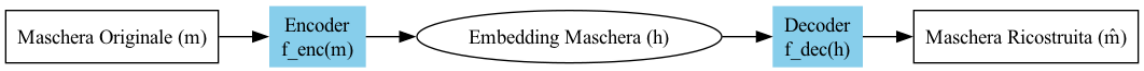

Figure 5: Schema encoder–decoder per le maschere. L'encoder $f_{\text{enc}}(\mathbf{m})$ restituisce un embedding di dimensione $n = 32$, mentre il decoder f_{dec} genera la maschera ricostruita $\hat{\mathbf{m}}$.

Formalmente, dopo l'addestramento, l'encoder con pesi congelati diventa:

$$f^{\star}_{\text{enc}}(\cdot) : \mathbb{R}^{W \times H} \to \mathbb{R}^n,$$

e, data una maschera $\mathbf{m}$, il suo embedding $\mathbf{h} \in \mathbb{R}^n$ è:

$$\mathbf{h} = f^{\star}_{\text{enc}}(\mathbf{m}).$$

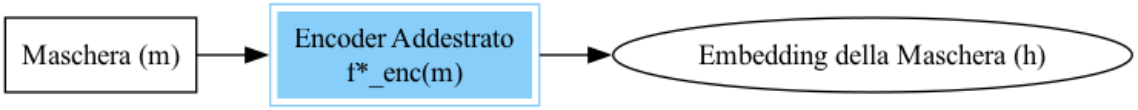

Figure 6: Encoder addestrato e congelato che produce l'embedding della maschera $\mathbf{h} \in \mathbb{R}^n$.

Il vettore $\mathbf{h}$ così calcolato può essere unito (concatenato) con il vettore $\mathbf{v}$ proveniente dalla CNN che estrae le feature dall'immagine, o con eventuali altri *input* numerici (informazioni extra) per formare l'ingresso definitivo al *perceptron* di classificazione. Se si indicano con $\mathbf{x}_2$ le altre feature e con $\mathbf{v} = \mathbf{x}_1$ l'output della CNN, l'input complessivo $\mathbf{z}$ alla rete densa comprende anche $\mathbf{h}$:

$$\mathbf{z} = [\mathbf{x}_1; \mathbf{h}; \mathbf{x}_2],$$

dove $[\cdot; \cdot]$ denota la concatenazione di vettori. In questo modo, l'architettura sfrutta contemporaneamente: 1) Le feature visuali estratte dalla CNN, 2) L'embedding della maschera da encoder, 3) Le eventuali informazioni ausiliarie (distanza, dimensioni originali, parametri geometrici, ecc.).

Questo arricchimento dell'input può fornire un contesto più ampio e una migliore capacità di classificazione rispetto allo studio di ablazione.

2.5 Zero-shot Annotation Coaching

L'approccio presentato integra tecniche di *zero-shot object detection* [20] per automatizzare il processo di annotazione in domini specifici ad esempio uffici o mezzi di trasporto (treni, autobus, ecc), al fine di generare dataset utilizzabili per addestrare modelli più efficienti, come YOLOv8. L'obiettivo principale è minimizzare la necessità di annotazioni manuali e ottenere modelli efficienti e robusti, gestendo automaticamente grandi quantità di dati.

2.5.1 Estrapolazione dei frame dai video

La prima fase consiste nell'acquisizione e analisi di video rappresentativi del dominio applicativo specifico. Ciascun video V_i viene rappresentato come una sequenza temporale di frame:

$$V_i = \{f_{i,1}, f_{i,2}, ..., f_{i,N_i}\}, \quad i = 1, 2, ..., M \tag{1}$$

dove $f_{i,j}$ rappresenta il j-esimo frame del video i, N è il numero totale di frame per video e M è il numero totale di video considerati. Opzionalmente si può decidere di compilare a frequenza costante al fine massimizzare la copertura spazio-temporale e di minimizzare la ridondanza delle informazioni tra due frame troppo simili.

2.5.2 Generazione automatica delle annotazioni

I frame estratti vengono sottoposti ad inferenza utilizzando OWLv2[12, 13], un modello basato sulla tecnica di *open-world object detection*[21] che permette il rilevamento di oggetti senza necessità di pre-allenamento specifico. OWLv2 sfrutta una descrizione testuale delle classi selezionate per indirizzare il processo di detection; ciò significa che, specificando testualmente le classi desiderate $C = \{c_1, c_2, ..., c_K\}$, il modello può generare detection mirate a tali classi. Formalmente, la detection per ciascun frame f si esprime come:

$$\sum_{c=1,..,k} \{(\hat{y}_{f,c}, \hat{b}_{f,c})\}_{c \in C} = \text{OWLv2}(f \mid C) \tag{2}$$

dove $\hat{y}_{f,c}$ indica la confidenza predetta per la presenza della classe c e $\hat{b}_{f,c}$ rappresenta il bounding box dell'oggetto rilevato corrispondente alla classe c. Queste annotazioni automatiche vengono successivamente utilizzate per addestrare modelli più leggeri e veloci.

2.5.3 Creazione automatizzata del dataset e data augmentation

Sulla base delle annotazioni generate da OWLv2, viene costruito un dataset di immagini che include etichette e posizioni degli oggetti rilevati. È possibile usare diversi approcci per bilanciare il numero delle classi all'interno del dataset; applicando una revisione manuale oppure mediante funzioni di perdita che tengano conto del peso di quella classe sull'intero dataset, come la BCELoss[22].

Per aumentare la capacità del modello di adattarsi a scenari reali, si applicano numerose tecniche di data augmentation, come rotazioni casuali, modifiche nella luminosità e nel contrasto, distorsioni prospettiche e aggiunta di rumore. Queste procedure ampliano la variabilità dei dati e rendono il modello più robusto rispetto alle differenti condizioni operative.

$$D_{\text{aug}} \;=\; \bigcup_{T \in \mathcal{T}} \big\{\, (T(f),\, z) \;\mid\; (f, z) \in D \big\}, \tag{3}$$

Dove:

- D rappresenta il dataset originale.
- $\mathcal{T}$ è l'insieme delle trasformazioni utilizzate nella data augmentation.
- $T(f)$ indica l'immagine f dopo l'applicazione della trasformazione T a al frame f.
- z rappresenta le carrattersitiche aggiuntive che rigaurdano l'annotazione.

2.5.4 Addestramento della rete YOLOv8

L'addestramento di YOLOv8 prevede l'utilizzo del dataset generato tramite annotazioni automatiche di OWLv2. In questo modo, il modello YOLOv8 viene addestrato sulle annotazioni rilevate dal modello OWLv2 che risulta più leggero e veloce nell'inferenza.

Nel dettaglio, il processo si può considerare come una forma di *distillazione*: attraverso il dataset annotato, YOLOv8 apprende a classificare e localizzare le stesse classi che OWLv2 aveva già identificato. Il modello finale è quindi in grado di svolgere la detection in maniera più efficiente, pur mantenendo buona parte della competenza derivata dalla fase di annotazione. Possiamo rappresentare l'obiettivo di apprendimento nel modo seguente:

$$\theta_{\text{YOLOv8}} \;=\; \arg\min_{\theta} \sum_{(x,\, a_{\text{OWL}}) \,\in\, D_{\text{aug}}} \mathcal{L}\big(f(x;\theta),\, a_{\text{OWL}}\big)\,, \tag{4}$$

Dove:

- D_{aug} è il dataset annotato da OWLv2, costituito da immagini e relative etichette (classi e bounding box).
- x indica l'immagine di input.
- a_{OWL} rappresenta l'annotazione generata dal modello OWLv2.
- $f(x;\theta)$ è la funzione di predizione di YOLOv8, parametrizzata da θ.
- $\mathcal{L}$ è la funzione di perdita complessiva, che tiene conto di errori di classificazione, localizzazione e confidenza.

Tale formulazione illustra come i parametri θ di YOLOv8 vengano ottimizzati per minimizzare le discrepanze tra le predizioni del modello e le annotazioni provenienti da OWLv2, trasferendo così parte della conoscenza del modello più complesso a un sistema più rapido ed efficiente.

In sintesi, l'inclusione di informazioni provenienti da OWLv2 durante l'addestramento di YOLOv8 consente di ottenere un modello più leggero, in grado di eseguire inferenze rapide, senza dover sacrificare eccessivamente la capacità di rilevare correttamente gli oggetti di interesse.

3 Attività sperimentali

3.1 Creazione del Dataset mediante unione

Il primo approccio sperimentato per la creazione di un dataset da utilizzare in carenza di dati originali si è basato sulla unione di dataset preesistenti. L'unione di più dataset ha permesso di creare un set di dati più ricco e variegato, utile per migliorare la capacità di generalizzazione del modello. Il processo di unione ha evidenziato alcune problematiche, come la gestione di differenti risoluzioni e formati di annotazione.

3.1.1 Scelta dei dataset di partenza

Sono stati selezionati quattro dataset provenienti dalla piattaforma *Roboflow*, ciascuno rappresentante di una specifica classe di oggetti: **plastic bottle, dirt, packet, phone**.

1. **Dataset classe Plastic bootle**: Questo dataset contiene un totale di 3.773 immagini con 5.750 annotazioni. Le immagini rappresentano una vasta gamma di bottiglie di plastica, comprese varianti in diverse condizioni di illuminazione e angolazioni. La sua ampia diversità lo rende ideale per il riconoscimento di molte tipologie di bottiglie [23].

2. **Dataset classe Dirty (Macchie di Sporco)**: La presenza di dataset specifici per lo sporco è piuttosto limitata, il che ha ridotto le opzioni disponibili. Tuttavia, è stato individuato un dataset contenente 388 immagini con 553 annotazioni. Nonostante le dimensioni ridotte, questo dataset fornisce un buon punto di partenza per il riconoscimento delle macchie di sporco [24].

3. **Dataset classe Packet**: Il dataset relativo ai pacchetti contiene 230 immagini, ma con un totale di 3.216 annotazioni. Questo suggerisce un'alta densità di oggetti per immagine, il che potrebbe rendere questa classe particolarmente difficile da riconoscere a causa dell'elevata varianza nei tipi di buste rappresentate (ad esempio, confezioni di patatine, involucri di plastica, ecc.) [25].

4. **Dataset classe Phone** Il dataset per la classe telefono include 2.199 immagini con un totale di 3.161 annotazioni. Questo dataset rappresenta molte tipologie di telefoni, fornendo una varietà di casi utili per il riconoscimento [26].

3.1.2 Esportazione ed Unione dei Dataset

Mediante l'algoritmo 1 viene effettuato il *merge* dei dataset in input, assicurando che tutte le classi vengano integrate correttamente. Per garantire la compatibilità con *YOLOv8*, tutti i dataset sono stati esportati in questo formato. Il relativo script Python riportato in appendice D.1 permette di combinare più dataset YOLOv8, garantendo un'uniformità nella struttura del dataset.

3.2 Fine-tuning del modello di IA

3.2.1 Scelta del modello di partenza

La selezione del modello di *object detection* di partenza è un passaggio cruciale per garantire un buon compromesso tra accuratezza e velocità di inferenza. Numerosi modelli sono

Algorithm 1 MergeYoloDatasets

```
datasetPaths ← ["$D_1$", "$D_2$", "$D_3$", "$D_4$"]
outputPath ← "dataset"
createDir(outputPath)
for all split in ["train", "valid", "test"] do
    createDir(joinPath(outputPath, split, "images"))
    createDir(joinPath(outputPath, split, "labels"))
end for
allClassNames ← empty set
for all dsPath in datasetPaths do
    dataYamlPath ← joinPath(dsPath, "data.yaml")
    if fileNotExists(dataYamlPath) then
        printWarning(dsPath)
        continue
    end if
    dataDict ← loadYaml(dataYamlPath)
    if "names" in dataDict then
        for all c in dataDict["names"] do
            add c to allClassNames
        end for
    end if
    for all split in ["train", "valid", "test"] do
        origImagesDir ← joinPath(dsPath, split, "images")
        origLabelsDir ← joinPath(dsPath, split, "labels")
        destImagesDir ← joinPath(outputPath, split, "images")
        destLabelsDir ← joinPath(outputPath, split, "labels")
        if dirExists(origImagesDir) then
            for all file in listDir(origImagesDir) do
                copyFile(joinPath(origImagesDir, file), destImagesDir)
            end for
        end if
        if dirExists(origLabelsDir) then
            for all file in listDir(origLabelsDir) do
                copyFile(joinPath(origLabelsDir, file), destLabelsDir)
            end for
        end if
    end for
end for
mergedData ← "train" : joinPath("train", "images"), "val" : joinPath("valid",
"images"), "test" : joinPath("test", "images"), "nc" : size(allClassNames), "names"
: sort(allClassNames)
saveYaml(joinPath(outputPath, "data.yaml"), mergedData)
print("Merge completed!")
```

stati sviluppati per questo scopo, ognuno con caratteristiche peculiari che li rendono più o meno adatti a specifiche applicazioni. Nel contesto del *dirty detection*, i modelli della famiglia YOLO (You Only Look Once)[27] si sono affermati come una delle soluzioni più efficaci, grazie alla loro rapidità e adattabilità al task.

La scelta di un modello YOLO specifico richiede un'attenta valutazione. Tradizionalmente, molti studi nel settore utilizzano YOLOv5[28], poiché rappresenta una soluzione consolidata, ampiamente adottata dalla community e da diverse organizzazioni. Tuttavia, nel momento in cui questo report è stato scritto, lo stato dell'arte dell'*object detection* è rappresentato da YOLOv11[29], che introduce significativi miglioramenti in termini di accuratezza e riduzione dei tempi di inferenza rispetto alle versioni precedenti. Il principale svantaggio di YOLOv11, così come del suo predecessore YOLOv10[30], è la relativa giovinezza: trattandosi di modelli di recente sviluppo, il loro supporto e ottimizzazione per vari framework e dispositivi hardware risultano ancora limitati.

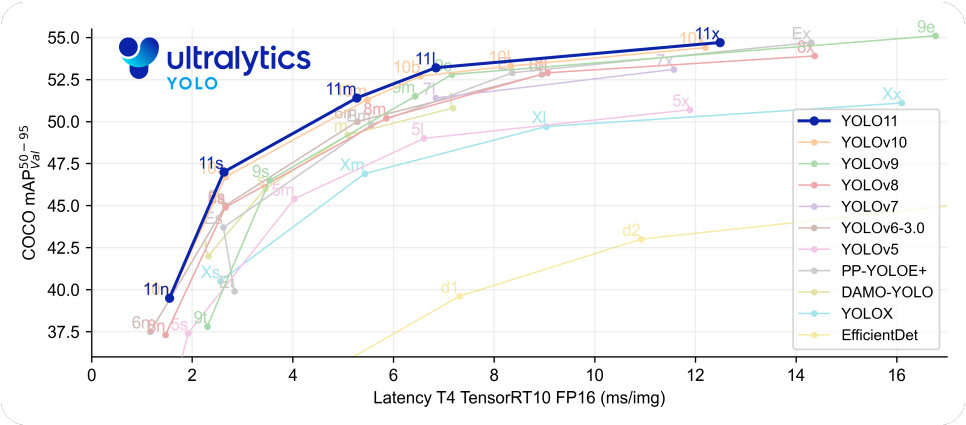

Figure 7: Confronto delle prestazioni tra le diverse versioni di YOLO in termini di accuratezza e velocità di inferenza(Fonte Ultralytics[31]).

3.2.2 Implementazione su NVIDIA Jetson con DeepStream

Un altro aspetto fondamentale nella selezione del modello riguarda la sua compatibilità con le piattaforme hardware di destinazione. Nel nostro caso, è stato necessario valutare la possibilità di implementare il modello su dispositivi Jetson[32] di NVIDIA[33], sfruttando la tecnologia DeepStream[34] per l'accelerazione hardware. Non tutti i modelli YOLO sono compatibili con questa piattaforma, poiché l'inferenza su Jetson richiede la conversione del modello in formato TensorRT[35], un'ottimizzazione specifica per GPU[36] NVIDIA.

Dopo un'attenta analisi, si è deciso di utilizzare YOLOv8[37], un modello ormai consolidato e ottimizzato per l'integrazione con Jetson e DeepStream. Questo modello, rilasciato da Ultralytics, ha ricevuto un forte supporto dalla community, risultando una scelta affidabile sia in termini di prestazioni che di compatibilità con ambienti di produzione. La conversione in TensorRT è stata effettuata utilizzando il software di terze parti DeepStream-YOLO[38], sviluppato da Marcos Luciano, uno degli strumenti più utilizzati per questo tipo di ottimizzazione. La procedura dettagliata per l'utilizzo del framework Deepstream è descritta nell'appendice C.

3.2.3 Tipologia di Modello

La scelta di YOLOv8 è stata guidata da un compromesso tra prestazioni, supporto della community e compatibilità con le piattaforme Jetson. Nonostante le versioni più recenti di YOLO offrono miglioramenti in termini di accuratezza, le limitazioni dovute alla scarsa ottimizzazione per hardware embedded[32] e la mancanza di supporto da parte di strumenti consolidati come DeepStream-YOLO hanno portato a preferire una soluzione più stabile e testata.

Il modello YOLOv8 supporta diversi compiti di visione artificiale, tra cui *Detection, Instance Segmentation, Pose Estimation, Oriented Detection* e *Classification* [37]. La scelta della tipologia di task dipende fortemente dalla natura del problema affrontato.

Nel contesto di questo progetto, l'opzione più adatta è risultata essere la **detection**, poiché permette di identificare la presenza di specifici oggetti senza la necessità di segmentarne i contorni esatti. Sebbene l'*Instance Segmentation* fornisca una segmentazione più dettagliata, il suo utilizzo avrebbe reso il processo di annotazione molto più complesso e dispendioso in termini di tempo e risorse. La *detection*, invece, consente di generare dataset più rapidamente, riducendo la soglia di errore nelle etichette e garantendo maggiore uniformità nelle annotazioni.

3.2.4 Varianti di YOLOv8 per il Task di Detection

Il modello YOLOv8 è disponibile in diverse varianti, ognuna con caratteristiche specifiche in termini di accuratezza, velocità di inferenza e complessità computazionale. La scelta della versione più adatta dipende da fattori come le capacità hardware disponibili e i requisiti in termini di latenza e precisione. La Tabella 1 riporta le principali specifiche delle varianti di YOLOv8 per il task di *object detection* [37].

Modello	**Dimensione**	**$mAP_{50\text{-}95}$**	**A100 (TensorRT) [ms]**	**Parametri (M)**
YOLOv8n	640	37.3	0.99	3.2
YOLOv8s	640	44.9	1.20	11.2
YOLOv8m	640	50.2	1.83	25.9
YOLOv8l	640	52.9	2.39	43.7
YOLOv8x	640	53.9	3.53	68.2

Table 1: Prestazioni e specifiche delle varianti di YOLOv8 per object detection.

La scelta della variante specifica è approfondita nei prossimi paragrafi, poiché le performance possono variare significativamente in base al tipo di dati utilizzato e alla configurazione hardware.

3.2.5 Installazione del software Yolov8

Per l'addestramento e la valutazione del modello, è stata utilizzata la libreria Ultralytics[31], che fornisce un'implementazione diretta di YOLOv8[37]. L'installazione della libreria è avvenuta tramite il comando:

Listing 1: Installazione di Ultralytics

```
pip install ultralytics
```

Per la registrazione e il monitoraggio degli esperimenti, si è scelto di utilizzare ClearML[39], una piattaforma di gestione degli esperimenti che consente di tracciare metriche, configurazioni e risultati. L'installazione è stata eseguita con:

Listing 2: Installazione di Clearml

```
pip install clearml
```

Dopo la creazione di un account su ClearML, è stato necessario configurare l'ambiente lanciando il seguente comando nel terminale:

Listing 3: Inizializzazione di Clearml

```
clearml-init
```

Una volta completata l'inizializzazione, Ultralytics rileva automaticamente la presenza di ClearML e salva i risultati degli esperimenti sulla piattaforma, permettendo un monitoraggio costante.

3.2.6 Esperimenti sulla risoluzione e architettura

Il primo obiettivo degli esperimenti è stato testare le diverse varianti di modello e risoluzione, valutando l'impatto su accuratezza e tempi di inferenza. Sono stati eseguiti più addestramenti con diverse architetture di YOLOv8. La Tabella 2 riporta i risultati ottenuti:

Modello	**Size (px)**	**mAP$_{50}$**	**mAP$_{50\text{-}95}$**	**Inference Time (ms)**
YOLOv8m	640	0.9234	0.7913	1.535
YOLOv8x	**640**	**0.9268**	**0.8085**	**3.752**
YOLOv8m	1280	0.8739	0.6591	5.926
YOLOv8l	1280	0.8914	0.6845	9.456
YOLOv8x	1280	0.8976	0.6902	13.656
YOLOv8x	1920	0.7572	0.4300	31.861

Table 2: Risultati dei test con YOLOv8 su diverse configurazioni di training.

L'addestramento dei modelli è stato eseguito utilizzando il comando:

Listing 4: Train del modello yolov8m-640

```
yolo task=detect mode=train model=yolov8m.pt data=dataset/data.
   yaml
epochs=350 device=0,1 cache=ram --workers=16 --batch=152
```

La dimensione del batch[40] e il numero di GPU[36] utilizzati dipendono dall'hardware a disposizione. Un valore maggiore del batch richiede più memoria video, mentre la risoluzione delle immagini influisce sia sulla richiesta di memoria che sul tempo di inferenza. Per test a risoluzioni elevate, come 1920x1080, è stato necessario ridurre il batch size per evitare problemi di memoria:

Listing 5: Train del modello yolov8x-1920

```
yolo task=detect mode=train model=yolov8x.pt data=dataset/data.
   yaml
epochs=350 device=0,1 cache=ram workers=16 batch=12 imgsz=1920
```

3.2.7 Confronto della Precisione tra Modelli

Per valutare le differenze di accuratezza tra le varie configurazioni, è stato generato un grafico. La Figura 8 mostra il confronto della precisione dei modelli YOLOv8 con diverse dimensioni e parametri negli esperimenti effettuati.

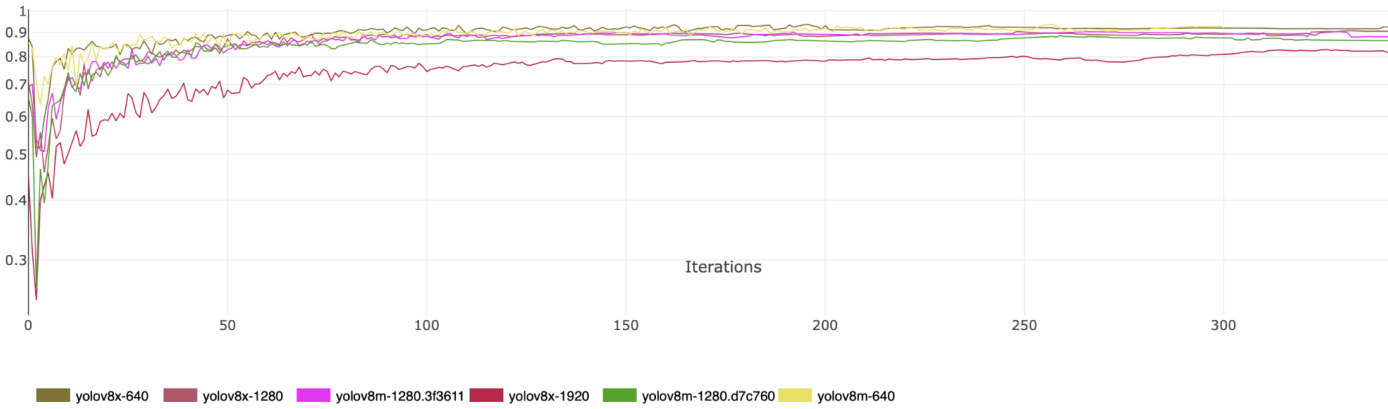

Figure 8: Confronto della precisione tra diverse configurazioni di YOLOv8.

Dai risultati ottenuti si osserva che i modelli offrono le migliori prestazioni quando utilizzano una risoluzione di 640 pixel. Questo comportamento è strettamente legato alla natura del dataset, poiché la risoluzione scelta rappresenta un buon compromesso tra dettaglio dell'immagine e capacità del modello di generalizzare le caratteristiche degli oggetti rilevati. Risoluzioni più elevate non hanno portato a miglioramenti significativi della $mAP_{50\text{-}95}$, mentre hanno incrementato il tempo di addestramento e i requisiti di memoria.

Un altro aspetto critico emerso dai test è l'importanza dei tempi di inferenza, specialmente in contesti di edge computing. In scenari in cui il modello deve operare su dispositivi embedded, come le piattaforme NVIDIA Jetson[32], l'ottimizzazione dell'inferenza è un fattore determinante per garantire sia le prestazioni in tempo reale che l'efficienza energetica del sistema.

3.3 Esperimenti sullo Zero-shot Annotation Coaching

Gli esperimenti sono stati finalizzati alla valutazione delle capacità del modello OWLv2 per il compito di Zero-shot Annotation Coaching (vedi 2.5). Questo approccio consente di annotare automaticamente immagini senza la necessità di un dataset di addestramento specifico, riducendo significativamente il costo temporale richiesto nella generazione di dataset annotati.

3.3.1 Installazione OWLv2

Come primo step è necessaria l'installazione della libreria OWLv2, un modello pre-addestrato disponibile sulla piattaforma Hugging Face [41].

Installazione delle dipendenze:

```
pip install transformers torch torchvision
```

Successivamente è necessario scaricare il modello specifico OWLv2 da Hugging Face tramite il seguente snippet in Python:

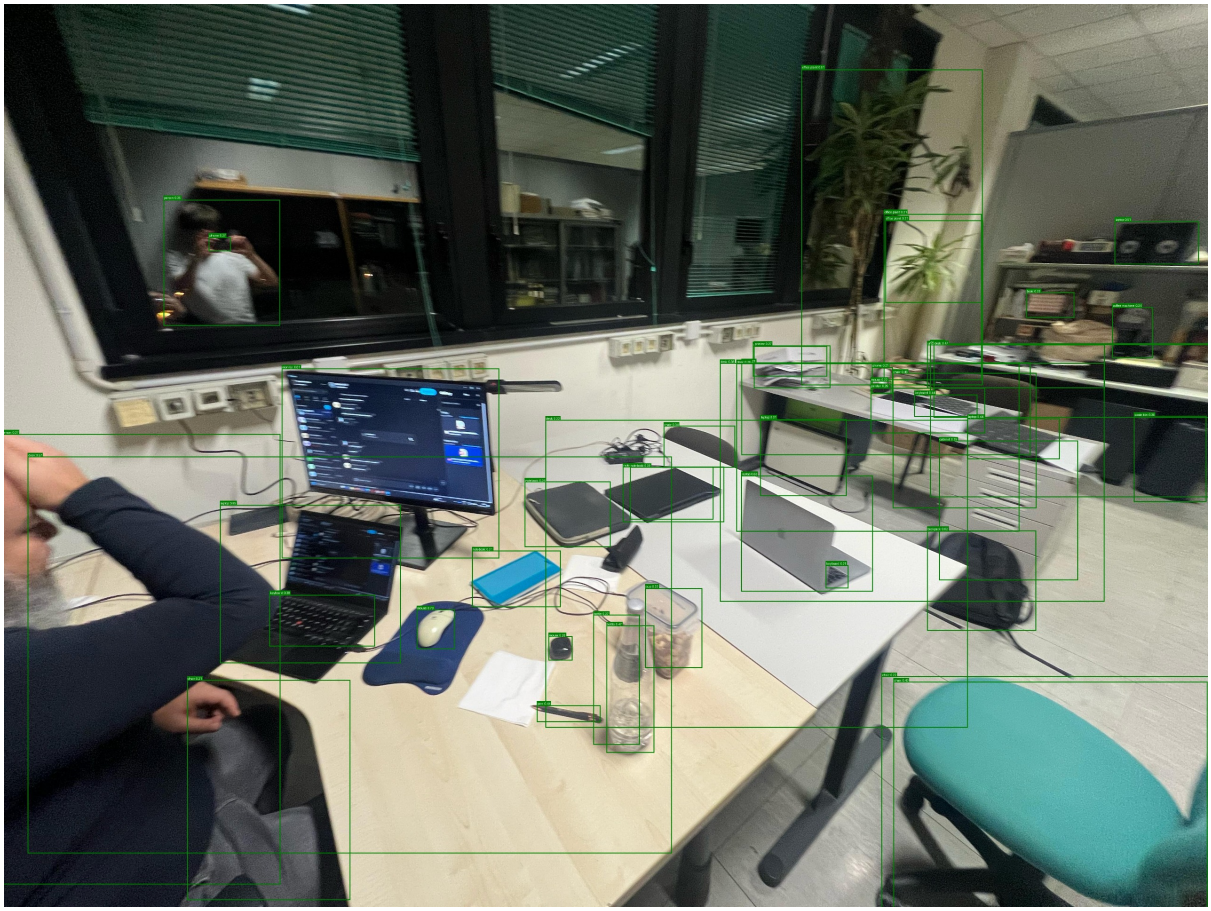

Figure 9: Immagine annotata automaticamente usando il modello owlv2.

```
from transformers import Owlv2ForObjectDetection

model = Owlv2ForObjectDetection.from_pretrained("google/owlv2-base-patch16")
```

3.3.2 Estrapolazione dei frame dai video

Per ottenere i singoli frame necessari per l'annotazione, è stato implementato uno script Python D.2 che automatizza il processo di estrazione dei frame da video sorgente in formato MP4. Tale script consente una configurazione flessibile della frequenza di estrazione (numero di frame al secondo), facilitando un utilizzo ottimale del materiale visivo disponibile. Lo pseudo-codice completo è mostrato nell'algoritmo 2.

3.3.3 Generazione automatica delle annotazioni

I frame estratti vengono successivamente passati allo script Python D.3.1 che utilizza il modello OWLv2 per generare annotazioni automatiche come mostrato in fig.9. Lo script implementato richiede in input una lista di classi target (token che il modello deve riconoscere), e restituisce come output un set di immagini annotate nel formato YOLOv8 standard. Lo pseudo-codice utilizzato per la generazione automatica delle annotazioni viene mostrato nell'algoritmo 3.

Le annotazioni sono state generate sfruttando come risorse hardware un server Dell PowerEdge R630[42] dotato delle seguenti specifiche:

Algorithm 2 ExtractFrames

```
1: if not DirExists(outputFolder) then
2:    createDir(outputFolder)
3: end if
4: files = listAllMP4(inputFolder)
5: totalFramesSaved = 0
6: for file in files do
7:    videoPath = joinPath(inputFolder, file)
8:    cap = openVideo(videoPath)
9:    totalFramesInVideo = getFrameCount(cap)
10:   for frameIdx = 0 to totalFramesInVideo do
11:      (ret, frame) = readFrame(cap)
12:      if not ret then
13:         break
14:      end if
15:      if (frameIdx mod 5) = 0 then
16:         frameFile = concat(totalFramesSaved, ".jpg")
17:         framePath = joinPath(outputFolder, frameFile)
18:         writeImage(framePath, frame)
19:         totalFramesSaved = totalFramesSaved + 1
20:      end if
21:   end for
22:   releaseVideo(cap)
23: end for
24: return totalFramesSaved
```

Algorithm 3 CreateAnnotations

```
device ← 'cpu'
model ← loadModel(owlv2).to(device)
processor ← loadProcessor(owlv2)
labels ← [c_1, c_2, c_3, ..., c_k]
framesFolder ← 'Frame'
outputFolder ← 'datasetOwl'
if not dirExists(outputFolder) then
    createDir(outputFolder)
end if
frameFiles ← listFrames(framesFolder)
for all file ∈ frameFiles do
    img ← loadImage(joinPath(framesFolder, file))
    inputs ← processor(labels, img)
    outputs ← modelForward(model, inputs)
    rawResults ← postProcessOD(outputs, 0.2)
    resultsNms ← applyNms(rawResults, 0.05)
    annotatedImg ← drawBoxes(img, resultsNms, labels)
    saveImage(joinPath(outputFolder, 'annotated_' + file), annotatedImg)
    annotationData ← buildAnnotations(file, resultsNms, labels)
    saveJson(joinPath(outputFolder, replaceExt(file, '.json')), annotationData)
end for
```

- CPU: 2x Intel Xeon E5-2680 v4 (28 core, 56 thread totali)
- Memoria RAM: 128 GB DDR4
- Storage: SSD NVMe 256GB

L'utilizzo di tale hardware ha permesso di elaborare contemporaneamente molteplici immagini, ottimizzando significativamente il tempo totale necessario per completare la fase di annotazione automatica.

3.3.4 Data augmentation del dataset annotato

L'algoritmo 4 descrive dettagliatamente l'implementazione del processo di data augmentation. Una volta prodotte le annotazioni iniziali, è necessario passare alla fase di data augmentation, essenziale per migliorare la generalizzazione del modello addestrato e prevenire fenomeni di overfitting. La data augmentation implementata comprende tecniche quali rotazione, ridimensionamento, traslazione, variazione della luminosità e contrasto, e flipping orizzontale e verticale. Tutti questi processi sono stati attentamente configurati per evitare artefatti visivi e l'uscita delle regioni di interesse dai limiti dell'immagine originale. La implementazione in linguaggio Python è visibile in D.3.2.

Al fine di massimizzare l'efficienza computazionale, l'intero processo di augmentation è stato parallelizzato utilizzando la libreria Python `multiprocessing`, consentendo di sfruttare pienamente le risorse CPU del server utilizzato. Tale approccio parallelo ha garantito un aumento significativo della produttività nella generazione di dataset estesi e diversificati per successive applicazioni di apprendimento automatico.

Algorithm 4 AugmentImagesWithAnnotations

```
framesDir ← 'Frame'
annDir ← 'datasetOwl'
outDir ← 'datasetAug'
createDirIfNotExists(outDir)
augmentations ← { T_1, T_2, T_3, ..., T_len(T) }
frameFiles ← listImages(framesDir)
for all file in frameFiles do
    imagePath ← joinPath(framesDir, file)
    annPath ← joinPath(annDir, replaceExt(file, '.json'))
    image ← loadImage(imagePath)
    annot ← loadJson(annPath)
    boxes ← getBoxesFrom(annot)
    for all (augName, augFunction) in augmentations do
        (augImg, augBoxes) ← augFunction(image, boxes)
        outImgName ← concat(baseName(file), '.format')
        outJsonName ← replaceExt(outImgName, '.json')
        saveImage(joinPath(outDir, outImgName), augImg)
        newAnn ← buildAnnotations(outImgName, annot, augBoxes)
        saveJson(joinPath(outDir, outJsonName), newAnn)
    end for
end for
```

3.3.5 Allenamento della rete YOLOv8

Il modello YOLOv8l composto da 43 milioni di parametri è stato allenato utilizzando 3 GPU A100 40Gb ed ha richiesto 114 ore per allenare il modello su 350 epcohe su un dataset composto da 13414 immagini per la fase di training, 2912 nella fase di validation e 2912 nella fase di test.

Il comando per poter allenare la rete è stato il seguente:

```
!yolo task=detect mode=train model=yolov8l.pt
 data=YOLOv8_dataset/data.yml epochs=350 device=0,1,2
  cache=ram workers=16 batch=18 imgsz=1920
```

A AIPassport

AIPassport è uno strumento progettato per aiutare a gestire e tracciare gli esperimenti di apprendimento automatico integrando Data Version Control (DVC) e MLflow. Semplifica il processo di organizzazione del progetto in diverse fasi, tracciando le modifiche e registrando i risultati. Questa guida contiene i passaggi per installare AIPassport, impostare il tuo modello personalizzato e gestire efficacemente i tuoi esperimenti di apprendimento automatico.

AIPassport sfrutta DVC[1] e MLflow[44] per tracciare le modifiche e gestire gli esperimenti. DVC è utilizzato per il data versioning e la gestione della pipeline, mentre MLflow gestisce il tracciamento degli esperimenti e la gestione dei modelli.

A.1 Come installare AIPassport

È possibile scaricare e installare AIPassport direttamente del repository GitHub con il seguente comando:

```
$ pip install git+https://github.com/Degik/aimodelpassport-colab-
   dev.git
```

Questo comando usa `pip` per installare il pacchetto AIPassport direttamente dal repository GitHub.

A.1.1 Usare l'ambiente Anaconda

Se stai lavorando con un ambiente virtuale (Anaconda, ecc), è importante attivare l'ambiente prima di procedere con l'installazione di AIPassport per assicurarsi che avvenga nella cartella giusta. Puoi attivare il tuo ambiente con il comando (nel caso di Anaconda):

```
$ conda activate env-name
```

Sostituisci `env-name` con il nome del tuo ambiente Anaconda.

A.2 Impostare il proprio modello custom con AIPassport

Per usare AIPassport è necessario dividire la struttura del tuo progetto in fasi o stage. Spesso gli stage in progetti di machine learning includono il preprocessing dei dati, l'addestramento e la produzione. Questo approccio modulare rende più semplice gestire e tenere traccia di ogni componente del progetto separatamente.

A.2.1 Configurare il tracking degli stage con MLflow

Una volta suddiviso il tuo progetto in fasi, è necessario inizializzare MLflow seguire nel dettaglio ogni fase del progetto:

```
from AIPassport import MLflowTracker
RUN_NAME = "PHASE_NAME"
tracker = MLflowTracker()
tracker.ActivateMLflow(RUN_NAME)
```

[1] Data Version Control[43]

A.2.2 Inizializzare AIPassport

Prima di avviare l'esperimento è necessario inizializzare AIPassport. Per farlo utilizza il seguente comando:

Sostituisci `PASH_NAME` con il nome della fase (esempio: "preporcess", "train", "production").

Successivamemte, è possibile inizializzare AIPassport usando:

```
from AIPassport import Passport
passport = Passport()
```

Chiamare il costrutture `Passport` inizializza AIPassport e genera diversi file di configurazione che sono essenziali per la gestione del progetto.

A.2.3 File di configurazione generato per AIPassport

Nello specifico i dati generati sono i seguenti:

- **datasetinfo.yaml** - Contiene tutte le informazioni riguardanti i dataset utilizzati nel progetto. È possibile compilare questo file con i dettagli riguardanti le fonti dei dati, i passaggi di data preprocessing e qualsiasi altra informazione che sia rilevante.
- **params.yaml** - Contiene tutti i parametri e le configurazioni per le diverse fasi presenti nel progetto. Questo file è cruciale per poter parametrizzare i vari stage della pipeline di esecuzione, Un esempio di `params.yaml` è possibile vederlo più in basso in A.4.
- **modelinfo.yaml** - Contiene le informazioni riguardo ai modelli utilizzati nel progetto, incluso quindi dettagli sull'architettura, iperparametri e ogni parametro custom.

A.2.4 Creare stage della pipeline con DVC

Dopo aver inizializzato AIPassport è possibile definire gli stage della pipeline utilizzando DVC. Per farlo è necessario chiamare il metodo `addStage`:

```
passport.AddStage()
```

Questo metodo legge la configurazione del file `params.yaml` ed in base a questo aggiorna il file `dvc.yaml`. DVC utilizza il file `dvc.yaml` per gestire la pipeline di esecuzine, tracciare le dipendenze e assicurare la riproducibilità dell progetto.

A.2.5 Eseguire gli esperimenti

Con gli stage della pipeline definiti, è possibile eseguire l'esperimento usando:

```
passport.RunExperiment()
```

Questo comando esegue gli stage della pipeline che sono stati definiti nel `dvc.yaml` file, Gestisce il controllo delle versioni dei dati, tiene traccia delle modifiche e registra i risultati su MLflow.

A.3 Come visualizzare gli esperimenti

Dopo aver eseguito gli esperimenti, è possibile visualizzare i risultati e tenere traccia dei progressi sull'interfaccia utente di MLflow; questo rende fornisce un'interfaccia web che permette di visualizzare metriche, parametri e artefatti associati agli esperimenti.

Per avviare MLflow UI, è necessario eseguire il comando:

```
$ mlflow ui
```

Di default MLflow sarà accessibile all'indirizzo `http://localhost:5000`; aprendo sarà possibile accedere all'interfaccia web.

A.3.1 Registrazione dei modelli custom in PyTorch

Nel caso si utilizzi PyTorch per costruire i propri modelli, MLflow non è in grado raccogliere autonomamente gli artefatti di default. Per essere sicuri che i modelli vengano registrati è necessario eseguire il log esplicitamente usando il modulo MLflow per PyTorch; è necessario quindi implementare il seguente script nel codice per il training:

```
signature = mlflow.models.infer_signature(
    image.detach().cpu().numpy(),
    output.detach().cpu().numpy()
)

mlflow.pytorch.log_model(
    bestModel,
    "model",
    signature=signature,
    code_paths=["include/modelstr"]
)
```

Nel codice:

- `infer_signature` definisce lo schema di input e output del modello, utile per il model serving..
- `mlflow.pytorch.log_model` registra il modello PyTorch su MLflow, incluso lo stato del modello e tutte le dipendenze del codice necessarie.
- `code_paths` dovrebbe includere qualsiasi libreria di modelli personalizzati o codice necessario per riprodurre il modello. Sostituisci `"include/modelstr"` con il percorso al codice del tuo modello.

Se il progetto utilizza librerie o strutture di modelli personalizzate, è importante includerle nel parametro `code_paths` quando si registra il modello. Ciò garantisce che MLflow abbia tutto il codice necessario per riprodurre e servire il modello in futuro.

A.4 How to Restart the Project

Se necessario, per riavviare il progetto o rieseguire gli esperimenti, AIPassport e DVC semplificano la riproduzione dei tuoi risultati. È necessario assicurarsi che i file di configurazione (`params.yaml`, `datasetinfo.yaml`, `modelinfo.yaml`) siano impostati correttamente con i parametri desiderati.

Per rieseguire la pipeline da zero, è possibile utilizzare i comandi DVC:

```
$ dvc repro
```

Questo comando indica a DVC di riprodurre le fasi della pipeline, eseguendo tutti i passaggi modificati o le cui dipendenze sono cambiate.

In alternativa può essere utilizzato:

```
passport.RunExperiment()
```

Questo rieseguirà gli esperimenti con le configurazioni correnti.

È necessario eliminare il file dvc.yaml e dvc.lock prima di rieseguirlo.

A.5 Esempio di `params.yaml`

Il file `params.yaml` è un file di configurazione YAML che definisce i parametri per MLflow, nonché ogni fase della pipeline. Di seguito è riportato un esempio:

```yaml
flow:
activate: True
experiment_name: "faster_rcnn_experiment_new"
pipeline_name: "faster_rcnn_experiment_new"

rocess:
task: preprocess
params:
    output_dir: "data/processed"
depends_on:
- data/raw/image/
- data/raw/mask/
- src/preprocess.py
output:
- data/processed/
execution: "python3 src/preprocess.py ${preprocess.params.output_dir}"

n:
task: train
params:
    images_train: "dataset/train/"
    anns_train: "dataset/train/_annotations.coco.json"
    images_val: "dataset/valid/"
    anns_val: "dataset/valid/_annotations.coco.json"
    batch_size: 14
    learning_rate: 0.05
    num_epochs: 2
    split: 0.2
    confidence: 0.8
    list_classes: ["person", "backpack", "luggage", "wheelchair"]
    eval_path: "train"

depends_on:
```

```
- data/raw/image/
- data/raw/mask/
- src/train.py
output:
- train/
execution: "python3 src/train.py ${train.params.images_train}
${train.params.anns_train} ${train.params.images_val}
${train.params.anns_val} ${train.params.batch_size}
${train.params.learning_rate} ${train.params.num_epochs}
${train.params.split} ${train.params.confidence}
${train.params.eval_path} fast_rcnn http://127.0.0.1:8080"


uction:
task: production
params:
    images_train: "dataset/train/"
    anns_train: "dataset/train/_annotations.coco.json"
    images_val: "dataset/valid/"
    anns_val: "dataset/valid/_annotations.coco.json"
    batch_size: 14
    learning_rate: 0.05
    num_epochs: 1
    split: 0.01
    confidence: 0.8
    list_classes: ["person", "backpack", "luggage", "wheelchair"]
    eval_path: "production"
depends_on:
- data/raw/image/
- data/raw/mask/
- src/production.py
output:
- production/
- best_model.pth
execution: "python3 src/production.py ${production.params.images_train}
${production.params.anns_train} ${production.params.images_val}
${production.params.anns_val} ${production.params.batch_size}
${production.params.learning_rate} ${production.params.num_epochs}
${production.params.split} ${production.params.confidence}
${production.params.eval_path} fast_rcnn http://127.0.0.1:8080"

```

Spiegazione delle voci su `params.yaml`

- La sezione `mlflow` configura le impostazioni MLflow, ad esempio se attivare o meno il tracciamento MLflow, il nome dell'esperimento e il nome della pipeline.
- Ogni fase (ad esempio, `preprocess`, `train`, `production`) definisce:

- `task`: Il nome del task.
- `params`: Parametri specifici per quella fase.
- `depends_on`: Un elenco di file o directory da cui dipende la fase. DVC lo usa per determinare se una fase deve essere rieseguita.
- `output`: Il risultato ottenuto dallo stage.
- `execution`: Il comando per eseguire la fase utilizzando i parametri definiti.

Esempio di parametri per una fase di train

- Nella fase `train`:
  - `images_train`, `anns_train`: Percorsi per immagini di addestramento e annotazioni.
  - `images_val`, `anns_val`: Percorsi per immagini di convalida e annotazioni.
  - `batch_size`, `learning_rate`, `num_epochs`: Iperparametri per l'allenamento.
  - `split`: La proporzione del dataset per la fase di validation.
  - `confidence`: Soglia per le predizioni.
  - `list_classes`: Lista delle classi nel dataset.
  - `eval_path`: Percorso per memorizzare i risultati della valutazione.

Utilizzo dei parametri nei comandi di esecuzione

I parametri vengono utilizzati nei comandi di esecuzione con una sintassi simile a `${train.params.batch_size}`. Ciò consente di modificare i parametri nel file `params.yaml` senza modificare gli script.

B Nvidia DGX (Deep GPU Xceleration) server

Il sistema di DGX[45] sviluppato da Nvidia ha molti benefici, in particolare quello disponibile presso l'ISTI-CNR è composta da 8 GPU A100[46] da 40Gb ognuna. Un totale di 320 Gb che può essere sfruttato per simulazioni che richiedono un grosso quantitativo di dati e l'uso specifico nell'IA. Questo è possibile grazie alla tecnologia NVLink[47] che permette lo scambio di dati in modo veloce ed efficiente tra le GPU sfruttando al massimo l'elevato parallelismo del sistema. Il sistema dispone di due processori AMD Rome 7742 da 64 core l'uno e 128 thread; un totale di 128 core e 256 thread.

B.1 Ambiente condiviso

È importante notare che la maggior parte degli incarichi non impiegheranno tutte le risorse a disposizione. Essendo questo un sistema condiviso da chi ne fa richiesta all'interno del laboratorio, in quanto tale ha dei limiti e possono essere lo spazio, ma sopratutto l'uso delle GPU. Se già impegnata una GPU sarà necessario cambiare la propria configurazione o addirittura attendere. Questo può capitare spesso, poiché molto utilizzato, ma quello che può influenzare in positivo è un corretto utilizzo delle risorse.

B.2 Come accedere alla DGX

Per ottenere l'accesso alla macchina occorre scrivere allo staff tecnico. A differenza di altri servizi messi a disposizione dal CNR, la DGX richiede la generazione di una chiave RSA privata e pubblica, da dare al tecnico addetto alla registrazione sulla macchina.

B.2.1 Generazione della chiave RSA

Per generare la propria chiave RSA è necessario usare il comando sottostante:

```
$ ssh-keygen -t rsa -b 4096
```

Il comando ssh-keygen dispone di diverse opzioni, le più importanti sono -t che permette di specificare l'algoritmo di generazione della chiave; in questo caso rsa. Mentre l'opzione -b permette di specificare la grandezza della chiave in bits; nell'esempio 4096.

Di default la chiave pubblica verrà generata col nome id_rsa.pub nella cartella ~/.ssh. Sarà necessario passare questo file al tecnico adetto della macchina.

B.2.2 Generazione della passphrase (Opzionale)

Nel processo di generazione della chiave verrà chiesto se si desidera generare la chiave con una passphrase. Questo non incide in alcuno modo sul lavoro e neanche sulla creazione della chiave, a discrezione dell'utente può essere una protezione aggiuntiva, nel caso il computer finisse nelle mani sbagliate. Senza passhphrase non si può accedere alla macchina, ovviamente anche se opzionale, è suggerito adottarla.

B.2.3 Login sulla DGX

Una volta creato l'utente sulla macchina basterà eseguire il comando **ssh**[48] sottostante:

```
$ ssh nomeutente@indirizzodgx
```

Una volta eseguito il comando potrebbe essere necessario inserire la passphrase B.2.1, ma dipende da come avete generato la chiave.

B.2.4 Impostazioni personalizzate (Opzionale)

È possibile personalizzare il login su connessione remota ssh tramite il file **config** situato nella cartella ~/**.ssh**

```
Host CNRDGX           # Nome custom per l'host
HostName host.address # Indirizzo di collegamento all'host
User nomeutente       # Nome utente dell'host
Port                  # Per porte custom ssh
```

In casi più particolari è possibile spostare la propria chiave privata o personalizzare l'accesso ad una specifica chiave. Può essere utile per più connessioni ssh che richiedono una chiave RSA.

Basta generare o rinominare una chiave con un nome custom. Per esempio **id_rsa_custom**, che può essere utilizzata aggiungendo la seguente linea al file di configurazione dell'host:

```
IdentityFile ~/.ssh/id_rsa_custom
```

Una volta impostato il file config, è possibile utilizzare il comando sottostante per accedere:

```
$ ssh CNRDGX
```

È possibile utilizzare questa configurazione con il comando **scp** [49].

B.3 Gestione delle risorse della DGX

Usare un sistema condiviso richiede molta attenzione specialmente se non presente una politica per le risorse del sistema. Quando viene creato un container può essere necessario specificare il numero di cpu o gpu disponibili nell'ambiente da poter usare (vedi B.4.3).

nvidia-smi Questo comando che restituisce una visualizzazione più semplice della situazione delle gpu. Il problema di questo comando risulta nell'assenza delle informazioni relative a carico cpu e ram, inoltre stampa una volta e non aggiorna periodicamente; questo può essere risolto con la combinazione di comandi *watch -n 1 nvidia-smi*[2] (per interrompere il comando premere CTRL+C).

nvtop Permette una visualizzazione in tempo reale delle risorse impiegate (cpu, ram, gpu), quando questo viene usato è possibile osservare con attenzione la memoria usata al momento su ciascuna gpu, ma cosa più importante a differenza di altri comandi permette di vedere con molta precisione i carichi di lavoro sulle gpu. Può essere molto utile per chiarire se la nostra gpu sul task impiegato stia usando le risorse con efficienza, oppure presenti dei colli di bottiglia che possano essere risolti con una scrittura del software più accurata (per terminare il comando premere q).

[2] Aggiorna ogni secondo l'ouput del comando su schermo

ncdu Questo permette di avere una visione chiara di dove sono allocate la maggior parte delle risorse in termini di spazio di archiviaizone. Il comando può richiedere un pò di tempo, inoltre il punto in cui viene eseguito è importante, per esempio per avere una visione chiara della partizione */raid* è necessario trovarsi li mentre si esegue (per terminare il comando premere q).

B.4 Utilizzo di Docker sulla DGX

Come spiegato nel paragrafo precedente [B.1] la DGX è un ambiente condiviso. Questo implica che per poterla usare in modo efficiente è necessario avere una buona padronanza dei concetti chiave di Docker[50]. Questo lavoro su dei container, ambienti isolati che permettono di limitare le risorse specifiche per un compito specifico. In particolare è possibile creare un container Docker partendo da un'immagine; questa andrà costruita mediante l'uso di un Dockerfile.

B.4.1 La struttura della DGX

Ogni utente avrà a propria disposizione solo e solamente la propria **home directory**. Questo rende necessario l'utilizzo dei container, così da poter installare tutte le dipendenze necessarie in un ambiente isolato senza intaccare il lavoro di altre persone. La DGX è costituita da due partizioni principali:

- */dev/md0 montata su /*
- */dev/md127 montata su /raid*

È necessario utilizzare la partizione */raid* per eseguire i propri progetti, questo perché è la più capiente e l'unica che permette la scrittura. **Installare elementi su / impossibiliterebbe gli utenti dal creare e modificare i container per mancanza di spazio**

B.4.2 Come creare un Dockerfile

La prima cosa da fare è usare il comando touch[51] per creare un file denominato Dockerfile:

```
$ touch Dockerfile
```

Una volta creato il file usando uno dei comandi (gedit, vim, ecc..) è possibile modificarlo. Normalmente è possibile richiamare container già esistenti e non dover lavorare da zero. Sul web è possibile trovare molti di questi su siti come Docker hub e Nvidia Non è insolito trovare all'inizio di ogni Dockerfile l'istruzione **FROM** con accanto un riferimento all'immagine di partenza, per esempio:

```
FROM conda/miniconda3
```

Una volta importato il container, è possibile successivamente installare eventuali pacchetti aggiuntivi; in questo caso, un server jupyter[52]. Essendo moltissime le istruzioni disponibili nella creazione di un Dockerfile ecco un esempio semplice per far sì che possa essere intuitiva la metodologia d'uso, anche se normalmente richiede molta pratica configurare immagini complesse.

```
# Richiama un container gia esistente con sopra jupyter e anaconda
FROM quay.io/jupyter/scipy-notebook:2024-05-27

# Seleziona un utente all'interno del container
USER root # In questo caso root

EXPOSE 8888
```

Per maggiori informazioni sulle istruzioni, è consigliato guardare la guida ufficiale di Docker.

B.4.3 Generare l'immagine dal Dockerfile

Una volta configurato il Dockerfile, è necessario eseguire il seguente comando per generare l'immagine:

```
docker build -t nome_immagine:tag .
```

Per verificare se l'immagine è presente si può utilizzare il comando:

```
docker image ls
```

B.4.4 Esempio di avvio di un Container Docker

Ecco un esempio di script necessario ad avviare un container Docker:

```
docker run \
    -d \
    --rm \
    --shm-size=16G \
    --cpus 64 \
    --gpus '"device=0,1,7"' \
    --group-add users \
    --user $(id -u):$(id -g) \
    --volume /usr/lib/x86_64-linux-gnu:/usr/lib/x86_64-linux-gnu:ro \
    --volume /usr/lib/x86_64-linux-gnu/libnvidia-ml.so:/usr/lib/x86_64-linux-
        gnu/libnvidia-ml.so:ro \
    --volume /usr/lib/x86_64-linux-gnu/libcuda.so:/usr/lib/x86_64-linux-gnu/
        libcuda.so:ro \
    -p 10000:8888 \
    -e GRANT_SUDO=yes \
    jupyter_test:latest
```

Il comando `docker run` viene utilizzato per avviare un container basato su un'immagine specificata. Di seguito la spiegazione dei parametri utilizzati:

- `-d`: esegue il container in modalità detached (in background).
- `-rm`: rimuove automaticamente il container alla chiusura.
- `-shm-size=16G`: assegna 16 GB di memoria condivisa.
- `-cpus 64`: limita il numero di CPU utilizzabili dal container a 48.
- `-gpus '"device=0,1,7"'`: specifica quali GPU assegnare al container in base al loro id.
- `-group-add users`: aggiunge il container al gruppo `users`.

- `-user $(id -u):$(id -g)`: esegue il container con l'ID utente e gruppo dell'utente corrente.
- `-volume` : montano risorse necessarie all'interno del container
- `-p 10000:8888`: mappa la porta 8888 del container sulla porta 10000 dell'host.
- `-e GRANT_SUDO=yes`: abilita i privilegi sudo nel container.
- `jupyter_test`: specifica l'immagine da utilizzare per il container.

Per eseguire il container utilizzando questo script, basta salvare il codice in un file chiamato, ad esempio, `run_container.sh`, per poi eseguire il comando:

```
chmod +x drun.sh
./drun.sh
```

Questo avvierà il container con i parametri definiti.
Al momento dell'avvio viene generato un id del container:

```
$ ./drun.sh
1b5fab0819d641d593b03e52a647a0812614fa482540e3fbbc99e3e14d130c78
```

B.4.5 Come recuperare la chiave jupyter

Nel caso di un server jupyter la chiave verrà visualizzata al momento dell'avvio, per tanto è possibile recuperarla con il comando:

```
$ docker container logs 1
    b5fab0819d641d593b03e52a647a0812614fa482540e3fbbc99e3e14d130c78
```

Una volta eseguito il comando verrà visualizzata ad un certo punto dell'output la seguente linea:

```
Or copy and paste one of these URLs:
    http://1b5fab0819d6:8888/lab?token=3
        f454e3a4c5e1d9b317ba30ee0edab5bf6b85e939278ffc6
    http://127.0.0.1:8888/lab?token=3
        f454e3a4c5e1d9b317ba30ee0edab5bf6b85e939278ffc6
```

Figure 10: Esempio di output del container con il server jupyter

Recuperando il token sarà possibile accedere alla piattaforma web del proprio jupyter server all'indirizzo *hostname:port*.

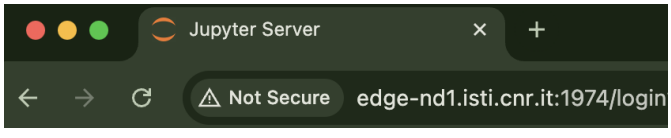

Figure 11: Esempio di indirizzo sul server alla porta 1974

È possibile effettuare il login direttamente con il token (vedi immagine B.4.5)

Figure 12: Login tramite token

Una procedura più consigliata è creare la password tramite il token di accesso (vedi immagine B.4.5), quando il container viene terminato la password è il token non saranno più validi perciò sarà necessario ripetere la procedura.

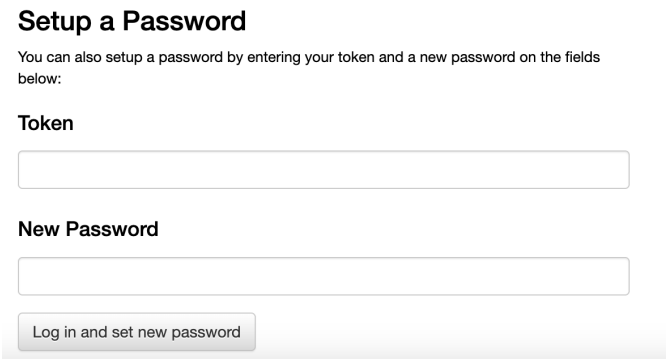

Figure 13: Creazione della password

B.4.6 Come terminare il container

Per motivi di risorse, sicurezza e manutenzione può essere necessario uccidere il container. Per farlo è possibile usare il comando *docker ps* per vedere i container attivi; è possibile cercare il proprio dal nome per poi ricavare il container ID. Infine basta eseguire il comando:

```
docker container kill container_id
```

C NVidia Deepstream

DeepStream, sviluppato da NVIDIA, è un framework altamente scalabile per l'elaborazione e l'inferenza in tempo reale su flussi video. È progettato per integrare librerie come TensorRT e fornire pipeline ottimizzate per il rilevamento di oggetti, la classificazione e altre attività di visione artificiale. Questo capitolo descrive le funzionalità principali di DeepStream, la configurazione ideale per pipeline basate su modelli YOLO e le sfide affrontate durante l'implementazione.

C.1 DeepStream e l'integrazione con GStreamer

DeepStream utilizza un'architettura a pipeline composta da plugin modulari, ciascuno dei quali svolge un ruolo specifico nella gestione del flusso video. Il plugin centrale per l'inferenza dei modelli è `nvinfer`, che supporta modelli in formati ottimizzati come TensorRT, ONNX e altri. DeepStream si basa sull'infrastruttura di **GStreamer**, un framework open-source progettato per creare pipeline multimediali modulari ed efficienti. GStreamer organizza una pipeline in una serie di plugin, chiamati *elementi*, ognuno dei quali svolge un ruolo specifico. Questi elementi comunicano tra loro attraverso dei *pad*, che trasportano i dati in modo sincrono all'interno della pipeline.

Ogni elemento è responsabile di una fase del flusso multimediale, come la decodifica, il preprocessamento, l'inferenza e la visualizzazione dei risultati. DeepStream utilizza GStreamer per implementare pipeline complesse ottimizzate per applicazioni di visione artificiale in tempo reale.
Tra i plugin principali utilizzati in DeepStream troviamo:

- `nvstreammux`, che raccoglie flussi video multipli, li sincronizza e li converte in un formato uniforme.
- `nvinfer`, che esegue inferenze utilizzando modelli ottimizzati con TensorRT.
- `nvdsosd`, che sovrappone informazioni rilevanti ai frame video, come bounding box ed etichette.
- `nveglglessink`, che consente di visualizzare i risultati.

Un esempio di pipeline costruita con GStreamer per eseguire inferenze su un flusso video è la seguente:

```
gst-launch-1.0 filesrc location=input.mp4 ! decodebin ! nvstreammux ! nvinfer \
! nvdsosd ! nveglglessink
```

Ogni plugin è configurabile tramite parametri o file di configurazione dedicati. Ad esempio, il plugin `nvinfer` richiede un file di configurazione che specifichi i dettagli del modello, i parametri di inferenza e le risorse hardware utilizzate. DeepStream arricchisce GStreamer con plugin specifici per l'accelerazione GPU e l'inferenza in tempo reale, mantenendo la flessibilità e l'efficienza di GStreamer come infrastruttura di base.
Un esempio di configurazione ideale per un modello YOLOv8 è il seguente:

```
[primary-gie]
enable=1
gpu-id=0
model-engine-file=../models/yolov8.engine
onnx-file=../models/yolov8.onnx
```

```
batch-size=1
interval=0
gie-unique-id=1
network-type=0
num-detected-classes=80
```

Questo file definisce il modello da utilizzare (`onnx-file`), il file precompilato con TensorRT (`model-engine-file`) e altri parametri chiave come la dimensione del batch e il numero di classi. Il preprocessamento è configurato tramite parametri aggiuntivi, che assicurano che i frame siano normalizzati alle dimensioni richieste dal modello.

Durante l'esecuzione della pipeline, DeepStream elabora ogni frame secondo il seguente flusso:

- Il video viene acquisito e suddiviso in frame dal plugin sorgente.
- Ogni frame viene preprocessato per adattarsi alle specifiche del modello.
- Il plugin `nvinfer` utilizza TensorRT per eseguire l'inferenza, generando predizioni come bounding box e punteggi di confidenza.
- I risultati vengono visualizzati o salvati tramite i plugin di post-elaborazione e output.

C.2 Conversione di modelli ONNX in TensorRT Engine

DeepStream richiede che i modelli utilizzati per l'inferenza siano convertiti in formato `TensorRT Engine` (`.engine`). Questo formato, ottimizzato per l'hardware NVIDIA, consente di migliorare le prestazioni dell'inferenza tramite tecniche come la fusione dei layer, la riduzione della precisione numerica (FP16 o INT8) e l'ottimizzazione della memoria.

La conversione di un modello `ONNX` in TensorRT Engine avviene attraverso un processo composto da più fasi:

1. **Parsing del modello**: TensorRT legge il file `.onnx` e lo traduce in un grafo di esecuzione.
2. **Ottimizzazione**: TensorRT applica tecniche di ottimizzazione, come la riorganizzazione dei layer e la riduzione della precisione numerica.
3. **Generazione del file `.engine`**: Il modello ottimizzato viene salvato in un formato eseguibile direttamente dal plugin `nvinfer`.

C.2.1 Utilizzo di DeepStream-YOLO

Nel caso di modelli YOLO, il repository DeepStream-YOLO fornisce script specifici per semplificare la conversione dei modelli ONNX in TensorRT Engine. Questo repository offre una serie di strumenti per superare limitazioni di TensorRT, come la gestione di layer personalizzati e il supporto per dimensioni di input non standard. La conversione di un modello YOLO in TensorRT Engine utilizzando il repository DeepStream-YOLO segue i seguenti passi:

- **Preparazione del modello ONNX**: Verifica che il modello YOLO sia in formato ONNX e che i layer personalizzati siano supportati.

- **Utilizzo degli script di conversione**: Il repository include script per adattare il modello ONNX al formato TensorRT. Ad esempio:

```
python3 yolo_to_onnx.py -m yolov5s.onnx
python3 onnx_to_tensorrt.py -m yolov5s.onnx -o yolov5s.engine
```

- **Generazione del file di configurazione**: Il repository fornisce template per configurare il file `config_infer_primary.txt`, che può essere personalizzato in base alle specifiche del modello YOLO.

C.2.2 Problemi tecnici

Durante la conversione, sono stati riscontrati problemi tecnici:

- **Incompatibilità tra versioni**: Modelli YOLO più recenti richiedono modifiche manuali agli script di conversione.
- **Limiti di TensorRT**: Alcuni layer, come `IShuffle`, hanno causato errori durante il parsing.
- **Dimensioni di input non standard**: Modelli YOLO specifici richiedono dimensioni di input particolari, che devono essere configurate correttamente nel file `config_infer_primary.txt`.

Nonostante le difficoltà, l'utilizzo del repository DeepStream-YOLO ha notevolmente semplificato l'integrazione di modelli YOLO con DeepStream, garantendo un flusso di lavoro più efficiente.

C.2.3 Dettagli dell'implementazione

In prima istanza per implementare DeepStream è stato scelto il modello YOLOv5 per conoscenza pregressa di tale tecnologia utilizzata in altri lavori.. Il modello è stato prima convertito dal formato PyTorch (`.pt`) a ONNX (`.onnx`), successivamente ottimizzato con TensorRT per l'uso nel plugin `nvinfer`. La configurazione iniziale ha seguito i principi descritti nella sezione precedente, con adattamenti specifici al contesto.

La conversione è stata effettuata utilizzando lo script ufficiale `export.py`, disponibile nel repository YOLOv5. Il comando utilizzato è:

```
python export.py --weights yolov5s.pt --img 640 --batch 1 --simplify
```

Questo processo genera un modello semplificato in formato ONNX, ma durante la conversione sono emersi problemi di compatibilità con TensorRT. Gli errori principali riscontrati includevano:

```
ERROR: [TRT]: Your ONNX model has been generated with INT64 weights, while
TensorRT does not natively support INT64. Attempting to cast down to INT32.
ERROR: [TRT]: IShuffleLayer /model.24/Reshape: reshaping failed for tensor.
```

Questi problemi erano dovuti a pesi salvati in formato `INT64` e a incompatibilità nelle dimensioni di alcuni layer.

Per la soluzione del problema si è provato il repository esterno BlueMirrors Yolov5-TensorRT, che prometteva una conversione più stabile. Tuttavia, molte librerie risultavano deprecate.

Per risolvere problemi di compatibilità, è stato adottato DeepStream 7.1 con JetPack 5.1.2. Questo aggiornamento ha migliorato significativamente la stabilità della pipeline e la compatibilità con modelli YOLOv5 ottimizzati. L'ambiente è stato configurato utilizzando un'immagine Docker personalizzata:

```
FROM nvcr.io/nvidia/deepstream:7.1-samples

RUN apt-get update && apt-get install -y \
    python3-pip \
    python3-opencv \
    && pip3 install pyds
```

Si è verificato in seguito che modelli più recenti, come YOLOv8, offrono in modo nativo maggiore compatibilità con TensorRT.

Per migliorare le prestazioni del modello, è stata implementata la calibrazione INT8, un processo che riduce il consumo di risorse ottimizzando il modello per lavorare con numeri interi a 8 bit. Questo richiede un dataset di calibrazione rappresentativo.

Il file di calibrazione generato è stato specificato nella configurazione di `nvinfer`:

```
int8-calib-file=../calibration_data/calib_cache.txt
```

D Script List

In questo appendice vengono riportati tutti gli script in linguaggio Python utilizzati durante l'attività sperimentale svolta nel tirocinio.

D.1 Merge-Dataset script

Il codice Python utilizzato per l'unione dei dataset preesistenti.

```
import os
import shutil
import yaml

def merge_datasets(dataset_paths, output_path="dataset"):
    """
    Unisce piu' dataset YOLO (train/valid/test con immagini e
       annotazioni)
    in un unico dataset consolidato.
    """
    os.makedirs(output_path, exist_ok=True)

    for split in ["train", "valid", "test"]:
        os.makedirs(os.path.join(output_path, split, "images"),
           exist_ok=True)
        os.makedirs(os.path.join(output_path, split, "labels"),
           exist_ok=True)

    all_class_names = set()

    for ds_path in dataset_paths:
        data_yaml_path = os.path.join(ds_path, "data.yaml")
        if not os.path.isfile(data_yaml_path):
            print(f"[WARNING] No data.yaml found in {ds_path}. Skipping
               ...")
            continue

        with open(data_yaml_path, "r") as f:
            data_dict = yaml.safe_load(f)

        if "names" in data_dict:
            for c in data_dict["names"]:
                all_class_names.add(c)

        for split in ["train", "valid", "test"]:
            orig_images_dir = os.path.join(ds_path, split, "images")
            orig_labels_dir = os.path.join(ds_path, split, "labels")

            dest_images_dir = os.path.join(output_path, split, "images"
               )
            dest_labels_dir = os.path.join(output_path, split, "labels"
               )

            if not os.path.isdir(orig_images_dir):
                continue

            for filename in os.listdir(orig_images_dir):
                src_file = os.path.join(orig_images_dir, filename)
```

```
                if os.path.isfile(src_file):
                    shutil.copy2(src_file, dest_images_dir)

            if os.path.isdir(orig_labels_dir):
                for filename in os.listdir(orig_labels_dir):
                    src_file = os.path.join(orig_labels_dir, filename)
                    if os.path.isfile(src_file):
                        shutil.copy2(src_file, dest_labels_dir)

    merged_data = {
        "train": os.path.join("train", "images"),
        "val": os.path.join("valid", "images"),
        "test": os.path.join("test", "images"),
        "nc": len(all_class_names),
        "names": sorted(list(all_class_names))
    }

    with open(os.path.join(output_path, "data.yaml"), "w") as f:
        yaml.dump(merged_data, f, sort_keys=False)

    print("Merge completed!")
    print(f"Merged dataset path: {output_path}")
    print(f"Detected classes: {merged_data['names']}")

if __name__ == "__main__":
    datasets_to_merge = [
        "Dataset_Dirt",
        "Dataset_Packets",
        "Dataset_Phone",
        "Dataset_PlasticBottle",
    ]
    merge_datasets(datasets_to_merge, output_path="dataset")
```

D.2 Frames extractor script

Il codice Python utilizzato per l'estrazione dei frame da un video in input

```
import os
import cv2
from tqdm import tqdm

# Percorsi delle cartelle
cartella_input = "VideoDataset/"
cartella_output = "Frame"

# Crea la cartella di output se non esiste
if not os.path.exists(cartella_output):
    os.makedirs(cartella_output)

# Ottieni la lista dei file .mp4 nella cartella di input
files = [f for f in os.listdir(cartella_input) if f.lower().endswith(".
    mp4")]

total_frames_saved = 0

# Itera sui video con una barra di avanzamento
for file in tqdm(files, desc="Elaborazione video"):
    percorso_video = os.path.join(cartella_input, file)
    nome_video = os.path.splitext(file)[0]

    cap = cv2.VideoCapture(percorso_video)
    total_frames_video = int(cap.get(cv2.CAP_PROP_FRAME_COUNT))


    for frame_idx in tqdm(range(total_frames_video), desc=f"Frame di {
        nome_video}", leave=False):
        ret, frame = cap.read()
        if not ret:
            break  # Fine del video

        # Salva il frame ogni 5 frame (supponendo 30fps)
        if frame_idx % 5 == 0:
            #nome_file_frame = f"{nome_video}_frame{frame_idx}_{
                total_frames_saved}.jpg"
            nome_file_frame = f"{total_frames_saved}.jpg"
            percorso_frame = os.path.join(cartella_output,
                nome_file_frame)
            cv2.imwrite(percorso_frame, frame)
            total_frames_saved += 1

    cap.release()

print(f"Totale frame salvati: {total_frames_saved}")
```

D.3 Zero-shot Annotation Coaching

In questa sezione sono stati raggruppati tutti gli script utilizzati nell'approccio descritto in2.5

D.3.1 Annotation script

Il codice Python utilizzato per creare le annotazione in automatico usando OWLv2.

```
import os
import json
import torch
from transformers import Owlv2Processor, Owlv2ForObjectDetection
from PIL import Image, ImageDraw, ImageFont
import torchvision.ops as ops
from tqdm import tqdm

# Dispositivo e modelli
device = "cpu"
model = Owlv2ForObjectDetection.from_pretrained("google/owlv2-base-
    patch16-ensemble").to(device)
processor = Owlv2Processor.from_pretrained("google/owlv2-base-patch16-
    ensemble")

# Label da rilevare
texts = [
    "chair", "desk", "keyboard", "monitor", "mouse", "printer",
    "phone", "office plant", "conference table", "person",
    "cabinet", "laptop", "book", "backpack", "cup", "bottle", "pen",
    "notebook", "waste bin", "box", "container", "lamp", "pc case",
    "paper", "stereo", "charger", "locker", "window", "door", "coat",
    "whiteboard", "coat hanger", "audio speaker", "cable", "eraser",
    "headphone case", "camera", "knife", "kitchen knife"
]

# Percorsi
frames_folder = "Frame"
output_folder = "dataset_owl"
if not os.path.exists(output_folder):
    os.makedirs(output_folder)
frame_files = [f for f in os.listdir(frames_folder) if f.lower().
    endswith((".jpg", ".jpeg", ".png"))]
font = ImageFont.load_default()

# Elaborazione dei frame
for frame_file in tqdm(frame_files, desc="Elaborazione frame"):
    frame_path = os.path.join(frames_folder, frame_file)
    image = Image.open(frame_path).convert("RGB")

    inputs = processor(text=texts, images=image, return_tensors="pt").
        to(device)
    with torch.no_grad():
        outputs = model(**inputs)

    target_sizes = torch.tensor([image.size[::-1]], device=device)
    results = processor.post_process_object_detection(outputs,
        target_sizes=target_sizes, threshold=0.2)[0]
    boxes = results["boxes"]
```

```
    scores = results["scores"]
    labels = results["labels"]

    # Non-Maximum Suppression
    nms_threshold = 0.05
    keep_indices = []
    for current_label in labels.unique():
        inds = (labels == current_label).nonzero(as_tuple=False).
            squeeze(1)
        boxes_class = boxes[inds]
        scores_class = scores[inds]
        keep = ops.nms(boxes_class, scores_class, nms_threshold)
        keep_indices.append(inds[keep])
    if len(keep_indices) > 0:
        keep_indices = torch.cat(keep_indices)
    else:
        keep_indices = torch.tensor([])
    results_nms = {
        "boxes": boxes[keep_indices] if keep_indices.numel() > 0 else
            torch.empty((0, 4)),
        "scores": scores[keep_indices] if keep_indices.numel() > 0 else
            torch.empty((0,)),
        "labels": labels[keep_indices] if keep_indices.numel() > 0 else
            torch.empty((0,), dtype=torch.int64)
    }

    # Annotazione immagine
    image_draw = image.copy()
    draw = ImageDraw.Draw(image_draw)
    for box, score, label in zip(results_nms["boxes"], results_nms["
        scores"], results_nms["labels"]):
        box_list = [round(i, 2) for i in box.tolist()]
        draw.rectangle(box_list, outline="green", width=3)
        text_label = f"{texts[label]}: {score:.2f}"
        bbox = draw.textbbox((0, 0), text_label, font=font)
        text_width, text_height = bbox[2]-bbox[0], bbox[3]-bbox[1]
        draw.rectangle((box_list[0], box_list[1]-text_height-4,
            box_list[0]+text_width+4, box_list[1]), fill="green")
        draw.text((box_list[0]+2, box_list[1]-text_height-2),
            text_label, fill="white", font=font)
    annotated_image_path = os.path.join(output_folder, f"annotated_{
        frame_file}")
    image_draw.save(annotated_image_path)

    # Salvataggio annotazioni JSON
    annotation_dict = {"file": frame_file, "annotations": []}
    for box, score, label in zip(results_nms["boxes"], results_nms["
        scores"], results_nms["labels"]):
        annotation = {
            "box": [round(i, 2) for i in box.tolist()],
            "score": round(score.item(), 2),
            "label": texts[label]
        }
        annotation_dict["annotations"].append(annotation)
    annotation_file_path = os.path.join(output_folder, f"{os.path.
        splitext(frame_file)[0]}.json")
    with open(annotation_file_path, "w") as f:
        json.dump(annotation_dict, f, indent=4)
```

```
print("Elaborazione completata.")
```

D.3.2 Data augmentation script

Il codice Python utilizzato per effettuare le data-augmentations partendo dai frames estratti dal video originale.

```
import os
import json
import math
import numpy as np
from PIL import Image, ImageEnhance, ImageFilter
from concurrent.futures import ThreadPoolExecutor, as_completed
from tqdm import tqdm

# Configurazione dei percorsi
frames_dir = "Frame"              # Cartella dei frame "puliti"
annotations_dir = "dataset_owl"   # Cartella dei file JSON di
    annotazione
output_dir = "dataset_aug"        # Cartella di output per immagini
    augmentate e annotazioni

os.makedirs(output_dir, exist_ok=True)

# Funzioni di trasformazione geometrica
def rotate_image_and_boxes(image, boxes, angle):
    angle_rad = math.radians(angle)
    cos_theta = math.cos(angle_rad)
    sin_theta = math.sin(angle_rad)
    w, h = image.size
    cx, cy = w / 2, h / 2
    new_w = int(math.ceil(w * abs(cos_theta) + h * abs(sin_theta)))
    new_h = int(math.ceil(w * abs(sin_theta) + h * abs(cos_theta)))
    ncx, ncy = new_w / 2, new_h / 2
    rotated_image = image.rotate(angle, expand=True)
    new_boxes = []
    for box in boxes:
        xmin, ymin, xmax, ymax = box
        corners = np.array([
            [xmin, ymin],
            [xmax, ymin],
            [xmax, ymax],
            [xmin, ymax]
        ])
        corners[:, 0] -= cx
        corners[:, 1] -= cy
        new_corners = np.zeros_like(corners)
        new_corners[:, 0] = corners[:, 0] * cos_theta - corners[:, 1] *
            sin_theta
        new_corners[:, 1] = corners[:, 0] * sin_theta + corners[:, 1] *
            cos_theta
        new_corners[:, 0] += ncx
        new_corners[:, 1] += ncy
        new_boxes.append([
            float(np.min(new_corners[:, 0])), float(np.min(new_corners
                [:, 1])),
```

```
            float(np.max(new_corners[:, 0])), float(np.max(new_corners
                [:, 1]))
        ])
    return rotated_image, new_boxes

def hflip_image_and_boxes(image, boxes):
    w, h = image.size
    flipped_image = image.transpose(Image.FLIP_LEFT_RIGHT)
    new_boxes = []
    for box in boxes:
        xmin, ymin, xmax, ymax = box
        new_boxes.append([w - xmax, ymin, w - xmin, ymax])
    return flipped_image, new_boxes

def vflip_image_and_boxes(image, boxes):
    w, h = image.size
    flipped_image = image.transpose(Image.FLIP_TOP_BOTTOM)
    new_boxes = []
    for box in boxes:
        xmin, ymin, xmax, ymax = box
        new_boxes.append([xmin, h - ymax, xmax, h - ymin])
    return flipped_image, new_boxes

# Funzioni composite di trasformazione geometrica
def identity(image, boxes):
    return image, boxes

def rot90(image, boxes):
    return rotate_image_and_boxes(image, boxes, 90)

def rot180(image, boxes):
    return rotate_image_and_boxes(image, boxes, 180)

def rot270(image, boxes):
    return rotate_image_and_boxes(image, boxes, 270)

def hflip(image, boxes):
    return hflip_image_and_boxes(image, boxes)

def vflip(image, boxes):
    return vflip_image_and_boxes(image, boxes)

def hflip_rot90(image, boxes):
    img, boxes = hflip(image, boxes)
    return rot90(img, boxes)

def vflip_rot90(image, boxes):
    img, boxes = vflip(image, boxes)
    return rot90(img, boxes)

# Trasformazione geometrica: shear orizzontale
def shear_x_image_and_boxes(image, boxes, shear_factor=0.2):
    w, h = image.size
    offset_x = 0
    if shear_factor < 0:
        offset_x = int(round(abs(shear_factor) * h))
    new_w = int(round(w + abs(shear_factor) * h))
    new_h = h
```

```
    matrix = (1, shear_factor, offset_x, 0, 1, 0)
    sheared_image = image.transform((new_w, new_h), Image.AFFINE,
       matrix, resample=Image.BICUBIC)
    new_boxes = []
    for box in boxes:
        xmin, ymin, xmax, ymax = box
        corners = np.array([
            [xmin, ymin],
            [xmax, ymin],
            [xmax, ymax],
            [xmin, ymax]
        ])
        new_corners = np.zeros_like(corners)
        new_corners[:, 0] = corners[:, 0] + shear_factor * corners[:,
           1] + offset_x
        new_corners[:, 1] = corners[:, 1]
        new_boxes.append([
            float(np.min(new_corners[:, 0])), float(np.min(new_corners
               [:, 1])),
            float(np.max(new_corners[:, 0])), float(np.max(new_corners
               [:, 1]))
        ])
    return sheared_image, new_boxes

# Trasformazioni non geometriche (non alterano le bounding box)
def gaussian_noise(image, boxes, mean=0, std=10):
    np_image = np.array(image).astype(np.float32)
    noise = np.random.normal(mean, std, np_image.shape)
    noisy_image = np.clip(np_image + noise, 0, 255).astype(np.uint8)
    return Image.fromarray(noisy_image), boxes

def brightness_increase(image, boxes, factor=1.2):
    enhancer = ImageEnhance.Brightness(image)
    return enhancer.enhance(factor), boxes

def brightness_decrease(image, boxes, factor=0.8):
    enhancer = ImageEnhance.Brightness(image)
    return enhancer.enhance(factor), boxes

def contrast_increase(image, boxes, factor=1.5):
    enhancer = ImageEnhance.Contrast(image)
    return enhancer.enhance(factor), boxes

def contrast_decrease(image, boxes, factor=0.7):
    enhancer = ImageEnhance.Contrast(image)
    return enhancer.enhance(factor), boxes

def blur(image, boxes, radius=2):
    return image.filter(ImageFilter.GaussianBlur(radius)), boxes

# Dizionario delle augmentations
augmentations = {
    "identity": identity,
    "hflip": hflip,
    "vflip": vflip,
    "rot90": rot90,
    "rot180": rot180,
    "rot270": rot270,
```

```
    "hflip_rot90": hflip_rot90,
    "vflip_rot90": vflip_rot90,
    "gaussian_noise": gaussian_noise,
    "bright_increase": brightness_increase,
    "bright_decrease": brightness_decrease,
    "contrast_increase": contrast_increase,
    "contrast_decrease": contrast_decrease,
    "blur": blur,
    "shear_x": shear_x_image_and_boxes
}

# Processing di un'immagine con le relative annotazioni
def process_image(frame_filename):
    try:
        frame_path = os.path.join(frames_dir, frame_filename)
        annotation_filename = os.path.splitext(frame_filename)[0] + ".
           json"
        annotation_path = os.path.join(annotations_dir,
           annotation_filename)

        image = Image.open(frame_path).convert("RGB")
        with open(annotation_path, "r") as f:
            ann = json.load(f)
        boxes = [a["box"] for a in ann.get("annotations", [])]

        generated_files = []
        for aug_name, aug_func in augmentations.items():
            aug_image, aug_boxes = aug_func(image, boxes)
            base_name = os.path.splitext(frame_filename)[0]
            out_img_name = f"{base_name}_aug_{aug_name}.jpg"
            out_json_name = f"{base_name}_aug_{aug_name}.json"
            out_img_path = os.path.join(output_dir, out_img_name)
            out_json_path = os.path.join(output_dir, out_json_name)
            aug_image.save(out_img_path)
            new_ann = {
                "file": out_img_name,
                "annotations": []
            }
            for original, new_box in zip(ann.get("annotations", []),
               aug_boxes):
                new_ann["annotations"].append({
                    "box": [round(coord, 2) for coord in new_box],
                    "score": original.get("score", 1.0),
                    "label": original.get("label", "")
                })
            with open(out_json_path, "w") as f:
                json.dump(new_ann, f, indent=4)
            generated_files.append(out_img_name)
        return frame_filename, generated_files
    except Exception as e:
        return frame_filename, str(e)

# Elaborazione in multithread dei file
def main():
    frame_files = [f for f in os.listdir(frames_dir) if f.lower().
       endswith((".jpg", ".jpeg", ".png"))]
    results = []
    with ThreadPoolExecutor(max_workers=os.cpu_count()) as executor:
```

```
        futures = {executor.submit(process_image, filename): filename
            for filename in frame_files}
        for future in tqdm(as_completed(futures), total=len(futures),
            desc="Processing images"):
            res = future.result()
            results.append(res)
    print("Processing completed.")
    for orig, outcome in results:
        if isinstance(outcome, list):
            print(f"{orig}: generated {len(outcome)} images.")
        else:
            print(f"{orig}: error -> {outcome}")

if __name__ == "__main__":
    main()
```

D.3.3 Inference script

Il codice Python per effettuare l'inferenza sul modello Yolo allenato.

```
import argparse
import os
import cv2
import numpy as np
from ultralytics import YOLO

# Lista custom delle classi
CUSTOM_LABELS = [
    "chair", "desk", "keyboard", "monitor", "mouse", "printer",
    "phone", "office plant", "conference table", "person",
    "cabinet", "laptop", "book", "backpack", "cup", "bottle", "pen",
    "notebook", "waste bin", "box", "container", "lamp", "pc case",
    "paper", "stereo", "charger", "locker", "window", "door", "coat",
    "whiteboard", "coat hanger", "audio speaker", "cable", "eraser",
    "headphone case", "camera", "knife", "kitchen knife"
]

def annotate_image_custom(image, results):
    """
    Disegna le bounding box e le label custom sull'immagine.
    """
    boxes = results[0].boxes.xyxy.cpu().numpy()  # [x1, y1, x2, y2]
    confidences = results[0].boxes.conf.cpu().numpy()
    classes = results[0].boxes.cls.cpu().numpy().astype(int)
    for box, conf, cls in zip(boxes, confidences, classes):
        x1, y1, x2, y2 = map(int, box)
        label = CUSTOM_LABELS[cls] if cls < len(CUSTOM_LABELS) else str
            (cls)
        text = f"{label}: {conf:.2f}"
        cv2.rectangle(image, (x1, y1), (x2, y2), (0, 255, 0), 2)
        cv2.putText(image, text, (x1, y1 - 10),
                    cv2.FONT_HERSHEY_SIMPLEX, 0.6, (0, 255, 0), 2)
    return image

def process_image_file(model, source, output, conf_threshold):
    results = model.predict(source, conf=conf_threshold)
    image = cv2.imread(source)
```

```
    if image is None:
        print(f"Errore nel caricamento dell'immagine: {source}")
        return
    annotated_img = annotate_image_custom(image, results)
    if output:
        cv2.imwrite(output, annotated_img)
        print(f"Immagine annotata salvata in: {output}")
    if os.environ.get("DISPLAY", None):
        try:
            cv2.imshow("YOLOv8 Inference", annotated_img)
            cv2.waitKey(0)
        except cv2.error:
            print("cv2.imshow non supportato in questo ambiente.")
    try:
        cv2.destroyAllWindows()
    except cv2.error:
        pass

def process_video_file(model, source, output, conf_threshold):
    cap = cv2.VideoCapture(source)
    if not cap.isOpened():
        print("Errore nell'apertura del video.")
        return
    fps = cap.get(cv2.CAP_PROP_FPS)
    width  = int(cap.get(cv2.CAP_PROP_FRAME_WIDTH))
    height = int(cap.get(cv2.CAP_PROP_FRAME_HEIGHT))
    fourcc = cv2.VideoWriter_fourcc(*'mp4v')
    writer = None
    if output:
        writer = cv2.VideoWriter(output, fourcc, fps, (width, height))
    print("Elaborazione del video. Premi 'q' per uscire (se
       visualizzazione abilitata).")
    while True:
        ret, frame = cap.read()
        if not ret:
            break
        results = model(frame, conf=conf_threshold)
        annotated_frame = annotate_image_custom(frame.copy(), results)
        if writer:
            writer.write(annotated_frame)
        if os.environ.get("DISPLAY", None):
            try:
                cv2.imshow("YOLOv8 Inference", annotated_frame)
                if cv2.waitKey(1) & 0xFF == ord('q'):
                    break
            except cv2.error:
                pass
    cap.release()
    if writer:
        writer.release()
    try:
        cv2.destroyAllWindows()
    except cv2.error:
        pass
    if output:
        print(f"Video annotato salvato in: {output}")

def main():
```

```
    parser = argparse.ArgumentParser(
        description="Inferenze con YOLOv8 fine-tuned e label custom")
    parser.add_argument("--model", type=str, required=True,
                        help="Percorso al modello YOLOv8 fine-tuned (
                            file .pt)")
    parser.add_argument("--source", type=str, required=True,
                        help="Percorso all'immagine o al video di input
                            ")
    parser.add_argument("--output", type=str, default="",
                        help="(Opzionale) Percorso per salvare l'output
                             annotato")
    parser.add_argument("--conf", type=float, default=0.25,
                        help="Soglia di confidenza (default: 0.25)")
    args = parser.parse_args()
    model = YOLO(args.model)
    ext = os.path.splitext(args.source)[1].lower()
    if ext in [".jpg", ".jpeg", ".png"]:
        process_image_file(model, args.source,
                           args.output if args.output else None, args.
                              conf)
    elif ext in [".mp4", ".avi", ".mov", ".mkv"]:
        process_video_file(model, args.source,
                           args.output if args.output else None, args.
                              conf)
    else:
        print("Formato file non supportato. Usa immagine (.jpg, .jpeg,
            .png) o video (.mp4, .avi, .mov, .mkv).")

if __name__ == "__main__":
    main()
```

References

[1] "Fostering Artificial Intelligence Trust for Humans towards the optimization of trustworthiness through large-scale pilots in critical domains (FAITH)," 2024. Last retrieved December 11, 2025.

[2] "ISO/IEC TS 5723:2022 Trustworthiness—vocabulary," 2022. Last retrieved December 11, 2025.

[3] NIST, "Engineering trustworthy secure systems," 2022. NIST Special Publication NIST SP 800-160v1r1.

[4] S. Barocas, M. Hardt, and A. Narayanan, *Fairness and Machine Learning: Limitations and Opportunities*. MIT Press, 2023.

[5] A. Jobin, M. Ienca, and E. Vayena, "The global landscape of AI ethics guidelines," *Nature Machine Intelligence*, vol. 1, no. 9, pp. 389–399, 2019.

[6] K. Greff, F. Belletti, L. Beyer, C. Doersch, Y. Du, D. Duckworth, D. J. Fleet, D. Gnanapragasam, F. Golemo, C. Herrmann, T. Kipf, A. Kundu, D. Lagun, I. Laradji, Hsueh-Ti, Liu, H. Meyer, Y. Miao, D. Nowrouzezahrai, C. Oztireli, E. Pot, N. Radwan, D. Rebain, S. Sabour, M. S. M. Sajjadi, M. Sela, V. Sitzmann, A. Stone, D. Sun, S. Vora, Z. Wang, T. Wu, K. M. Yi, F. Zhong, and A. Tagliasacchi, "Kubric: A scalable dataset generator," 2022.

[7] B. O. Community, *Blender - a 3D modelling and rendering package*. Blender Foundation, Stichting Blender Foundation, Amsterdam, 2018.

[8] A. Kar, A. Prakash, M.-Y. Liu, E. Cameracci, J. Yuan, M. Rusiniak, D. Acuna, A. Torralba, and S. Fidler, "Meta-sim: Learning to generate synthetic datasets," in *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, October 2019.

[9] Roboflow, "Roboflow." Accessed: 2025-01-27. Roboflow is a platform for building computer vision applications, providing tools for image annotation, dataset management, model training, and deployment.

[10] N. Ravi, V. Gabeur, Y.-T. Hu, R. Hu, C. Ryali, T. Ma, H. Khedr, R. Rädle, C. Rolland, L. Gustafson, E. Mintun, J. Pan, K. V. Alwala, N. Carion, C.-Y. Wu, R. Girshick, P. Dollár, and C. Feichtenhofer, "Sam 2: Segment anything in images and videos," 2024.

[11] B. Sekachev, N. Manovich, M. Zhiltsov, A. Zhavoronkov, D. Kalinin, B. Hoff, TOsmanov, D. Kruchinin, A. Zankevich, DmitriySidnev, M. Markelov, Johannes222, M. Chenuet, a andre, telenachos, A. Melnikov, J. Kim, L. Ilouz, N. Glazov, Priya4607, R. Tehrani, S. Jeong, V. Skubriev, S. Yonekura, vugia truong, zliang7, lizhming, and T. Truong, "opencv/cvat: v1.1.0," Aug. 2020.

[12] A. Dosovitskiy, L. Beyer, A. Kolesnikov, D. Weissenborn, X. Zhai, T. Unterthiner, M. Dehghani, M. Minderer, G. Heigold, S. Gelly, J. Uszkoreit, and N. Houlsby, "An image is worth 16x16 words: Transformers for image recognition at scale," 2021.

[13] M. Minderer, A. Gritsenko, and N. Houlsby, "Scaling open-vocabulary object detection," 2024.

[14] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, "Attention is all you need," 2023.

[15] F. Rosenblatt, "The perceptron: A probabilistic model for information storage and organization in the brain [j]," *Psychol. Review*, vol. 65, pp. 386 – 408, 11 1958.

[16] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[17] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," *CoRR*, vol. abs/1502.03167, 2015.

[18] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: A simple way to prevent neural networks from overfitting," *Journal of Machine Learning Research*, vol. 15, no. 56, pp. 1929–1958, 2014.

[19] C. H. Sudre, W. Li, T. Vercauteren, S. Ourselin, and M. J. Cardoso, "Generalised dice overlap as a deep learning loss function for highly unbalanced segmentations," *CoRR*, vol. abs/1707.03237, 2017.

[20] A. Bansal, K. Sikka, G. Sharma, R. Chellappa, and A. Divakaran, "Zero-shot object detection," 2018.

[21] K. J. Joseph, S. Khan, F. S. Khan, and V. N. Balasubramanian, "Towards open world object detection," 2021.

[22] P. Documentation, "Bceloss — pytorch 2.0 documentation." `https://pytorch.org/docs/stable/generated/torch.nn.BCELoss.html`. Accessed: 26-03-2025.

[23] "Plastic bottle dataset." URL: https://universe.roboflow.com/fyp-li8zz/plastic-bottle-2.0. Accessed: 2025-01-27.

[24] "Dirty dataset." URL: https://universe.roboflow.com/dirt-dn7iu/dirt-oq0xp. Accessed: 2025-01-27.

[25] "Packet dataset." URL: https://universe.roboflow.com/twoseptember/packets. Accessed: 2025-01-27.

[26] "Phone dataset." URL: https://universe.roboflow.com/objectdetection-lryar/dataset-un2vt. Accessed: 2025-01-27.

[27] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You only look once: Unified, real-time object detection," 2016.

[28] G. J. et al., "Yolov5 by ultralytics," 2021. Open-source object detection framework for real-time object detection.

[29] Ultralytics, "Yolov11: State-of-the-art object detection," 2024. Most recent YOLOv5 model, offering cutting-edge object detection performance.

[30] Ultralytics, "Yolov10: Real-time end-to-end object detection," 2024. Eliminates the need for Non-Maximum Suppression (NMS) during inference by employing consistent dual assignments for training. This approach reduces inference latency and improves prediction efficiency.

[31] Ultralytics, "Ultralytics: Yolov5, yolov8, yolov10, yolov11." Accessed: 2025-01-27. Ultralytics is a company specializing in artificial intelligence, particularly known for their work on the YOLO (You Only Look Once) object detection models. They offer a range of resources and tools for developers, including pre-trained models, datasets, and training pipelines.

[32] N. Corporation, "Nvidia jetson: Embedded platforms for ai at the edge." Accessed: 2025-01-27. NVIDIA Jetson is a family of embedded computing boards designed for AI applications at the edge, enabling high-performance machine learning and computer vision tasks in a compact form factor.

[33] N. Corporation, "Nvidia: The ai computing company." Accessed: 2025-01-27. NVIDIA is a technology company that designs graphics processing units (GPUs) for the gaming and professional markets, as well as system on a chip units (SoCs) for the mobile computing and automotive market.

[34] NVIDIA, "Deepstream sdk for intelligent video analytics." Software development kit for building real-time video analytics applications.

[35] N. Corporation, "Nvidia tensorrt: High-performance deep learning inference." Accessed: 2025-01-27. TensorRT is an SDK for high-performance deep learning inference. It includes a deep learning inference optimizer and runtime that delivers low latency and high throughput for deep learning inference applications.

[36] K. Fatahalian, J. Sugerman, and P. Hanrahan, "Understanding the efficiency of gpu algorithms for matrix-matrix multiplication," *Proceedings of the ACM SIGGRAPH/EUROGRAPHICS conference on Graphics hardware*, pp. 133–137, 2004. This paper provides a detailed analysis of the efficiency of GPU algorithms for matrix-matrix multiplication, a fundamental operation in many graphics and scientific computing applications. It discusses the architectural features of GPUs that contribute to their performance and provides insights into how to optimize algorithms for these architectures.

[37] Ultralytics, "Yolov8: Next-generation object detection," 2023. Latest version of YOLOv5, offering improved accuracy and speed.

[38] M. Luciano, "Deepstream-yolo: Tensorrt optimization for yolo models," 2021. TensorRT-based optimization for deploying YOLO models on NVIDIA DeepStream.

[39] A. AI, "Clearml: Open-source mlops platform." Accessed: 2025-01-27. ClearML is an open-source MLOps platform that helps automate and manage machine learning workflows, including experiment tracking, dataset versioning, model deployment, and monitoring.

[40] Y. Bengio, "Practical recommendations for gradient-based training of deep architectures," *Neural networks: Tricks of the trade*, pp. 437–478, 2012. This paper provides

a comprehensive overview of practical recommendations for training deep neural networks, including a discussion on the role of batch size in stochastic gradient descent and its impact on training stability and generalization performance.

[41] H. Face, "Owlv2." https://huggingface.co/docs/transformers/model_doc/owlv2. Accessed on March 26, 2025.

[42] Dell, "Dell poweredge r630 specification sheet."

[43] Iterative, "Dvc website." https://dvc.org. Last access: 25-09-2024.

[44] Iterative, "Mlflow website." https://mlflow.org. Last access: 25-09-2024.

[45] Nvidia, "Nvidia dgx a100 the universal system for ai infrastructure." https://www.nvidia.com/content/dam/en-zz/Solutions/Data-Center/nvidia-dgx-a100-datasheet.pdf. Last Access: 27-5-2024.

[46] N. Corporation, "Nvidia a100 tensor core gpu: The flagship gpu for ai and hpc." White Paper, May 2020. Available: https://images.nvidia.com/aem-dam/en-zz/Solutions/data-center/nvidia-ampere-architecture-whitepaper.pdf.

[47] Nvidia, "Switch nvlink e nvswitch." https://www.nvidia.com/it-it/data-center/nvlink/. Last Access: 27-5-2024.

[48] "ssh(1) — linux manual page." https://man7.org/linux/man-pages/man1/ssh.1.html. Last Access: 28-5-2024.

[49] "scp(1) - linux man page." https://linux.die.net/man/1/scp. Last Access: 28-5-2024.

[50] "Overview of the get started guide." https://docs.docker.com/get-started/. Last Access: 4-6-2024.

[51] "touch(1) - linux man page." https://www.man7.org/linux/man-pages/man1/touch.1.html. Last Access: 4-6-2024.

[52] P. Jupyter, "Jupyter notebook." https://jupyter.org/, 2015–2025.