

Dalla Rete YOLO al One-Shot Learning: Sviluppo di un Sistema per il Rilevamento di una Barca

Stefano Monte 152122

2 ottobre 2024

Indice

1 Rete YOLO	2
1.1 Obiettivo del progetto	2
1.2 YOLOv5	2
1.2.1 Architettura e Vantaggi	2
1.3 Preparazione del dataset	3
1.3.1 Estrazione di Immagini dai Video	3
1.3.2 Data augmentation	3
1.3.3 Annotazioni del Dataset	4
1.3.4 Conversione delle annotazioni	4
1.3.5 Processo di Normalizzazione	5
1.3.6 Suddivisione del dataset	5
1.4 Addestramento della rete	5
1.5 Validazione della rete	6
1.6 Conclusioni	6
2 Few-shot/One-shot learning	9
2.1 Obiettivo	9
2.2 Stato dell'arte	9
2.2.1 Few-shot learning (FSL)	9
2.2.2 One-Shot Learning (OSL)	10
2.2.3 One-Shot Object Detection (OSOD)	10
2.3 Tecnica utilizzata	11
2.4 Libreria Detectron2	11
2.5 Ambiente di sviluppo	11
2.6 Sviluppo della rete	12
2.7 Test della rete	12
2.8 Conclusioni e sviluppi futuri	14

Capitolo 1

Rete YOLO

1.1 Obiettivo del progetto

La prima parte del progetto ha come obiettivo la creazione di una rete neurale convoluzionale basata sull'architettura YOLO (You Only Look Once), progettata per il rilevamento in tempo reale di un'imbarcazione. Questa rete sarà utilizzata in combinazione con un drone dotato di telecamera, il cui scopo è localizzare e identificare con precisione la barca all'interno dell'ambiente circostante. Il drone, sfruttando il modello YOLO, sarà in grado di processare i frame video acquisiti dalla sua telecamera, individuando l'imbarcazione anche in contesti dinamici, come il movimento del drone o la variazione delle condizioni ambientali.

1.2 YOLOv5

YOLOv5 [4] è una delle versioni più recenti e avanzate della famiglia di algoritmi YOLO (You Only Look Once), utilizzati per il rilevamento di oggetti in immagini e video in tempo reale. Sviluppato da Ultralytics, YOLOv5 si distingue per la sua combinazione di alta precisione e velocità di elaborazione, rendendolo una scelta ideale per applicazioni che richiedono un rilevamento rapido e accurato.

1.2.1 Architettura e Vantaggi

In termini di architettura, YOLOv5 sfrutta una rete neurale convoluzionale profonda (CNN) per individuare oggetti nelle immagini. Rispetto alle versioni precedenti, YOLOv5 ha introdotto una serie di miglioramenti che lo rendono particolarmente adatto per il rilevamento in scenari reali, caratterizzati da condizioni ambientali complesse come riflessi sull'acqua, variazioni di luminosità e movimenti non prevedibili. Alcuni dei principali vantaggi di YOLOv5 sono:

- Modello più leggero e performante: YOLOv5 è stato progettato per essere più efficiente rispetto alle versioni precedenti, grazie alla sua architettura ottimizzata. Questo lo rende ideale per l'implementazione su dispositivi con risorse limitate, come i droni, che richiedono modelli leggeri ma altamente efficaci.
- Miglioramenti nell'accuratezza: Grazie a ottimizzazioni nell'architettura della rete neurale e all'addestramento su set di dati più ampi e diversificati, YOLOv5 offre una maggiore precisione nel rilevamento degli oggetti.
- Facilità di utilizzo e personalizzazione: YOLOv5 è stato progettato con particolare attenzione alla facilità d'uso. Grazie all'implementazione open-source fornita da Ultralytics, completa di script per l'addestramento e la valutazione, è possibile personalizzare il modello in modo relativamente semplice.
- Flessibilità e adattabilità: Un ulteriore punto di forza di YOLOv5 è la sua flessibilità. La rete può essere personalizzata in base a requisiti specifici, come la dimensione degli oggetti da rilevare, il numero di classi o i parametri di addestramento.

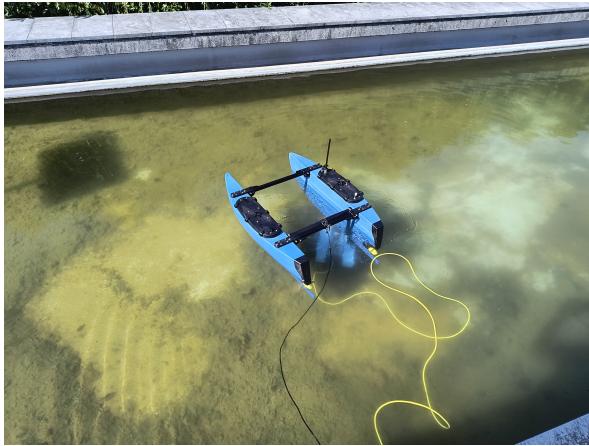


Figura 1.1: Immagine della barca da riconoscere

- Supporto per l'elaborazione parallela: Grazie alla sua architettura ottimizzata, YOLOv5 sfrutta appieno le moderne GPU e i processori multi-core per l'elaborazione parallela. Questo consente di elaborare i dati video in tempo reale.

Nel caso specifico, è stata utilizzata la versione YOLOv5s, una delle varianti più leggere e veloci di YOLOv5. Questa versione offre un ottimo compromesso tra accuratezza e velocità, mantenendo un tempo di inferenza ridotto, essenziale per il rilevamento in tempo reale su dispositivi come i droni. YOLOv5s è ottimizzato per rilevare oggetti di varie dimensioni all'interno delle immagini, il che è particolarmente vantaggioso per individuare barche di diverse forme e dimensioni a differenti distanze.

1.3 Preparazione del dataset

Per la realizzazione del dataset destinato al rilevamento dell'imbarcazione, è stata adottata una strategia che prevede l'utilizzo di immagini e video dell'imbarcazione in movimento sull'acqua. Questa scelta è motivata dalla necessità di simulare un contesto reale in cui la barca si trova in un ambiente acquatico, rendendo così il modello di rilevamento più efficace e realistico durante le fasi di test e implementazione.

1.3.1 Estrazione di Immagini dai Video

Per ottenere una quantità sufficiente di immagini rappresentative dal materiale video, è stato deciso di estrarre un frame ogni secondo. Questa metodologia garantisce che le immagini ottenute siano sufficientemente distanti l'una dall'altra in termini di contenuto visivo, consentendo una maggiore varietà nel dataset. Infatti, l'estrazione di frame consecutivi potrebbe risultare in immagini simili, limitando la diversità necessaria per un efficace addestramento del modello. Pertanto, adottando un intervallo di un secondo tra le estrazioni, si assicura che i frame siano significativamente differenti, facilitando la classificazione delle immagini come esemplari unici.

Questo primo processo ha permesso di ottenere circa 200 immagini di partenza.

1.3.2 Data augmentation

Per espandere il dataset iniziale di circa 200 immagini fino a circa 1000 immagini, è stata implementata una strategia di data augmentation utilizzando le librerie PyTorch e imgaug nello script python *DataAugmentation.py*. Questo passaggio è stato sviluppato per aumentare la varietà e la quantità dei dati disponibili per l'addestramento della rete neurale, migliorando così la capacità del modello di generalizzare su nuove immagini.

Configurazione delle Trasformazioni di PyTorch

Utilizzando transforms di torchvision, sono state impostate diverse tecniche di data augmentation:

- Rotazione: Le immagini possono essere ruotate fino a 40 gradi, introducendo variazioni angolari nel dataset.
- Traslazione: Viene applicata una traslazione orizzontale e verticale fino al 30% delle dimensioni dell'immagine originale, simulando spostamenti in varie direzioni.
- Zoom: L'uso di RandomResizedCrop permette di applicare zoom, ridimensionando casualmente le immagini.
- Ribaltamento: Le immagini possono essere ribaltate orizzontalmente, aumentando la varietà angolare del dataset.
- Conversione in Tensor: Alla fine, l'immagine viene convertita in un tensor, che è il formato richiesto da PyTorch per il successivo addestramento.

Applicazione di Augmentations Aggiuntive

In aggiunta alle trasformazioni offerte da torchvision, sono state implementate altre tecniche tramite imgaug per aumentare ulteriormente la variabilità delle immagini:

- Blur Gaussiano: Applicato al 30% delle immagini, per simulare condizioni di visibilità variabile.
- Rumore Gaussiano: Introduce rumore in modo casuale, aumentando la robustezza del modello rispetto a disturbi.
- Modifica della Luminosità: L'intensità luminosa delle immagini è variata, simulando diverse condizioni di illuminazione.
- Rotazioni Aggiuntive: Permette rotazioni supplementari nell'intervallo di -20 a 20 gradi.

Output

Il codice procede a caricare ogni immagine dalla cartella di input utilizzando la libreria PIL e applicando le trasformazioni di PyTorch. Successivamente, le immagini risultanti vengono elaborate tramite imgaug per generare le versioni augmentate. Per ogni immagine originale, vengono generate 5 varianti augmentate e salvate nella cartella di output con un prefisso per distinguerle.

1.3.3 Annotazioni del Dataset

Le annotazioni delle immagini sono state eseguite utilizzando il software open source VGG Annotation. Questo strumento è particolarmente utile per il compito di annotazione delle immagini, poiché offre un'interfaccia intuitiva che semplifica il processo di disegno delle bounding box attorno agli oggetti di interesse—in questo caso, la barca. Il processo di annotazione ha comportato l'apertura di ciascuna immagine nel software, dove sono stati disegnati manualmente i riquadri attorno alle barche presenti. Ogni annotazione è stata accompagnata da un'etichetta che identifica l'oggetto, consentendo al modello di apprendere a riconoscere la barca durante la fase di addestramento. VGG Annotation supporta l'esportazione delle annotazioni in vari formati, facilitando la creazione di un dataset compatibile con le esigenze del modello YOLO.

1.3.4 Conversione delle annotazioni

Uno dei passaggi fondamentali per l'addestramento del modello YOLOv5 è la preparazione delle etichette nel formato corretto. YOLOv5 richiede infatti che, per ogni immagine, sia presente un file di annotazione in formato .txt, che contiene le coordinate della bounding box e l'ID della classe corrispondente all'oggetto rilevato. Le informazioni all'interno di ciascun file .txt devono essere strutturate nel seguente formato:

$$\langle \text{class_id}, \text{x_center}, \text{y_center}, \text{width}, \text{height} \rangle$$

- class_id: rappresenta l'ID della classe dell'oggetto da rilevare (nel nostro caso, la barca).

- `x_center` e `y_center`: sono le coordinate del centro della bounding box, normalizzate rispetto alla larghezza e all'altezza dell'immagine. Queste coordinate, quindi, non vengono espresse in pixel assoluti, ma come frazioni relative alle dimensioni dell'immagine, il che consente al modello di essere robusto a diverse risoluzioni.
- `width` e `height`: rappresentano rispettivamente la larghezza e l'altezza della bounding box, anche queste normalizzate rispetto alle dimensioni dell'immagine.

1.3.5 Processo di Normalizzazione

Il processo di normalizzazione delle etichette è stato eseguito attraverso uno script Python denominato `NormalizeForYOLO.py`. Questo script ha il compito di:

1. Leggere i dati di annotazione da un file JSON (prodotto in fase di annotazione delle immagini, ad esempio tramite il software VGG Annotation).
2. Convertire le coordinate assolute delle bounding box, riportate in pixel, in coordinate normalizzate, secondo le regole del formato YOLOv5. La normalizzazione viene effettuata dividendo la posizione del centro della bounding box e le sue dimensioni per la larghezza e l'altezza totali dell'immagine.
3. Generare un file .txt per ogni immagine, contenente i dati normalizzati nel formato richiesto. Il file .txt viene creato con lo stesso nome dell'immagine corrispondente, ma con estensione .txt, in modo che YOLOv5 possa associarli facilmente durante l'addestramento.

1.3.6 Suddivisione del dataset

Per l'addestramento del modello, si è scelto di suddividere il dataset in due parti con un rapporto di 80:20, destinando l'80% delle immagini alla fase di training e il restante 20% alla fase di validation.

Per fare ciò è stato sviluppato uno script chiamato `TrainTestSplit.py` il quale:

1. Definisce i percorsi delle cartelle per immagini e annotazioni.
2. Crea cartelle per le immagini e annotazioni di training e test, se non esistono già.
3. Raccoglie tutti i file immagine con estensione .jpg e .jpeg.
4. Mescola casualmente le immagini utilizzando un seme fisso per ripetibilità.
5. Divide le immagini in base a un rapporto 80% training e 20% test.
6. Copia le immagini e le annotazioni corrispondenti nelle rispettive cartelle (training e test).

1.4 Addestramento della rete

Durante il processo di training, il modello è stato addestrato per 30 epoche con parametri ottimizzati per migliorare progressivamente le prestazioni di rilevamento dell'oggetto target, una barca. Il processo di apprendimento è stato avviato con un learning rate iniziale di 0.01, associato a un fattore di decadimento dei pesi (weight decay) pari a 0.0005. Per garantire una stabilità iniziale durante l'apprendimento, sono stati utilizzati 3 epoch di warmup, permettendo al modello di adattarsi gradualmente ai dati prima di aumentare l'aggressività dell'aggiornamento dei pesi. Nel corso delle epoche, si è osservato un costante miglioramento delle perdite (loss) associate al rilevamento della box (`box_loss`), alla classificazione dell'oggetto (`cls_loss`), e alla presenza dell'oggetto (`obj_loss`). Ad esempio, la box loss è diminuita costantemente da 0.1147 nell'epoca iniziale fino a 0.0151 nell'ultima epoca, indicando che il modello ha appreso in modo accurato la posizione e le dimensioni della barca all'interno delle immagini. Anche la perdita relativa alla presenza dell'oggetto è calata da 0.0291 a 0.0052, suggerendo che il modello ha affinato la sua capacità di distinguere tra oggetti rilevanti e non rilevanti. Le metriche di valutazione come la precisione, il recall e l'Average Precision (AP) hanno mostrato risultati estremamente positivi. Nelle prime fasi, la precisione si è attestata intorno al 70%, ma è migliorata rapidamente fino a raggiungere valori prossimi al 100% nell'ultima epoca, con una precisione del 99.98%.

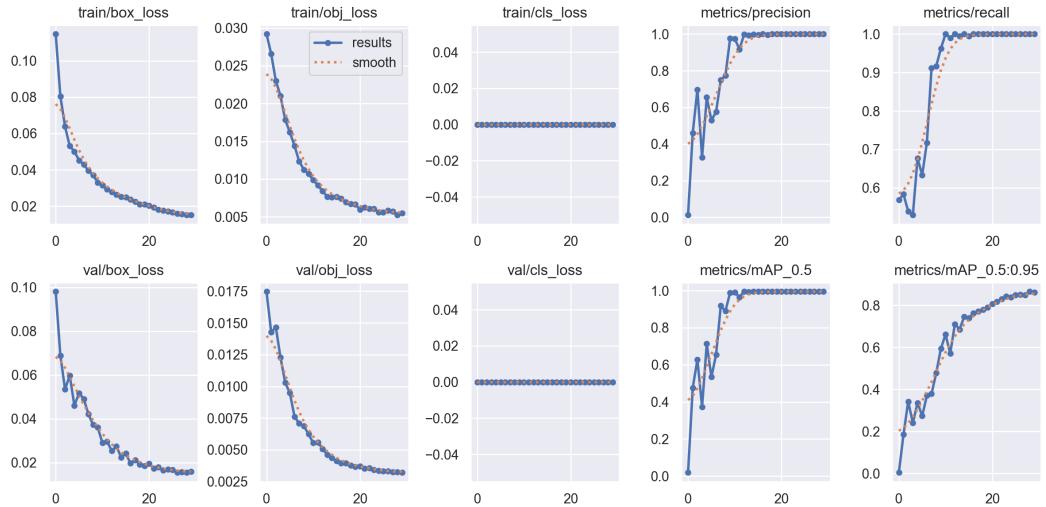


Figura 1.2: Risultati della rete YOLOv5s

Il recall, ovvero la capacità del modello di rilevare correttamente tutte le occorrenze dell'oggetto, ha raggiunto il 100%, suggerendo che il modello è riuscito a identificare correttamente tutte le barche nel dataset di validazione. Il Mean Average Precision (mAP), un indicatore chiave della performance complessiva del modello su diverse soglie di confidenza, ha registrato valori superiori al 99%, confermando l'alta accuratezza del modello nel riconoscere e localizzare l'oggetto target in diverse condizioni.

1.5 Validazione della rete

Il test finale del modello è stato condotto utilizzando due video inediti, non inclusi nel set di addestramento né nel set di validazione. Questi video simulavano riprese effettuate da un drone aereo, fornendo una sfida aggiuntiva al modello, dato che le immagini presentavano variazioni di angolazione, movimento e condizioni di illuminazione differenti rispetto alle immagini statiche utilizzate in precedenza. I risultati ottenuti sono stati molto positivi: la rete ha dimostrato una notevole capacità di mantenere una confidenza stabile del 90% lungo l'intera durata dei video. Ciò indica che il modello è in grado di rilevare in maniera affidabile la barca in contesti dinamici e realistici, rispondendo correttamente ai cambiamenti di prospettiva e movimento della camera senza significative fluttuazioni nelle sue predizioni.

1.6 Conclusioni

Dopo aver effettuato diversi test con la rete YOLOv5 per il rilevamento dell'imbarcazione, i risultati si sono dimostrati particolarmente promettenti. Il modello ha raggiunto una percentuale di precisione media stabile alta e stabile, dimostrando un'elevata capacità di identificare e localizzare correttamente l'oggetto di interesse all'interno delle immagini. Questo livello di accuratezza, mantenuto in maniera costante durante le prove, evidenzia la robustezza del modello e la sua efficacia in scenari reali di rilevamento. L'utilizzo della YOLOv5 si è quindi rivelato una scelta adeguata per applicazioni di object detection richieste dal progetto.

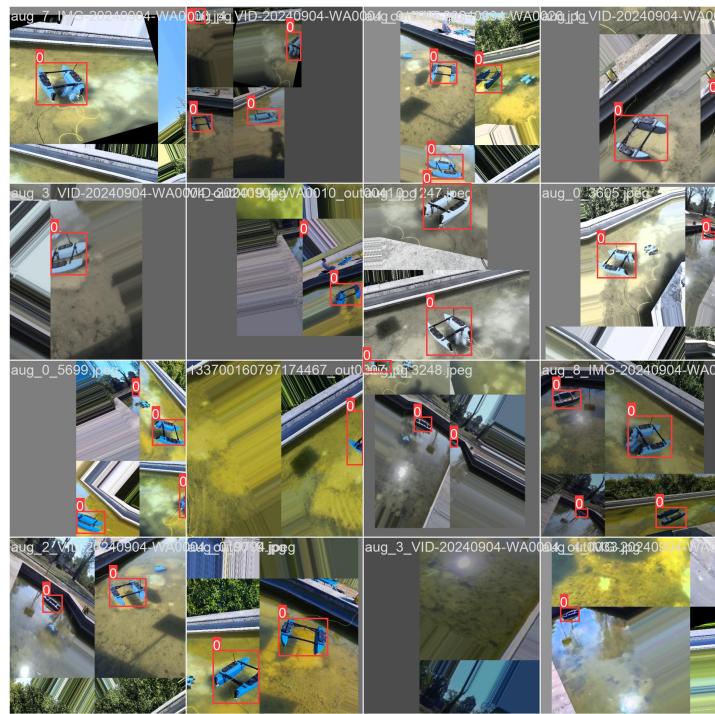


Figura 1.3: Esempio di train batch con labels

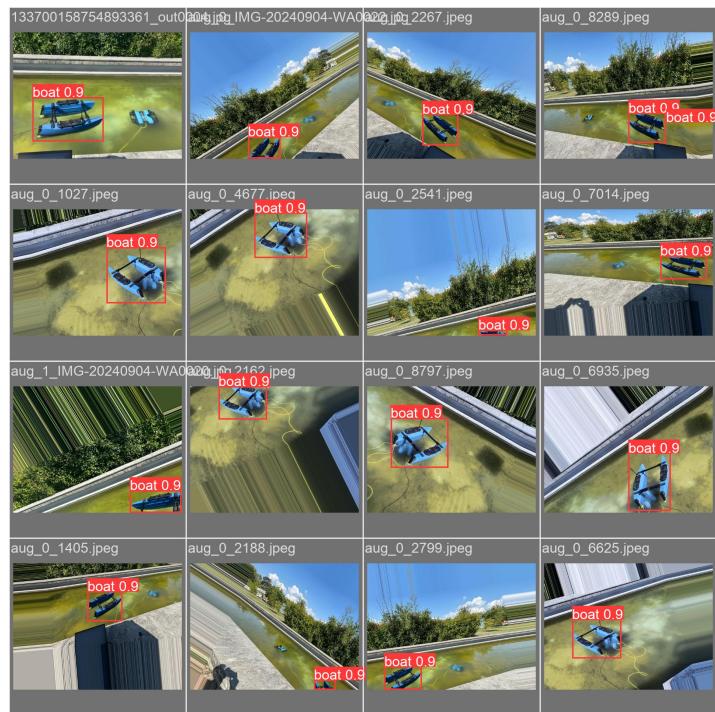


Figura 1.4: Esempio di validation batch con predizioni



Figura 1.5: Frame di un video inedito di validazione

Capitolo 2

Few-shot/One-shot learning

2.1 Obiettivo

Con questa seconda parte del progetto si intende sperimentare tecniche Few e One-Shot Object Detection per ottimizzare i tempi di training e semplificare la creazione di dataset per progetti futuri relativi al rilevamento di imbarcazioni.

2.2 Stato dell'arte

2.2.1 Few-shot learning (FSL)

Il Few-Shot Learning (FSL) [11] è una tecnica avanzata di apprendimento automatico che permette ai modelli di generalizzare a nuove classi o compiti utilizzando solo pochi esempi per classe. Questo approccio è particolarmente utile in contesti dove la disponibilità di grandi quantità di dati di addestramento è limitata, come nel riconoscimento di immagini, il rilevamento di oggetti, e la classificazione di testo.

Tecniche di FSL

Le tecniche di FSL si basano su diverse strategie:

- Approcci basati su metriche: Questi metodi apprendono uno spazio di rappresentazione in cui è possibile misurare la somiglianza tra gli esempi. Modelli come Matching Networks, Prototypical Networks, e Relation Networks sono esempi tipici di questo approccio. L'idea centrale è che i nuovi esempi siano classificati in base alla loro vicinanza agli esempi di classi note all'interno di uno spazio delle caratteristiche, facilitando la classificazione con pochi dati.
- Meta-learning (learning to learn): In questo paradigma, i modelli imparano a "imparare" in modo efficace da pochi dati. Un esempio rilevante è il Model-Agnostic Meta-Learning (MAML), che ottimizza i modelli affinché possano adattarsi rapidamente a nuovi compiti con un numero ridotto di iterazioni. MAML permette ai modelli di imparare un insieme di parametri generali, che possono essere velocemente personalizzati per nuovi compiti.
- Approcci generativi: Tecniche che sfruttano modelli generativi, come i Variational Autoencoders (VAE) o le Generative Adversarial Networks (GAN), possono essere impiegate per generare esempi sintetici. Questi esempi aggiuntivi aumentano la varietà dei dati disponibili e possono migliorare l'efficacia dell'addestramento con pochi esempi reali.

Le reti siamese

Un approccio rilevante nel contesto del Few-Shot Learning è quello basato sulle reti siamese [2]. Le reti siamese sono composte da due rami gemelli che condividono i pesi e che ricevono in input coppie di immagini. Il compito della rete è apprendere a distinguere tra esempi simili e dissimili calcolando la distanza tra le loro rappresentazioni. Questa tecnica è efficace per il riconoscimento e la verifica di

identità, poiché permette al modello di apprendere una funzione di somiglianza che può generalizzare a nuovi esempi non visti durante l'addestramento.

2.2.2 One-Shot Learning (OSL)

Il one-shot learning (OSL) [5] è una forma estrema di few-shot learning, in cui il modello deve riconoscere e generalizzare a nuove classi utilizzando un solo esempio. Questo è ancora più complesso rispetto al few-shot learning, poiché richiede un'alta capacità di generalizzazione con una quantità minima di informazioni. Gli approcci per OSL sono simili a quelli del FSL ma estremizzati, con particolare enfasi sugli approcci metrici e di meta-learning. Un modello deve essere in grado di capire le caratteristiche chiave che differenziano una classe con un solo esempio di addestramento.

2.2.3 One-Shot Object Detection (OSOD)

La One-Shot Object Detection (OSOD) è una variante del one-shot learning specificamente applicata al rilevamento di oggetti. Mentre nel one-shot learning classico l'obiettivo è classificare un oggetto con un solo esempio, il rilevamento di oggetti aggiunge la complessità di localizzare l'oggetto nell'immagine tramite bounding box. Questo rende il problema più difficile, poiché il modello deve non solo riconoscere nuovi oggetti con un singolo esempio, ma anche identificare correttamente la loro posizione.

Tecniche per OSOD

Le tecniche per la OSOD possono essere suddivise in diverse categorie, ognuna delle quali mira a superare la sfida del rilevamento con pochissimi esempi:

- Meta-learning per OSOD: Il meta-learning, già utilizzato nel few-shot learning, è stato adattato al rilevamento di oggetti attraverso architetture come Meta YOLO, un'estensione del modello YOLO (You Only Look Once). Questi modelli apprendono a rilevare oggetti in un framework one-shot tramite task-specific learning. Vengono addestrati su una serie di compiti di rilevamento utilizzando poche immagini per ogni task, e successivamente sono in grado di generalizzare a nuovi task utilizzando un solo esempio per classe.
- Prototypical Networks per la rilevazione di oggetti: Le Prototypical Networks, conosciute per il few-shot learning, sono state adattate per la OSOD. In questo approccio, il modello apprende un prototipo per ogni classe basato su un singolo esempio e lo utilizza per rilevare oggetti simili nelle immagini future. La rappresentazione del prototipo agisce come un centroide nello spazio delle caratteristiche, facilitando la localizzazione degli oggetti con pochi dati.
- Modelli basati su reti convoluzionali (CNN) con attenzione: Per migliorare la capacità del modello di rilevare oggetti a partire da un solo esempio, vengono spesso integrati meccanismi di attenzione nelle architetture CNN. Questi meccanismi permettono al modello di focalizzarsi sulle caratteristiche più rilevanti dell'immagine di riferimento, migliorando la capacità di rilevamento su nuovi oggetti mai visti prima.
- Framework R-CNN adattati: Modelli ampiamente utilizzati come R-CNN e Faster R-CNN sono stati modificati per funzionare in contesti one-shot. Questi approcci mirano a migliorare il rilevamento di nuovi oggetti con un singolo esempio affinando le reti basate su region proposals, utilizzando metriche di somiglianza apprese in precedenza. In questo modo, il modello è in grado di localizzare gli oggetti più rapidamente ed efficacemente, anche in presenza di pochi esempi di addestramento.

Stato attuale della OSOD

Nonostante i progressi, le tecniche di One-Shot Object Detection rappresentano ancora una nicchia rispetto ad altri ambiti del one-shot learning, come la classificazione. Il rilevamento di oggetti richiede una maggiore complessità computazionale e la necessità di apprendere simultaneamente sia la classe che la posizione dell'oggetto. Per questo motivo, la ricerca e lo sviluppo in OSOD sono ancora in evoluzione, con sfide significative da superare per raggiungere le prestazioni ottenute in altre applicazioni del one-shot learning. Tuttavia, grazie agli approcci emergenti come il meta-learning e le reti basate su attenzione, l'OSOD sta guadagnando terreno e promette miglioramenti futuri [6].

2.3 Tecnica utilizzata

Per questa analisi, è stato scelto di utilizzare un modello pre-addestrato sul vasto dataset COCO (Common Objects in Context) [7], noto per contenere milioni di immagini annotate con diverse categorie di oggetti. Questo dataset è ampiamente utilizzato nel campo del deep learning per il rilevamento di oggetti, la segmentazione e la classificazione, in quanto offre una solida base per trasferire la conoscenza a task specifici [12]. Il modello pre-addestrato su COCO è stato selezionato per la sua capacità di apprendere caratteristiche generalizzabili da un'ampia varietà di oggetti e scene. Successivamente, è stato eseguito un fine-tuning mirato per adattare il modello al compito specifico del Few-Shot e One-Shot Object Detection (OSOD) della barca in questione. Questo processo ha permesso al modello di specializzarsi su un task ristretto, utilizzando un set limitato di dati specifici per la barca. Il fine-tuning è stato essenziale per consentire al modello di apprendere non solo le caratteristiche visive della barca, ma anche la sua localizzazione nell'immagine tramite bounding box. Il fine-tuning consiste nell'aggiornare i pesi del modello pre-addestrato, rendendolo capace di adattarsi a un nuovo dominio con pochi esempi. In particolare, grazie all'uso di tecniche di one-shot learning, il modello ha imparato a riconoscere e rilevare la barca con un numero molto ridotto di immagini di addestramento. Questo approccio ha sfruttato la ricchezza di caratteristiche apprese dal dataset COCO, che, tramite l'addestramento aggiuntivo, sono state riconfigurate per adattarsi a un dominio più ristretto.

2.4 Libreria Detectron2

Detectron2 [13] è una piattaforma di ricerca modulare e flessibile per l'apprendimento automatico sviluppata da Facebook AI Research (FAIR), specificamente progettata per compiti di visione artificiale come il rilevamento di oggetti, la segmentazione semantica e instance segmentation. Basato su PyTorch, Detectron2 offre una serie di implementazioni all'avanguardia di modelli come Faster R-CNN, Mask R-CNN, e varianti basate su RetinaNet. La piattaforma è ottimizzata per essere altamente scalabile e facilmente adattabile a diversi dataset e configurazioni, rendendola ideale per esperimenti con tecniche come il few-shot learning e il fine-tuning di modelli pre-addestrati su dataset limitati. In particolare, Detectron2 supporta nativamente il formato COCO per le annotazioni e offre un'infrastruttura solida per personalizzare i modelli in base a esigenze specifiche, ad esempio nel contesto della rilevazione di oggetti con pochi esempi, come richiesto dal progetto.

2.5 Ambiente di sviluppo

Uno degli aspetti negativi di Detectron2 è la quantità significativa di dipendenze che devono essere installate per farlo funzionare correttamente. Questo può rendere l'installazione particolarmente complessa, soprattutto su sistemi operativi come Windows, dove il supporto nativo per Detectron2 è limitato. In particolare, il sistema operativo utilizzato per questo progetto è Windows 11, ma per facilitare la gestione delle dipendenze e sfruttare meglio le capacità del framework, è preferibile utilizzare un ambiente basato su Linux o macOS. Per superare queste difficoltà e garantire un ambiente di sviluppo isolato e gestibile, si è scelto di utilizzare un contenitore Docker [8] con sistema operativo Ubuntu. Docker consente di creare un ambiente virtualizzato, separato dal sistema operativo principale, garantendo al contempo la possibilità di replicare facilmente l'ambiente su diverse macchine. L'installazione del contenitore prevede la configurazione di un ambiente SSH per l'accesso remoto, l'installazione di strumenti di gestione come Miniconda [1] per creare e gestire ambienti virtuali Python [3], e l'installazione di PyTorch [10] con il supporto di CUDA [9] per sfruttare l'accelerazione GPU. In questo modo, si è creato un ambiente controllato in cui poter installare Detectron2 e tutte le sue dipendenze senza interferire con il sistema operativo host. L'uso di conda per gestire gli ambienti virtuali consente di mantenere le diverse versioni delle librerie necessarie, come PyTorch, compatibili con la versione corretta di CUDA. Inoltre, sono stati installati strumenti di sviluppo essenziali come GCC e Git, necessari per il build e l'installazione di Detectron2 direttamente dal codice sorgente o da repository di versioni stabili, riducendo così i problemi di compatibilità e ottimizzando il processo di sviluppo. Si lascia la lista dei comandi da seguire per la creazione dell'ambiente nel file con nome *Detectron2Env.txt*

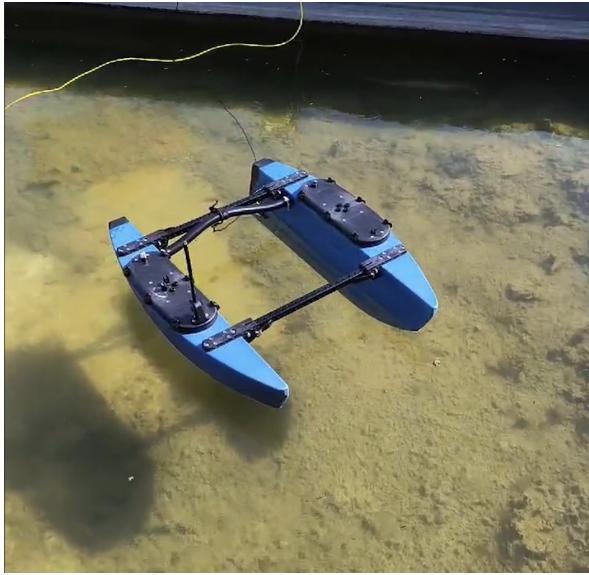


Figura 2.1: Immagine usata per il training (OSOD)

2.6 Sviluppo della rete

Ci si è concentrati sull'implementare la One-shot object detection della barca con Detectron2, è stato sviluppato uno script che utilizza un modello pre-addestrato, precisamente Faster R-CNN con backbone ResNet-50 in configurazione C4. Questo modello, già allenato sul dataset COCO, è stato scelto per la sua capacità di garantire un'accurata rilevazione anche con un numero limitato di immagini di addestramento. Tale caratteristica lo rende particolarmente adatto al fine-tuning su un dataset specifico, come quello della barca, con poche immagini annotate. Lo script segue una serie di passaggi fondamentali. Innanzitutto, il modello è stato configurato per rilevare una singola classe (la barca), con una soglia di confidenza definita per migliorare la precisione delle predizioni. Parametri come il batch size, il learning rate e il numero massimo di iterazioni sono stati ottimizzati per adattarsi al contesto few-shot, permettendo al modello di apprendere rapidamente dalle poche immagini disponibili, senza incorrere nel rischio di sovra-addestramento. Il dataset è stato registrato in formato COCO, suddiviso in un set per l'addestramento e uno per la validazione. L'addestramento del modello è gestito attraverso il framework di Detectron2, utilizzando il modulo di valutazione COCOEvaluator per misurare le performance del modello sui dati di validazione, assicurando che i risultati siano accurati. Una volta conclusa la fase di addestramento, il modello può essere utilizzato per l'inferenza. Lo script permette di caricare i pesi del modello addestrato e di creare un predictor utilizzabile sia per l'elaborazione di immagini statiche che di video. Un aspetto rilevante dello script è la possibilità di elaborare video frame-by-frame, applicando il modello per il rilevamento degli oggetti e generando un video di output che mantiene la risoluzione e il frame rate originali, mostrando le predizioni visive in modo efficace e dettagliato.

2.7 Test della rete

È stata selezionata un'immagine rappresentativa della barca, ma non di altissima qualità, per favorire una maggiore capacità di generalizzazione del modello. Il primo addestramento è stato eseguito con 500 iterazioni e un batch size di 2. Tuttavia, i risultati non sono stati soddisfacenti in termini di detection: il modello risultava sovra-allenato, rilevando erroneamente, oltre alla barca, altre tipologie di oggetti come la stessa categoria. Successivamente, è stato avviato un processo di fine-tuning per migliorare l'apprendimento del modello sull'immagine di training. Dopo varie prove, è stato stabilito che il miglior risultato si otteneva con 300 iterazioni e un batch size di 4. Questo ha portato a una migliore precisione durante la rilevazione della barca in video inediti registrati sull'acqua, sebbene i video avessero caratteristiche simili all'immagine di training, come il colore dell'acqua di sfondo e luce simile. Per testare ulteriormente l'efficacia del modello, sono stati registrati dei video all'interno del

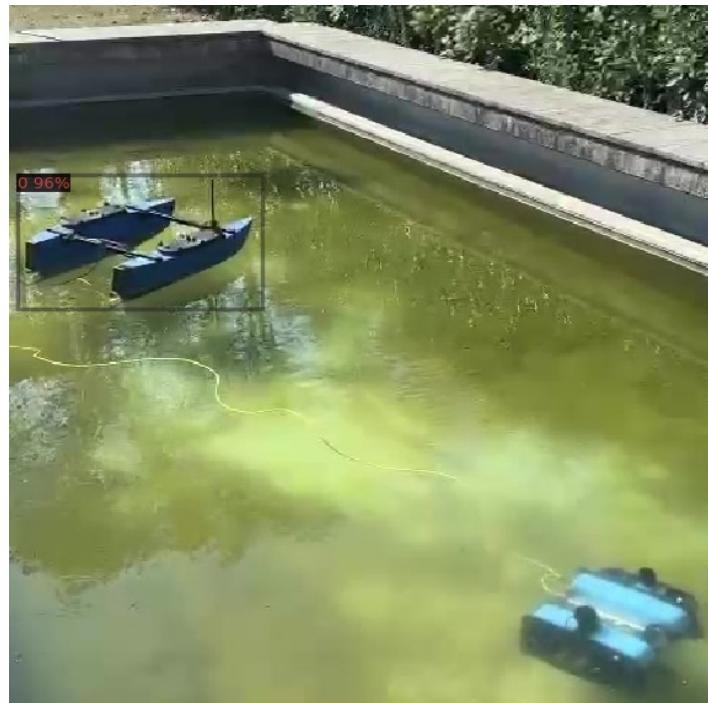


Figura 2.2: Frame di un video della barca con OSOD in esterna



Figura 2.3: Frame di un video di validazione con OSOD in interna

laboratorio. In queste condizioni, la rete ha dimostrato una buona capacità di predizione della barca, con un livello di confidenza di circa il 90% per la maggior parte del tempo in cui l'oggetto era visibile. Tuttavia, in alcuni frame, il modello identifica erroneamente altri oggetti come barche, probabilmente a causa dell'influenza del dataset COCO utilizzato per il backbone. Questo problema, tuttavia, non risulta critico nell'uso pratico, poiché in ambiente marino oggetti come scrivanie o tastiere non sono presenti, rendendo tali falsi positivi irrilevanti per lo scopo del progetto.

2.8 Conclusioni e sviluppi futuri

L'esplorazione iniziale del Few-Shot Learning applicato alla object detection ha dimostrato che questa tecnica rappresenta una strada promettente e percorribile. Il principale vantaggio di questo approccio risiede nella rapidità con cui è possibile addestrare un modello per il rilevamento di oggetti. In un confronto diretto con il modello YOLO sviluppato in precedenza, è emerso che, mentre gran parte del tempo veniva impiegato per la creazione e annotazione del dataset di allenamento, il Few-Shot Learning permette di ridurre drasticamente questa fase, richiedendo un numero molto inferiore di immagini per ottenere risultati comparabili. Questo non solo accelera il processo di sviluppo, ma libera risorse e tempo prezioso che possono essere destinati ad altre attività, come l'ottimizzazione del modello o la sperimentazione con altre tecniche. Tuttavia, nonostante i vantaggi legati alla velocità di addestramento e alla flessibilità del metodo, il modello Few-Shot attuale presenta alcune limitazioni, in particolare in termini di complessità e peso computazionale. Rispetto alla rete YOLO, questa architettura richiede una potenza di calcolo superiore per essere eseguita in modo efficiente, rendendo difficoltosa la sua implementazione su dispositivi con risorse limitate, come i droni più piccoli ed economici. Questo rappresenta un ostacolo importante, soprattutto quando si considera che tali droni sono spesso utilizzati in contesti in cui leggerezza, autonomia e capacità di elaborazione limitata sono fattori chiave. Per il futuro, uno dei principali obiettivi sarà quindi lavorare non solo sul miglioramento della precisione e delle prestazioni della rete, ma anche sulla sua ottimizzazione in termini di efficienza computazionale. La riduzione delle dimensioni del modello e la diminuzione del suo fabbisogno di risorse sono fondamentali per rendere questa soluzione applicabile su hardware più modesto, mantenendo comunque un livello accettabile di accuratezza e velocità di rilevamento. Questo permetterà di sfruttare i benefici del Few-Shot Learning anche in ambienti con limitate capacità di elaborazione, ampliando così le possibili applicazioni della tecnologia, specialmente in scenari operativi su piattaforme mobili e a basso costo.

Bibliografia

- [1] Inc. Anaconda. Conda: A cross-platform, language-independent package manager. <https://docs.conda.io/>, 2023. Accessed: 2024-10-02.
- [2] Jane Bromley, Isabelle Guyon, Yann LeCun, Alessio Saffiotti, and P. Shah. Signature verification using a "siamese" time delay neural network. In *Advances in Neural Information Processing Systems*, volume 5, pages 737–744, 1993.
- [3] Python Software Foundation. Python. <https://www.python.org/>, 2023. Accessed: 2024-10-02.
- [4] Glenn Jocher et al. YOLOv5 by ultralytics. <https://github.com/ultralytics/yolov5>, 2020. Accessed: 2024-10-02.
- [5] Gregor Koch, Richard Zemel, and Ruslan Salakhutdinov. Siamese neural networks for one-shot image recognition. In *Proceedings of the 17th International Conference on Artificial Intelligence and Statistics (AISTATS)*, pages 222–230, 2015.
- [6] Xinyu Li. Awesome few-shot object detection. <https://github.com/lxn96/awesome-few-shot-object-detection>, 2020. Accessed: 2024-10-02.
- [7] Tsung-Yi Lin, Michael Maire, Serge Belongie, Lubomir Bourdev, Ross Girshick, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C Lawrence Zitnick. Microsoft coco: Common objects in context. In *European Conference on Computer Vision (ECCV)*, pages 740–755. Springer, 2014.
- [8] Dirk Merkel. Docker: Lightweight linux containers for consistent development and deployment. <https://www.docker.com/>, 2014. Accessed: 2024-10-02.
- [9] NVIDIA Corporation. Cuda toolkit documentation. <https://developer.nvidia.com/cuda-toolkit>, 2023. Accessed: 2024-10-02.
- [10] Adam Paszke, Sam Gross, Francisco Massa, A. Lerer, James Bradbury, G. Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. Pytorch: An imperative style, high-performance deep learning library. <https://pytorch.org/>, 2019. Accessed: 2024-10-02.
- [11] Mengqi Xue Jie Song Mingli Song† Qihan Huang, Haofei Zhang. A survey of deep learning for low-shot object detection. *arXiv preprint arXiv:2112.02814*, 2022.
- [12] Xin Wang, Thomas E Huang, Trevor Wang, Trevor Darrell, and Joseph E Gonzalez. Frustratingly simple few-shot object detection. *arXiv preprint arXiv:2011.04267*, 2021.
- [13] Yuxin Wu, Alexander Kirillov, Francisco Massa, Wan-Yen Lo, and Ross Girshick. Detectron2. <https://github.com/facebookresearch/detectron2>, 2019. Accessed: 2024-10-02.