

Sistema di detection di rifiuti plastici su specchi d'acqua

De Ramundo Marco - Perani Xavier

Settembre 2024

Abstract

Obiettivo: ottenere un modello che possa individuare e classificare oggetti sulla superficie di uno specchio d'acqua come un fiume o un lago per poter riconoscere e individuare rifiuti plastici come bottiglie e buste di plastica.

Introduzione

L'inquinamento ambientale dovuto ai rifiuti plastici è un problema che influenza tutti noi, nella fattispecie per quanto riguarda la salvaguardia degli oceani. Le due principali fonti di inquinamento di plastica negli oceani deriva dalla pesca e dai rifiuti gettati nei fiumi. Per poter contrastare la seconda causa è necessario poter effettuare operazioni di monitoraggio per prevenire che rifiuti dannosi possano raggiungere il mare mentre per la prima occorrono strumenti che rendano il recupero dei rifiuti più semplice e meno dispendioso. Anche solo riconoscere, identificare e individuare i rifiuti darebbe una grossa mano allo scopo. Per questo motivo può aver senso adoperare soluzioni quali modelli convoluzionali per effettuare detection degli oggetti in questione per poi agire di conseguenza con la raccolta e la rimozione dall'acqua della plastica.

1 Dataset scelto

Il dataset scelto per il nostro scopo è kili-technology/plastic_in_river che raccoglie immagini di laghetti e bacini d'acqua con oggetti e/o rifiuti in superficie. Il dataset è stato creato per la Kili's Community Challenge, sfida nata per invogliare la community di sviluppatori a sfruttare i modelli di deep learning per

contrastare il problema dell'inquinamento degli oceani. La sfida si è svolta nel febbraio 2022 e il dataset è rimasto accessibile pubblicamente anche dopo la fine dell'evento¹.

Il dataset è composto da un totale di 4259 immagini, già divisi in tre sottoinsiemi per training, validazione e test. Nella tabella 1 è possibile vedere la ripartizione delle istanze per sottoinsieme. Ogni immagine è corrisposta da un file di testo con la lista degli oggetti presenti all'interno, classe dell'oggetto e posizione spaziale rispettando il data format usato da YOLO.

| Subset | Numero immagini |
|--------|-----------------|
| Train | 3407 |
| Val | 425 |
| Test | 427 |
| Totale | 4259 |

Tabella 1: Distribuzione delle immagini negli insiemi di train, val e test

Le categorie a disposizione per poter classificare gli oggetti identificati sono 4 e sono:

- 0 : PLASTIC_BAG
- 1 : PLASTIC_BOTTLE
- 2 : OTHER_PLASTIC_WASTE
- 3 : NOT_PLASTIC_WASTE

La distribuzione delle classi tra le istanze non è uniforme ed è uno dei possibili problemi da affrontare durante l'addestramento del modello. Il rischio è quello di avere difficoltà nel riconoscimento delle categorie che sono meno rappresentate all'interno del dataset. Nella tabella 2 è possibile vedere il numero di istanze per classe all'interno dei vari subsets.

Le dimensioni delle immagini sono varie ma generalmente di qualità alta: la dimensione in larghezza supera i 1000 pixel e gli 800 pixel in altezza. Di seguito nella tabella 3 un riassunto delle dimensioni delle immagini presenti all'interno del dataset.

¹Attualmente (luglio-agosto 2024) si riscontrano problemi nell'acquisizione del dataset tramite API. Probabilmente il proprietario del dataset ha spostato o tolto il dataset dal proprio server. Per il momento non si sa se il dataset tornerà a disposizione senza problemi. *Edit:* in data 28 Agosto 2024 il proprietario ha annunciato la rimozione del dataset. Attualmente il dataset non è più disponibile

| Classe | Train | Val | Test |
|---------------------|--------------|------------|-------------|
| PLASTIC_BAG | 1250 | 167 | 85 |
| PLASTIC_BOTTLE | 6276 | 785 | 854 |
| OTHER_PLASTIC_WASTE | 3345 | 296 | 122 |
| NOT_PLASTIC_WASTE | 1414 | 212 | 111 |

Tabella 2: Distribuzione delle istanze all'interno del dataset

| Dimensione Immagine | Training-Set | Validation-Set | Test-Set | Totale |
|----------------------------|---------------------|-----------------------|-----------------|---------------|
| 1280x720 (16:9) | 2365 | 300 | 283 | 2948 |
| 1280x640 (16:8) | 600 | 77 | 82 | 759 |
| 1920x1080 (16:9) | 401 | 45 | 53 | 499 |
| 3840x2160 (16:9) | 19 | 1 | 6 | 26 |
| 2560x1440 (16:9) | 22 | 2 | 3 | 27 |
| Totale | 3407 | 425 | 427 | 4259 |

Tabella 3: Numero di immagini per dimensione (in pixel) nei vari subset e totale

Nella immagine 1 è possibile in ordine vedere la distribuzione delle istanze nel dataset, la grandezza dei box per istanza all'interno dell'immagine, la distribuzione delle coordinate x e y del box nell'immagine, infine la grandezza del box per istanza.

Da notare come la maggior parte delle istanze hanno una dimensione della box rispetto all'immagine ridotta, in genere inferiore al 20% della larghezza e/o altezza dell'immagine. In genere la maggior parte delle istanze, essendo rifiuti sulla superficie dell'acqua, sono concentrati orizzontalmente nella parte centrale dove è plausibile trovare l'orizzonte dello specchio idrico.

La dimensione ridotta del box potrebbe essere utile da tenere in considerazione per quanto riguarda l'iperparametro `imgsz`, image size, ovvero le dimensioni delle immagini in ingresso passate al modello.

2 Modello per Object Detection

L'obiettivo del lavoro è quello di effettuare detection sui rifiuti plastici su immagini e per questo si è optato su una architettura basata su rete convoluzionale che allo stato dell'arte sia efficacie e abbia anche facilità nell'utilizzo. La scelta è ricaduta su YOLO, You Only Look Once, in quanto risulta una delle architetture

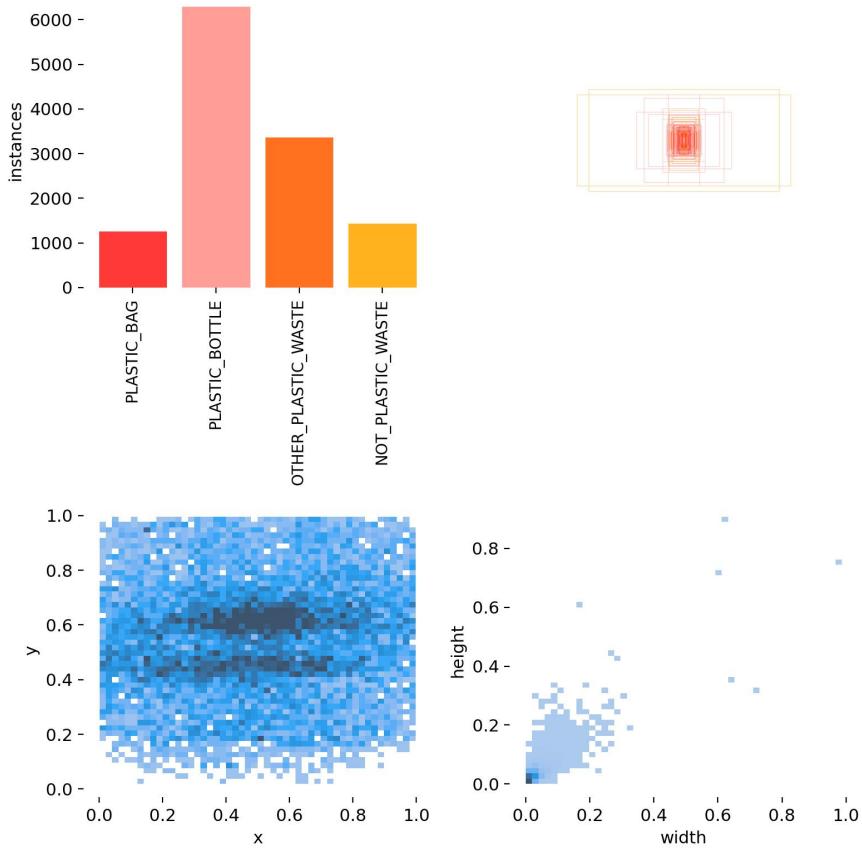


Figura 1: Caratteristiche istanze nel dataset

più utilizzate e performanti. A maggior ragione, il dataset a nostra disposizione era già predisposto per essere compatibile con YOLO.

Ci siamo basati sulla libreria YOLO di ultralytics in quanto possiede molti metodi e strumenti per lavorare al meglio con il modello. In particolare molto utile la gestione interna per fare data augmentation direttamente durante la fase di training.

All'inizio del progetto era a disposizione la versione 8 di YOLO che è stata poi la versione da noi utilizzata sebbene in questi ultimi mesi sono state presentate le versioni 9 e 10. Abbiamo preferito mantenere per continuità con il lavoro la versione 8, che di recente ha visto anche l'aggiornamento alla versione 8.2, una versione che mantiene la stessa struttura per quanto riguarda l'architettura ma ha performance migliori.

Architettura di YOLOv8

L'architettura di YOLOv8 si basa su un approccio end-to-end che suddivide l'immagine in una griglia e applica convoluzioni multiple per predire le bounding box e le classi degli oggetti all'interno di ciascuna cella della griglia. YOLOv8 utilizza tecniche avanzate come la *Path Aggregation Network* (PANet) per migliorare l'integrazione delle informazioni a diversi livelli di profondità della rete, il che è fondamentale per rilevare oggetti di varie dimensioni e scale. Nella figura 2 è riportato un diagramma semplificato dell'architettura.

Il modello YOLOv8 incorpora anche una *Neck* e una *Head* ottimizzati per massimizzare la capacità di rilevamento attraverso una migliore aggregazione delle caratteristiche e una maggiore flessibilità nelle predizioni finali. L'uso di meccanismi come le *Depthwise Separable Convolutions* contribuisce a ridurre il numero di operazioni computazionali, mantenendo un'elevata capacità espressiva del modello.

Dimensioni delle architetture usate

Nell'ambito del progetto, si è scelto di utilizzare le varianti **small**, **medium** e **nano** di YOLOv8 per il task di object detection, in quanto queste architetture offrono un buon compromesso tra capacità computazionale e performance, risultando particolarmente adatte per l'esecuzione su hardware con risorse limitate ovvero quelle a noi disponibili.

Vantaggi di YOLOv8 per il Riconoscimento dei Rifiuti Plastici

La scelta di YOLOv8 per il riconoscimento dei rifiuti plastici nei fiumi è giustificata da diversi fattori:

- **Rilevamento in Tempo Reale:** YOLOv8 è noto per la sua capacità di eseguire object detection in tempo reale, il che è cruciale per applicazioni di monitoraggio continuo nei fiumi. La possibilità di rilevare e classificare i rifiuti plastici in tempo reale consente interventi immediati, riducendo il rischio che i rifiuti possano propagarsi ulteriormente.
- **Efficienza Computazionale:** Le versioni di YOLO sono ottimizzate per girare su hardware con risorse limitate, come droni o sistemi embedded. Questo le rende particolarmente adatte per applicazioni sul campo, dove la potenza di calcolo potrebbe essere un vincolo.
- **Accuratezza Elevata:** Nonostante la sua velocità, YOLOv8 mantiene un'accuracy competitiva grazie all'uso di tecniche avanzate come l'addestra-

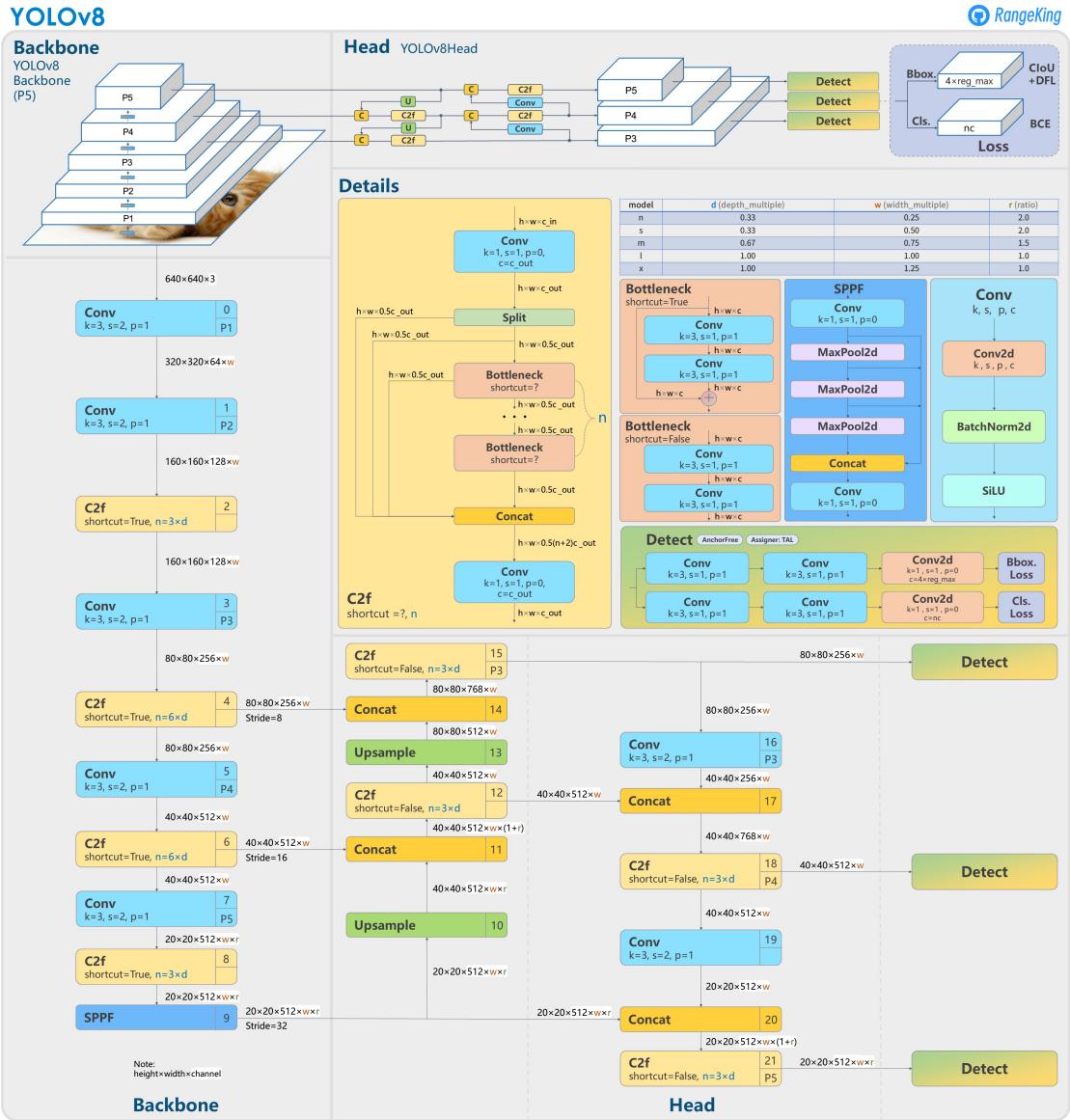


Figura 2: Schema dell'architettura di YOLOv8

mento con *label smoothing* e l'adozione di una griglia più fine per la previsione delle bounding box. Questo è essenziale per distinguere efficacemente i rifiuti plastici da altri detriti naturali presenti nei fiumi, come è possibile vedere in un esempio alla figura 3.

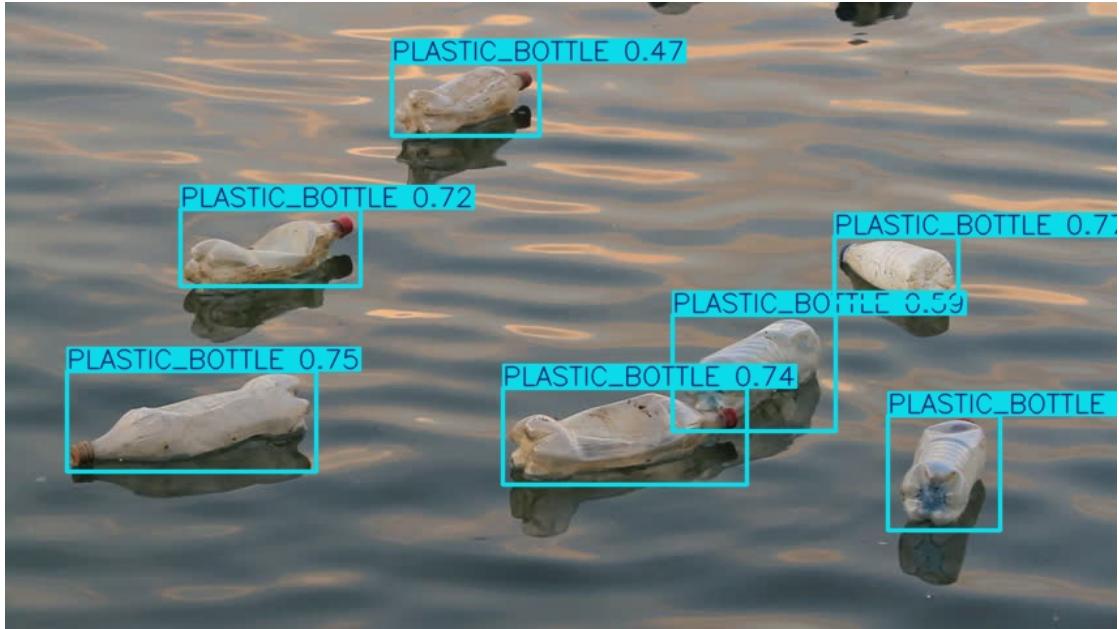


Figura 3: Esempio di output di YOLOv8 per il rilevamento di rifiuti plastici nei fiumi.

3 Metriche per valutazione performance

Nell’ambito del riconoscimento automatico dei rifiuti plastici nei fiumi utilizzando la rete convoluzionale YOLO (You Only Look Once), l’accurata valutazione delle performance del modello è cruciale. Il contesto operativo, caratterizzato da variabilità ambientali e la presenza di oggetti confondenti, rende fondamentale l’impiego di metriche di valutazione che forniscano una visione completa dell’efficacia del modello. In questa sezione, discuteremo in dettaglio le metriche principali utilizzate per valutare il modello, giustificando la loro rilevanza rispetto all’obiettivo specifico di riconoscimento dei rifiuti plastici.

Precision

La *Precision* è una metrica fondamentale nel contesto del riconoscimento dei rifiuti plastici. Questa misura indica la percentuale di rifiuti classificati correttamente tra tutti quelli identificati come rifiuti dal modello. In altre parole, la precision ci dice quanto possiamo fidarci delle predizioni positive effettuate da YOLO. La formula per calcolarla è la seguente:

$$\text{Precision} = \frac{TP}{TP + FP} \quad (1)$$

Nel contesto dei fiumi, dove possono essere presenti detriti naturali o altre forme di inquinamento, una precisione elevata è essenziale per ridurre i falsi allarmi. Un'elevata precisione significa che il modello è in grado di distinguere efficacemente i rifiuti plastici da altri oggetti, riducendo il rischio di identificare erroneamente materiali innocui come inquinanti, il che è cruciale per evitare sprechi di risorse in operazioni di pulizia.

Recall

La *Recall* misura la capacità del modello di identificare correttamente tutti i rifiuti plastici presenti. La recall è particolarmente importante in scenari dove la priorità è minimizzare il numero di rifiuti plastici non rilevati (falsi negativi), che possono avere un impatto ambientale significativo se lasciati nei corsi d'acqua. La formula per calcolare la recall è la seguente:

$$\text{Recall} = \frac{TP}{TP + FN} \quad (2)$$

Nel contesto del riconoscimento dei rifiuti plastici, un alto valore di recall significa che il modello è efficace nel rilevare la maggior parte dei rifiuti, riducendo al minimo il rischio che rifiuti plastici possano sfuggire all'attenzione. Tuttavia, un aumento della recall potrebbe portare a una diminuzione della precisione, poiché il modello potrebbe iniziare a classificare erroneamente più oggetti come plastica per evitare di perdere i rifiuti effettivi. Pertanto, è fondamentale trovare un equilibrio tra precisione e recall.

F1-Score

L'*F1-Score* è la media armonica tra precisione e recall, offrendo una misura bilanciata che tiene conto sia dei falsi positivi che dei falsi negativi. Questa metrica è particolarmente utile in scenari come il riconoscimento dei rifiuti plastici, dove il dataset può presentare classi sbilanciate e dove è importante non sovraccaricare il modello per una sola metrica a scapito dell'altra. La formula per l'*F1-Score* è:

$$F1 = \frac{2 \cdot \text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}} \quad (3)$$

Nel contesto del riconoscimento dei rifiuti plastici, l'*F1-Score* è **particolarmente indicato**, poiché consente di valutare l'efficacia complessiva del modello, bilanciando l'accuratezza con la capacità di rilevamento. Un *F1-Score* elevato indica che il modello YOLO è in grado di mantenere un buon equilibrio tra identificare correttamente i rifiuti plastici e minimizzare gli errori.

mean Average Precision (mAP)

Il *mean Average Precision* (mAP) è una delle metriche più utilizzate e riconosciute nella valutazione delle performance di modelli di *object detection*. Il mAP rappresenta la media delle *Average Precision* (AP) calcolate su tutte le classi considerate nel modello. La *Average Precision*, a sua volta, è l'area sotto la curva Precision-Recall per ciascuna classe. Il mAP è particolarmente rilevante per il nostro caso di studio, poiché:

- Riflette le prestazioni complessive del modello su più classi di rifiuti plastici (se categorizzati in diverse tipologie);
- Tiene conto della capacità del modello di rilevare i rifiuti plastici in condizioni variabili, come differenti condizioni di illuminazione, presenza di acqua torbida, e variazioni nella forma e dimensione dei rifiuti;
- Considera le variazioni nella soglia di rilevamento del modello, fornendo un'indicazione della sua robustezza e capacità di generalizzazione.

Nel caso del riconoscimento dei rifiuti plastici, un mAP elevato indica che il modello è in grado di rilevare correttamente i rifiuti in diverse situazioni operative, mantenendo un buon equilibrio tra precisione e richiamo per ciascuna classe considerata. Questo è cruciale per garantire che il sistema di monitoraggio automatico sia affidabile in ambienti reali e possa contribuire efficacemente alla riduzione dell'inquinamento nei corsi d'acqua.

4 Esperimenti effettuati

In questa sezione vengono esposti i vari esperimenti di addestramento svolti con annessi risultati e considerazioni.

Modello 1: v8m12 e v8m12bis

I primi esperimenti sono serviti per raccogliere informazioni utili per avere un fase di addestramento che fosse il più efficace possibile. Innanzitutto si è cercato di capire quanto tempo avrebbe ogni iterazione, ogni epoca e in genere l'addestramento richiesto e capire in particolare quante risorse computazionali erano richieste e quante epoche fornire al modello durante la fase di training.

Questi test, così come molti dei seguenti esperimenti, sono stati eseguiti su un notebook jupyter eseguito su server colab, sfruttando l'accelerazione GPU, in genere una scheda T4, fornita giornalmente per un massimo di 3 ore da parte di Google.

Una delle prime prove svolte riguarda due esecuzioni di training denominate v8m12 e v8m12bis. La prima è stata un'esecuzione di sole 30 epoche per vedere quanto velocemente le funzioni di loss scendessero di valore e stimare quante epoche concedere successivamente. Il modello utilizzato è quello medio preaddestrato con il dataset COCO, come disponibile dalla libreria di ultralytics (YOLOv8m.pt). La seconda esecuzione ha visto sempre un numero di epoche pari a 30 ma utilizzando come modello di partenza il miglior modello dell'addestramento precedente in quanto dai risultati come è possibile vedere nella figura 4 c'era ancora margine di miglioramento.

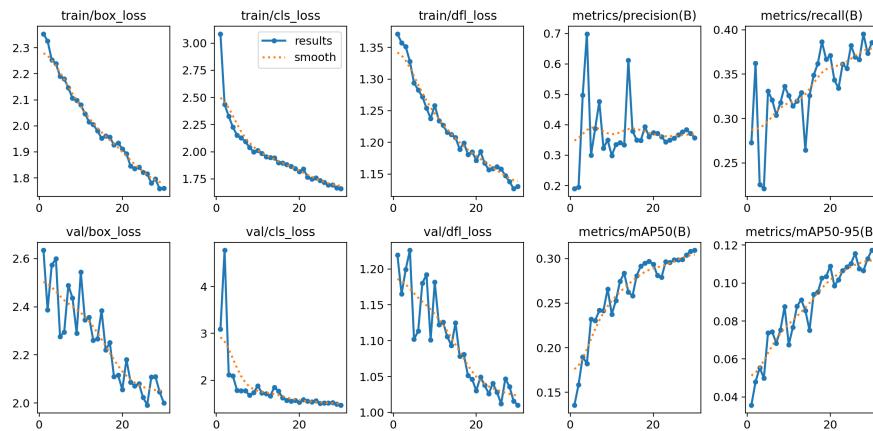


Figura 4: Andamento funzioni di loss e metriche durante l'esecuzione di v8m12

Molti degli iperparametri, se non tutti, sono stati mantenuti con il valore di default della libreria di ultralytics. L'unico iperparametro impostato per curiosità è `freeze`: in breve abbiamo tenuto 7 layer del modello di YOLO congelati in modo da ridurre il tempo necessario per l'addestramento. Per la bontà dell'esperimento si è dimostrato di poca utilità e negli addestramenti successivi tale iperparametro è stato ignorato, ovvero la modifica dei pesi avveniva su tutto il modello.

In ogni caso, a prescindere dai risultati ottenuti, che per la natura dell'esperimento non sono fondamentali, si è visto come il numero di epoche necessario per l'addestramento dovrebbe essere molto più alto per poter raggiungere quanto meno un'efficacia del modello accettabile, così come è stato fatto poi per gli esperimenti successivi.

Modello 2 - small-1202 e small-1203

Avendo sperimentato con i vari modelli, abbiamo iniziato a effettuare alcuni addestramenti aumentando il numero di epoche. Si è deciso quindi di proce-

dere con il training su tutti i parametri della rete e non congelare alcuni layer con l'opzione `freeze` ma bilanciando l'operazione optando per la dimensione del modello più piccola quale la versione small YOLOv8s. Il numero di epoch per esecuzione è stato impostato a 120 e per tale motivo i vari modelli sono stati nominati come `small-120x`.

Il primo dei due modelli, `small-1202`, ha visto un'esecuzione non ottimale visto che alcuni dei valori di loss rispetto alla validazione non sono stati salvati per alcuni problemi durante l'esecuzione su colab. Ciononostante si è visto, come in figura 5, che i valori delle funzioni di loss continuavano a decrescere mentre le metriche o rimanevano costanti, come precision e recall, o andavano a migliorare, come mAP50.

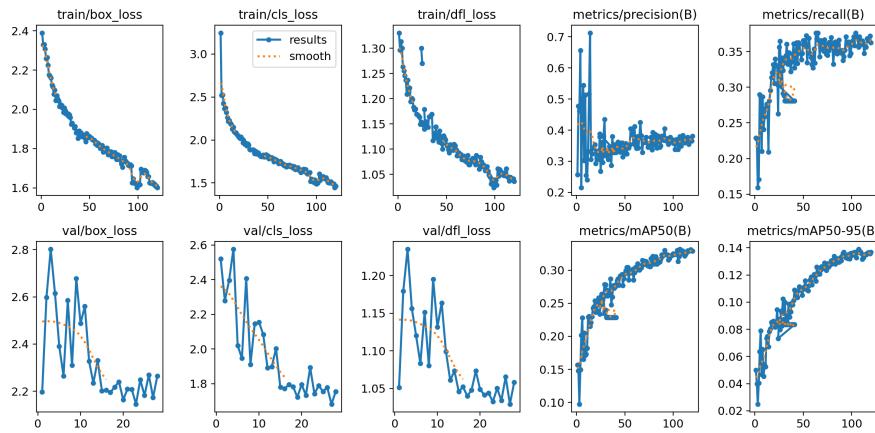


Figura 5: Andamento funzioni di loss e metriche durante l'esecuzione di `small-1202`

La fase di training è stata pertanto ripetuta con l'esecuzione `small-1203` che rispetto al nome assegnatogli ha avuto 300 epocha a disposizione per l'addestramento mantenendo gli stessi iperparametri del tentativo precedente.

Di seguito, nella tabella 4, sono riportati tutti gli iperparametri più importanti che caratterizzano questi addestramenti:

In particolare da notare l'aggiunta dell'iperparametro `dropout` rispetto alla configurazione del modello `small-1202` per poter introdurre una certa regolarizzazione all'interno della fase di apprendimento. Abbiamo poi sfruttato il parametro `patience` che indica quante epocha aspettare prima di effettuare early stopping nel caso in cui non ci fossero stati miglioramenti delle prestazioni.

Quello che si è potuto vedere dai risultati e dalle performance di questi modelli, come è possibile consultare dai grafici in figura 6 e in tabella 5, è che la rete tende ad avere discreti valori di precisione e richiamo ma non troppo soddisfacenti. In particolare se si considera una delle metriche più significative

| Iperparametro | Valore |
|---------------------|------------|
| epoch | 300 |
| optimizer | auto (SGD) |
| learning rate (lr0) | 0.01 |
| learning rate (lrf) | 0.01 |
| momentum | 0.937 |
| weight_decay | 0.0005 |
| imgsz | 640 |
| dropout | 0.15 |
| patience | 100 |
| hsv_h | 0.7 |
| degrees | 120 |
| shear | 55 |

Tabella 4: Configurazione iperparametri del modello small-1203 per il training

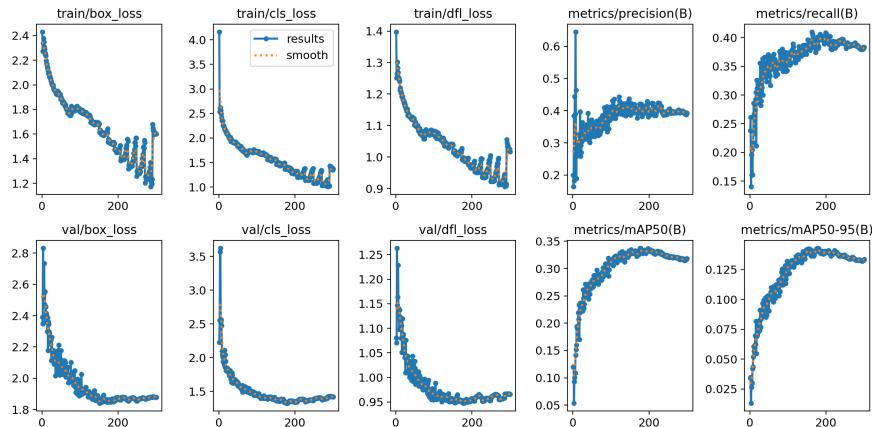


Figura 6: Andamento funzioni di loss e metriche durante l'esecuzione di small-1203

quali mAP50 vediamo che si raggiunge il valore di 37.4% nel caso migliore che potrebbe essere sì considerato un risultato accettabile ma che presenta alcuni dubbi sull'efficacia del modello.

Considerando anche gli andamenti delle funzioni di errore durante l'addestramento (figura 6) si può notare come i valori di loss con il set di validazione vengono plafonati mentre quelli inerenti al training set in genere si riducono ma avendo un comportamento oscillante. Questo può indicare un momento di stallo che può portare a un indesiderato overfitting della rete.

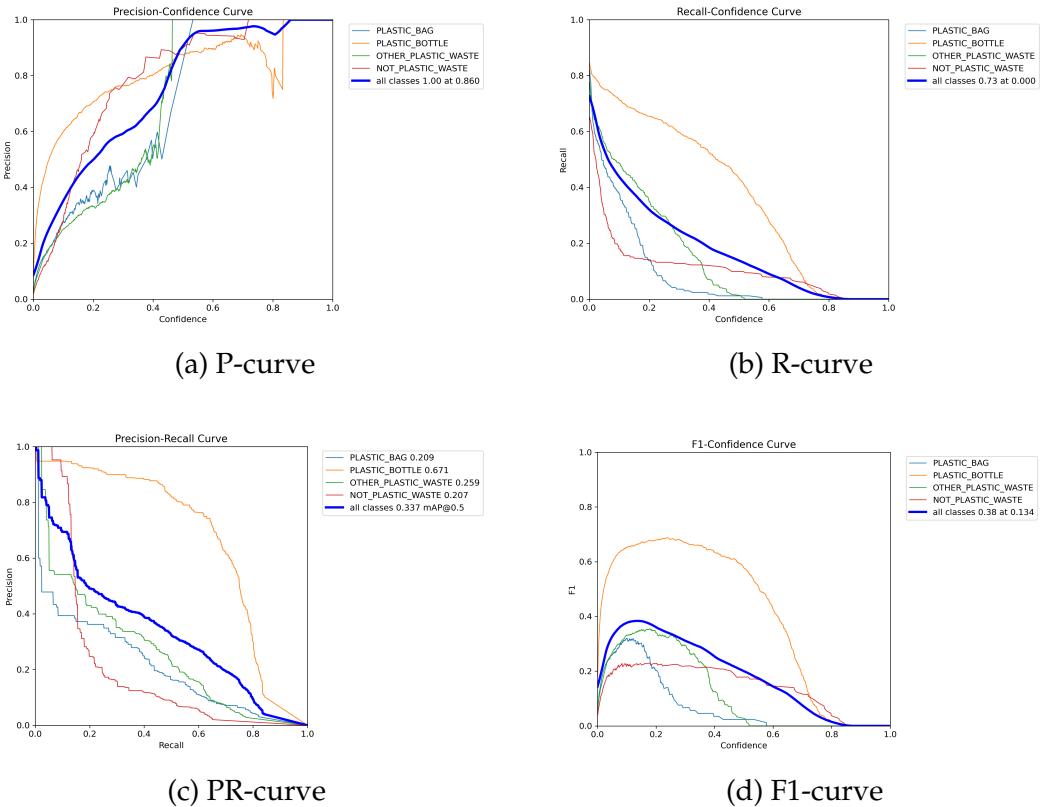


Figura 7: Da in alto a sinistra in poi, curva di confidenza della precisione, curva di confidenza del richiamo, curva precisione-richiamo e curva di confidenza F1 per il modello small-1203

È anche possibile vedere dalla tabella 5 e dalla matrice di confusione normalizzata in figura 8 come la distribuzione sbilanciata delle classi comporta si ottimi risultati per la classe PLASTIC_BOTTLE ma risultati discreti nelle altre se non addirittura pessime considerando la classe OTHER_PLASTIC_WASTE. Questo fenomeno si è visto in tutti gli esperimenti e non è stato possibile rimuoverlo ma solo arginarlo in parte.

Anche per quanto riguarda l'analisi delle curve di confidenza delle varie metriche, figura 7, si può notare come la classe PLASTIC_BOTTLE, rappresentata dalle curve arancioni, abbia performance nettamente migliori rispetto alla media.

Se il modello fosse pensato per il solo riconoscimento delle bottiglie di plastica probabilmente si potevano raggiungere ottimi risultati ma di contro sarebbero stati poco utili poi all'atto pratico.

| Class | P | R | mAP50 | mAP50-95 |
|---------------------|-------|-------|-------|----------|
| ALL | 0.452 | 0.450 | 0.374 | 0.172 |
| PLASTIC_BAG | 0.402 | 0.388 | 0.344 | 0.120 |
| PLASTIC_BOTTLE | 0.684 | 0.727 | 0.724 | 0.344 |
| OTHER_PLASTIC_WASTE | 0.140 | 0.377 | 0.106 | 0.0367 |
| NOT_PLASTIC_WASTE | 0.582 | 0.306 | 0.320 | 0.187 |

Tabella 5: Risultati delle metriche sul test set per small-1203

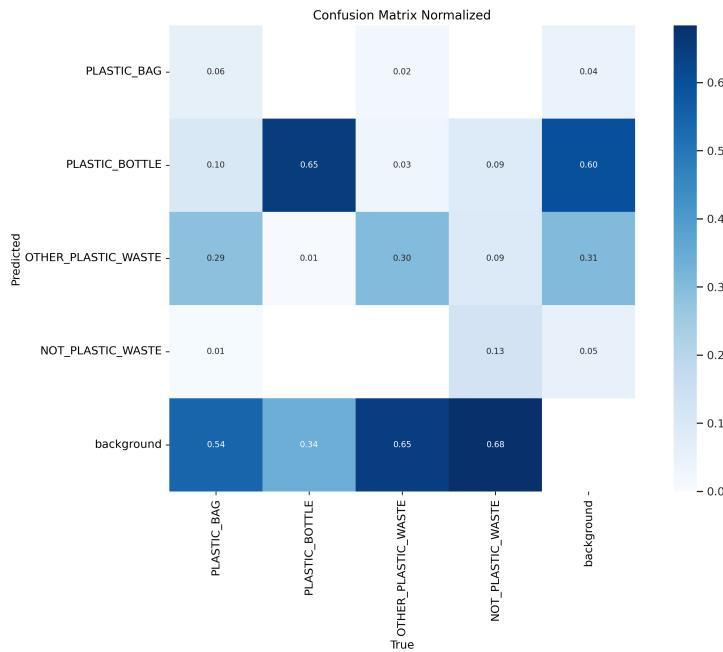


Figura 8: Matrice di confusione normalizzata data dal modello small-1203

Modello 3 - medium-200-0

L'idea per l'esperimento successivo è stata quella di utilizzare un modello con più parametri e pertanto optare per una dimensione di YOLO più grande, nello specifico la dimensione media YOLOv8m. Se è vero che il numero di epoche rimane alto, dall'altra parte si spera di poter raggiungere prestazioni migliori con una rete più profonda.

Il numero di epoche è stato impostato a 200 perché vedendo il precedente tentativo con il modello small ci aspettavamo quel tanto di epoche prima che il modello iniziasse a overfittare i dati.

Nella tabella 6 sono riportati gli iperparametri di questo esperimento. Ri-

spetto al modello `small-1203` non ci sono particolari differenze se non il numero di epoche e la dimensione del modello di YOLO.

| Iperparametro | Valore |
|---------------------|------------|
| epoch | 200 |
| optimizer | auto (SGD) |
| learning rate (lr0) | 0.01 |
| learning rate (lrf) | 0.01 |
| momentum | 0.937 |
| weight_decay | 0.0005 |
| imgsz | 640 |
| dropout | 0.15 |
| patience | 100 |
| hsv_h | 0.7 |
| degrees | 120 |
| shear | 55 |

Tabella 6: Configurazione iperparametri del modello `medium-200-0` per il training

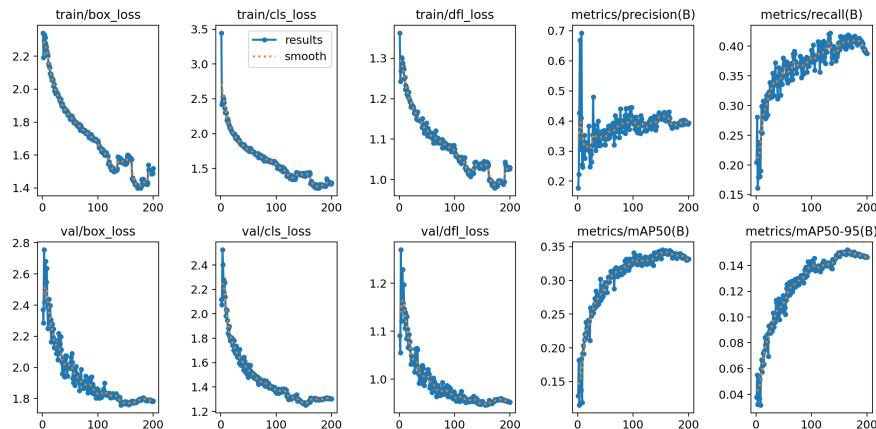


Figura 9: Andamento funzioni di loss e metriche durante l'esecuzione di `medium-200-0`

La fase di addestramento è stata eseguita su server Google colab sfruttando le GPU T4 a disposizione, esattamente come è stato per gli esperimenti precedenti.

I risultati ottenuti sono simili e leggermente migliori rispetto a quelli ricavati dai modelli più piccoli. Si è raggiunto un valore discreto di mAP pari a 39.1%

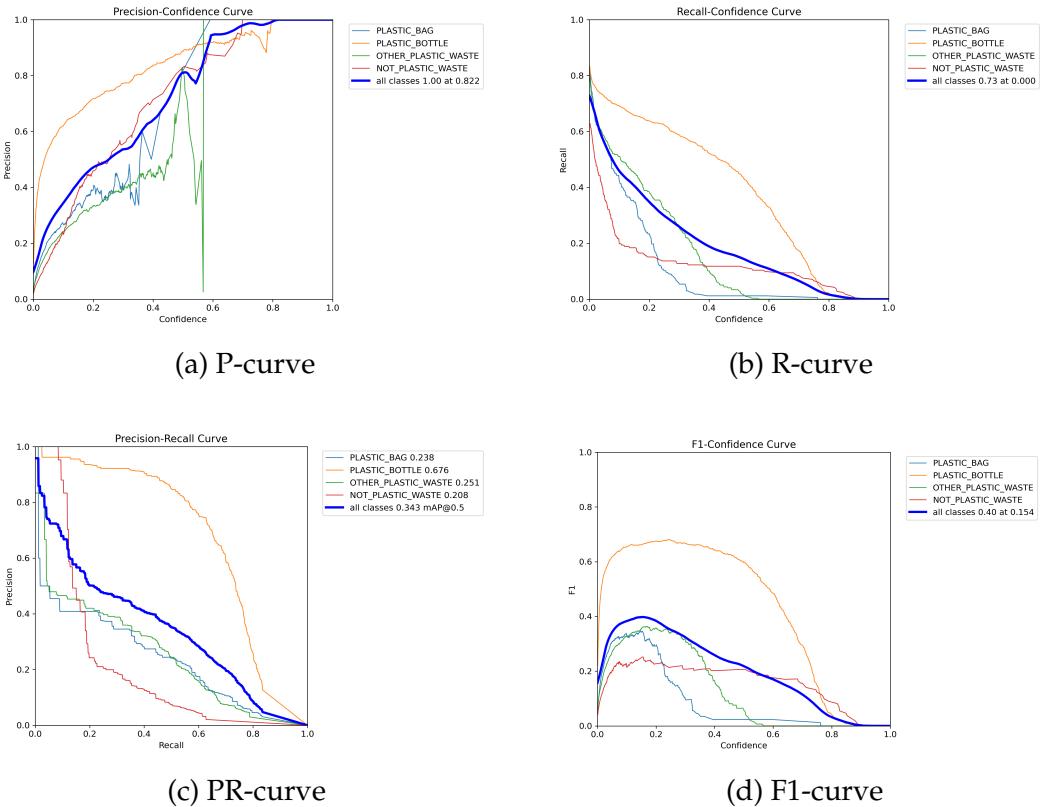


Figura 10: Andamento funzioni di loss e metriche durante l'esecuzione di medium-200-0

mentre tutte le problematiche dovute alla distribuzione sbilanciata delle istanze rimangono.

In genere il modello tende a riconoscere per lo più bottiglie di plastica e a confondere tra loro in alcuni casi le altre classi. Tali problematiche hanno portato a cercare altre soluzioni con l'obiettivo di migliorare i risultati finora raggiunti.

| Class | P | R | mAP50 | mAP50-95 |
|---------------------|-------|-------|-------|----------|
| ALL | 0.456 | 0.459 | 0.391 | 0.186 |
| PLASTIC_BAG | 0.483 | 0.482 | 0.384 | 0.142 |
| PLASTIC_BOTTLE | 0.756 | 0.712 | 0.744 | 0.374 |
| OTHER_PLASTIC_WASTE | 0.150 | 0.361 | 0.131 | 0.0483 |
| NOT_PLASTIC_WASTE | 0.435 | 0.279 | 0.305 | 0.178 |

Tabella 7: Risultati delle metriche sul test set per medium-200-0

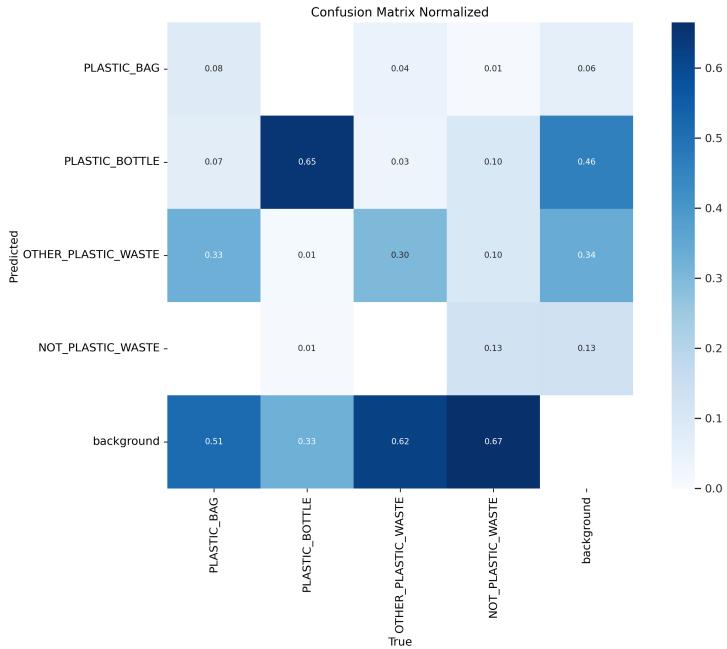


Figura 11: Matrice di confusione normalizzata data dal modello medium-200-0

Modello 4 - small-tune-03 e small-tune-04

Nella ricerca di ulteriori soluzioni per ottenere migliori risultati si è quindi deciso di guardare se altre persone avessero lavorato sullo stesso dataset e quali prestazioni hanno in tal caso raggiunto. Il motivo di ciò era duplice: in primo luogo per avere soluzioni differenti a quelle finora intraprese; in secondo luogo per ottenere almeno delle prestazioni di riferimento che era auspicabile ambire nel nostro lavoro. Inoltre abbiamo pensato che essendo il dataset rilasciato per una sfida pubblica ci sarebbe stata la chance di poter visionare altri lavori².

Peccato che le soluzioni reperibili pubblicamente non sono state così numerose se non per un articolo pubblicato per una conferenza sostenuta quest'anno a Leopoli, Ucraina, ovvero COLINS-2024: 8th International Conference on Computational Linguistics and Intelligent Systems.

²In generale non ci siamo limitati alla ricerca ristretta al nostro dataset ma anche a tutti i lavori inerenti al nostro problema di rifiuti su superfici d'acqua. Molti dei lavori sono stati molto interessanti ma si basavano su approcci totalmente differenti quali l'analisi di immagini satellitari o di droni per valutare lo stato di salute dei fiumi. Uno degli articoli trovati, *Detection of River Plastic Using UAV Sensor Data and Deep Learning* è disponibile qui

L'articolo, *Plastic Waste on Water Surfaces Detection Using Convolutional Neural Networks*³, scritto da Yurii Kryvenchuk e Andrii Marusyk, mostra i risultati ottenuti dai due ricercatori proprio sul dataset su cui stiamo lavorando.

I risultati pubblicati sembravano molto promettenti e i loro accorgimenti potevano risultare molto utili. Hanno eseguito gli addestramenti con una GPU Nvidia GeForce RTX 2070 quindi non erano avvantaggiati per quanto riguarda la potenza computazionale.

Hanno lavorato con i modelli di YOLO di dimensione nano e medium, ed effettuando una piccola fase di fine tuning spaziando con i valori del numero di epoche, learning rate, momentum e dimensione delle immagini in ingresso. Le tecniche di fine tuning non erano complesse, avendo per lo più effettuato una grid search su questi pochi iperparametri.

Le due soluzioni più interessanti colte dall'articolo sono state:

- Procedere con una fase di fine tuning magari più complessa e completa (soluzione già presa in considerazione)
- Modificare il parametro `imgsz` per permettere l'uso di immagini con una risoluzione maggiore

La seconda soluzione in particolare era intrigante perché rispondeva a una delle possibili problematiche del nostro dataset: poichè molte delle istanze nelle immagini sono di piccole dimensioni, utilizzare una risoluzione 640x640 pixel in ingresso poteva far sì che le istanze fossero poco riconoscibili e perdessero informazioni utili al modello per un corretto riconoscimento.

Questo aspetto unito al fatto che gli autori hanno raggiunto ottimi risultati con un modello nano YOLOv8n, learning rate molto basso, pari a 0.0001 e momentum a 0.98, ci hanno convinto ad aumentare anche per il nostro esperimento `imgsz` a una dimensione di 800x800 pixel. Nell'articolo, il Model 2 arrivava ad avere un valore di mAP pari al 68.6%, quasi il doppio del nostro miglior risultato ottenuto finora.

Abbiamo pertanto impostato alcuni degli iperparametri come indicato nell'articolo nella speranza di ottenere buoni risultati. Si è lavorato con il modello small per compensare il numero di pesi dovuto all'aumento di `imgsz` e i due modelli qui presentati sono nominati come `small-tune-0x` e qui di seguito nella tabella 8 è possibile visionare i valori degli iperparametri.

I due tentativi si distinguono per alcuni dettagli. Il primo modello, `small-tune-03`, ha mantenuto le 200 epoche e l'ottimizzatore automatico. Quest'ultimo parametro si è rivelato controproducente perché la funzione di ottimizzazione `auto` della libreria ultralytics permette di decidere la migliore funzione ma ignorando altri iperparametri tra cui il learning rate.

³Link dell'articolo disponibile qui

| Iperparametro | Valore st-03 | Valore st-04 |
|---------------------|--------------|--------------|
| epoch | 200 | 400 |
| optimizer | auto (SGD) | Adam |
| learning rate (lr0) | 0.0001 | 0.0001 |
| learning rate (lrf) | 0.01 | 0.01 |
| momentum | 0.98 | 0.98 |
| weight_decay | 0.0005 | 0.0005 |
| imgsz | 800 | 800 |
| dropout | 0.015 | 0.015 |
| patience | 100 | 50 |
| hsv_h | 0.7 | 0.7 |
| degrees | 120 | 120 |
| shear | 55 | 55 |

Tabella 8: Configurazione iperparametri dei modelli small-tune-03 e small-tune-04 per il training

Pertanto il secondo modello, small-tune-04, è stato configurato con l'ottimizzatore Adam ed è stato aumentato il numero di epoch a disposizione, ma riducendo il numero di epoch per l'early stopping.

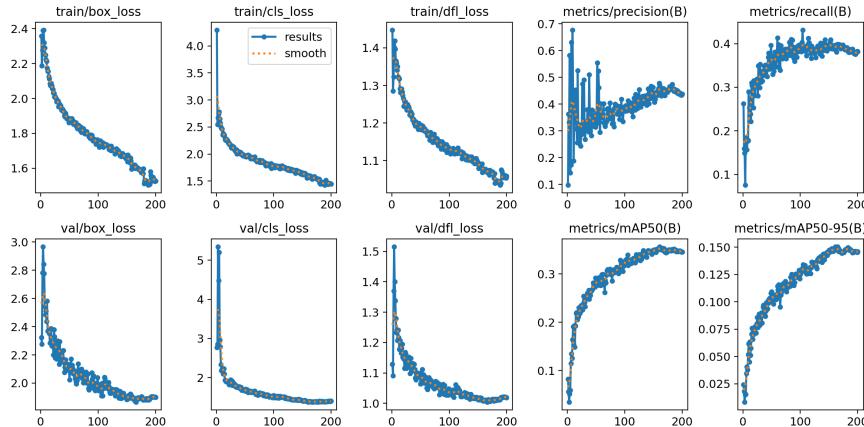


Figura 12: Andamento funzioni di loss e metriche durante l'esecuzione di small-tune-03

Il modello small-tune-03 ha visto risultati (figura 12) in linea con gli esperimenti precedenti ma nettamente distanti da quelli indicati dall'articolo. L'andamento delle funzioni di errore è sempre risultata discendente seppure verso la fine con una velocità ridotta, mentre le metriche hanno via via subito

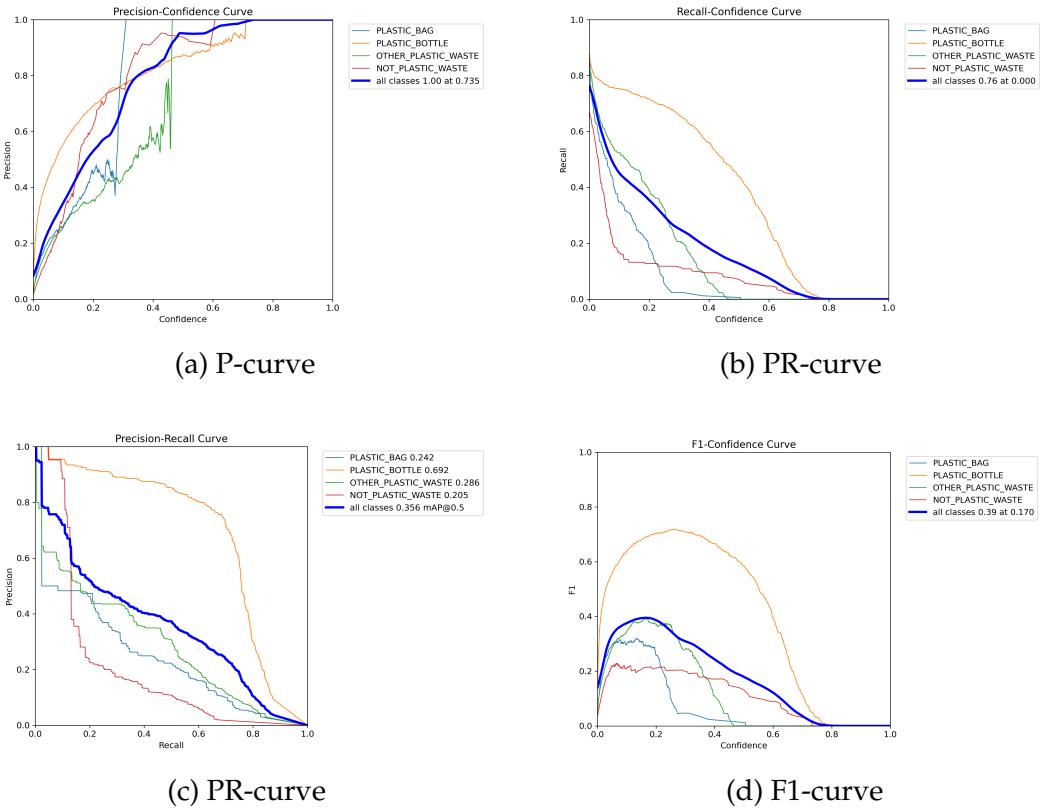


Figura 13: Andamento funzioni di loss e metriche durante l'esecuzione di small-tune-03

uno stallo. Confidando in una ricerca migliore con gli iperparametri impostati si è deciso di effettuare un nuovo training partendo da uno di quelli ottenuto dall'esperimento small-tune-03.

Il modello small-tune-04 è stato il tentativo di sistemare le imprecisioni dovute al comportamento della funzione dello strumento di ottimizzazione di YOLO della libreria. Questo tentativo si è rivelato di poca utilità in quanto il modello aveva già raggiunto il livello di apprendimento possibile e continuare l'addestramento con un learning rate molto basso, nonostante permettesse di esplorare lo spazio delle ipotesi con maggior dettaglio, non permetteva di andare molto lontano con i risultati già ottenuti.

Così come si può vedere dalla figura 14 le funzioni di errore e le metriche durante la fase di training sono state molto oscillanti e non hanno portato miglioramenti significativi. Dalla tabella 9 si può notare come le prestazioni siano pressoché le stesse e che non ci siano differenze sostanziali tra i due tentativi.

Entrambi i modelli sono stati eseguiti sui server colab con i vincoli per l'u-

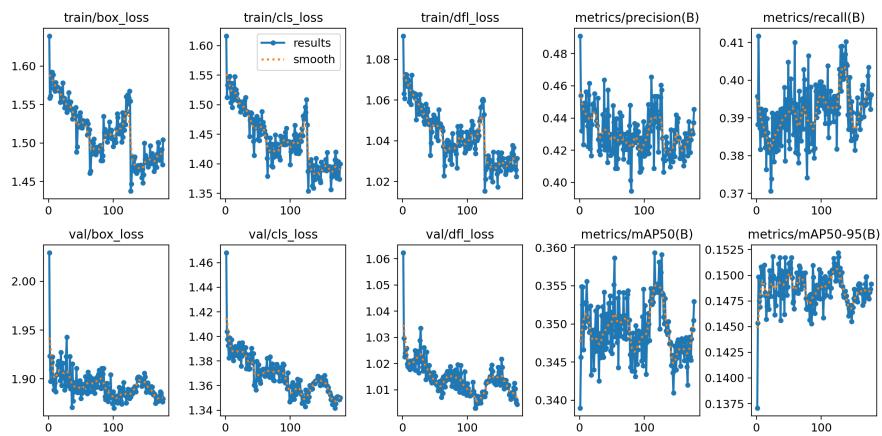


Figura 14: Andamento funzioni di loss e metriche durante l'esecuzione di small-tune-04

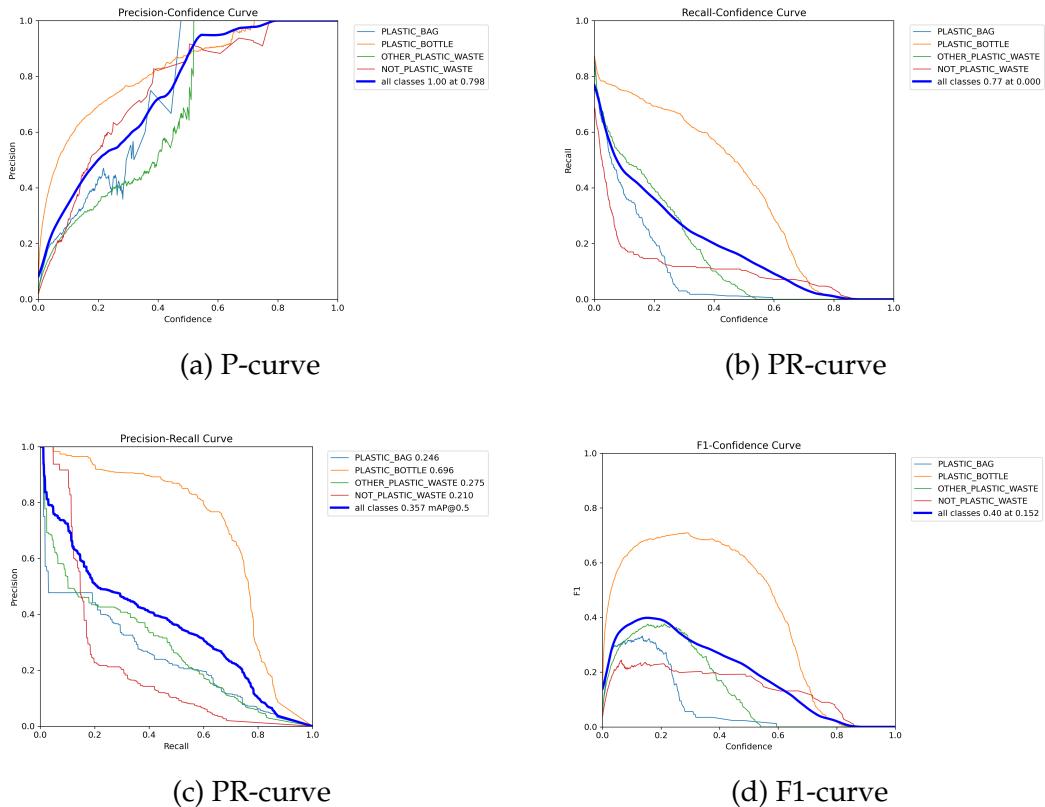


Figura 15: Andamento funzioni di loss e metriche durante l'esecuzione di small-tune-04

| Class | P | R | mAP50 | mAP50-95 |
|---------------------|-------|-------|--------|----------|
| small-tune-03 | | | | |
| ALL | 0.450 | 0.451 | 0.382 | 0.170 |
| PLASTIC_BAG | 0.436 | 0.428 | 0.394 | 0.157 |
| PLASTIC_BOTTLE | 0.672 | 0.769 | 0.731 | 0.333 |
| OTHER_PLASTIC_WASTE | 0.113 | 0.320 | 0.0934 | 0.0353 |
| NOT_PLASTIC_WASTE | 0.576 | 0.288 | 0.311 | 0.154 |
| small-tune-04 | | | | |
| ALL | 0.457 | 0.449 | 0.382 | 0.172 |
| PLASTIC_BAG | 0.405 | 0.412 | 0.352 | 0.122 |
| PLASTIC_BOTTLE | 0.701 | 0.776 | 0.753 | 0.349 |
| OTHER_PLASTIC_WASTE | 0.131 | 0.345 | 0.102 | 0.0373 |
| NOT_PLASTIC_WASTE | 0.590 | 0.261 | 0.322 | 0.181 |

Tabella 9: Risultati delle metriche sul test set per small-tune-03 sopra e small-tune-03 sotto

tilizzo delle risorse imposto da Google. I risultati dell’articolo non sono stati replicati e inizialmente la causa poteva ricadere su una gestione non accurata delle fasi di addestramento, sulla gestione approssimativa degli iperparametri e su uno scarso rigore nel replicare un esperimento svolto da altri ricercatori.

Per questo motivo risultava necessario ripetere queste esecuzioni applicando più attenzione sulle condizioni e sugli iperparametri. L’obiettivo diveniva quello di effettuare delle esecuzioni il più fedeli possibili a quelle descritte per poter replicare quei risultati che sembravano tanto promettenti.

Modelli 5 - *size-ukra-0*

Per replicare i risultati si è deciso di essere il più fedeli possibili con gli iperparametri noti e a disposizione nell’articolo dei ricercatori ucraini e di testare le 3 dimensioni di modelli di YOLO che fino a quel momento sono stati sfruttati: nano (YOLOv8n), small (YOLOv8s) e medium (YOLOv8m). Pertanto i nomi dei modelli addestrati fanno riferimento come *size-ukra-0* per evidenziare come questi modelli siano svolti prendendo in riferimento le configurazioni degli esperimenti degli autori dell’articolo.

Le fasi di addestramento sono stati svolti non più su server colab ma sui server interni all’università dove venivano messi a disposizione 2 GPU T4. Il motivo principale dell’utilizzo dei server era dovuto non tanto a questo espe-

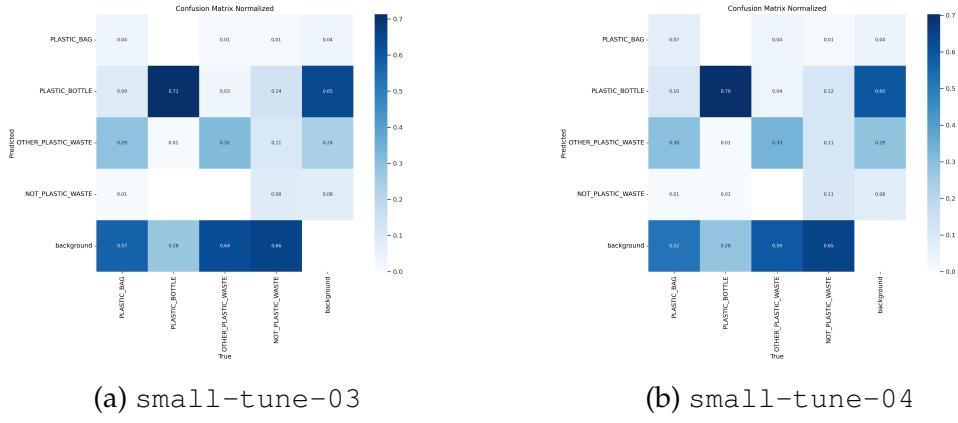


Figura 16: Matrice di confusione normalizzata data dai due modelli small-tune-03 e small-tune-04

rimento ma a quello successivo comprendente una fase di tuning che risultava particolarmente onerosa.

Gli iperparametri, tabella 10, sono stati impostati esattamente con i valori citati nell'articolo e impostati ai valori di default qualora l'iperparametro non venisse menzionato. Il numero di epoche è stato inizialmente impostato a 600 con il modello `small` e patience pari a 300. Avendo poi visionato i risultati si è deciso di ridurre il numero di epoche a 200 e la patience a 50 per le altre due dimensioni. Il modello `medium` ha poi subito un arresto verso la fine dell'addestramento. Non abbiamo pertanto i grafici dell'addestramento a disposizione ma solo le metriche ricavate dalla validazione sul test set.

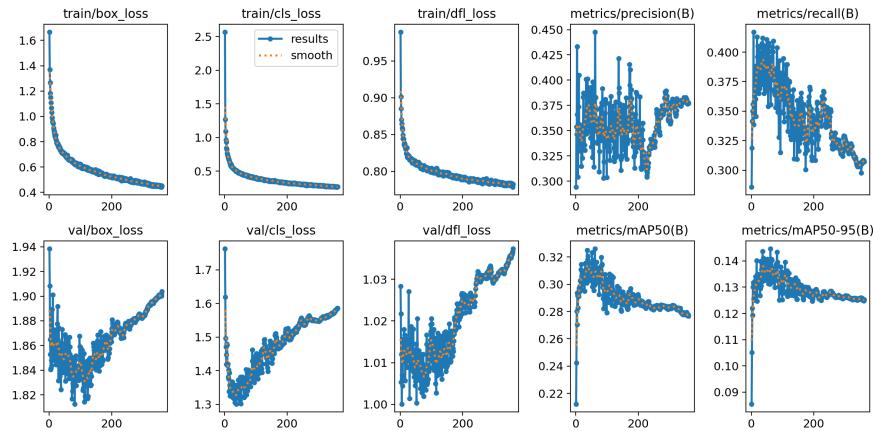


Figura 17: Andamento funzioni di loss e metriche durante l'esecuzione di small-ukra-0

| Iperparametro | nano | small | medium |
|---------------------|--------|--------|--------|
| epoch | 200 | 600 | 200 |
| optimizer | Adam | Adam | Adam |
| learning rate (lr0) | 0.0001 | 0.0001 | 0.0001 |
| learning rate (lrf) | 0.01 | 0.01 | 0.01 |
| momentum | 0.98 | 0.98 | 0.98 |
| weight_decay | 0.0005 | 0.0005 | 0.0005 |
| imgsz | 800 | 800 | 800 |
| dropout | 0.0 | 0.0 | 0.0 |
| patience | 50 | 300 | 50 |
| hsv_h | 0.4 | 0.4 | 0.4 |
| degrees | 0.0 | 0.0 | 0.0 |
| shear | 0.0 | 0.0 | 0.0 |

Tabella 10: Configurazione iperparametri dei modelli *size-ukra-0* per il training

Come si può notare dalle figure 17 e 19, l’andamento delle funzioni di errore durante la fase di training scendono velocemente fino a raggiungere valori molto al di sotto di quelli trovati finora. Ma se si guarda l’andamento delle funzioni di errore nelle fasi di validazione si può vedere come l’errore risulta più simile ai precedenti tentativi rispetto ai risultati mostrati nell’articolo. Anche le metriche raggiungono abbastanza velocemente uno stato di saturazione delle prestazioni che però sono ben al di sotto di quelle indicate.

Dalla tabella 11 possiamo vedere come il modello migliore sia quello *small* ma questo non supera il valore di mAP del 34.9%, al di sotto di alcuni risultati raggiunti nei tentativi precedenti. Questa significativa discrepanza di prestazioni tra i nostri valori e quelli indicati nell’articolo suggeriscono un qualche problema di fondo.

Da una parte è plausibile un qualche nostro errore ma strano che porti a risultati tanto disastrosi. Dall’altra parte è possibile che i risultati pubblicati dai ricercatori non siano affidabili ma in quel caso sarebbe necessario capire in che modo abbiano raggiunto le prestazioni dichiarate. Dopo vari tentativi, notando per lo più gli andamenti delle funzioni di errore del modello migliore nell’articolo, si è iniziato a dubitare che gli autori abbiano utilizzato correttamente il dataset per la validazione e la fase di test.

Abbiamo pertanto provato a effettuare test di validazione sul train set scoprendo così come i valori di mAP50 andavano così ad avere lo stesso valore se non addirittura migliore, pari a 78.3%. Quindi è plausibile che abbiano usato il train set per la fase di test.

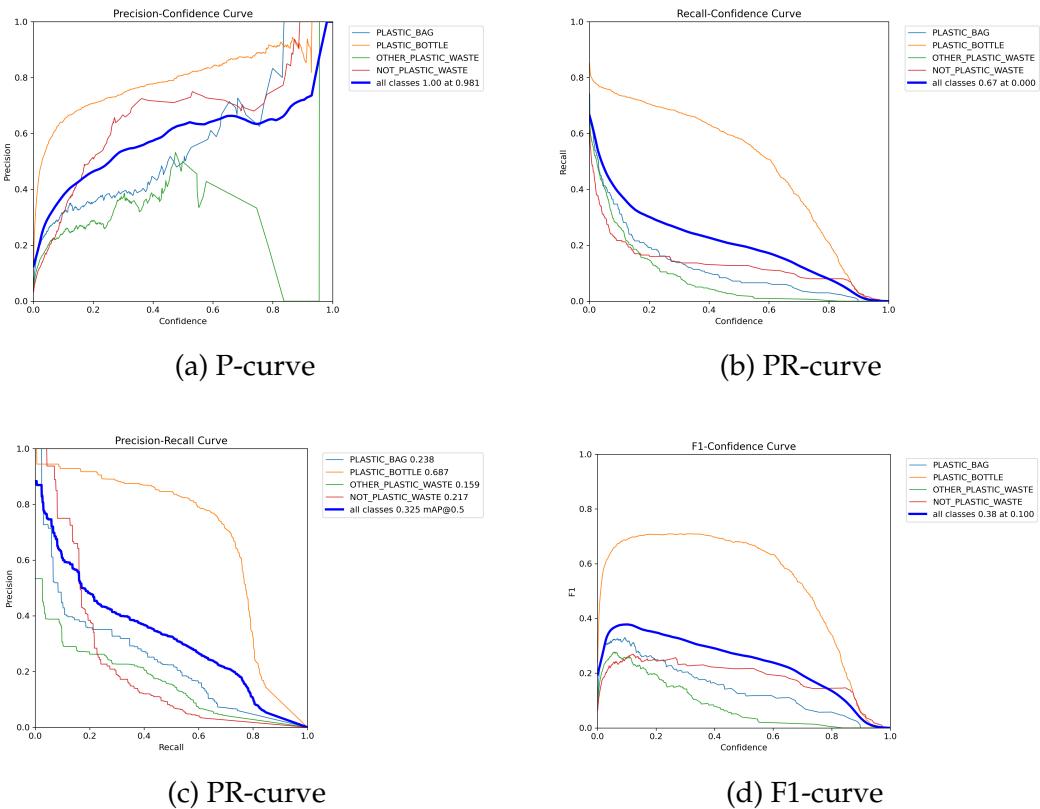


Figura 18: Andamento funzioni di loss e metriche durante l'esecuzione di small-ukra-0

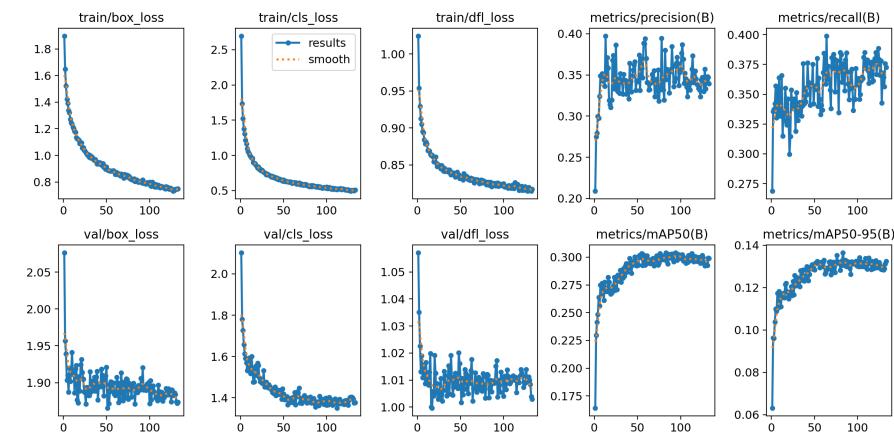


Figura 19: Andamento funzioni di loss e metriche durante l'esecuzione di nano-ukra-0

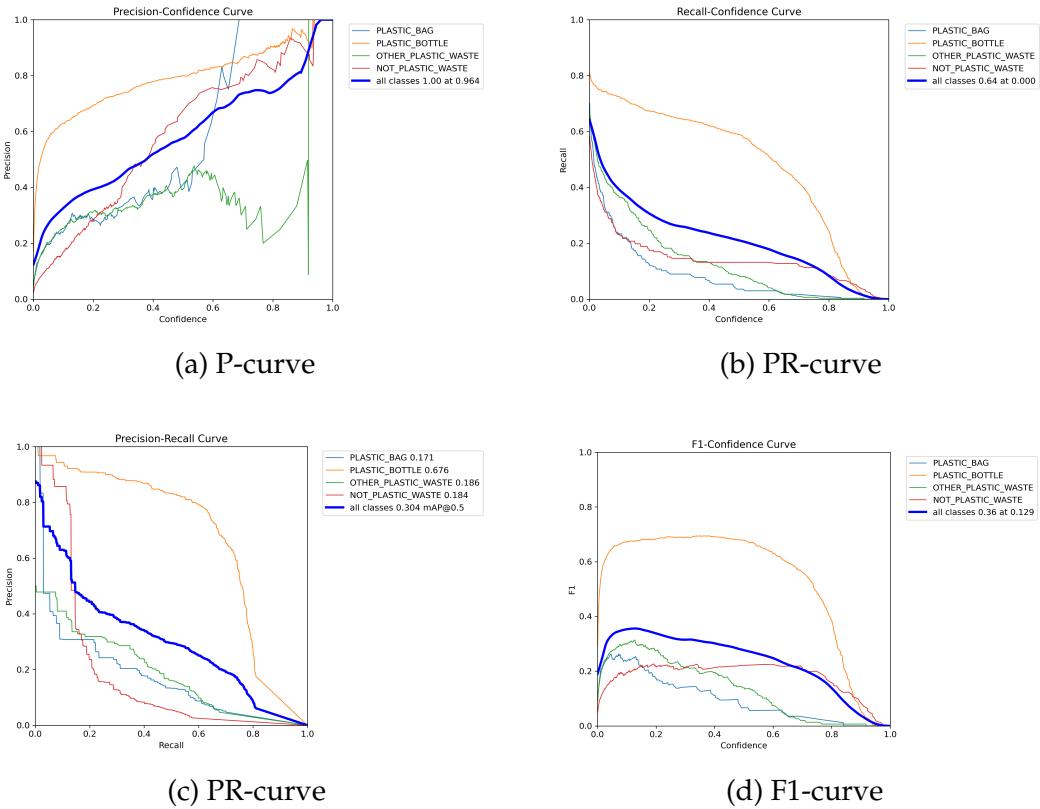


Figura 20: Andamento funzioni di loss e metriche durante l'esecuzione di nano-ukra-0

Per verificare lo stesso comportamento nella fase di validazione durante l'addestramento è stato eseguito un ulteriore tentativo su un modello nano impostando che il train set venisse usato come validation set. I risultati possono essere osservati nella figura 21.

In questo modo è stato possibile replicare i risultati e scoprire come le prestazioni millantate nell'articolo siano viziose da un errore di impostazione dei sottoinsiemi del dataset: gli autori hanno usato lo stesso set (probabilmente il train set) sia per l'addestramento, che per la validazione che per il test. Il modello quindi non poteva raggiungere un alto valore di mAP50 come indicato nell'articolo.

Modelli 6 - tuning iperparametri

L'ultimo esperimento è quello di effettuare una regolazione degli iperparametri tramite fine tuning. Abbiamo pertanto utilizzato lo strumento tuner della

| Class | P | R | mAP50 | mAP50-95 |
|---------------------|-------|-------|-------|----------|
| small-ukra-0 | | | | |
| ALL | 0.432 | 0.406 | 0.349 | 0.178 |
| PLASTIC_BAG | 0.304 | 0.329 | 0.201 | 0.0628 |
| PLASTIC_BOTTLE | 0.725 | 0.747 | 0.763 | 0.414 |
| OTHER_PLASTIC_WASTE | 0.168 | 0.197 | 0.100 | 0.0309 |
| NOT_PLASTIC_WASTE | 0.533 | 0.351 | 0.332 | 0.205 |
| nano-ukra-0 | | | | |
| ALL | 0.330 | 0.379 | 0.308 | 0.149 |
| PLASTIC_BAG | 0.212 | 0.235 | 0.155 | 0.0506 |
| PLASTIC_BOTTLE | 0.695 | 0.737 | 0.744 | 0.372 |
| OTHER_PLASTIC_WASTE | 0.143 | 0.254 | 0.104 | 0.0293 |
| NOT_PLASTIC_WASTE | 0.271 | 0.288 | 0.229 | 0.144 |
| medium-ukra-0 | | | | |
| ALL | 0.405 | 0.394 | 0.324 | 0.168 |
| PLASTIC_BAG | 0.209 | 0.235 | 0.124 | 0.0418 |
| PLASTIC_BOTTLE | 0.738 | 0.754 | 0.771 | 0.415 |
| OTHER_PLASTIC_WASTE | 0.146 | 0.246 | 0.089 | 0.0280 |
| NOT_PLASTIC_WASTE | 0.527 | 0.342 | 0.311 | 0.186 |

Tabella 11: Risultati delle metriche sul test set per *size-ukra-0*

libreria ultralytics per poter ricavare i valori migliori per gli iperparametri. Tale tuner effettua una serie di training con dei valori di iperparametri iniziali. Ogni iterazione consiste in una manciata di epoche di training necessarie per poter ottenere un modello da valutare. Alla fine di ogni iterazione si valuta tale modello in relazione ai suoi iperparametri. Con la nuova iterazione gli iperparametri vengono impostati a dei nuovi valori che vengono stabiliti dall'algoritmo del tuner. L'evoluzione degli iperparametri segue un algoritmo genetico in modo da poter indirizzare i valori verso le prestazioni migliori.

Per far sì che potessimo ottenere dei risultati consistenti anche dopo la fase di training, abbiamo effettuato la fase di tuning con il validation set in modo che successivamente i risultati non fossero influenzati da un overfitting degli iperparametri su training set. La successiva fase di addestramento ha visto l'uso normale dei sottoinsiemi del dataset per train, validazione e test.

La fase di tuning è stata quella che ha richiesto più risorse, sia in termini di tempo che di potenza di calcolo. Nella figura 22 è possibile vedere l'andamento

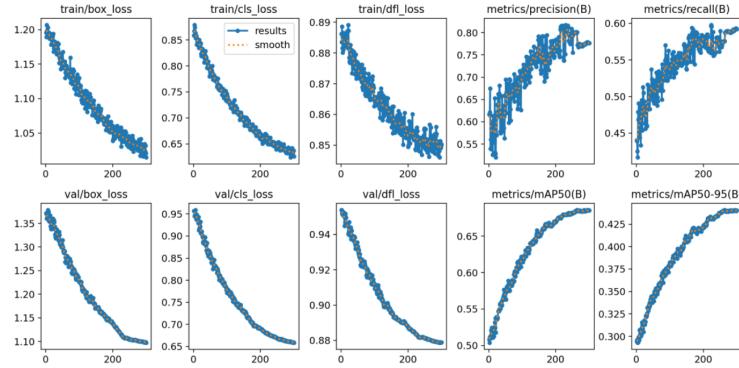


Figure 21: Overall Results of The Best Model.

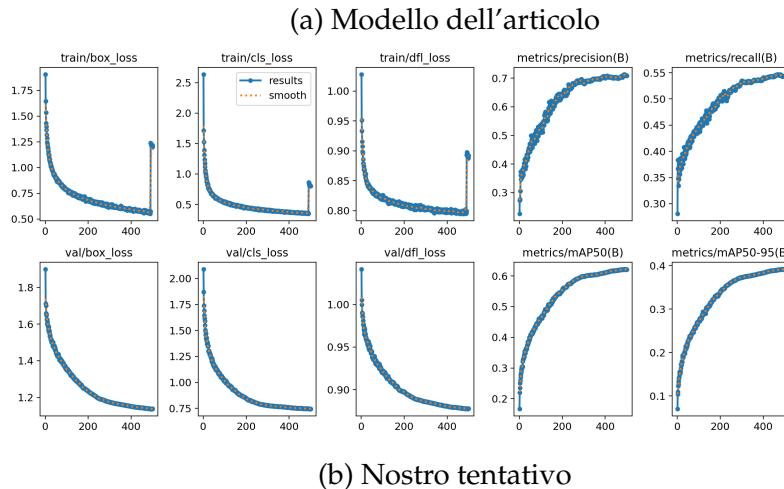


Figura 21: Comparazione tra l'andamento delle funzioni di errore durante il training del miglior modello dell'articolo e il nostro tentativo di replicare i risultati

della funzione di fitness, funzione che valutava la bontà degli iperparametri utilizzati, mentre nella figura 23 l'andamento dei valori degli iperparametri nel corso del tuning.

Questa fase ha visto l'utilizzo del modello nano di YOLO, YOLOv8n, per 300 iterazioni ciascuna da 30 epoche di addestramento. Gli iperparametri migliori, poi utilizzati nella fase di training, sono elencati nella tabella 12.

Questa configurazione è stata utilizzata per l'esecuzione di 3 addestramenti, per i modelli nano, small e medium di YOLO. Ogni training ha avuto a disposizione 500 epoche con 200 epoche di patience prima di procedere con l'early stopping. `imgsz` è rimasto impostato a 800 pixel mentre come ottimizzatore è stato utilizzato AdamW. Il nome dato ai modelli per questi training è

| Iperparametro | Valori risultati dal Tuning | Valore Predefinito | Descrizione |
|-----------------|-----------------------------|--------------------|--|
| lr0 | 0.00252 | 0.01 | Learning rate iniziale |
| lrf | 0.00814 | 0.01 | Learning rate finale |
| momentum | 0.86722 | 0.937 | Fattore di momentum |
| weight_decay | 0.00058 | 0.0005 | Termine di regolarizzazione per i pesi |
| warmup_epochs | 2.28985 | 3.0 | Numero di epoche di warmup |
| warmup_momentum | 0.78103 | 0.8 | Momentum iniziale durante il warmup |
| box | 5.6359 | 0.05 | Guadagno della loss per il box |
| cls | 1.08241 | 0.5 | Guadagno della loss per la classificazione |
| dfl | 0.86478 | 1.5 | Guadagno della Distribution Focal Loss |
| hsv_h | 0.00914 | 0.015 | Fattore di aumento della tonalità |
| hsv_s | 0.29432 | 0.7 | Fattore di aumento della saturazione |
| hsv_v | 0.16061 | 0.4 | Fattore di aumento del valore |
| translate | 0.11669 | 0.1 | Fattore di traslazione dell'immagine |
| scale | 0.52052 | 0.5 | Fattore di scala dell'immagine |
| fliplr | 0.27039 | 0.5 | Probabilità di ribaltamento orizzontale |
| mosaic | 0.9728 | 29 | 1.0 Fattore di aumento mosaic |

Tabella 12: Risultati della sintonizzazione degli iperparametri di YOLOv8

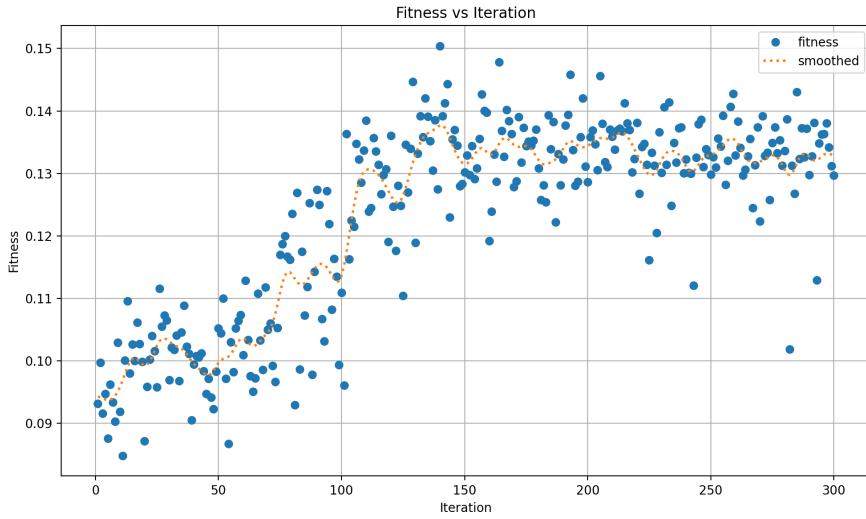


Figura 22: Andamento funzioni di fitness durante l'esecuzione del tuning degli iperparametri

`size-tune-2 . 0.`

Come è possibile vedere nelle figure 24, che mostrano l'andamento delle funzioni di errore e delle metriche durante la fase di addestramento, e nella tabella 13, che mostra le prestazioni su test set, i risultati sono in linea con quelli già ottenuti e non hanno mostrato miglioramenti significativi. Il modello che ha portato a prestazioni migliori è stato `small-tune-2 . 0` che ha raggiunto un valore di mAP50 pari al 34.5%.

5 Conclusioni

Nella tabella 14 abbiamo riassunto tutte le metriche dei modelli realizzati. Tra tutti i modelli, `medium-200-0` è risultato quello con precisione, richiamo e mAP più alti.

I modelli che seguivano il metodo indicato dall'articolo hanno avuto le prestazioni peggiori mentre il nostro tentativo con la regolazione dei iperparametri non ha portato ai risultati sperati.

In generale comunque le prestazioni tra i vari tentativi sono abbastanza simili, non ci sono grosse discrepanze se non qualche tendenza di un modello a essere migliore di un altro.

Possiamo supporre che l'iperparametro `dropout` ha aiutato a migliorare le performance e che per poter aumentare l'efficienza del modello sarebbe necessario mettere mano al dataset per contrastare meglio la distribuzione sbilanciata

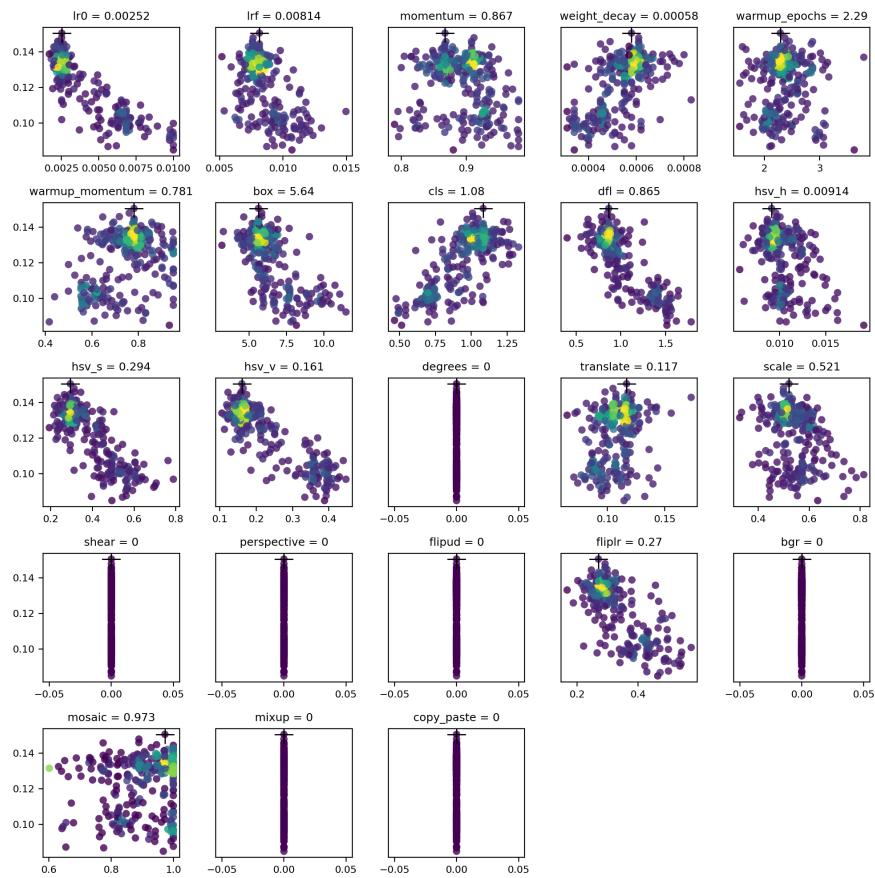
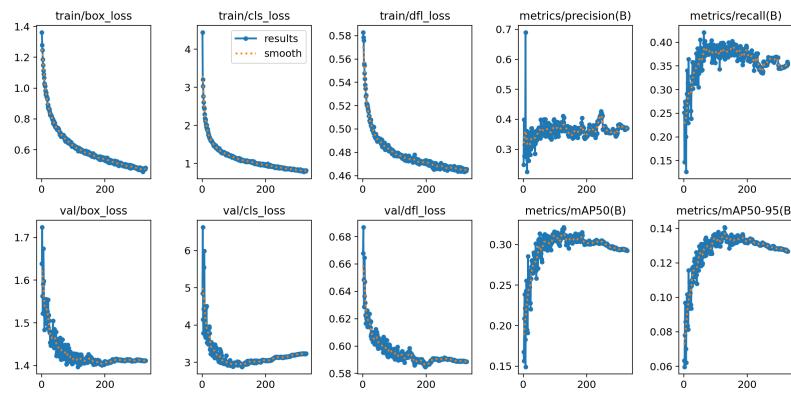


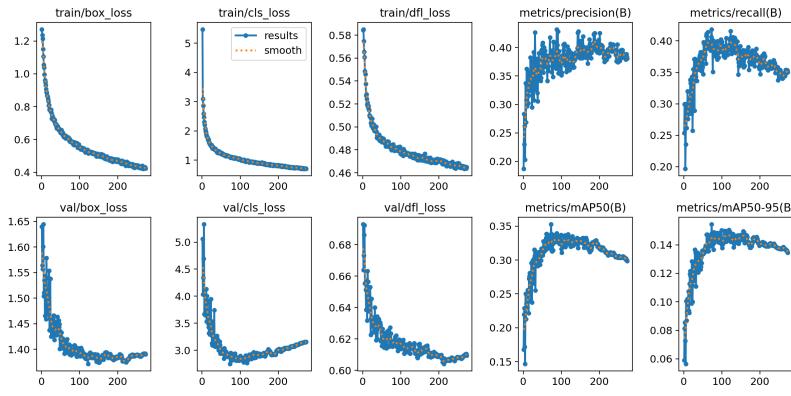
Figura 23: Andamento valori degli iperparametri durante la fase di tuning

delle classi nelle immagini. I metodi di augmentation della libreria ultralytics non sono così sufficienti a contrastare questa problematica nonostante sia stato d'aiuto.

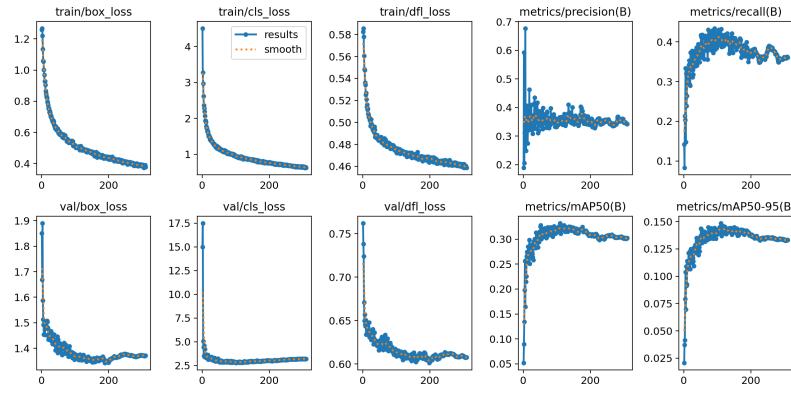
In ultimo, i modelli ottenuti possono essere utili per il riconoscimento dei rifiuti, soprattutto per quanto riguarda le bottiglie di plastica, ma devono essere perfezionati per ridurre la percentuale di errore e di mancata individuazione degli oggetti interessati.



(a) Nano



(b) Small



(c) Medium

Figura 24: Comparazione tra l'andamento delle funzioni di errore durante il training dei modelli `size-tune-2 .0`

| Class | P | R | mAP50 | mAP50-95 |
|---------------------|-------|-------|--------|----------|
| nano-tune-2.0 | | | | |
| ALL | 0.372 | 0.428 | 0.320 | 0.159 |
| PLASTIC_BAG | 0.223 | 0.329 | 0.158 | 0.0463 |
| PLASTIC_BOTTLE | 0.714 | 0.737 | 0.751 | 0.385 |
| OTHER_PLASTIC_WASTE | 0.141 | 0.385 | 0.103 | 0.0319 |
| NOT_PLASTIC_WASTE | 0.409 | 0.261 | 0.268 | 0.174 |
| small-tune-2.0 | | | | |
| ALL | 0.400 | 0.389 | 0.345 | 0.172 |
| PLASTIC_BAG | 0.273 | 0.188 | 0.205 | 0.0662 |
| PLASTIC_BOTTLE | 0.741 | 0.724 | 0.757 | 0.395 |
| OTHER_PLASTIC_WASTE | 0.152 | 0.293 | 0.0948 | 0.0297 |
| NOT_PLASTIC_WASTE | 0.433 | 0.351 | 0.321 | 0.198 |
| medium-tune-2.0 | | | | |
| ALL | 0.363 | 0.487 | 0.335 | 0.166 |
| PLASTIC_BAG | 0.254 | 0.372 | 0.194 | 0.0628 |
| PLASTIC_BOTTLE | 0.737 | 0.768 | 0.769 | 0.397 |
| OTHER_PLASTIC_WASTE | 0.111 | 0.475 | 0.0952 | 0.0303 |
| NOT_PLASTIC_WASTE | 0.350 | 0.334 | 0.280 | 0.173 |

Tabella 13: Risultati delle metriche sul test set per *size-tune-2.0*

| Model | Precision | Recall | mAP50 | mAP50-95 |
|-----------------|-----------|--------|-------|----------|
| small-1203 | 0.452 | 0.450 | 0.374 | 0.172 |
| medium-200-0 | 0.456 | 0.459 | 0.391 | 0.186 |
| small-tune-03 | 0.450 | 0.451 | 0.382 | 0.170 |
| small-tune-04 | 0.457 | 0.449 | 0.382 | 0.172 |
| nano-ukra-0 | 0.330 | 0.379 | 0.308 | 0.149 |
| small-ukra-0 | 0.432 | 0.406 | 0.349 | 0.178 |
| medium-ukra-0 | 0.405 | 0.394 | 0.324 | 0.168 |
| nano-tune-2.0 | 0.372 | 0.428 | 0.320 | 0.159 |
| small-tune-2.0 | 0.400 | 0.389 | 0.345 | 0.172 |
| medium-tune-2.0 | 0.363 | 0.487 | 0.335 | 0.166 |

Tabella 14: Confronto validazione su test set tra i modelli testati

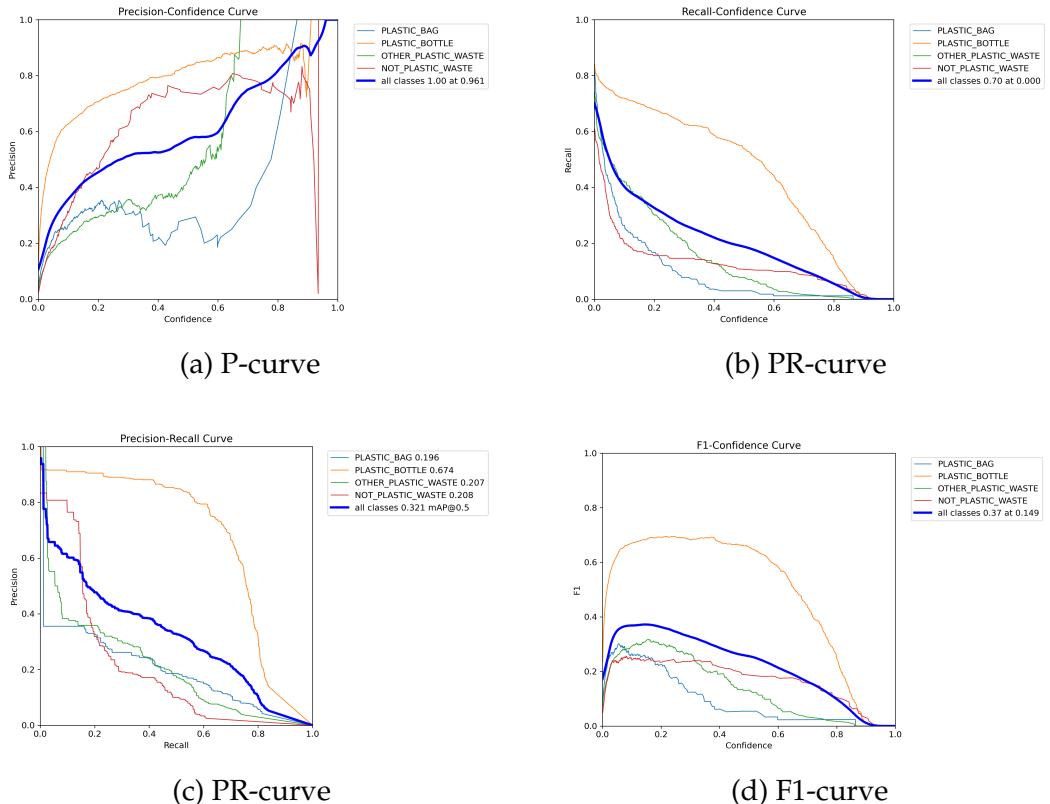


Figura 25: Andamento funzioni di loss e metriche durante l'esecuzione di nano-tune-2.0

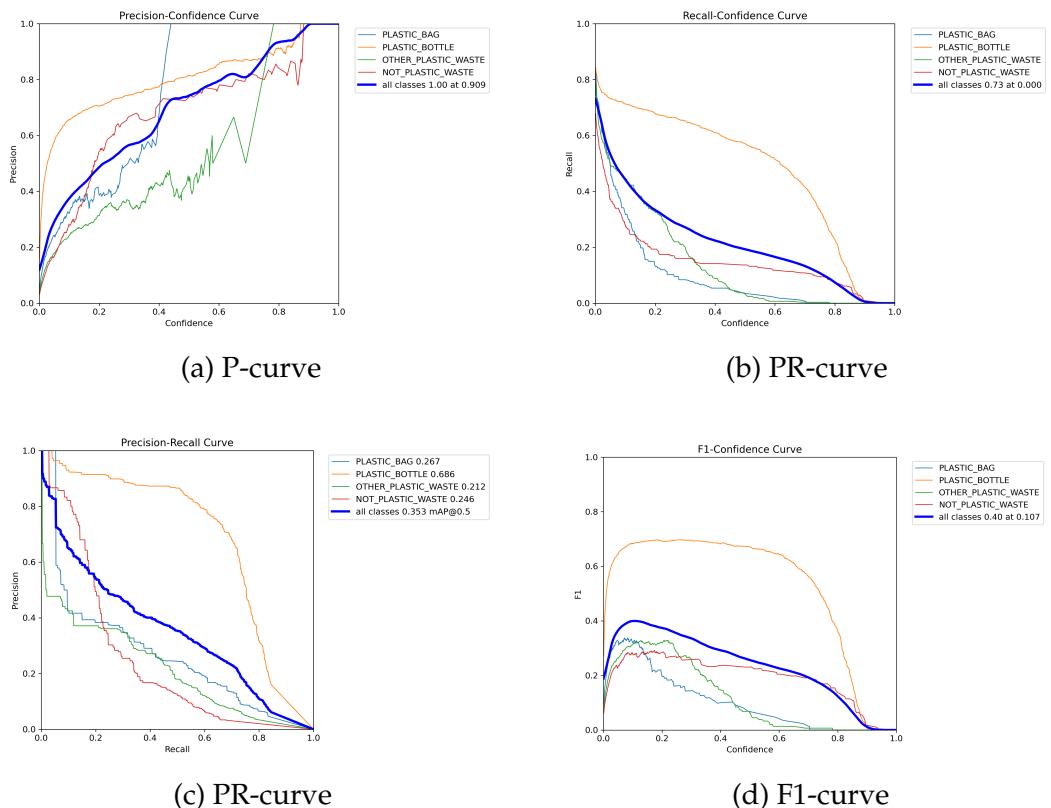


Figura 26: Andamento funzioni di loss e metriche durante l'esecuzione di small-tune-2.0

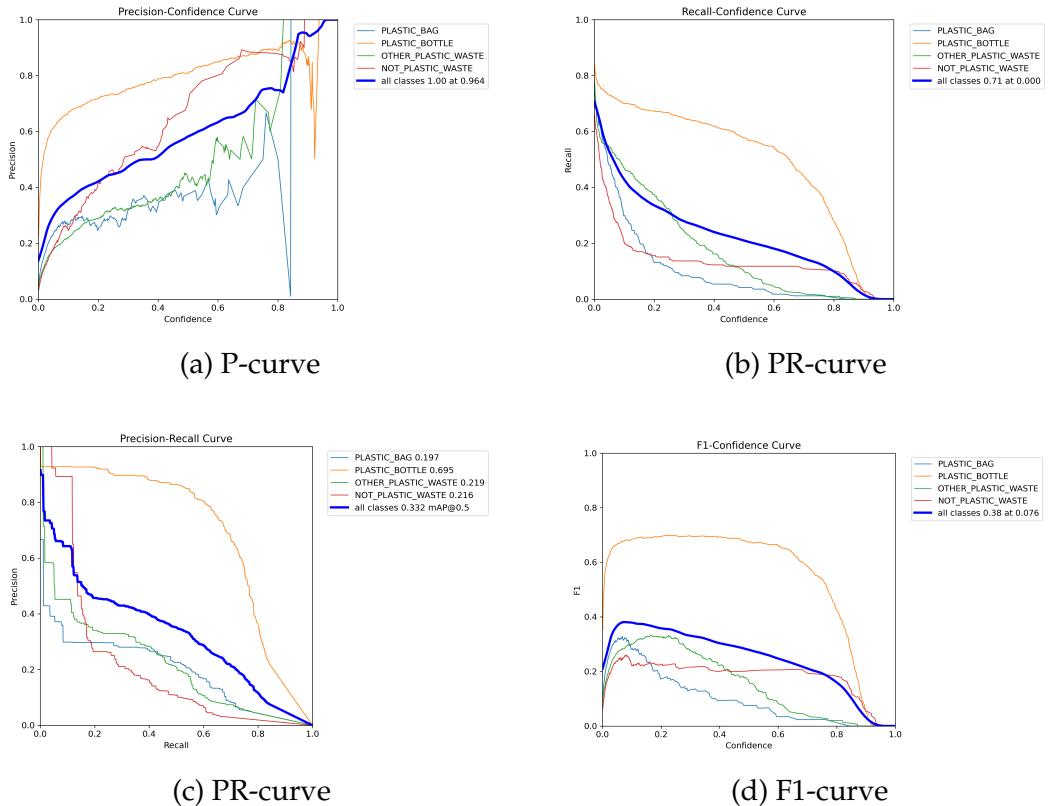


Figura 27: Andamento funzioni di loss e metriche durante l'esecuzione di medium-tune-2.0

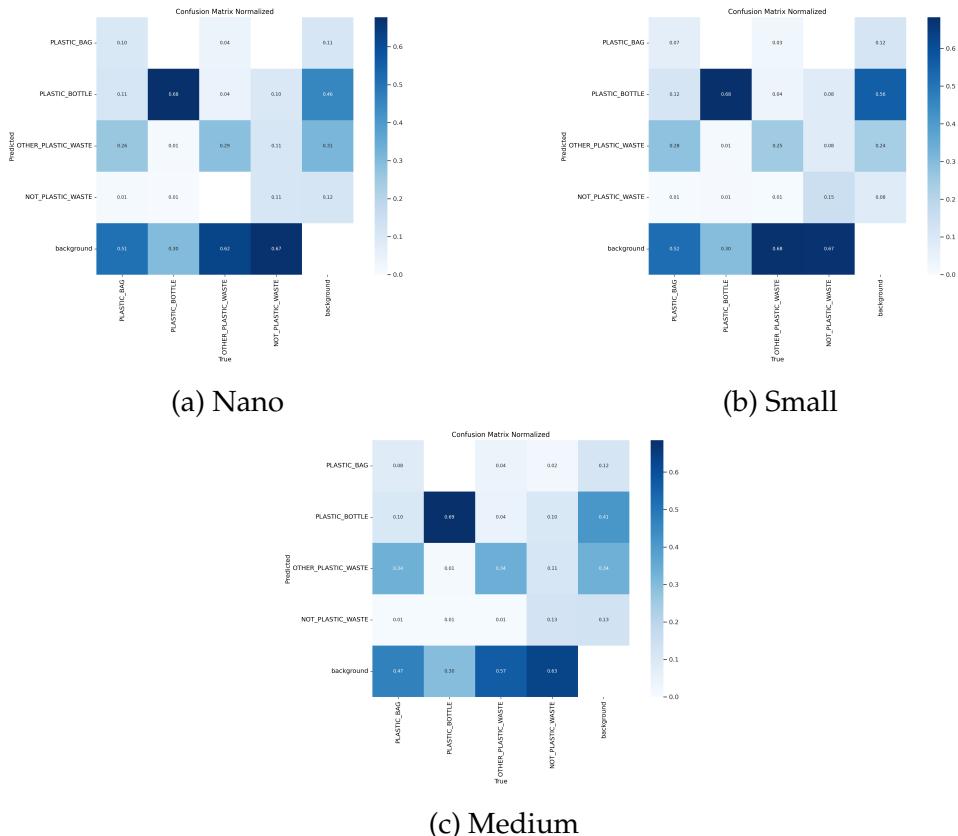


Figura 28: Comparazione tra le matrici di confusione normalizzate dei modelli `size-tune-2.0`