Università Politecnica delle Marche

FACOLTÀ DI INGEGNERIA
Corso di Laurea Magistrale in Ingegneria Informatica e dell'Automazione

Tesi di laurea magistrale

Analisi e implementazione di tecniche rule-based e AI-based per Machine Vision in ambito robotico

Analysis and Implementation of Rule-Based and AI-Based Techniques for Machine Vision in Robotic Applications

Candidato:
Cecilia De Padova
Matricola 1107274

Relatore:
Giacomo Palmieri

Anno Accademico 2022-2023

Indice

1 Introduzione alla machine vision **2**
1.1 Tecniche di illuminazione 2
1.2 Camera 3
1.3 Software 5

2 Applicazione della machine vision per la presa robot **6**

3 Algoritmi di visione rule-based e algoritmi di visione AI-based **8**
3.1 Algoritmi rule-based 9
3.2 Algoritmi AI-based 10
3.2.1 Valutazione apprendimento di una rete neurale 12

4 Strumenti e Software **14**
4.1 Strumenti 14
4.1.1 SensoPart Visor Robotic 14
4.1.2 Cognex In-Sight 2802 15
4.2 Alimentatore Omron S8VK-C12024 15
4.3 Software 16
4.3.1 Visor Vision Sensor 16
4.3.2 InSight Vision Suite 17
4.3.3 Visual Studio Code 18
4.3.4 Labelme 19

5 Applicazione reale **20**
5.1 Algoritmi rule-based 21
5.1.1 Camera SensoPart 22
5.1.2 Camera Cognex 23
5.2 Algoritmi AI-based 28
5.2.1 YOLOv8 28
5.2.2 RetinaNet 34
5.2.3 Confronto tra YOLOv8 e RetinaNet 44

6 Algoritmi su setup sperimentale **47**
6.1 I nuovi algoritmi rule-based per il setup sperimentale 47
6.1.1 Prima tipologia di cartoni 47
6.1.2 Seconda tipologia di cartoni 48
6.1.3 Terza tipologia di cartoni 50
6.1.4 Quarta tipologia di cartoni 52
6.2 I nuovi algoritmi AI-based per il setup sperimentale 54

7 Confronto algoritmi rule-based e AI-based **55**
7.1 Implementazione programmi per confronto 55
7.2 Confronto tra gli algoritmi 59

8 Conclusioni **70**

Elenco delle figure

1 Spettro elettromagnetico 2
2 Rappresentazione della lunghezza focale 3
3 Rappresentazione del filtro di Bayer 4
4 Posizionamento sistema di visione 6
5 Confronto tra algoritmi *rule-based* e algoritmi *AI-based* 8
6 Applicazione pattern matching 9
7 Applicazione contour 10
8 Applicazione blob 10
9 Rappresentazione rete neurale 11
10 SensoPart Visor Robotic 14
11 Uscite per collegamenti camera SensoPart 14
12 Alimentazione camera SensoPart 15
13 Cognex In-Sight 2802 15
14 Alimentazione camera Cognex 15
15 Alimentatore Omron S8VK-C12024 16
16 SensoFind 16
17 SensoConfig 17
18 EasyBuilder 18
19 SpreadSheet 18
20 Labelme 19
21 Punti di presa del primo cartone 20
22 Punti di presa del secondo cartone 20
23 Punti di presa del terzo cartone 21
24 Punti di presa del quarto cartone 21
25 Algoritmo SensoPart per la prima tipologia di cartone 22
26 Algoritmo SensoPart per la seconda tipologia di cartone 23
27 Spreadsheet funzioni applicate 24
28 Riferimento per allineamento 24
29 Allineamento su immagine d'esempio 24
30 Riferimento primo punto di presa 25
31 Parametri primo pattern di riferimento 25
32 Parametri ricerca del primo pattern 26
33 Ricerca pattern immagine di riferimento 26
34 Secondo punto di presa 27
35 Terzo punto di presa 27
36 Quarto punto di presa 27
37 Grafici risultati allenamento YOLOv8 per primo cartone 33
38 Grafici risultati allenamento YOLOv8 per secondo cartone 33

39 Grafici risultati allenamento YOLOv8 per terzo cartone 33
40 Grafici risultati allenamento YOLOv8 per quarto cartone 34
41 Grafici *loss* allenamento primo cartone 39
42 Grafici *loss* allenamento secondo cartone 40
43 Grafici *loss* allenamento terzo cartone 40
44 Grafici *loss* allenamento quarto cartone 41
45 Risultati AP e AR allenamento primo cartone 42
46 Risultati AP e AR allenamento secondo cartone 43
47 Risultati AP e AR allenamento terzo cartone 43
48 Risultati AP e AR allenamento quarto cartone 43
49 Confronto primo cartone 45
50 Confronto secondo cartone 45
51 Confronto terzo cartone 46
52 Confronto quarto cartone 46
53 Posizionamento blocchi prima tipologia nel setup sperimentale . 47
54 Posizionamento blocchi seconda tipologia nel setup sperimentale 48
55 Posizionamento blocchi seconda tipologia nel setup sperimentale 49
56 Algoritmo seconda tipologia su setup sperimentale 50
57 Posizionamento blocchi terza tipologia primo layer nel setup sperimentale 51
58 Posizionamento blocchi terza tipologia secondo layer nel setup sperimentale 51
59 Riferimento per discriminare primo *layer* 52
60 Riferimento per discriminare secondo *layer* 52
61 Posizionamento blocchi quarta tipologia nel setup sperimentale . 53
62 Algoritmo quarta tipologia su setup sperimentale 53
63 Interfaccia grafica 58
64 Grafico frequenza errore algoritmo rule-based primo cartone . . . 60
65 Grafico frequenza errore algoritmo AI-based primo cartone 60
66 Grafico dispersione centri primo cartone 61
67 Grafico frequenza errore algoritmo rule-based secondo cartone . . 61
68 Grafico frequenza errore algoritmo AI-based secondo cartone . . 62
69 Grafico dispersione centri secondo cartone 62
70 Grafico frequenza errore algoritmo rule-based terzo cartone . . . 63
71 Grafico frequenza errore algoritmo AI-based terzo cartone 63
72 Grafico dispersione centri terzo cartone 64
73 Grafico frequenza errore algoritmo rule-based quarto cartone . . 64
74 Grafico frequenza errore algoritmo AI-based quarto cartone . . . 65
75 Grafico dispersione centri quarto cartone 65
76 Esempio di immagine test non riconosciuta cartone tipologia uno 66

77 Fallimento algoritmo rule-based cartone tipologia uno 66
78 Risultato con algoritmo AI-based cartone tipologia uno 66
79 Esempio di immagine test non riconosciuta cartone tipologia due 67
80 Fallimento algoritmo rule-based cartone tipologia due 67
81 Risultato con algoritmo AI-based cartone tipologia due 67
82 Esempio di immagine test non riconosciuta cartone tipologia tre 68
83 Fallimento algoritmo rule-based cartone tipologia tre 68
84 Risultato con algoritmo AI-based cartone tipologia tre 68
85 Esempio di immagine test non riconosciuta cartone tipologia quattro 69
86 Fallimento algoritmo rule-based cartone tipologia quattro 69
87 Risultato con algoritmo AI-based cartone tipologia quattro . . . 69

Listato

1 Funzione di img_augmentation.py per aumentare il dataset . . . 29
2 Funzione di jsontotxt.py per la creazione dei file nel formato YOLO 31
3 main.py . . . 32
4 config.yaml . . . 32
5 Funzione get_dicts contenuta in utilities.py . . . 35
6 Funzione register_datasets contenuta in utilities.py . . . 36
7 Configurazione parametri in train.py per allenamento di RetinaNet 36
8 Addestramento RetinaNet in train.py con calcolo metriche d'interesse . . . 38
9 Funzione di loss.py per calcolo loss su set di validazione . . . 38
10 Estrazione dati e generazione grafici in plot.py . . . 41
11 Codice in predict_yolo.py per predizione . . . 44
12 Codice in predict_retinanet.py per predizione . . . 44
13 Funzione principale di object_detector.py per calcolo e ordinamento centro . . . 55
14 Funzione per gestire i clic del mouse nell'interfaccia in errore_centro.py 57
15 Calcolo centro in errore_centro.py . . . 57
16 Funzione per l'esecuzione degli script con i parametri selezionati in gui.py . . . 58

Sommario

La presente tesi si focalizza sulla progettazione di un sistema di visione destinato a un robot industriale. L'obiettivo di questo sistema è identificare con precisione la posizione del centro di diversi tipi di cartoni disposti su pallet, al fine di trasmettere tali coordinate al robot. Questo permetterà al robot di eseguire l'operazione di presa nel modo più accurato possibile. Il sistema di visione sarà realizzato mediante l'utilizzo di diverse tecnologie che comprenderanno sia algoritmi basati sul *deep learning* che algoritmi basati su regole.

La tesi è suddivisa in otto sezioni per fornire una panoramica completa del sistema realizzato. La prima sezione introduce la *machine vision* e le sue componenti principali, definendo il contesto tecnologico trattato. Nella seconda sezione si tratta l'applicazione della *machine vision* alla robotica. Nella terza sezione vengono esposti i riferimenti teorici relativi agli algoritmi impiegati. Nella quarta sezione sono illustrati tutti gli strumenti utilizzati. La quinta sezione si concentra sulla realizzazione del sistema di visione, delineando le soluzioni che sono state sviluppate. La sesta sezione analizza il sistema implementato all'interno del setup sperimentale. La settima sezione consiste in un confronto tra i risultati ottenuti con algoritmi *rule-based* e *AI-based*. Infine, l'ottava sezione costituisce le conclusioni della tesi.

Durante il corso dell'elaborato verranno riportati degli estratti del codice per chiarificare ciò che è stato implementato.

1 Introduzione alla machine vision

La *machine vision* [1] comprende tutte le applicazioni industriali e non industriali in cui una combinazione di hardware e software fornisce una guida operativa ai dispositivi nell'esecuzione delle loro funzioni basate sulla cattura e l'elaborazione di immagini.

La *machine vision* è basata sui sistemi di visione, ovvero sistemi che permettono di acquisire immagini, trasferirle al computer e produrre quindi un risultato. I componenti di un sistema di visione sono: l'illuminazione, la camera e il software.

1.1 Tecniche di illuminazione

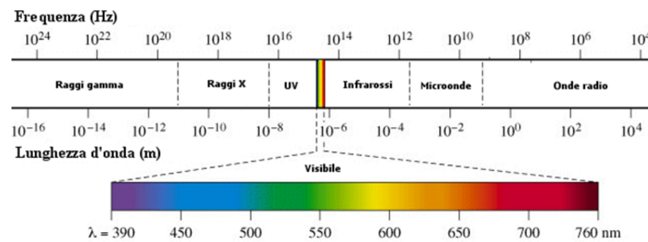


Figura 1: Spettro elettromagnetico

Lo spettro visibile comprende le lunghezze d'onda tra 400 e 700 nanometri circa, mentre la parte di spettro che viene definita luce va dai raggi X alle microonde. I colori percepiti dall'occhio umano rappresentano le lunghezze d'onda riflesse da un oggetto. Un'illuminazione adeguata di un oggetto gioca un ruolo cruciale nell'ottimizzazione di un algoritmo, in quanto contribuisce all'amplificazione del contrasto e, di conseguenza, alla miglior leggibilità dell'immagine risultante. La corretta illuminazione di un'immagine dipende sostanzialmente da due fattori fondamentali: la lunghezza d'onda della luce utilizzata e la geometria dell'illuminazione applicata.

In base alle lunghezze d'onda che l'oggetto riflette si decide di applicare un filtro o di illuminare con la lunghezza d'onda opportuna. Ad esempio, su un oggetto di colore rosso, l'applicazione di un filtro blu può notevolmente accentuarne la visibilità, poiché il rosso assorbe la luce rossa e riflette quella blu. L'uso di filtri può estendersi anche a regioni non visibili dello spettro, come nel caso dei filtri NIR (*Near Infrared*) e SWIR (*Short-wave Infrared*). Il filtro SWIR, ad esempio, è in grado di evidenziare opacità e trasparenze degli oggetti. Spostandosi a frequenze ancora maggiori dello spettro si trova il LWIR (*Long-wave Infrared*) il quale genera un'immagine monocromatica che converte il freddo in tonalità di blu e il caldo in tonalità di rosso. Tale filtro non registra la luce

riflessa, ma l'emissione di radiazione termica. Una tipica applicazione riguarda il controllo della temperatura dei circuiti. Un'altra tipologia di illuminazione è data dall'utilizzo di luce ultravioletta. Questa forma di illuminazione induce la fluorescenza in alcuni oggetti, evidenziando particolari con caratteristiche fluorescenti. Ad esempio, viene applicata per l'analisi di banconote o di difetti su particolari magnetici: gli oggetti vengono immersi in materiale fluorescente e successivamente asciugati per evidenziare eventuali difetti. Un'ulteriore metodologia di illuminazione si ottiene con l'impiego dei raggi X che permettono di opacizzare le parti più dense.

Le caratteristiche del materiale, quali la riflessione, la trasmissione e l'assorbimento della luce quando interagisce con la superficie, guidano la scelta della geometria di illuminazione. Alcune delle geometrie di illuminazione più conosciute includono il *dome*, la circolina e il *doal*.

1.2 Camera

La camera rappresenta lo strumento per l'acquisizione delle immagini ed è composta da un insieme di lenti le cui caratteristiche possono differire. L'insieme di lenti montate sulla camera prende il nome di ottica. I parametri che hanno più rilevanza su una lente sono: la lunghezza focale, il *mount* e la dimensione massima del sensore.

La lunghezza focale è la distanza dalla lente al punto in cui convergono i raggi luminosi.

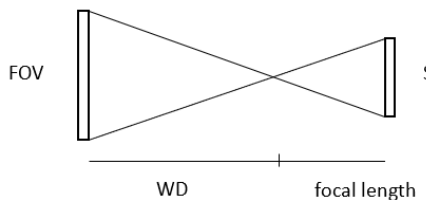


Figura 2: Rappresentazione della lunghezza focale

Nell'immagine sovrastante WD rappresenta la distanza alla quale si vuole acquisire l'immagine, FOV lo spazio che si vuole inquadrare, S la grandezza del sensore e *focal length* la lunghezza focale. I triangoli sono simili, quindi vale la seguente relazione: $\frac{WD}{FOV} = \frac{focallength}{S}$.

Il *mount* rappresenta l'interfaccia meccanica che permette di collegare la lente alla camera. Esistono ottiche con interfaccia meccanica predefinita, come ad esempio *f-mount* e *c-mount*, mentre esistono ottiche senza un'interfaccia predefinita di cui viene specificato solo il passo. La distinzione tra queste interfacce risiede nel fatto che, nel primo caso, il piano di fuoco viene regolato median-

te la rotazione di una ghiera che ricolloca le lenti, mentre nel secondo caso la regolazione avviene mediante uno spostamento meccanico della lente stessa.

È inoltre importante conoscere la dimensione del sensore al fine di scegliere l'apposita lente e prevenire l'effetto vignettatura. La lente viene scelta in base alla diagonale del sensore.

Oltre alle peculiarità già elencate ci sono ulteriori caratteristiche importanti delle lenti. La profondità di campo (DOF) rappresenta l'area in cui l'oggetto risulta nitido, mentre la funzione di trasferimento della modulazione (MTF) fornisce un'indicazione dell'efficienza di trasferimento ottico, permettendo così la misurazione della capacità di risoluzione dell'ottica.

Dopo aver definito le caratteristiche delle lenti e quindi dell'ottica si procede all'analisi delle camere e dei sensori. Lo scopo dei sensori e delle camere è la trasduzione del segnale; ne esistono di vari tipi in base alla lunghezza d'onda di interesse. Un sensore è definito come un insieme di pixel dove ogni pixel è un fotorecettore. I sensori possono essere classificati come matriciali o lineari, a seconda che acquisiscano informazioni da aree o righe. Il pixel non è in grado di discriminare la lunghezza d'onda del fotone, ma converte il fotone stesso in un impulso elettrico. I sensori misurano l'impulso elettrico senza distinguere i colori, producendo quindi immagini in scala di grigio.

Per ottenere immagini a colori si ricorre al filtro di Bayer, che consiste in una serie di filtri inseriti su ogni pixel (R, G o B). Il filtro ha un pattern fisso, può solo cambiare il pixel di inizio. I pixel verdi sono il doppio degli altri poiché l'occhio umano è più sensibile al verde, richiedendo quindi una riproduzione più accurata. Per determinare le informazioni di un determinato pixel che filtra un altro colore si usa l'interpolazione, ovvero la media dei filtri adiacenti che filtrano il colore di interesse.

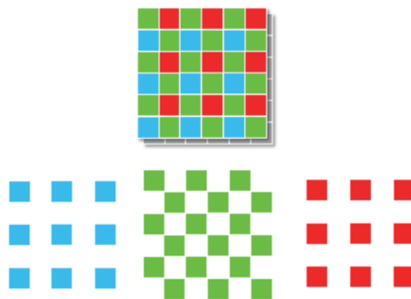


Figura 3: Rappresentazione del filtro di Bayer

1.3 Software

Esistono specifici software dedicati all'acquisizione e all'analisi delle immagini dalle camere. Tipicamente, ciascuna camera è dotata del proprio ambiente di sviluppo, il quale include algoritmi pre-implementati e interfacce grafiche che facilitano il loro utilizzo. In alternativa, esistono dei programmi esterni che permettono di configurare camere di vari modelli e che integrano funzioni e algoritmi complessi utili all'analisi delle immagini.

2 Applicazione della machine vision per la presa robot

L'integrazione della *machine vision* con i robot industriali [2] rappresenta un miglioramento nel settore dell'automazione industriale. L'utilizzo della visione per i robot consente di acquisire informazioni sull'ambiente circostante e sui pezzi da lavorare, permettendogli quindi di effettuare misurazioni e valutazioni. La capacità visiva permette ai robot di essere impiegati in applicazioni che richiedono l'identificazione degli oggetti. Inoltre, la visione consente ai robot di collaborare con le persone e integrare le informazioni provenienti dalla visione con quelle provenienti dai sensori.

Nel contesto dell'automazione industriale, i robot sono comunemente impiegati per sostituire il lavoro manuale, migliorando l'efficienza delle linee di produzione e riducendo i costi. Grazie all'introduzione della *machine vision* i robot possono realizzare operazioni di produzione più intelligenti e più ottimizzate. Questa integrazione non solo estende il campo di applicazione dei robot industriali, ma contribuisce anche a migliorare la qualità dei prodotti.

Per integrare il sistema di visione nell'applicazione robotica, esistono due configurazioni principali [3]: montare la camera su una superficie o montarla sul braccio del robot. Una camera montata sul robot è l'opzione più flessibile e garantisce migliori performance nella rilevazione dei componenti. Le telecamere fisse sono più facili da configurare e sono tipicamente adatte per applicazioni più semplici e di dimensioni ridotte.

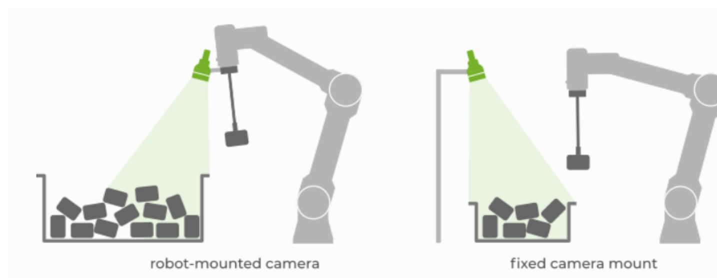


Figura 4: Posizionamento sistema di visione

Introdurre la visione per un robot fa nascere una problematica importante, ovvero come mappare le misurazioni della camera all'interno del sistema di coordinate dello spazio di lavoro del robot. Questo problema è noto come *hand-eye calibration* [4] e consiste in quattro fasi. Innanzitutto, c'è la fase di acquisizione della posa della camera basato sulla calibrazione della camera che consente di ottenere i parametri interni ed esterni della stessa. I parametri interni riguardano le caratteristiche intrinseche della camera come il fattore di scala e i coefficienti

di distorsione, necessari per convertire dal sistema di coordinate della camera alla rappresentazione in pixel. I parametri esterni, invece, descrivono la posizione della camera rispetto al sistema di coordinate del mondo. Successivamente, si procede con l'acquisizione della posa dell'*end-effector* del robot. Questo processo coinvolge la calibrazione del robot stesso che, attraverso lo studio della cinematica, identifica i parametri necessari per determinare la posizione precisa del punto d'interesse. La fase chiave è l'*hand-eye calibration*, con cui si ottiene la matrice di trasformazione omogenea che relaziona la camera e l'*end-effector* del robot. Infine, è fondamentale valutare l'accuratezza della calibrazione utilizzando metriche adeguate per garantire che il sistema operi con la precisione necessaria.

La *machine vision*, integrata con i robot, trova applicazioni [5] in varie aree. Nell'ambito dell'ispezione, la visione si impiega per analizzare finiture superficiali, dimensioni, etichettatura e individuare difetti sugli oggetti, offrendo una precisione e una velocità superiori rispetto all'ispezione umana. Nell'identificazione supporta i robot nel rilevare e classificare oggetti, consentendo loro di individuare gli elementi distintivi di un oggetto. Nei contesti di navigazione corregge i dati sensoriali per guidare i robot in ambienti dinamici, migliorando l'accuratezza dei loro movimenti. Nel controllo qualità si affida all'ispezione e all'identificazione per valutare la qualità dei prodotti in modo efficiente e accurato. Nell'assemblaggio facilita operazioni precise di *pick and place* per l'assemblaggio di componenti con un'alta precisione. Nella localizzazione delle parti i robot la utilizzano per individuare e identificare le parti necessarie, ottimizzando il processo di assemblaggio. Infine, nei sistemi di elaborazione dati, interpreta l'ambiente circostante per guidare i robot nel trasporto di pezzi.

3 Algoritmi di visione rule-based e algoritmi di visione AI-based

L'obiettivo del progetto consiste nel riconoscere la posizione del centro dell'oggetto analizzato al fine di trasferire questa informazione al robot per realizzare una presa precisa. Per far ciò si è deciso di adottare due approcci: *rule-based*, basato sugli algoritmi di *machine vision* tradizionali e *AI-based*, fondato sul *deep learning*.

Gli algoritmi tradizionali [6] sono molto efficaci in una scena strutturata caratterizzandosi per velocità, precisione e ripetibilità; possono facilmente ispezionare dettagli di oggetti troppo piccoli per essere visti dall'occhio umano garantendo una maggiore affidabilità e minori errori. In una linea di produzione, gli algoritmi di *machine vision* tradizionali possono ispezionare centinaia o migliaia di pezzi al minuto in modo affidabile e ripetitivo, superando notevolmente la capacità di ispezione umana. D'altra parte il *deep learning* è migliore per l'interpretazione qualitativa di una scena complessa e non strutturata, specialmente in presenza di difetti sottili e imperfezioni imprevedibili. La tecnologia del *deep learning* utilizza reti neurali che emulano l'intelligenza umana per distinguere anomalie, componenti e caratteri, tollerando le variazioni naturali in modelli complessi. In questo modo combina la flessibilità dell'essere umano con la velocità e la robustezza di un sistema computerizzato.

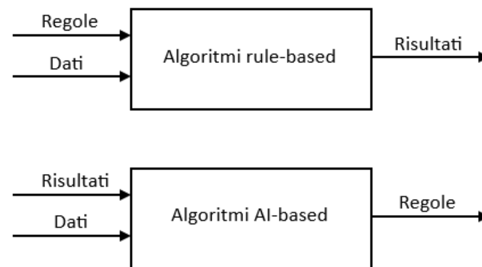


Figura 5: Confronto tra algoritmi *rule-based* e algoritmi *AI-based*

Nel confronto tra *deep learning* e i metodi tradizionali di *machine vision* [7] la differenza più significativa risiede nel modo in cui avviene l'estrazione delle caratteristiche dell'oggetto esaminato. Con gli algoritmi tradizionali bisogna decidere quali caratteristiche cercare per rilevare un determinato oggetto in un'immagine. Occorre, inoltre, selezionare il giusto insieme di caratteristiche per ciascuna classe di oggetti, ciò diventa complicato con l'aumentare delle classi. In contrasto, il *deep learning* istruisce l'algoritmo su cosa cercare in re-

lazione a ciascuna classe specifica. Analizzando immagini di esempio apprende automaticamente le caratteristiche più rappresentative per ciascuno oggetto.

3.1 Algoritmi rule-based

Gli algoritmi *rule-based* [8] rappresentano l'approccio classico basato su regole per l'identificazione e la classificazione degli oggetti in un'immagine. Le regole vengono create in base al risultato desiderato e il software segue queste regole per classificare gli oggetti e prendere decisioni. Questi algoritmi sono più adatti per applicazioni che richiedono regole semplici e dirette, ovvero per processi che richiedono una bassa variabilità e un'alta ripetibilità. Questa tipologia di algoritmi permette di ottenere un'acquisizione rapida e accurata delle informazioni.

Nei sistemi di *machine vision* tradizionali, gli algoritmi sono divisi in due fasi. Nella prima fase, comunemente chiamata estrazione delle caratteristiche, vengono eseguite una serie di operazioni matematiche sui valori dei pixel dell'immagine. Nella seconda fase, chiamata di elaborazione, le caratteristiche estratte nella prima fase vengono combinate per giungere a una decisione finale sull'immagine. Questa fase decisionale viene spesso realizzata utilizzando una combinazione di parametri o soglie regolate manualmente. Gli algoritmi tradizionali più usati sono il *pattern matching*, il *contour* e il *blob*.

Il *pattern matching* [9] si compone di due fasi, apprendimento e abbinamento. Durante la fase di apprendimento, l'algoritmo estrae informazioni sul valore di grigio e/o sul gradiente dei bordi dall'immagine modello. L'algoritmo organizza e memorizza le informazioni in modo da facilitare una ricerca più rapida nell'immagine. Durante la fase di abbinamento, l'algoritmo estrae le informazioni sul valore di grigio e/o sul gradiente dei bordi nell'immagine ispezionata e trova le corrispondenze individuando le regioni nell'immagine in cui si osserva la correlazione più elevata.

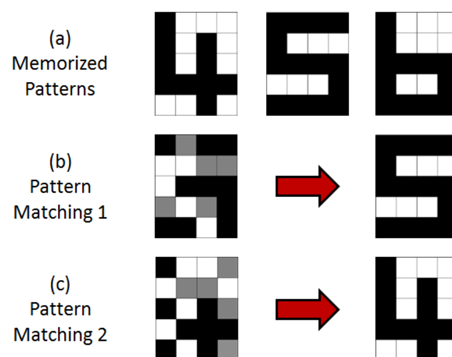


Figura 6: Applicazione pattern matching

Il *contour* [10] studia i contorni della parte d'interesse dell'immagine. In particolare, vengono estratti i contorni dell'area di interesse, viene studiata la curvatura del contorno trovato e viene confrontato con i contorni rilevati.

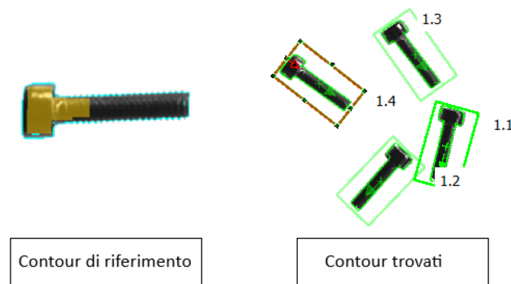


Figura 7: Applicazione contour

I *blob* rappresentano gruppi di pixel in un'immagine con alcune proprietà condivise, come il valore della scala di grigi. L'analisi dei *blob* inizia identificando queste aree nell'immagine con l'utilizzo di strumenti chiamati rilevatori di *blob*. Il rilevatore di *blob* trasforma l'immagine in una serie di immagini binarie, ciascun'immagine rappresenta una soglia diversa.

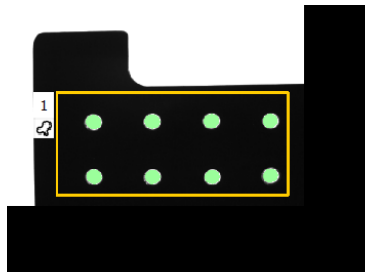


Figura 8: Applicazione blob

3.2 Algoritmi AI-based

Gli algoritmi *AI-based* rappresentano un approccio recente che utilizza reti neurali per imparare automaticamente come identificare e classificare oggetti in un'immagine. I programmi sono allenati su dataset composti da un grande numero di immagini e utilizzano la loro conoscenza per prendere decisioni su nuove immagini sconosciute. Questi algoritmi sono utilizzati in contesti complessi e con un'alta variabilità.

Prima di introdurre le tipologie di reti neurali è importante definire il concetto di rete neurale [11]: si tratta di un sistema informatico ispirato alle connes-

sioni del cervello umano. Durante il processo di addestramento, i dati vengono introdotti nello strato di input e attraversano strati computazionali successivi, ognuno costituito da connessioni complesse. Durante questo percorso, i dati subiscono trasformazioni significative fino a raggiungere lo strato di output. Durante l'addestramento, i pesi e le soglie vengono continuamente regolati affinché i dati di addestramento con le stesse etichette producano in modo coerente risultati simili.

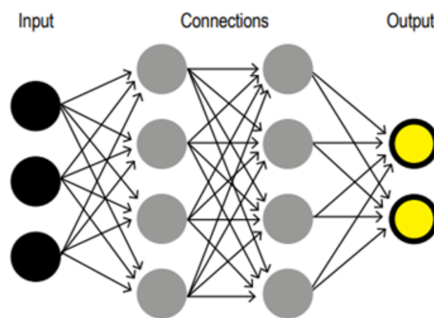


Figura 9: Rappresentazione rete neurale

Nel nostro caso la rete neurale addestrata dovrà essere in grado di riconoscere la presenza dell'oggetto al fine di rilevarne la posizione. Per raggiungere questo obiettivo è necessario introdurre il concetto di *object detection* [12], una tecnica di *computer vision* finalizzata a individuare istanze di oggetti in immagini o video. Gli algoritmi di *object detection* in genere sfruttano l'apprendimento automatico o il *deep learning* per produrre risultati significativi. Quando gli esseri umani guardano immagini o video, possono riconoscere e localizzare oggetti di interesse in pochi istanti. L'obiettivo dell'*object detection* è replicare questa capacità utilizzando algoritmi su computer. Le reti [13] per l'*object detection* possono essere classificate in due categorie: i metodi *region proposal-based* (R-CNN, Faster R-CNN, ecc.) e i metodi *classification-based* (*You Only Look Once* (YOLO), *single shot detector* (SSD), ecc.).

La FasterRCNN è composta da due reti: una rete per suggerire le posizioni potenziali degli oggetti da rilevare e una seconda rete per classificare e individuare gli oggetti basandosi sulle regioni proposte. Sebbene sia considerata una delle tecniche più accurate per l'*object detection*, richiede un notevole costo computazionale.

YOLO analizza le immagini in input e utilizza le *feature* di queste immagini per predire ogni *bounding box*. Partiziona ogni immagine in una griglia SxS e ciascuna cella anticipa N *bounding box* con i relativi *confidence score*. Il *confidence score* rileva l'accuratezza dei *bounding box*. YOLO è veloce ma ha il

vincolo di poter riconoscere soltanto una categoria all'interno di ogni cella della griglia.

SSD è progettata per rilevare numerosi oggetti in un'immagine con un singolo passaggio, ciò significa che la rete convoluzionale verrà eseguita una sola volta per trovare la *feature map* che utilizza, con risoluzioni diverse, per predire oggetti di dimensioni diverse. È molto più veloce dei metodi basati su due stadi.

RetinaNet è una tecnica avanzata che combina le caratteristiche di YOLO e SSD, è un sistema unificato e autonomo composto da una rete di base (*backbone*) e due sottoreti. La *backbone* crea una *feature map* su tutta l'immagine di input, mentre la prima sottorete si occupa della classificazione degli oggetti e la seconda sottorete della regressione dei *bounding box*, entrambe utilizzando l'output della rete di base. RetinaNet è nota per la sua capacità di gestire efficacemente oggetti di diverse dimensioni.

3.2.1 Valutazione apprendimento di una rete neurale

Le reti neurali, come accennato in precedenza, sono sottoposte a un processo di addestramento. La qualità di tale addestramento viene valutata attraverso il calcolo di diverse metriche, le quali consentono di valutare l'efficienza dell'*object detection*. Di seguito, verranno esaminate alcune di queste metriche che saranno impiegate nel corso del presente elaborato.

La *loss* (funzione di perdita) in una rete neurale è una misura dell'errore tra le previsioni della rete e i valori desiderati. L'obiettivo dell'addestramento è minimizzare questa *loss*, utilizzando algoritmi di ottimizzazione per aggiornare i pesi e i *bias* della rete, in modo che le previsioni diventino più accurate. La *loss* guida l'addestramento verso la migliore performance della rete e può essere calcolata sia sul dataset di addestramento che sul dataset di validazione.

Prima di definire le prossime metriche è importante dare delle definizioni ausiliarie. *True positive* (TP) [14] rappresenta il successo nell'identificare correttamente un elemento o una caratteristica desiderata. *False positive* (FP) rappresenta il rilevamento errato di un elemento inesistente o il rilevamento fuori luogo di un oggetto esistente. *False negative* (FN) rappresenta il non rilevamento di un elemento che in realtà è presente.

La precisione (*precision*) è una misura della frazione di istanze classificate da un modello come positive e che sono effettivamente positive, rispetto a tutte le istanze classificate come positive (sia vere positive che false positive). La formula è: $Precision = \frac{TP}{TP+FP}$. La precisione è utile per valutare quanto il modello sia preciso nella classificazione delle istanze positive. Una *precision* più alta indica una minore probabilità di avere falsi positivi.

La *recall* è una misura della frazione di istanze positive effettivamente classificate come positive dal modello rispetto a tutte le istanze effettivamente positive (sia vere positive che false negative). La formula è: $Recall = \frac{TP}{TP+FN}$. La *recall* è utile per valutare la capacità del modello di catturare correttamente tutte le istanze positive. Una *recall* più alta indica una minore probabilità di avere falsi negativi.

La IoU (*Intersection over Union*) [15], è una misura che quantifica la sovrapposizione tra un *bounding box* predetto e un *bounding box* di riferimento. Essa svolge un ruolo fondamentale nel valutare l'accuratezza della localizzazione degli oggetti.

L'AP (*Average Precision*), calcola l'area sotto la curva *precision-recall*, fornendo un singolo valore che riassume le prestazioni del modello in termini di *precision* e *recall*. L'*mAP50* è la media dell'AP calcolata con una soglia della IoU pari a 0,50. Questa metrica rappresenta l'accuratezza del modello considerando solo le rilevazioni facili. L'*mAP50-95* è la media dell'AP calcolata a diverse soglie di IoU, che vanno da 0,50 a 0,95. Questo fornisce una visione completa delle prestazioni del modello su diversi livelli di difficoltà nella rilevazione.

4 Strumenti e Software

Questa sezione è dedicata alla presentazione della strumentazione hardware e software utilizzata per il progetto.

4.1 Strumenti

4.1.1 SensoPart Visor Robotic

La SensoPart Visor Robotic [16] è una telecamera industriale a colori prodotta dall'azienda SensoPart. È impiegata principalmente nel campo della robotica grazie alle sue funzioni di calibrazione che semplificano l'orientamento nello spazio. Per il progetto è stata utilizzata una camera nella versione V20C caratterizzata da una risoluzione pari a 1440x1080 pixel.

Figura 10: SensoPart Visor Robotic

La telecamera dispone di due uscite [17]: una è destinata alla connessione tramite cavo Ethernet al PC o al PLC, mentre l'altra è dedicata al collegamento all'alimentazione.

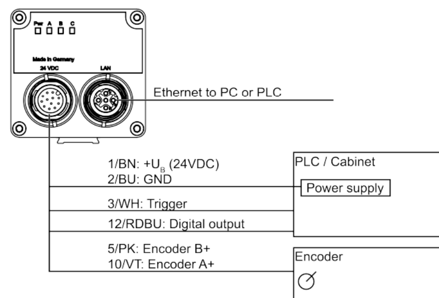


Figura 11: Uscite per collegamenti camera SensoPart

In dettaglio, dal cavo di alimentazione vengono utilizzati due fili per alimentare la telecamera: il filo blu viene collegato a ground (GND), mentre il filo marrone viene collegato alla tensione di 24V.

PIN	Color [3)]		Signal
1	BN		+ U_B (24 V DC)
2	BU		GND

Figura 12: Alimentazione camera SensoPart

4.1.2 Cognex In-Sight 2802

La telecamera industriale Cognex In-Sight 2802 [18] è di tipo monocromatico e presenta una risoluzione di 1920x1080 pixel. La sua illuminazione è multicolore, consente di illuminare l'oggetto con luce rossa, verde, blu o bianca. Un aspetto significativo di questa telecamera è la presenza di algoritmi basati sull'intelligenza artificiale.

Figura 13: Cognex In-Sight 2802

Anche la telecamera Cognex presenta due uscite: una destinata al collegamento e l'altra all'alimentazione. Per quanto riguarda l'alimentazione, vengono impiegati i cavi rosso e nero. Il cavo rosso viene collegato alla tensione di 24V, mentre il cavo nero è collegato a ground (GND).

+24 VDC	Red
GND	Black

Figura 14: Alimentazione camera Cognex

4.2 Alimentatore Omron S8VK-C12024

L'Omron S8VK-C12024 [19] è un alimentatore caratterizzato da un ingresso universale e un design compatto che ottimizza l'utilizzo dello spazio. Questo dispositivo consente di fornire l'alimentazione necessaria alle telecamere menzionate in precedenza, contribuendo così al corretto funzionamento del sistema.

Figura 15: Alimentatore Omron S8VK-C12024

4.3 Software

4.3.1 Visor Vision Sensor

Il software Visor Vision Sensor permette di acquisire immagini dalla telecamera SensoPart e di elaborarle. Esso è composto da due interfacce distinte: SensoFind e SensoConfig.

SensoFind offre la possibilità di selezionare un sensore o di simulare un sensore per la configurazione o la visualizzazione. Quest'interfaccia fornisce inoltre la possibilità di realizzare diversi settaggi di base.

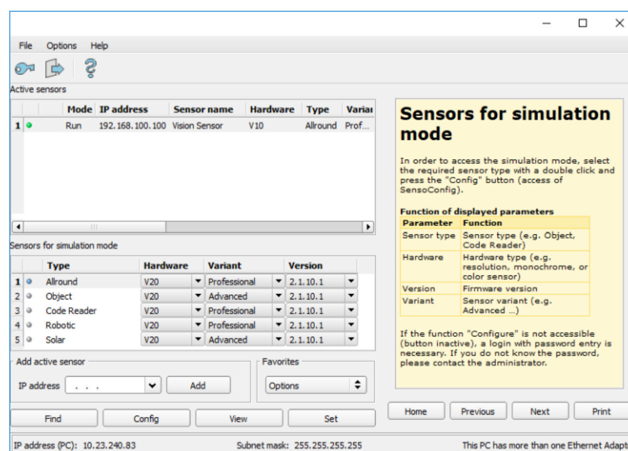


Figura 16: SensoFind

SensoConfig permette di creare gli algoritmi per l'analisi delle immagini acquisite. In particolare, ci sono 5 sezioni: *Job*, *Alignment*, *Detector*, *Output* e

Start Sensor. In *Job* sono racchiusi tutti gli algoritmi creati. In *Alignment* è presente la parte dell'algoritmo che permette di creare un allineamento rispetto ad un punto notevole, facilitando il corretto posizionamento delle immagini. In *Detector* sono riportati tutti gli algoritmi che si possono applicare per trovare le parti di interesse nell'immagine. In *Output* si ha l'interfaccia per scegliere quali valori, tra quelli calcolati, verranno rilasciati in output. *Start Sensor* permette di caricare sulla camera gli algoritmi creati.

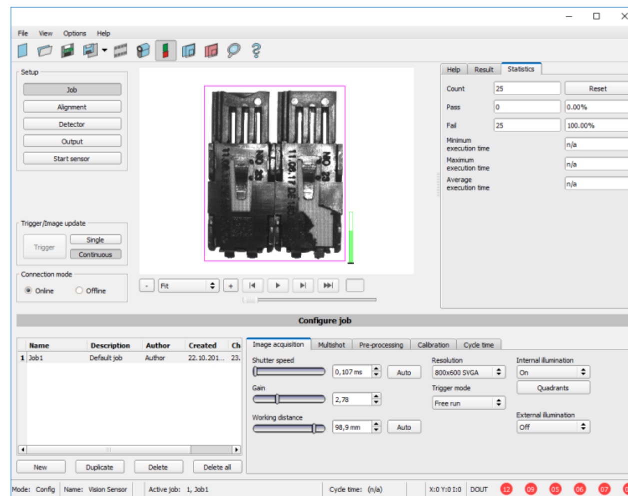


Figura 17: SensoConfig

4.3.2 InSight Vision Suite

InSight Vision Suite è il software dedicato all'acquisizione e all'elaborazione delle immagini provenienti dalla telecamera Cognex. Il software è suddiviso in: EasyBuilder e SpreadSheet. Il primo permette di realizzare algoritmi in modo rapido ma dispone di una lista di funzionalità molto limitata. Il secondo possiede molte più funzionalità ed ha una struttura basata su fogli di calcolo con celle. Per i fini del progetto viene impiegato EasyBuilder per la regolazione delle impostazioni di acquisizione mentre SpreadSheet è utilizzato per la realizzazione degli algoritmi.

EasyBuilder è composto da cinque sezioni. La sezione *Image* permette di regolare i parametri delle acquisizioni: l'illuminazione, la distanza di acquisizione dell'oggetto e l'esposizione. *Inspect* permette di aggiungere funzioni all'algoritmo in base alle esigenze richieste dalla specifica applicazione. *Communication* permette di configurare la connessione. *Output* permette di selezionare quali valori ottenuti dall'algoritmo devono essere restituiti in output. *HMI* permette di

configurare l'interfaccia che vedranno gli operatori addetti quando analizzeranno i risultati dell'algoritmo.

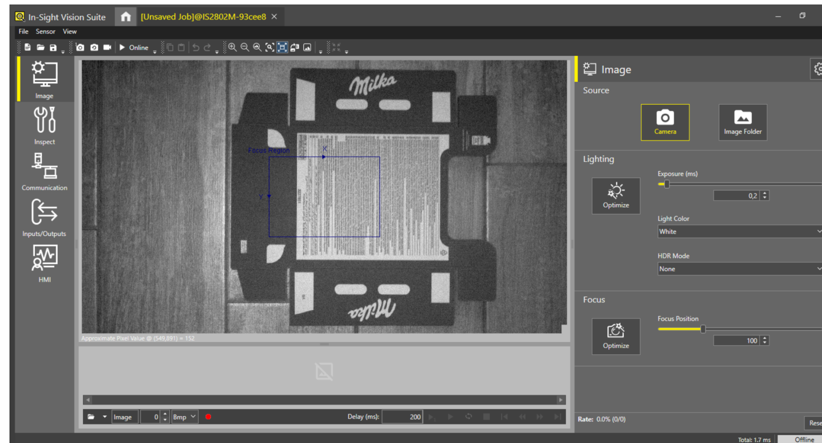


Figura 18: EasyBuilder

SpreadSheet offre la possibilità di utilizzare tutte le funzioni presenti in figura. Queste funzioni possono essere applicate aggiungendole direttamente al foglio di calcolo. Per configurare tali funzioni è sufficiente cliccare sulla funzione di interesse nel foglio di calcolo e regolare i parametri mediante l'interfaccia utente.

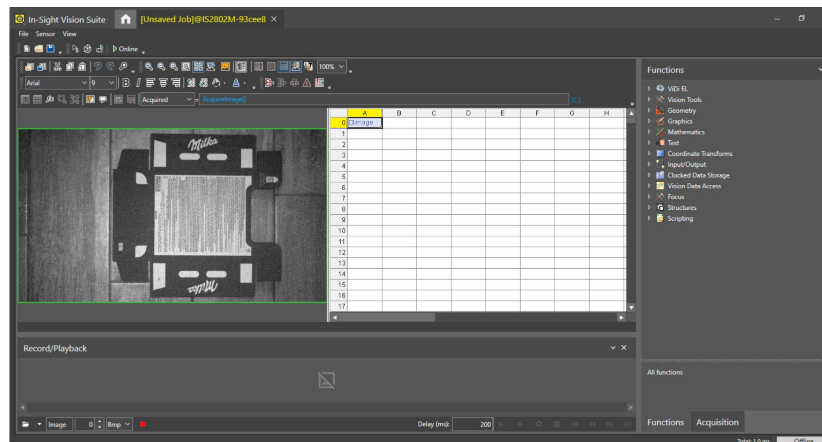


Figura 19: SpreadSheet

4.3.3 Visual Studio Code

Per addestrare le reti neurali si utilizza Python [20], linguaggio ad alto livello molto utilizzato in applicazioni di *deep learning*. Per l'implementazione dei programmi è stato scelto Visual Studio Code [21], editor di codice leggero ma po-

tente disponibile per Windows, macOS e Linux. É stato sviluppato da Microsoft e supporta vari linguaggi tra cui C++, C#, Java e Python.

4.3.4 Labelme

Labelme [22] è un tool grafico per l'annotazione delle immagini. L'annotazione è una fase necessaria per l'allenamento di una rete neurale; consiste nell'identificare e delimitare la posizione di specifici oggetti, caratteristiche o informazioni rilevanti nelle immagini. Labelme ha un'interfaccia molto intuitiva che permette di realizzare annotazioni per *bounding box detection*, *semantic segmentation* e *classification*.

Dalla sezione *Open* è possibile caricare le immagini che si desidera annotare. Dalla sezione *Create Polygons*, invece, è possibile selezionare il tipo di annotazione che si intende applicare.

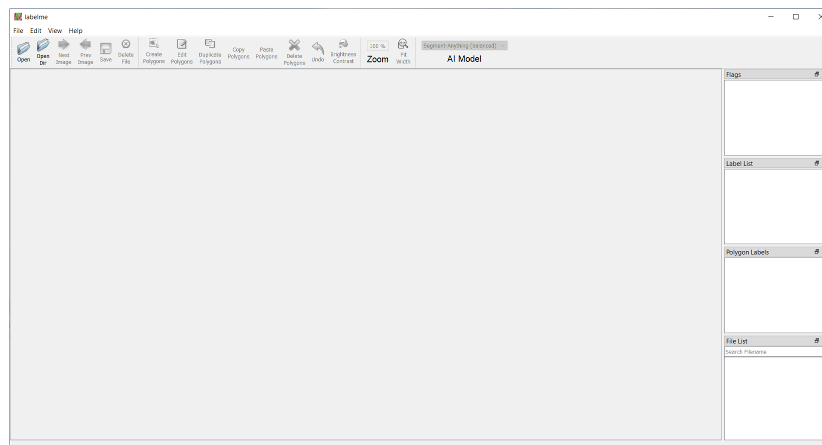


Figura 20: Labelme

5 Applicazione reale

L'obiettivo dell'applicazione analizzata è riconoscere la posizione di blocchi composti da cartoni al fine di fornire al robot una coordinata di presa il più accurata possibile. I cartoni che compongono ogni blocco possono essere leggermente disallineati tra loro. La precisione nella rilevazione della disposizione dei cartoni è fondamentale per garantire un'accurata coordinazione delle operazioni del robot. Si analizzano quattro tipologie di cartoni, ciascuno con specifici punti di presa che sono rappresentati nelle figure di seguito.

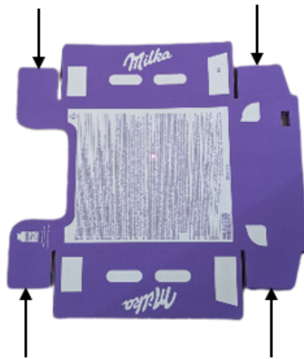


Figura 21: Punti di presa del primo cartone

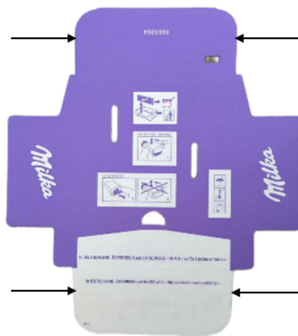


Figura 22: Punti di presa del secondo cartone

Figura 23: Punti di presa del terzo cartone

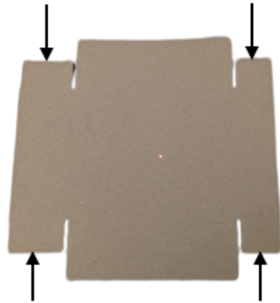


Figura 24: Punti di presa del quarto cartone

È importante precisare che, in caso di disallineamento dei cartoni che compongono un blocco, è essenziale individuare sempre il bordo del cartone esterno. In caso contrario il robot potrebbe entrare in contatto con i cartoni in maniera scorretta causando danni e impedendo una presa efficace.

5.1 Algoritmi rule-based

L'obiettivo degli algoritmi è creare un sistema di riconoscimento per quattro tipologie di cartoni al fine di individuarne i rispettivi centri; per fare ciò si individuano i punti di presa, tutti o solo due opposti. Il centro viene determinato calcolando la media delle coordinate dei punti, garantendo un posizionamento coerente con l'orientamento dei cartoni. Questo punto verrà poi passato al robot per permettergli di svolgere correttamente la presa. Vengono utilizzate entrambe le telecamere descritte in precedenza ognuna delle quali, grazie ai software proprietari, permette di sviluppare un algoritmo per ogni tipologia di scatola.

5.1.1 Camera SensoPart

Sono riportati gli algoritmi per la prima e la seconda tipologia di scatola, per il terzo e quarto cartone sono stati sviluppati degli algoritmi con la stessa struttura del primo. Per tutti gli algoritmi sono stati applicati tre filtri: *Erosion* con una *Property* 1x11, *Dilation* con una *Property* 11x1 e *Median* con una *Property* 11x11.

Prima tipologia di scatola

L'algoritmo sviluppato per la prima scatola è composto da una parte di *Alignment* e una parte di *Detector* per la rilevazione dei punti di interesse. L'allineamento è realizzato grazie a un *contour matching* mentre il rilevatore per i quattro bordi è basato su *pattern matching*. Per l'allineamento si è scelta come riferimento una caratteristica unica, presente solamente in un punto specifico rendendola un punto di riferimento ideale per l'allineamento. Il centro del riferimento scelto è indicato in figura dalle frecce colorate di azzurro. Per rilevare le parti di interesse nella scatola sono stati applicati quattro *pattern matching*, ciascuno mirato a rilevare uno degli angoli dell'oggetto. Utilizzando un offset è stato possibile ottenere, dai punti rilevati con il *pattern matching*, le coordinate dei punti di presa. Successivamente, è stato calcolato il centro dell'oggetto utilizzando una formula descritta nella figura nella sezione *Expressions*.

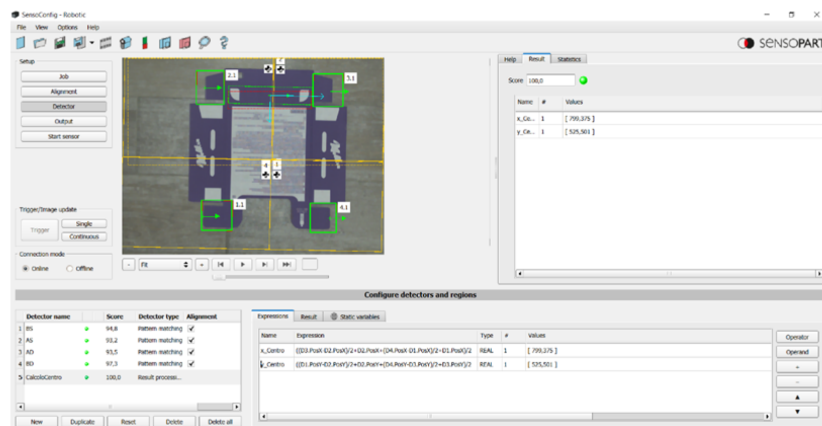


Figura 25: Algoritmo SensoPart per la prima tipologia di cartone

Seconda tipologia di scatola

L'algoritmo realizzato per la seconda tipologia di scatola è una combinazione di *contour matching* per l'allineamento e di *detector contour* per il rilevamento dei bordi. Il *detector contour* funziona molto bene per questa tipologia di scatola, garantisce che sia sempre individuato il bordo della scatola più esterna anche in situazioni di disallineamento. Anche in questo caso è stato applicato un offset per rilevare le coordinate dei punti di presa. Sono state svolte delle prove

anche sulle altre tipologie di scatole utilizzando l'approccio del *contour matching* nell'algoritmo. Tuttavia, i risultati ottenuti non sono stati soddisfacenti, poiché l'algoritmo tendeva a rilevare frequentemente i bordi dei cartoni più interni.

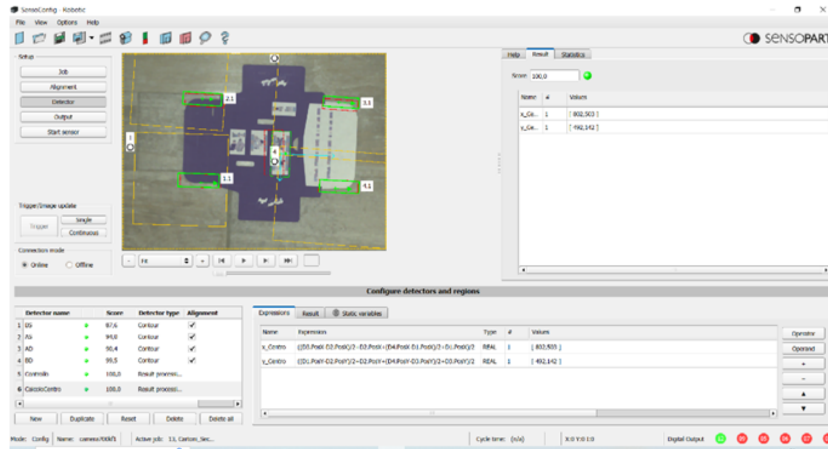


Figura 26: Algoritmo SensoPart per la seconda tipologia di cartone

5.1.2 Camera Cognex

Di seguito è riportato l'algoritmo sviluppato per la prima scatola. Gli algoritmi per le altre tre categorie di scatole sono stati creati seguendo lo stesso procedimento della prima. L'unica distinzione risiede nella quarta categoria di cartoni, per la quale è stata utilizzata una luce blu durante le acquisizioni al fine di ottenere un contrasto più elevato rispetto al pavimento.

Per tutte le scatole sono stati applicati i filtri: *Erode* 11x1, *Dilate* 1x11, *Close* 3x3 e *Fill Dark Holes*. Inoltre, durante l'acquisizione è stato applicato l'*HDR* per ottenere un risultato migliore.

Prima tipologia di scatola

L'algoritmo creato si divide in allineamento e rilevazione dei punti di interesse. Di seguito è riportato lo spreadsheet che contiene tutte le funzioni applicate.

	A	B	C	D	E	F	G	H	I	J
0	DImage									
1										
2		DImage								
3	DImage									
4	DImage									
5										
6	DPatterns	1,000								
7	DPatterns	0,000	741,507	133,975	1,891	100,000	93,193			
8										
9	DPatterns	1,000								
10	DPatterns	0,000	745,209	953,886	2,447	100,000	93,262			
11										
12	DPatterns	1,000								
13	DPatterns	0,000	1556,812	946,137	2,093	100,000	72,591			
14										
15	DPatterns	1,000								
16	DPatterns	0,000	1564,906	141,915	2,935	100,000	65,295			
17										
18										
19	DPatterns	1,000								
20	DPatterns	0,000	1148,781	176,546	1,799	100,000	98,601			
21										
22										

Figura 27: Spreadsheet funzioni applicate

In modo analogo a quanto è stato fatto per la camera SensoPart, è stata selezionata una caratteristica unica del cartone come punto di riferimento per l'allineamento.

Figura 28: Riferimento per allineamento

Nell'immagine catturata, che viene utilizzata come esempio di applicazione dell'algoritmo, l'allineamento viene identificato e posizionato nel seguente modo.

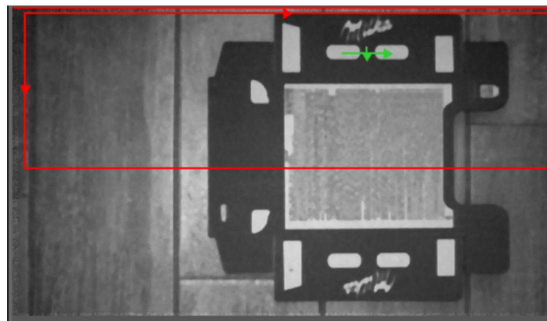


Figura 29: Allineamento su immagine d'esempio

Per identificare i punti di presa, sono stati introdotti quattro pattern di riferimento. Il pattern scelto per il primo punto è illustrato nella figura seguente.

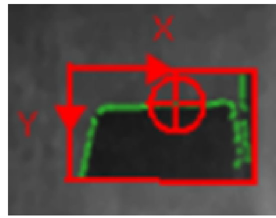


Figura 30: Riferimento primo punto di presa

La funzione utilizzata per creare un pattern di riferimento è la *TrainPathMaxRedLine*, a cui sono forniti i seguenti parametri:

- *Image*: parametro che richiede il riferimento alla cella contenente la struttura dati Immagine.
- *Pattern region*: parametro che consente di specificare la regione di interesse (*ROI*) che contiene il pattern che si vuole utilizzare come riferimento.
- *Pattern origin*: l'offset dal centro della *ROI* che permette di ottenere le coordinate del punto di interesse.
- *Fine Granularity*: parametro impiegato per trovare piccole caratteristiche e determinare con precisione il pattern.

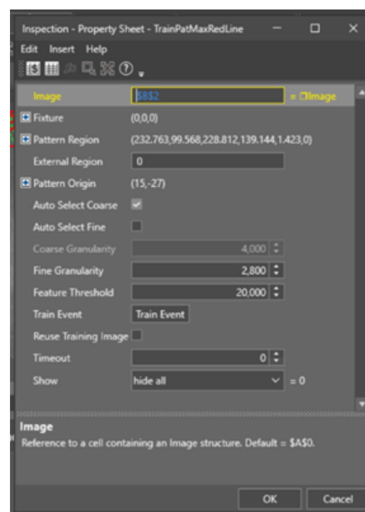


Figura 31: Parametri primo pattern di riferimento

Per determinare l'area in cui cercare il pattern di riferimento appena creato si utilizza la funzione *FindPatMaxRedLine*, nella quale vengono specificati i seguenti parametri:

- *Image*: analogo a quello scritto sopra.
- *Fixture*: definisce la regione di ricerca in relazione al riferimento fornito. L'impostazione della *ROI* relativa ad una *Fixture* garantisce che se la *Fixture* viene ruotata o traslata, la *ROI* verrà ruotata o traslata di conseguenza.
- *Find Region*: specifica l'area dell'immagine che viene sottoposta ad analisi.
- *Pattern*: deve fare riferimento a una cella del foglio di calcolo contenente una struttura dati *Patterns* valida restituita dalla funzione *TrainPatMaxRedLine*.

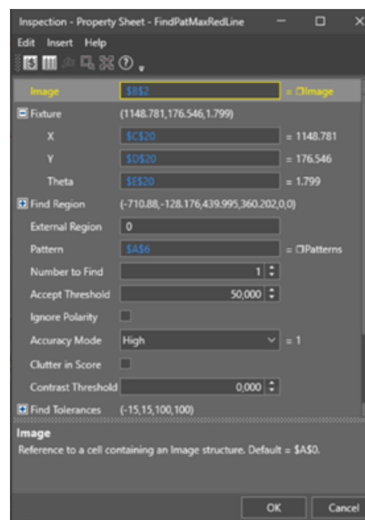


Figura 32: Parametri ricerca del primo pattern

Il risultato ottenuto nell'immagine utilizzata come esempio è il seguente.

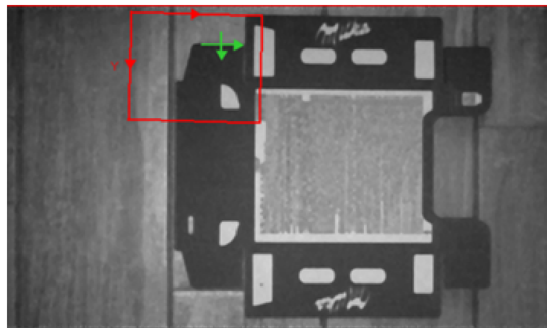


Figura 33: Ricerca pattern immagine di riferimento

A seguire, verranno presentati in modo conciso tutti i pattern di riferimento impiegati per ciascun punto, insieme ai risultati corrispondenti nell'immagine d'esempio.

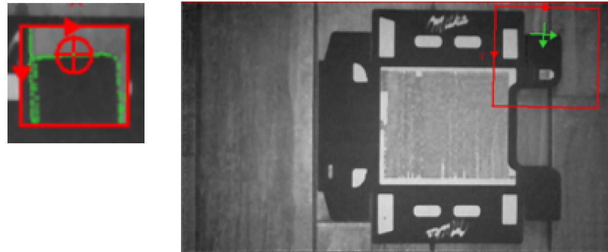


Figura 34: Secondo punto di presa

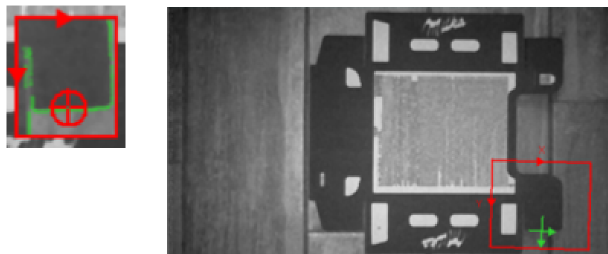


Figura 35: Terzo punto di presa

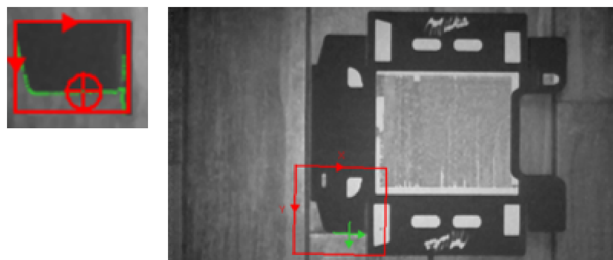


Figura 36: Quarto punto di presa

Dopo aver individuato i punti di interesse, si procede al calcolo delle coordinate del centro del cartone, utilizzando le seguenti formule. $x = ((C16 - C7)/2 + C7 + (C13 - C10)/2 + C10)/2$ e $y = ((D10 - D7)/2 + D7 + (D13 - D16)/2 + D16)/2$. È importante notare che queste formule acquisiscono i dati utilizzando i riferimenti alle celle.

5.2 Algoritmi AI-based

L'utilizzo del *deep learning* potrebbe fornire una soluzione alternativa, in grado di rilevare accuratamente il posizionamento della scatola anche in presenza di lievi disallineamenti. L'obiettivo degli algoritmi *AI-based* consiste nel rilevare la posizione della scatola trovando il *bounding box* e, attraverso l'uso delle coordinate di quest'ultimo, andare a calcolare la coordinata del centro da fornire al robot per la presa.

Si allena una rete neurale per ogni tipologia di scatola poichè il software con cui si andrà a interfacciare per realizzare la presa del robot sa già a priori quale tipologia di cartone gli sarà sottoposta. Quindi, per ridurre al massimo gli errori, non si è ritenuto necessario creare un'unica rete neurale che andasse a riconoscere la tipologia di scatola.

5.2.1 YOLOv8

Per l'allenamento è stata scelta una rete YOLOv8, ovvero l'ultima versione della rete disponibile. YOLOv8 è un miglioramento rispetto alla versione precedente di YOLO, che incrementa ulteriormente le prestazioni, rendendo il modello veloce, preciso e facile da usare. YOLOv8 ha introdotto un'architettura all'avanguardia che migliora notevolmente l'estrazione delle caratteristiche e le prestazioni di rilevazione degli oggetti. Inoltre, YOLOv8 si distingue per l'adozione di un approccio innovativo *anchor-free* che contribuisce a ottenere risultati più precisi e efficienti rispetto alle metodologie basate su *anchor*. Uno degli aspetti fondamentali di YOLOv8 è la sua capacità di bilanciare il trade-off tra la precisione e la velocità di rilevazione.

L'allenamento della rete YOLOv8 è stato fatto in Python tramite la libreria ultralytics [23] che permette di allenare la rete su un dataset customizzato. Il dataset customizzato è stato creato a partire da delle acquisizioni fatte sia con la camera SensoPart che con la camera Cognex. Quest'approccio ha consentito di creare un dataset diversificato e ampio, mirando a migliorare i risultati dell'addestramento. Inoltre, sono state create artificialmente ulteriori immagini per aumentare la dimensione del dataset ed ottenere risultati migliori. Per far ciò è stata usata la tecnica del *data augmentation* [24], con un approccio non supervisionato ovvero che non fa riferimento alle *labels* delle immagini. In particolare, si vanno a fare delle trasformazioni indipendenti dalla categoria dell'immagine andando così a creare nuove immagini che vanno a comporre il dataset. Le trasformazioni frequentemente impiegate includono: la riflessione dell'immagine lungo gli assi orizzontale o verticale, la rotazione dell'immagine verso un'orientazione casuale, il ritaglio di porzioni specifiche dall'immagine originale e lo spostamento dell'immagine lungo l'asse.

Per far ciò è stato sviluppato lo script *image_ augmentation.py* che contiene la funzione in cui sono riportate le trasformazioni da applicare alle immagini di partenza. Questa funzione viene applicata sia al dataset di addestramento che a quello di validazione.

```
def apply_image_augmentation(input_directory, output_directory):
    # Carica il dataset di immagini
    images = []
    images_path = glob.glob(os.path.join(input_directory, "*.bmp"))
    for img_path in images_path:
        img = cv2.imread(img_path)
        images.append(img)

    # Image Augmentation (applica modifiche al dataset di partenza)
    augmentation = iaa.Sequential([
        iaa.Fliplr(0.5),
        iaa.Flipud(0.5),
        iaa.Multiply((0.8, 1.2)),
        iaa.LinearContrast((0.6, 1.4)),
        iaa.Sometimes(0.5, iaa.GaussianBlur((0.0, 3.0))
        )
    ])

    # Salva tutte le immagini modificate
    os.makedirs(output_directory, exist_ok=True)

    for i, image in enumerate(images):
        augmented_images = augmentation(images=[image])
        augmented_image = augmented_images[0]
        output_path = os.path.join(output_directory, f"augmented_{i
    }qs.bmp")
        cv2.imwrite(output_path, augmented_image)

    print("Tutte le immagini sono state salvate")
```

Listato 1: Funzione di img_augmentation.py per aumentare il dataset

Per allenare la rete su un dataset customizzato è importante strutturare in modo corretto le cartelle contenenti il dataset.

```
data  --images  --train
                 --val
       --labels  --train
                  --val
```

Il dataset è suddiviso in due parti: il dataset di allenamento (*train*) è utilizzato per addestrare il modello di rete neurale, il dataset di validazione (*val*) è uti-

lizzato per valutare le prestazioni del modello addestrato. Per ognuno di essi devono essere presenti sia le immagini che i rispettivi file di annotazione.

Per la creazione dei file di annotazione è stato utilizzato il software Labelme con la funzione *Create Rectangle* che ha permesso di annotare le immagini andando a disegnare i *bounding box* di interesse per ogni immagine. Per ciascuna di queste immagini il programma realizza dei file JSON con la seguente struttura.

```
{
  "version": "5.3.1",
  "flags": {},
  "shapes": [
    {
      "label": "1",
      "points": [
        [
          x_point1
          y_point1
        ],
        [
          x_point2
          y_point2
        ]
      ],
      "group_id": null,
      "description": "",
      "shape_type": "rectangle",
      "flags": {}
    }
  ],
  "imagePath": "..\\path_immagine",
  "imageData":
  "imageHeight": altezza_immagine,
  "imageWidth": larghezza_immagine
}
}
```

Questo file JSON viene poi trasformato, tramite lo script creato appositamente, in un file txt nel formato richiesto dall'algoritmo YOLOv8 per l'*object detection*. Lo script che si occupa di questa trasformazione è *jsontotxt.py* che crea dei file txt nel formato: <class_id> <center_x> <center_y> <width>

<height>. Contiene una funzione che, per ciascun file JSON, estrae le informazioni di interesse e crea il file di testo nel formato YOLO corrispondente. Questa funzione viene applicata sia alle annotazioni del dataset di addestramento che a quelle di validazione.

```
def convert_json_to_txt(input_directory, output_directory):
    # Controlla se la directory di output esiste, altrimenti la crea
    if not os.path.exists(output_directory):
        os.makedirs(output_directory)

    # Legge tutti i file nella directory di input
    for filename in os.listdir(input_directory):
        if filename.endswith(".json"):
            input_json_path = os.path.join(input_directory, filename)
            output_txt_path = os.path.join(output_directory, os.path.splitext(filename)[0] + ".txt")

            # Apre i file json nella directory di input
            with open(input_json_path, 'r') as json_file:
                data = json.load(json_file)

            # Prende le informazioni di interesse dai file json
            #image_path = data["imagePath"]
            image_width = data["imageWidth"]
            image_height = data["imageHeight"]
            shapes = data["shapes"]

            # Apre il file di output
            with open(output_txt_path, 'w') as output_file:
                for shape in shapes:
                    label = shape["label"]
                    points = shape["points"]
                    x1, y1 = points[0]
                    x2, y2 = points[1]

                    # Trova le coordinate nel formato YOLO
                    x_center = (x1 + x2) / (2 * image_width)
                    y_center = (y1 + y2) / (2 * image_height)
                    width = (x2 - x1) / image_width
                    height = (y2 - y1) / image_height

                    # Scrive sul file txt
                    output_file.write(f"0 {x_center:.6f} {y_center:.6f} {width:.6f} {height:.6f}\n")

    print("Tutti i file sono stati convertiti")
```

Listato 2: Funzione di jsontotxt.py per la creazione dei file nel formato YOLO

Per l'allenamento sono state utilizzate 100 epoche che hanno permesso di ottenere buoni risultati e un tempo di allenamento abbastanza contenuto. Inoltre, considerando anche la grandezza del dataset che è di circa 100 immagini, ha permesso di evitare l'*overfitting*. Si è scelto di mantenere gli iperparametri di default in quanto si sono dimostrati efficaci nel raggiungere risultati soddisfacenti nella fase di allenamento della rete.

Per addestrare la rete è stato creato lo script *main.py* che permette l'allenamento sulla base alle informazioni contenute nel file di configurazione *config.yaml*, che include i percorsi di interesse relativi al dataset e la definizione delle classi. L'allenamento produce i file *last.pt* e *best.pt* che rappresentano rispettivamente i pesi calcolati con l'ultima epoca e i pesi migliori ottenuti.

```python
from ultralytics import YOLO

model = YOLO("yolov8n.yaml")  # Costruisce un nuovo modello

model.train(data="config.yaml", epochs=100)  # Allena il modello
```

Listato 3: main.py

```yaml

path: path_to\data  # Il percorso per la cartella che contiene le immagini e le annotazioni
train: images\train  # Il percorso per il dataset di train, relativo a path
val: images\val  # Il percorso per il dataset di validation, relativo a path

# Classi
names:
  0: cartoni
```

Listato 4: config.yaml

La durata dell'addestramento per tutte le reti è stata approssimativamente di un'ora e mezza.

Di seguito vengono riportati i grafici relativi ai risultati dell'allenamento.

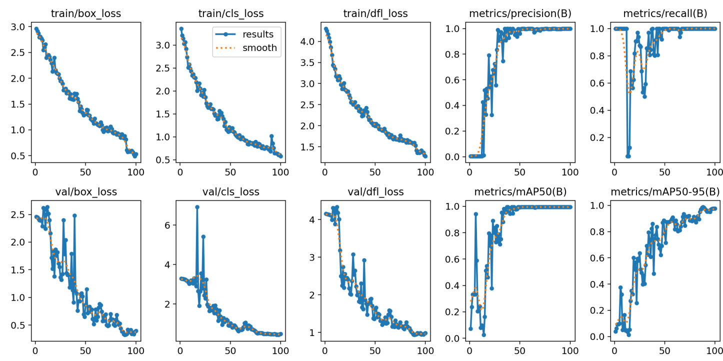


Figura 37: Grafici risultati allenamento YOLOv8 per primo cartone

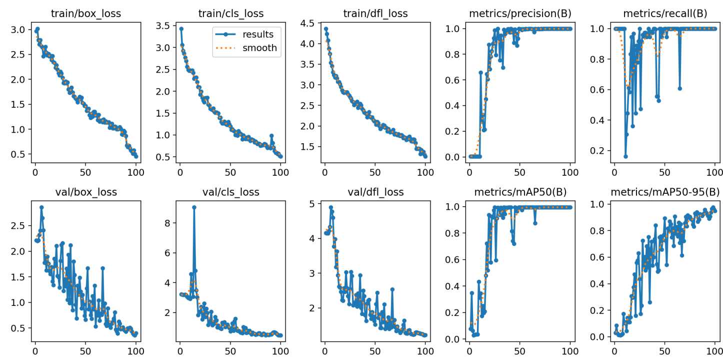


Figura 38: Grafici risultati allenamento YOLOv8 per secondo cartone

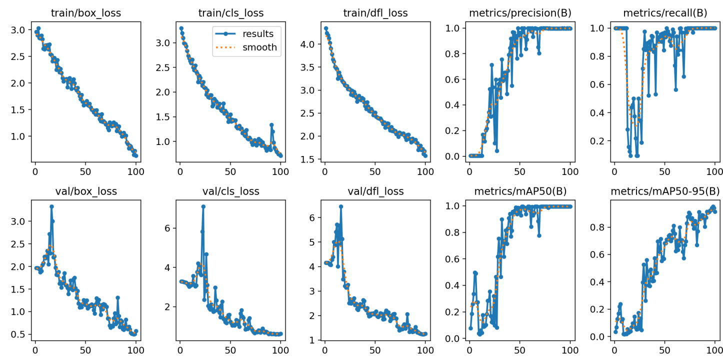


Figura 39: Grafici risultati allenamento YOLOv8 per terzo cartone

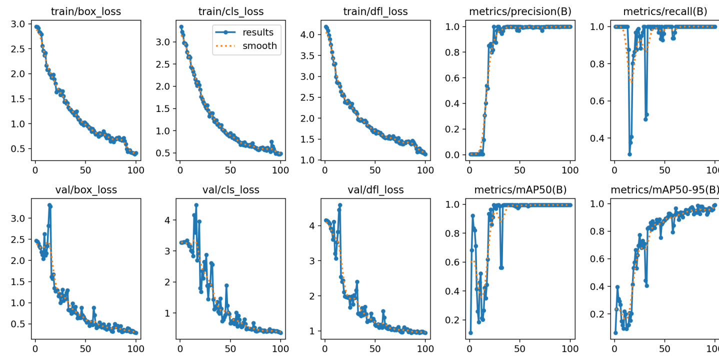


Figura 40: Grafici risultati allenamento YOLOv8 per quarto cartone

La libreria fornisce di default i grafici dei risultati ottenuti dall'addestramento salvandoli nella cartella ...\runs\detect\train. Nell'analisi delle prestazioni dell'allenamento della rete, l'attenzione è focalizzata sulla *loss* di validazione, sull'*mAP50* e sull'*mAP50-95*.

In particolare, ci si concentra sulla *loss* del rilevamento dei *bounding box* perchè ciò che interessa maggiormente nell'applicazione è la precisione nella determinazione della posizione dei cartoni. La *box_ loss* di validazione ha raggiunto valori di circa 0,5 dopo 100 epoche per tutte le categorie di scatole considerate. Questo risultato è soddisfacente, considerando la variazione nella creazione dei *bounding box* dovuta agli scostamenti delle scatole che compongono un blocco.

L'*mAP50* e l'*mAP50-95*, parametri rappresentativi della precisione complessiva del modello, si sono entrambi avvicinati al valore massimo di 1 per ogni tipologia di cartone. Questi risultati dimostrano che l'addestramento delle reti ha condotto a un livello significativo di precisione nella predizione, confermando l'efficacia dell'allenamento.

5.2.2 RetinaNet

Come alternativa alla rete YOLOv8 si è deciso di utilizzare la rete RetinaNet sviluppando un confronto tra le due per quanto riguarda accuratezza e velocità. Per allenare la nuova rete analizzata è stata utilizzata una libreria Detectron2 che permette di allenare reti neurali di diverso tipo preallenate su dataset molto grandi [25].

Detectron2 [26] è una libreria del team di ricerca *AI* di Facebook che rende disponibili gli algoritmi di *detection* e *segmentation* dello stato dell'arte attuale. Supporta molti dei progetti di ricerca di *computer vision* di Facebook ed è basato sulla libreria PyTorch.

Tramite questa libreria si è deciso di allenare una rete RetinaNet (R101) che è un'ottima alternativa alla YOLOv8, sia considerando le prestazioni che la

velocità. Per allenare queste nuove reti sono state utilizzate le stesse immagini utilizzate per allenare le reti YOLO (comprese quelle create tramite *augmentation*) e gli stessi file di annotazione. In questo caso varia però la struttura del percorso contenente i file relativi al dataset.

```
data  --train  --anns
               --imgs
       --val  --anns
               --imgs
```

Inoltre, è stato creato un ulteriore file che è *class.names* che contiene i nomi delle classi che si vogliono rilevare con la rete.

Per riutilizzare le annotazioni create per YOLO è stato sviluppato lo script *utilities.py* che contiene la funzione *get_ dicts* che legge le annotazioni del dataset nel formato YOLO e crea una lista di dizionari contenenti le informazioni relative ad ogni immagine.

```
1  # Legge le annotazioni in formato YOLO del dataset e crea una lista
       di dizionari contenenti informazioni per ogni immagine
2  def get_dicts(img_dir, ann_dir):
3
4      dataset_dicts = []
5      for idx, file in enumerate(os.listdir(ann_dir)):
6
7          record = {}
8
9          filename = os.path.join(img_dir, file[:-4] + '.bmp')
10         height, width = cv2.imread(filename).shape[:2]
11
12         record["file_name"] = filename
13         record["image_id"] = idx
14         record["height"] = height
15         record["width"] = width
16
17         objs = []
18         with open(os.path.join(ann_dir, file)) as r:
19             lines = [l[:-1] for l in r.readlines()]
20
21         for _, line in enumerate(lines):
22             if len(line) > 2:
23                 label, cx, cy, w_, h_ = line.split(' ')
24
25                 obj = {
26                     "bbox": [int((float(cx) - (float(w_) / 2)) *
    width),
27                              int((float(cy) - (float(h_) / 2)) *
    height),
```

```
28                              int(float(w_) * width),
29                              int(float(h_) * height)],
30                      "bbox_mode": BoxMode.XYWH_ABS,
31                      "category_id": int(label),
32                  }
33
34                  objs.append(obj)
35          record["annotations"] = objs
36          dataset_dicts.append(record)
37      return dataset_dicts
```

Listato 5: Funzione get_dicts contenuta in utilities.py

Sempre in *utilities.py* è stata creata la funzione *register_ datasets* che registra i dataset di addestramento e di validazione e restituisce il numero di classi presenti nel dataset.

```
1  # Registra il set di allenamento e validaione e ritorna il numero
       di classi
2  def register_datasets(root_dir, class_list_file):
3
4      with open(class_list_file, 'r') as reader:
5          classes_ = [l[:-1] for l in reader.readlines()]
6
7      for d in ['train', 'val']:
8          DatasetCatalog.register(d, lambda d=d: get_dicts(os.path.
       join(root_dir, d, 'imgs'), os.path.join(root_dir, d, 'anns')))
9          MetadataCatalog.get(d).set(thing_classes=classes_)
10
11     return len(classes_)
```

Listato 6: Funzione register_datasets contenuta in utilities.py

Per la fase di addestramento [27] della rete neurale, sono stati impostati gli iperparametri al fine di ottenere risultati ottimali. In particolare, il *batch* che indica il numero di campioni di addestramento impiegati in ciascuna iterazione è stato impostato pari a 4. Il *learning rate* ovvero la velocità con cui il modello aggiorna i pesi durante il processo di apprendimento è stato fissato a 0.00025. Il numero massimo di iterazioni è stato scelto pari a 60 al fine di garantire una copertura adeguata del dataset e un apprendimento sufficientemente approfondito. Nella prima parte dello script vengono configurati gli iperparametri per l'addestramento della rete.

```
1  # Configurazione dell'allenamento della rete
2  cfg = get_cfg()
3  # Si usa un modello pre-allenato sui cui si va ad allenare il
       modello custom https://github.com/facebookresearch/detectron2/
       blob/main/MODEL_ZOO.md
4  cfg.merge_from_file(model_zoo.get_config_file('COCO-Detection/
       faster_rcnn_R_101_FPN_3x.yaml'))
```

```
5
6 # Impostare i dataset da utilizzare
7 cfg.DATASETS.TRAIN = ("train",) # Posizione dei dati di allenamento
8 cfg.DATASETS.VAL = ("val",) # Posizione dei dati di validazione
9 cfg.DATASETS.TEST = ()
10
11 # Imposta il dispositivo su cui allenare
12 cfg.MODEL.DEVICE = 'cpu'
13
14 # Imposta il numero processi per il caricamento dei dati
15 cfg.DATALOADER.NUM_WORKERS = 2
16
17 # Imposta i pesi del modello sul modello pre-allenato
18 cfg.MODEL.WEIGHTS = model_zoo.get_checkpoint_url('COCO-Detection/
     faster_rcnn_R_101_FPN_3x.yaml')
19
20 # Imposta il numero di campioni di addestramento che vengono
     utilizzati in ogni iterazione
21 cfg.SOLVER.IMS_PER_BATCH = 4
22
23 # Imposta il periodo in cui vengono salvati i pesi come checkpoint
24 cfg.SOLVER.CHECKPOINT_PERIOD = 2
25
26 # Imposta il learning rate
27 cfg.SOLVER.BASE_LR = 0.00025
28
29 # Imposta il numero massimo di iterazioni per l'allenamento
30 cfg.SOLVER.MAX_ITER = 60
31
32 # Imposta lo Scheduler del learning rate ad una lista vuota, che
     significa che la velocita' di apprendimento non verra' ridotta
33 cfg.SOLVER.STEPS = []
34
35 # Imposta la dimensione del batch utilizzata durante l'
     addestramento della parte del modello responsabile delle
     regioni di interesse
36 cfg.MODEL.ROI_HEADS.BATCH_SIZE_PER_IMAGE = 128
37
38 # Imposta il numero di classi
39 cfg.MODEL.ROI_HEADS.NUM_CLASSES = register_datasets('./data', './
     class.names')
40
41 # Imposta la directory di output
42 cfg.OUTPUT_DIR = './output'
43
44 # Create un oggetto per la loss del set di validazione, di default
     calcola solo la loss sul set di allenamento
45 val_loss = ValidationLoss(cfg)
```

Listato 7: Configurazione parametri in train.py per allenamento di RetinaNet

All'interno del *main* viene aggiunto come valore da calcolare la *loss* della validazione. Successivamente, si procede con l'addestramento effettivo della rete. Alla fine, si calcolano le metriche di *Average Precision* (*AP*) e *Average Recall* (*AR*), che forniscono ulteriori indicazioni sulla qualità dell'addestramento. Dall'allenamento si ottengono i file dei pesi calcolati per ogni checkpoint e il file con i pesi ottenuti con il modello finale allenato, entrambi con estensione *.pth*.

```
if __name__ == "__main__":
    os.makedirs(cfg.OUTPUT_DIR, exist_ok=True)
    trainer = DefaultTrainer(cfg)
    # Registra l'oggetto per la loss di validazione come un hook
    trainer.register_hooks([val_loss])
    trainer._hooks = trainer._hooks[:-2] + trainer._hooks
    [-2:][::-1]
    trainer.resume_or_load(resume=False)
    # Parte l'allenamento
    trainer.train()

    # Calcola l'AP sui dati di valutazione
    evaluator = COCOEvaluator("val", ("bbox",), False, output_dir=
    cfg.OUTPUT_DIR)
    val_loader = build_detection_test_loader(cfg, "val")
    inference_on_dataset(trainer.model, val_loader, evaluator)

    # Stampa l'mAP
    metrics = evaluator._results
    print(f"mAP: {metrics['bbox']['AP']}")
```

Listato 8: Addestramento RetinaNet in train.py con calcolo metriche d'interesse

La libreria utilizzata per l'addestramento della rete fornisce informazioni sulla *loss* solamente ogni 20 iterazioni. Tali valori sono registrati nel file *metrics.json*, salvato nella directory dei risultati. È importante sottolineare che la libreria utilizzata non include il calcolo della *loss* sul dataset di validazione, né la valutazione dell'*AP* e dell'*AR*. Queste metriche sono state implementate separatamente: il calcolo di *AP* e *AR* è riportato nello script *train.py* mentre, per il calcolo della *loss* sul set di validazione è stato sviluppato lo script *loss.py* [28].

```
   # Calcolo la loss di validazione dopo ogni step di allenamento
    def after_step(self):
        data = next(self._loader)
        with torch.no_grad():

            loss_dict = self.trainer.model(data)

            losses = sum(loss_dict.values())
            assert torch.isfinite(losses).all(), loss_dict

            loss_dict_reduced = {"val_" + k: v.item() for k, v in
```

```
12                                          comm.reduce_dict(loss_dict).items
    ()}
13
14              losses_reduced = sum(loss for loss in loss_dict_reduced
    .values())
15
16              # Salva la loss di validazione
17              if comm.is_main_process():
18                  self.trainer.storage.put_scalars(total_val_loss=
    losses_reduced,
19                                                                  **
    loss_dict_reduced)
```

Listato 9: Funzione di loss.py per calcolo loss su set di validazione

La durata dell'addestramento per tutte le reti è stata approssimativamente di un'ora e mezza.

Di seguito, sono presentati i grafici derivanti dall'addestramento della rete RetinaNet.

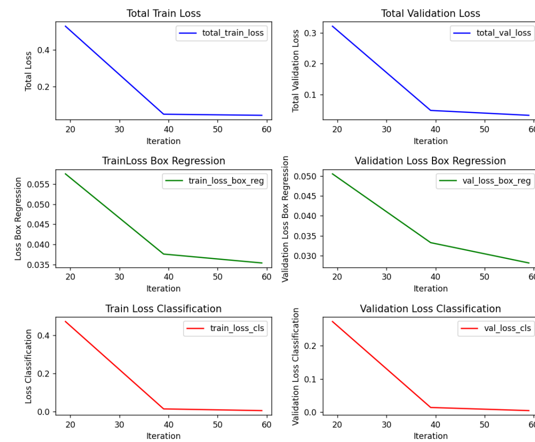


Figura 41: Grafici *loss* allenamento primo cartone

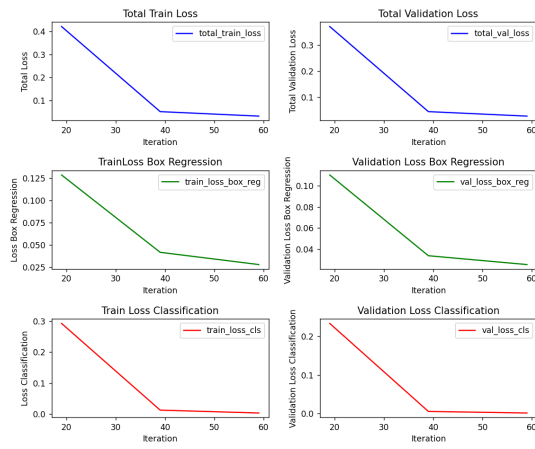


Figura 42: Grafici *loss* allenamento secondo cartone

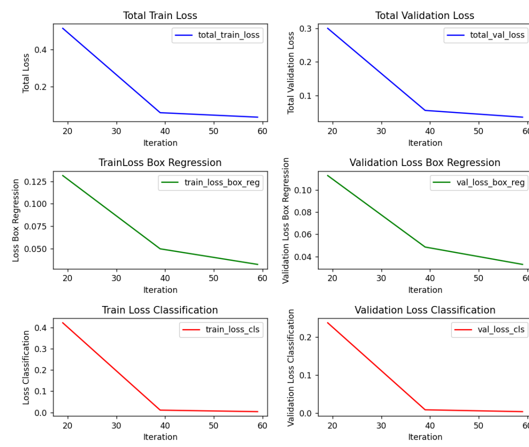


Figura 43: Grafici *loss* allenamento terzo cartone

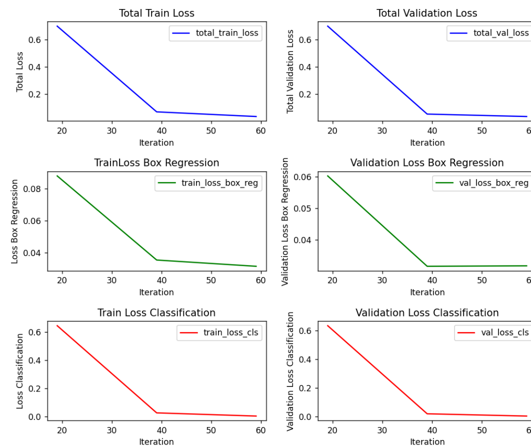


Figura 44: Grafici *loss* allenamento quarto cartone

Analogamente alla rete YOLO si concentra l'analisi sulla *loss* sul dataset di validazione, in particolare sulla *loss* sulla regressione delle *bounding box*. Nella fase di validazione si osservano valori estremamente bassi per la *loss* dei *bounding box* per ogni cartone, indicando un elevato livello di precisione. I grafici soprariportati sono generati dallo script *plot.py*.

```
1  # Estrae le informazioni di interesse
2  iterations = [entry["iteration"] for entry in data]
3  loss_box_reg = [entry["loss_box_reg"] for entry in data]
4  loss_cls = [entry["loss_cls"] for entry in data]
5  total_loss = [entry["total_loss"] for entry in data]
6  total_val_loss = [entry["total_val_loss"] for entry in data]
7  val_loss_box_reg = [entry["val_loss_box_reg"] for entry in data]
8  val_loss_cls = [entry["val_loss_cls"] for entry in data]
9
10 # Plotting
11 fig, axs = plt.subplots(3, 2, figsize=(15, 12))
12
13 axs[0, 0].plot(iterations, total_loss, label='total_train_loss',
       color='blue')
14 axs[0, 0].set_title('Total Train Loss')
15 axs[0, 0].set_xlabel('Iteration')
16 axs[0, 0].set_ylabel('Total Loss')
17 axs[0, 0].legend()
18
19 axs[1, 0].plot(iterations, loss_box_reg, label='train_loss_box_reg'
       , color='green')
20 axs[1, 0].set_title('TrainLoss Box Regression')
21 axs[1, 0].set_xlabel('Iteration')
22 axs[1, 0].set_ylabel('Loss Box Regression')
23 axs[1, 0].legend()
24
```

```python
axs[2, 0].plot(iterations, loss_cls, label='train_loss_cls', color='red')
axs[2, 0].set_title('Train Loss Classification')
axs[2, 0].set_xlabel('Iteration')
axs[2, 0].set_ylabel('Loss Classification')
axs[2, 0].legend()

axs[0, 1].plot(iterations, total_val_loss, label='total_val_loss', color='blue')
axs[0, 1].set_title('Total Validation Loss')
axs[0, 1].set_xlabel('Iteration')
axs[0, 1].set_ylabel('Total Validation Loss')
axs[0, 1].legend()

axs[1, 1].plot(iterations, val_loss_box_reg, label='val_loss_box_reg', color='green')
axs[1, 1].set_title('Validation Loss Box Regression')
axs[1, 1].set_xlabel('Iteration')
axs[1, 1].set_ylabel('Validation Loss Box Regression')
axs[1, 1].legend()

axs[2, 1].plot(iterations, val_loss_cls, label='val_loss_cls', color='red')
axs[2, 1].set_title('Validation Loss Classification')
axs[2, 1].set_xlabel('Iteration')
axs[2, 1].set_ylabel('Validation Loss Classification')
axs[2, 1].legend()

plt.tight_layout()
plt.show()
```

Listato 10: Estrazione dati e generazione grafici in plot.py

Di seguito, sono riportati i risultati di AP e AR ottenuti.

Figura 45: Risultati AP e AR allenamento primo cartone

```
[11/07 15:29:37 d2.evaluation.fast_eval_api]: Evaluate annotation type *bbox*
[11/07 15:29:37 d2.evaluation.fast_eval_api]: COCOeval_opt.evaluate() finished in 0.02 seconds.
[11/07 15:29:37 d2.evaluation.fast_eval_api]: Accumulating evaluation results...
[11/07 15:29:37 d2.evaluation.fast_eval_api]: COCOeval_opt.accumulate() finished in 0.01 seconds.
 Average Precision  (AP) @[ IoU=0.50:0.95 | area=   all | maxDets=100 ] = 1.000
 Average Precision  (AP) @[ IoU=0.50      | area=   all | maxDets=100 ] = 1.000
 Average Precision  (AP) @[ IoU=0.75      | area=   all | maxDets=100 ] = 1.000
 Average Precision  (AP) @[ IoU=0.50:0.95 | area= small | maxDets=100 ] = -1.000
 Average Precision  (AP) @[ IoU=0.50:0.95 | area=medium | maxDets=100 ] = -1.000
 Average Precision  (AP) @[ IoU=0.50:0.95 | area= large | maxDets=100 ] = 1.000
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=  1 ] = 1.000
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets= 10 ] = 1.000
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=100 ] = 1.000
 Average Recall     (AR) @[ IoU=0.50:0.95 | area= small | maxDets=100 ] = -1.000
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=medium | maxDets=100 ] = -1.000
 Average Recall     (AR) @[ IoU=0.50:0.95 | area= large | maxDets=100 ] = 1.000
[11/07 15:29:37 d2.evaluation.coco_evaluation]: Evaluation results for bbox:
|   AP    |  AP50   |  AP75   |  APs  |  APm  |   APl   |
|:-------:|:-------:|:-------:|:-----:|:-----:|:-------:|
| 100.000 | 100.000 | 100.000 |  nan  |  nan  | 100.000 |
[11/07 15:29:37 d2.evaluation.coco_evaluation]: Some metrics cannot be computed and is shown as NaN.
[11/07 15:29:37 d2.evaluation.coco_evaluation]: Per-category bbox AP:
| category   | AP      | category   | AP    |
|:-----------|:--------|:-----------|:------|
| cartone2   | 100.000 |            | nan   |
mAP: 100.0
```

Figura 46: Risultati AP e AR allenamento secondo cartone

```
[11/07 17:20:04 d2.evaluation.fast_eval_api]: Evaluate annotation type *bbox*
[11/07 17:20:04 d2.evaluation.fast_eval_api]: COCOeval_opt.evaluate() finished in 0.02 seconds.
[11/07 17:20:04 d2.evaluation.fast_eval_api]: Accumulating evaluation results...
[11/07 17:20:04 d2.evaluation.fast_eval_api]: COCOeval_opt.accumulate() finished in 0.00 seconds.
 Average Precision  (AP) @[ IoU=0.50:0.95 | area=   all | maxDets=100 ] = 0.994
 Average Precision  (AP) @[ IoU=0.50      | area=   all | maxDets=100 ] = 1.000
 Average Precision  (AP) @[ IoU=0.75      | area=   all | maxDets=100 ] = 1.000
 Average Precision  (AP) @[ IoU=0.50:0.95 | area= small | maxDets=100 ] = -1.000
 Average Precision  (AP) @[ IoU=0.50:0.95 | area=medium | maxDets=100 ] = -1.000
 Average Precision  (AP) @[ IoU=0.50:0.95 | area= large | maxDets=100 ] = 0.994
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=  1 ] = 0.997
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets= 10 ] = 0.997
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=100 ] = 0.997
 Average Recall     (AR) @[ IoU=0.50:0.95 | area= small | maxDets=100 ] = -1.000
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=medium | maxDets=100 ] = -1.000
 Average Recall     (AR) @[ IoU=0.50:0.95 | area= large | maxDets=100 ] = 0.997
[11/07 17:20:04 d2.evaluation.coco_evaluation]: Evaluation results for bbox:
|   AP   |  AP50   |  AP75   |  APs  |  APm  |  APl   |
|:------:|:-------:|:-------:|:-----:|:-----:|:------:|
| 99.440 | 100.000 | 100.000 |  nan  |  nan  | 99.440 |
[11/07 17:20:04 d2.evaluation.coco_evaluation]: Some metrics cannot be computed and is shown as NaN.
[11/07 17:20:04 d2.evaluation.coco_evaluation]: Per-category bbox AP:
| category   | AP     | category   | AP    |
|:-----------|:-------|:-----------|:------|
| cartone3   | 99.440 |            | nan   |
mAP: 99.43997524752474
```

Figura 47: Risultati AP e AR allenamento terzo cartone

```
[11/08 10:51:50 d2.evaluation.fast_eval_api]: Evaluate annotation type *bbox*
[11/08 10:51:50 d2.evaluation.fast_eval_api]: COCOeval_opt.evaluate() finished in 0.01 seconds.
[11/08 10:51:50 d2.evaluation.fast_eval_api]: Accumulating evaluation results...
[11/08 10:51:50 d2.evaluation.fast_eval_api]: COCOeval_opt.accumulate() finished in 0.00 seconds.
 Average Precision  (AP) @[ IoU=0.50:0.95 | area=   all | maxDets=100 ] = 1.000
 Average Precision  (AP) @[ IoU=0.50      | area=   all | maxDets=100 ] = 1.000
 Average Precision  (AP) @[ IoU=0.75      | area=   all | maxDets=100 ] = 1.000
 Average Precision  (AP) @[ IoU=0.50:0.95 | area= small | maxDets=100 ] = -1.000
 Average Precision  (AP) @[ IoU=0.50:0.95 | area=medium | maxDets=100 ] = -1.000
 Average Precision  (AP) @[ IoU=0.50:0.95 | area= large | maxDets=100 ] = 1.000
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=  1 ] = 1.000
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets= 10 ] = 1.000
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=100 ] = 1.000
 Average Recall     (AR) @[ IoU=0.50:0.95 | area= small | maxDets=100 ] = -1.000
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=medium | maxDets=100 ] = -1.000
 Average Recall     (AR) @[ IoU=0.50:0.95 | area= large | maxDets=100 ] = 1.000
[11/08 10:51:50 d2.evaluation.coco_evaluation]: Evaluation results for bbox:
|   AP    |  AP50   |  AP75   |  APs  |  APm  |   APl   |
|:-------:|:-------:|:-------:|:-----:|:-----:|:-------:|
| 100.000 | 100.000 | 100.000 |  nan  |  nan  | 100.000 |
[11/08 10:51:50 d2.evaluation.coco_evaluation]: Some metrics cannot be computed and is shown as NaN.
[11/08 10:51:50 d2.evaluation.coco_evaluation]: Per-category bbox AP:
| category   | AP      | category   | AP    |
|:-----------|:--------|:-----------|:------|
| cartone4   | 100.000 |            | nan   |
mAP: 100.0
```

Figura 48: Risultati AP e AR allenamento quarto cartone

L'*Average Precision* e l'*Average Recall* calcolati per l'allenamento delle scatola sono pari a 1 o comunque molto vicini a 1. Per gli oggetti di piccole e medie dimensioni, che non sono mai stati rilevati il risultato è -1, ciò è dovuto al fatto che la scatola è sempre della stessa tipologia e quindi mantiene costante le sue dimensioni. In contesti reali ottenere valori di AP e AR molto vicini a 1 è quasi impossibile ma, nell'applicazione considerata, la ripetibilità delle acqui-

sizioni consente di tollerare l'assenza sia di falsi positivi che di falsi negativi. Le acquisizioni per questo tipo di applicazione vengono effettuate in un contesto il più possibile uniforme, riducendo al minimo la variabilità tra le immagini.

5.2.3 Confronto tra YOLOv8 e RetinaNet

Confrontando le metriche calcolate le prestazioni della rete RetinaNet risultano notevolmente superiori a quelle della rete YOLOv8, in termini di precisione nella rilevazione dei *bounding box*. Questa maggiore precisione è evidente anche nei confronti diretti delle due reti sulla stessa immagine. Per realizzare la previsione su immagini delle due reti sono stati generati due script *predict_yolo.py* per la predizione con la rete YOLOv8 e *predict_retinanet.py* per la predizione con la rete RetinaNet.

```
1  start_time = time.time()
2
3  # Trova il percorso dove c'e' il modello creato con l'allenamento
4  model_path = os.path.join('.', 'runs', 'detect', 'train2', 'weights
   ', 'best.pt')
5
6  # Carica il modello creato dall'allenamento
7  model = YOLO(model_path)
8
9  results = model(source="image_name.bmp", show = False, conf = 0.2,
   save= False)
10 end_time = time.time()  # Registra l'istante finale
11
12 tempo_di_esecuzione = end_time - start_time
13 print(f"Tempo di esecuzione: {tempo_di_esecuzione} secondi")
```

Listato 11: Codice in predict_yolo.py per predizione

```
1  start_time = time.time()
2
3  # Carica un file di configuratione
4  cfg = get_cfg()
5  cfg.merge_from_file(model_zoo.get_config_file('COCO-Detection/
   retinanet_R_101_FPN_3x.yaml'))
6  cfg.MODEL.WEIGHTS = 'path_to\model_final.pth'
7  cfg.MODEL.DEVICE = 'cpu'
8
9  # Crea un'istanza del predittore
10 predictor = DefaultPredictor(cfg)
11
12 # Carica l'immagine
13 image = cv2.imread("path_input_image")
14
15 output_image_path = "path_to_output"
16
```

```
# Fa la predizione sull'immagine
outputs = predictor(image)

threshold = 0.5

preds = outputs["instances"].pred_classes.tolist()
scores = outputs["instances"].scores.tolist()
bboxes = outputs["instances"].pred_boxes

end_time = time.time()  # Registra l'istante finale

tempo_di_esecuzione = end_time - start_time
print(f"Tempo di esecuzione: {tempo_di_esecuzione} secondi")
```

Listato 12: Codice in predict_retinanet.py per predizione

Alcuni esempi di rilevazioni sono riportati di seguito. Le immagini a sinistra rappresentano la predizione con YOLOv8 mentre le immagini a destra rappresentano la predizione con RetinaNet.

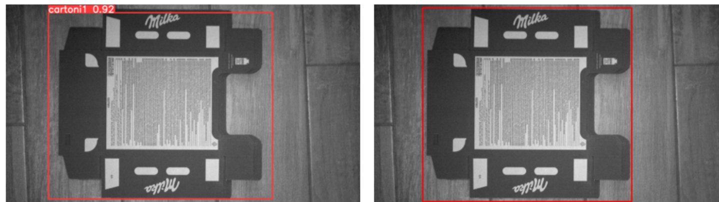


Figura 49: Confronto primo cartone

Figura 50: Confronto secondo cartone

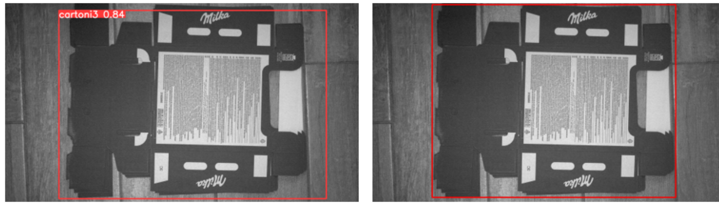


Figura 51: Confronto terzo cartone

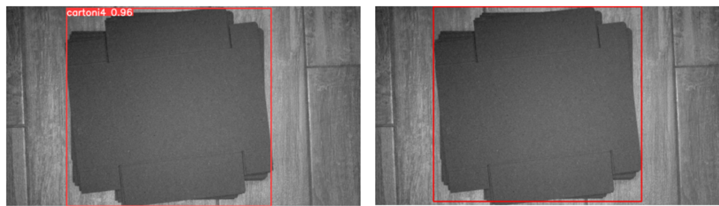


Figura 52: Confronto quarto cartone

Oltre alla precisione, un parametro cruciale da considerare è il tempo di esecuzione del programma per la predizione e per il calcolo del centro del *bounding box*. In media, la rete YOLOv8 presenta un tempo di esecuzione inferiore rispetto a RetinaNet. È importante notare che il tempo di esecuzione dipende anche dalle specifiche del computer su cui viene eseguito il programma. Tuttavia, nonostante il tempo di esecuzione leggermente superiore della RetinaNet, questo rimane comunque entro limiti accettabili che consentono al robot di operare in modo continuativo e senza interruzioni.

Per le prove nel setup sperimentale si è optato per l'utilizzo di RetinaNet anziché YOLOv8, questo per la maggiore precisione nella rilevazione della posizione offerta da RetinaNet.

6 Algoritmi su setup sperimentale

Il setup sperimentale è stato realizzato impiegando la telecamera SensoPart Visor Robotic fissata al soffitto. L'obiettivo principale consiste nel confrontare gli algoritmi *rule-based* con quelli *AI-based*. In particolare, si intende confrontare gli algoritmi sviluppati tramite il software Visor Vision Sensor con quelli derivati dalla rete RetinaNet. La scelta di utilizzare il software Visor Vision Sensor è motivata dalla costruzione del setup mentre l'adozione della rete RetinaNet è stata giustificata nel capitolo precedente.

Inizialmente sono stati adeguati gli algoritmi, sia di tipo *rule-based* che *AI-based*, alla nuova disposizione dei cartoni nel setup sperimentale. I cartoni sono impilati in modo da formare dei blocchi contenenti un certo numero di cartoni ognuno e, in ogni acquisizione, sono visibili sei blocchi. L'obiettivo principale è il riconoscimento non più di un singolo blocco, come precedentemente analizzato, bensì di sei blocchi di cartoni disposti seguendo una data configurazione.

6.1 I nuovi algoritmi rule-based per il setup sperimentale

6.1.1 Prima tipologia di cartoni

I sei blocchi sono disposti secondo la struttura illustrata nell'immagine seguente che rappresenta un esempio di acquisizione.

Figura 53: Posizionamento blocchi prima tipologia nel setup sperimentale

Inizialmente, sono stati aggiunti i filtri di *Erosion* e *Dilation* che verranno applicati a tutte le tipologie di cartoni analizzate. Successivamente, sono stati

utilizzati quattro *pattern matching* identificati come 1.1, 2.1, 3.1 e 4.1. Di ciascun pattern devono essere individuate tre istanze corrispondenti al numero di cartoni di ogni fila. I *pattern matching* sono risultati efficaci grazie alla netta differenza di colore tra i cartoni e il pallet. Questo contrasto è stato enfatizzato utilizzando il blu come canale di colore per i pattern di riferimento, contribuendo così a evidenziare la distinzione tra i cartoni e il pallet. Inoltre, è stata introdotta la possibilità di rotazione del riferimento in un intervallo compreso tra -10° e 10°, al fine di migliorare l'individuazione in presenza di eventuali disallineamenti. Infine, i risultati dei pattern sono stati disposti in ordine crescente rispetto alle coordinate x dei punti individuati.

Per calcolare il centro, avendo due soli punti, si è calcolata la media delle coordinate x e y. L'utilizzo di due punti permette di calcolare il centro tenendo conto di eventuali disallineamenti.

Figura 54: Posizionamento blocchi seconda tipologia nel setup sperimentale

6.1.2 Seconda tipologia di cartoni

I sei blocchi sono disposti seguendo la struttura dell'immagine, con la superficie stampata rivolta verso il pallet. Di seguito è riportato un esempio di acquisizione.

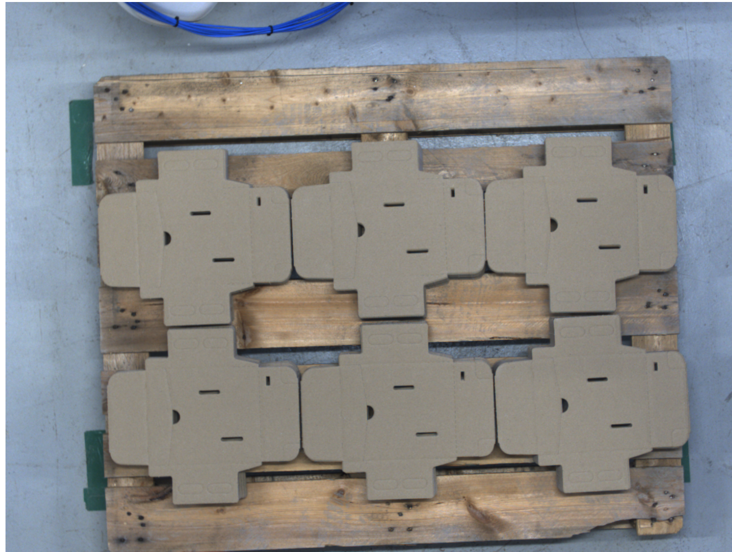


Figura 55: Posizionamento blocchi seconda tipologia nel setup sperimentale

La problematica in questo contesto è data dalla somiglianza di colore tra i cartoni e il pallet. Per facilitare il riconoscimento è stato introdotto un ulteriore filtro, oltre a *Erosion* e *Dilation*, chiamato *Mean*. A differenza delle altre tipologie di cartoni è stato impiegato un approccio basato su *Contour* poiché la limitata differenza di colore rende difficile l'analisi in scala di grigi del *Pattern Matching*.

Nel dettaglio, sono stati individuati i contorni delle aree evidenziate nelle figure come 1.1, 2.1, 3.1 e 4.1. Ogni contorno deve essere individuato tre volte all'interno dell'area di interesse pari al numero di scatole che compongono una singola fila.

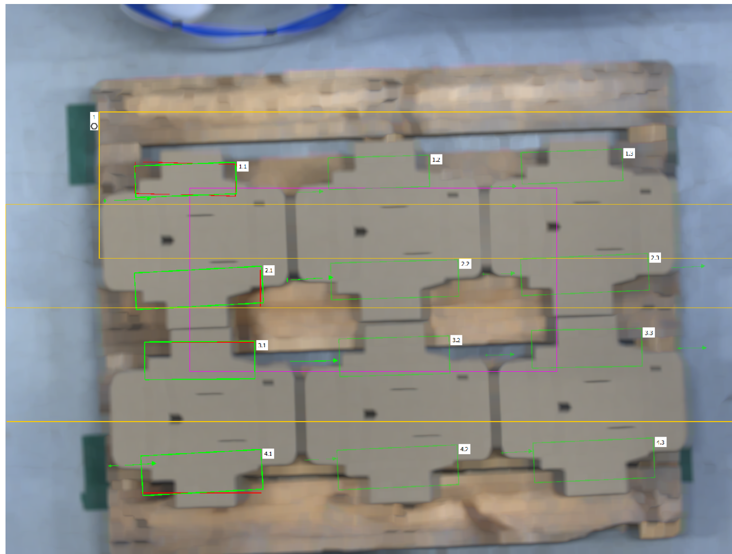


Figura 56: Algoritmo seconda tipologia su setup sperimentale

Per migliorare l'identificazione è stata introdotta la possibilità di rotazione di ogni riferimento nell'intervallo compreso tra -10° e 10°. Inoltre, è stata inclusa la possibilità di individuare riferimenti scalati con un fattore compreso tra 0,8 e 1,2. Questa caratteristica offre la flessibilità necessaria per individuare il riferimento anche in presenza di disallineamenti. I risultati dei contour sono stati disposti in ordine crescente rispetto alle coordinate x dei punti individuati.

Per determinare il centro si è calcolata la media delle coordinate x e y. L'utilizzo di questi due punti permette di individuare il centro tenendo conto di eventuali disallineamenti.

6.1.3 Terza tipologia di cartoni

I sei blocchi sono organizzati in due differenti configurazioni che si alternano sui vari livelli denominati *layer*. Di seguito sono riportate le disposizioni dei due *layer*.

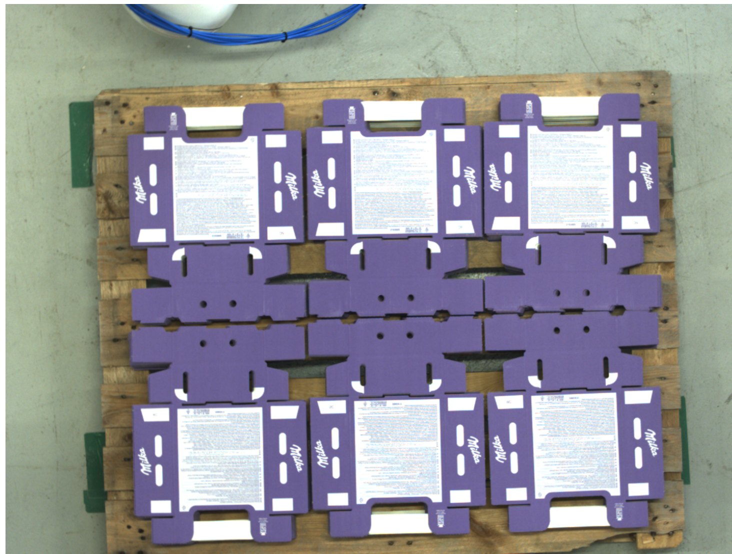


Figura 57: Posizionamento blocchi terza tipologia primo layer nel setup sperimentale

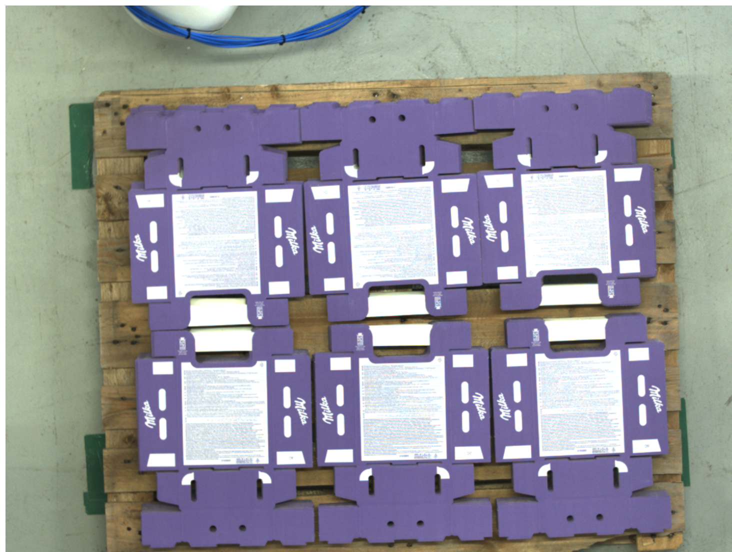


Figura 58: Posizionamento blocchi terza tipologia secondo layer nel setup sperimentale

Data la presenza di due *layer*, l'allineamento è stato escluso perchè non esiste un punto specifico sui cartoni che possa essere rilevato in entrambe le configurazioni.

Inizialmente, sono stati inclusi due algoritmi di *pattern matching* per identificare in quale dei due *layer* ci si trova. I pattern di riferimento sono stati

selezionati considerando la parte centrale dell'immagine, tenendo conto di tutte le scatole al fine di limitare l'errore dell'algoritmo.

Figura 59: Riferimento per discriminare primo *layer*

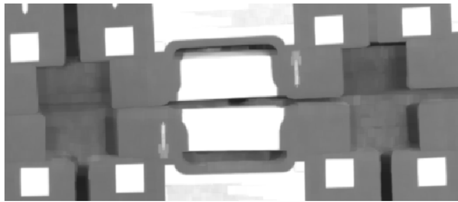


Figura 60: Riferimento per discriminare secondo *layer*

Per ciascun *layer* sono stati sviluppati quattro *pattern matching*, ciascuno configurato per riconoscere tre elementi nell'immagine. Questi *pattern matching* consentono di individuare la posizione di due angoli opposti per ogni blocco di cartoni così da poterne calcolare il centro. Inoltre, è stata introdotta la possibilità di rotazione del riferimento in un intervallo da -10° a 10° per gestire eventuali disallineamenti. I risultati sono stati disposti in ordine crescente rispetto alle coordinate x dei punti individuati.

In base ai risultati ottenuti dai pattern per discriminare il *layer*, il centro viene calcolato considerando i pattern del *layer* ottenuto. Per esempio, per determinare il centro del cartone in alto a sinistra viene utilizzata la seguente formula:

$centro_1 = if(D9; [(D1.PosX(1) + D2.PosX(1))/2; (D1.PosY(1)+$
D2.PosY(1))/2]; [(D5.PosX(1)+D6.PosX(1))/2; (D5.PosY(1)+D6.PosY(1))/2])

dove D9 rappresenta il risultato del *pattern matching* che riconosce il primo layer. Se ha successo il centro viene calcolato utilizzando i detector degli angoli del cartone relativi al *layer 1*, se fallisce vengono utilizzati i risultati dei detector del *layer 2*. Identificando due punti per ogni scatola, il centro si ottiene semplicemente calcolando la media tra questi due punti.

6.1.4 Quarta tipologia di cartoni

I sei blocchi sono disposti seguendo lo schema della figura seguente.

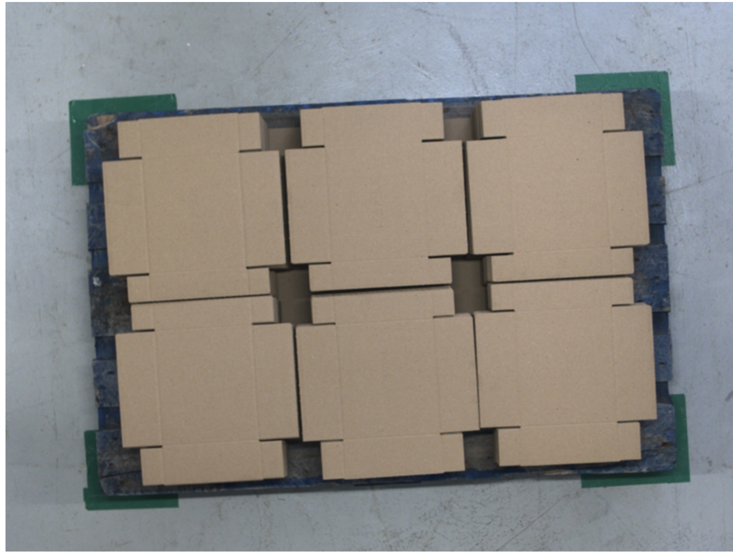


Figura 61: Posizionamento blocchi quarta tipologia nel setup sperimentale

Per l'algoritmo è stato adottato un approccio basato su *pattern matching*. Si è scelto di identificare due punti per ogni blocco di cartoni corrispondenti agli estremi di una delle due diagonali. Questi punti sono rappresentati dai riferimenti 1.1, 2.1, 3.1 e 4.1.

Per garantire una corretta identificazione in presenza di lievi disallineamenti è stato configurato un range di angoli compreso tra -6° e 6°.

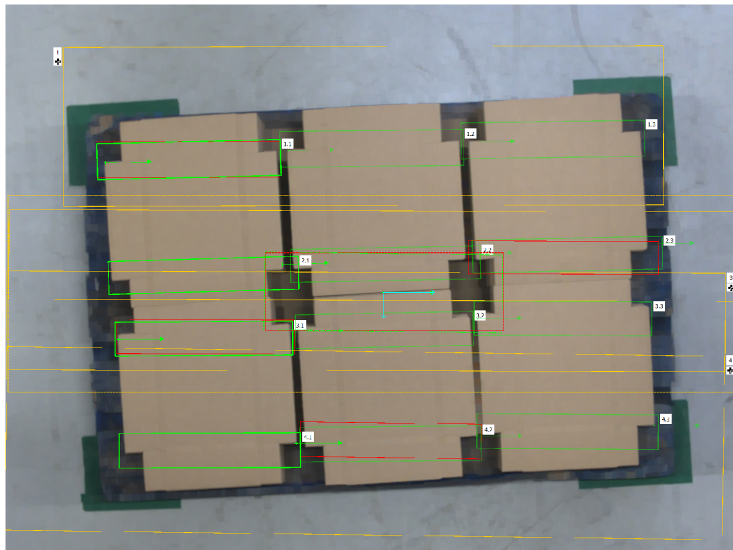


Figura 62: Algoritmo quarta tipologia su setup sperimentale

Per calcolare il centro, avendo due soli punti, si è calcolata la media delle coordinate x e y.

6.2 I nuovi algoritmi AI-based per il setup sperimentale

Per quanto riguarda gli algoritmi *AI-based* per individuare più blocchi all'interno di un'unica immagine sono stati eseguiti nuovi cicli di addestramento. Le immagini acquisite dal setup sperimentale sono state aggiunte al fine di migliorare l'addestramento di RetinaNet e facilitare l'identificazione dei sei blocchi di cartoni.

Il dataset utilizzato per la prima tipologia di cartoni consiste in 194 immagini per l'addestramento e 48 per la validazione. Per la seconda tipologia sono presenti 204 immagini nel set di addestramento e 52 nel set di validazione. La terza tipologia ha un totale di 182 immagini nel set di addestramento e 52 nel set di validazione. Infine, la quarta tipologia di cartoni è composta da 194 immagini nel set di addestramento e 56 nel set di validazione.

Gli addestramenti condotti con tutte le tipologie di cartoni, con l'introduzione di nuove immagini, hanno prodotto valori di AP superiori a 95 confermando l'efficacia dell'allenamento delle reti. È significativo notare che, per ciascuna categoria di cartone, i valori di *loss* relativi alla creazione dei *bounding box*, nonché per la rilevazione e l'identificazione, sia nel set di addestramento che in quello di validazione, si sono avvicinati a zero in tutti i casi. Questo indica un'elevata precisione nella generazione dei *bounding box* e nella capacità di rilevare e identificare correttamente le diverse tipologie di cartoni.

7 Confronto algoritmi rule-based e AI-based

7.1 Implementazione programmi per confronto

Per confrontare i risultati dei due algoritmi sono stati calcolati gli errori tra i centri ottenuti e i centri previsti. Per far ciò sono stati sviluppati appositi script in Python. Questi script consentono di salvare i centri calcolati utilizzando gli algoritmi di *deep learning* e i centri ideali all'interno di un file Excel. Inoltre, vengono inseriti all'interno del file Excel anche i centri derivati dagli algoritmi *rule-based*. Di seguito sono descritti i tre script creati.

Il primo script è *object_detector.py*, richiede tre parametri per essere eseguito: la cartella di input contenente le immagini da analizzare, la cartella di output dove verranno salvate le immagini con il *bounding box* disegnato e, infine, il nome del file Excel in cui verranno registrati i centri ottenuti tramite l'algoritmo *AI-based*. Per ogni immagine presente nella cartella di input lo script calcola il centro attraverso l'algoritmo *AI-based*. Una volta ottenuti i centri vengono riordinati in modo da seguire il seguente ordine.

```
1 3 5
2 4 6
```

Infine, salva tutti i valori nel file Excel.

```
1  def object_detector(input_folder, output_folder, excel_file):
2      # Carica un file di configuratione
3      cfg = get_cfg()
4      print(os.path)
5      cfg.merge_from_file(model_zoo.get_config_file('COCO-Detection/
       retinanet_R_101_FPN_3x.yaml'))
6      cfg.MODEL.WEIGHTS = os.getcwd() +'\output\model_final.pth'
7      cfg.MODEL.DEVICE = 'cpu'
8
9      # Crea un'istanza del predittore
10     predictor = DefaultPredictor(cfg)
11
12     # Creare il file Excel
13     file_excel_path = excel_file + ".xlsx"
14     file_excel = xlsxwriter.Workbook(file_excel_path)
15     foglio_excel = file_excel.add_worksheet()
16
17     # Inizializzare la riga corrente
18     current_row = 1
19
20     # Creare la cartella di output se non esiste
21     os.makedirs(output_folder, exist_ok=True)
22
23     # Itera attraverso tutte le immagini nella cartella di input
```

```
24     for filename in os.listdir(input_folder):
25         if filename.endswith(".bmp"):
26             # Carica l'immagine
27             image_path = os.path.join(input_folder, filename)
28             image = cv2.imread(image_path)
29
30             # Fa la predizione sull'immagine
31             outputs = predictor(image)
32
33             threshold = 0.8
34
35             preds = outputs["instances"].pred_classes.tolist()
36             scores = outputs["instances"].scores.tolist()
37             bboxes = outputs["instances"].pred_boxes
38             center = []
39             for j, bbox in enumerate(bboxes):
40                 bbox = bbox.tolist()
41                 score = scores[j]
42                 pred = preds[j]
43
44                 if score > threshold:
45                     x1, y1, x2, y2 = [int(i) for i in bbox]
46                     color = list(np.random.random(size=3) * 256)
47                     cv2.rectangle(image, (x1, y1), (x2, y2), color,
  1)
48
49                     x_center = int((x1 + x2) / 2)
50                     y_center = int((y1 + y2) / 2)
51                     center.append([x_center, y_center])
52
53             print(center)
54
55             # Ordina i cartoni
56             vettore_ordinato = sorted(center, key=lambda x: (x[0],
x[1]))
57             print(vettore_ordinato)
58             # Scrivere i valori nel foglio Excel solo se ci sono
centri per questa immagine
59             current_column = 12
60             if vettore_ordinato:
61                 # foglio_excel.write(current_row, current_column,
filename)
62                 for center in vettore_ordinato:
63                     foglio_excel.write(current_row, current_column
+ 1, center[0])
64                     foglio_excel.write(current_row, current_column
+ 2, center[1])
65                     current_column += 2
66                 current_row += 1
67
```

```
68          # Salva l'immagine con i risultati nella cartella di
    output
69          output_image_path = os.path.join(output_folder,
    filename.replace(".bmp", "_output.jpg"))
70          cv2.imwrite(output_image_path, image)
71
72  file_excel.close()
73
```

Listato 13: Funzione principale di object_detector.py per calcolo e ordinamento centro

Si prosegue con l'analisi di *errore_centro.py* in cui ogni immagine della cartella di input viene visualizzata in una finestra, fornendo un'interfaccia per selezionare i punti desiderati e ottenere i centri previsti. Infine, il programma apre il file Excel precedentemente creato e salva i centri previsti ottenuti.

```
1   def click_event(event, x, y, flags, params):
2       if event == cv2.EVENT_LBUTTONDOWN:
3           scale_factor = params['scale_factor']
4           original_x = int(x / scale_factor)
5           original_y = int(y / scale_factor)
6           z.append([original_x, original_y])
7
8           cv2.imshow('Scaled Image', scaled_img)
9
```

Listato 14: Funzione per gestire i clic del mouse nell'interfaccia in errore_centro.py

```
1       z = []
2
3       # Impostare il gestore del mouse per l'immagine scalata
4       cv2.setMouseCallback('Scaled Image', click_event, {'
    scale_factor': scale_factor})
5
6       # Attendere che venga premuto un tasto per chiudere l'
    immagine
7       cv2.waitKey(0)
8
9       # Calcolare i centri per gruppi di 4 punti consecutivi
10      centri = []
11      for i in range(0, len(z), 4):
12          x_centro = int(((z[i][0] + z[i + 1][0]) / 2 + (z[i +
    2][0] + z[i + 3][0]) / 2) / 2)
13          y_centro = int(((z[i][1] + z[i + 3][1]) / 2 + (z[i +
    1][1] + z[i + 2][1]) / 2) / 2)
14          centro = [x_centro, y_centro]
15          centri.append(centro)
16
```

```
17
```

Listato 15: Calcolo centro in errore_centro.py

In conclusione, viene creato *gui.py* script che consente la creazione di un'interfaccia grafica per selezionare: la cartella contenente le immagini da testare, la directory in cui memorizzare le immagini con il *bounding box* e il nome del file Excel da creare.

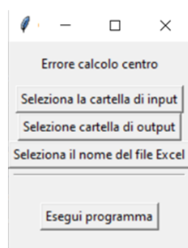


Figura 63: Interfaccia grafica

I parametri inseriti vengono poi passati alle funzioni da eseguire.

```
1        def execute_programs(self):
2            if hasattr(self, 'input_folder') and hasattr(self, '
    output_folder') and hasattr(self, 'excel_filename'):
3                # object_detector.py
4                object_detector_command = [
5                    'python',
6                    'object_detector.py',
7                    self.input_folder,
8                    self.output_folder,
9                    self.excel_filename
10               ]
11               subprocess.run(object_detector_command, check=True)
12
13               # errore_centro.py
14               errore_centro_command = [
15                   'python',
16                   'errore_centro.py',
17                   self.input_folder,
18                   self.excel_filename
19               ]
20               subprocess.run(errore_centro_command, check=True)
21
22               messagebox.showinfo("Info", "Il programma e' stato
    eseguito correttamente")
23           else:
24               messagebox.showwarning("Warning", "Inserisci tutti i
    parametri!")
```

```
25
```

Listato 16: Funzione per l'esecuzione degli script con i parametri selezionati in gui.py

7.2 Confronto tra gli algoritmi

La procedura per confrontare gli algoritmi prevede la creazione di set di dati di test, ognuno composto da 30 immagini corrispondenti a una tipologia di cartone. In particolare, le prime 20 immagini di ogni set rappresentano piccoli disallineamenti, mentre le ultime 10 rappresentano grandi disallineamenti. Per ciascuna immagine vengono calcolati i seguenti centri: i centri previsti calcolati tramite il programma spiegato precedentemente, i centri calcolati attraverso la predizione delle reti neurali addestrate e i centri calcolati mediante algoritmi *rule-based*. I valori ottenuti sono quindi salvati in un file Excel, uno per ogni tipologia di cartone, con la seguente struttura: tra le celle A2 e L31 sono presenti le coordinate dei 6 centri previsti; tra le celle N2 e Y31 sono riportati i centri calcolati con l'algoritmo *AI-based*; tra le celle N41 e Y70 sono indicati i centri calcolati con gli algoritmi *rule-based*.

Di seguito sono riportati i grafici che evidenziano il confronto tra i due algoritmi per ogni tipologia di cartoni. Sono presentati i due grafici che mostrano l'andamento dell'errore nel calcolo dei centri con i due algoritmi in relazione al numero di centri soggetti a tale errore. Questi grafici sono realizzati considerando intervalli di errore di grandezza pari a 0,5 pixel. Inoltre, è riportato il grafico che mostra la dispersione dei centri calcolati mediante entrambi gli algoritmi, normalizzati rispetto ai centri previsti. Sono riportati i valori di media, mediana e della deviazione standard. L'unità di misura dei risultati è il pixel (px).

Prima tipologia di cartoni

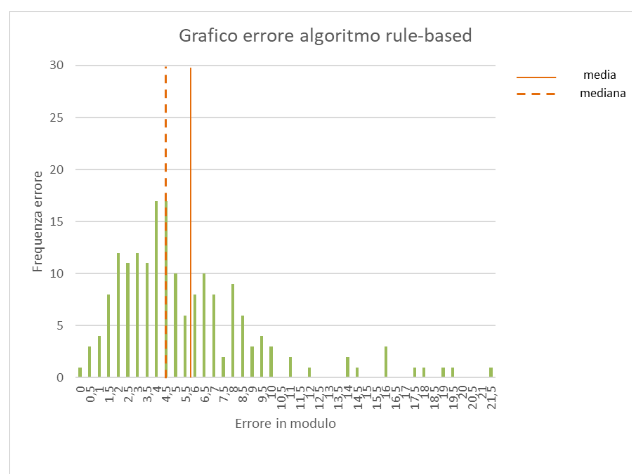


Figura 64: Grafico frequenza errore algoritmo rule-based primo cartone

La media è pari a 5,85 px, la mediana vale 4,5 px e la deviazione standard è di 3,89 px.

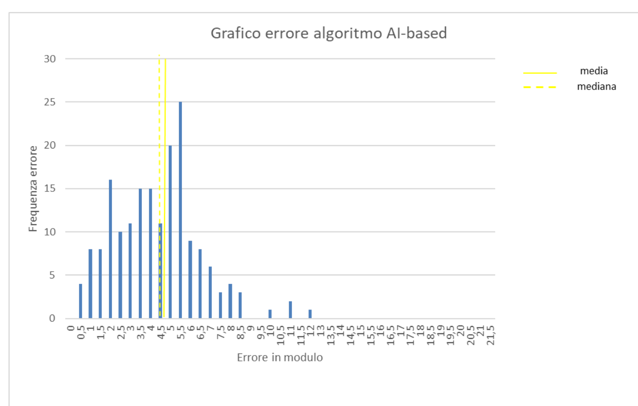


Figura 65: Grafico frequenza errore algoritmo AI-based primo cartone

La media è pari a 4,6 px, la mediana vale 4,5 px e la deviazione standard è di 2,14 px.

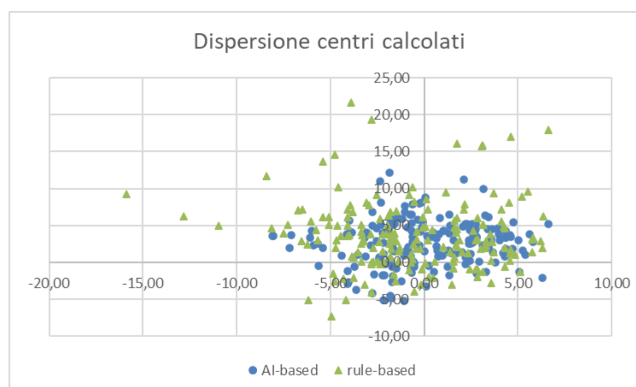


Figura 66: Grafico dispersione centri primo cartone

Seconda tipologia di cartoni

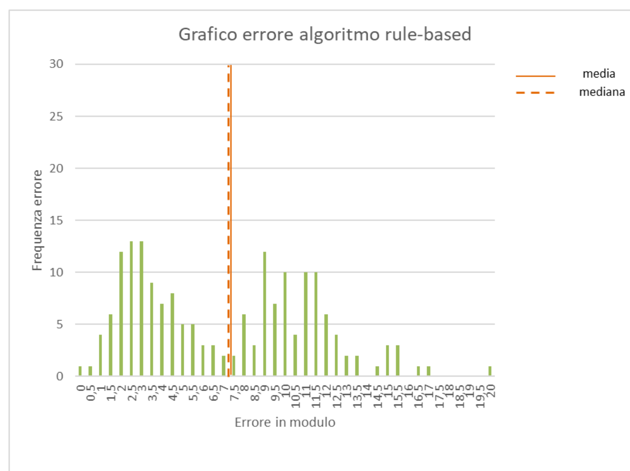


Figura 67: Grafico frequenza errore algoritmo rule-based secondo cartone

La media è pari a 7,32 px, la mediana vale 7,25 px e la deviazione standard è di 4,3 px.

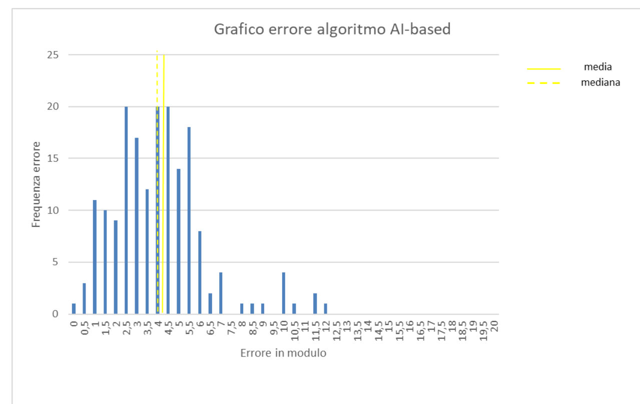


Figura 68: Grafico frequenza errore algoritmo AI-based secondo cartone

La media è pari a 4,27 px, la mediana vale 4 px e la deviazione standard è di 2,19 px.

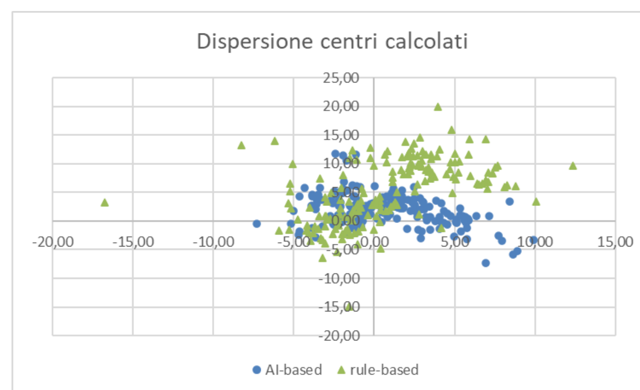


Figura 69: Grafico dispersione centri secondo cartone

Terza tipologia di cartoni

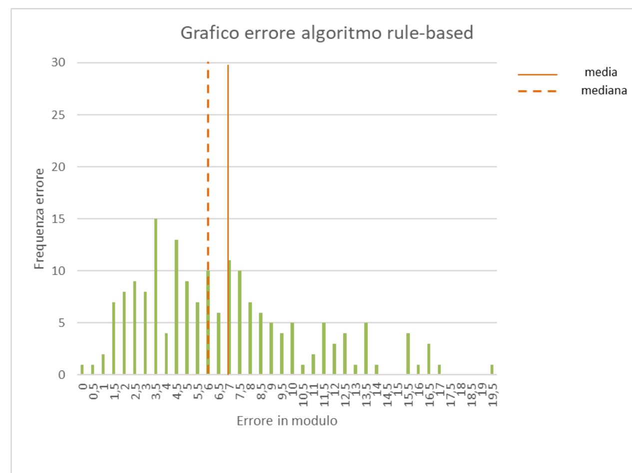


Figura 70: Grafico frequenza errore algoritmo rule-based terzo cartone

La media è pari a 6,98 px, la mediana vale 6 px e la deviazione standard è di 4,71 px.

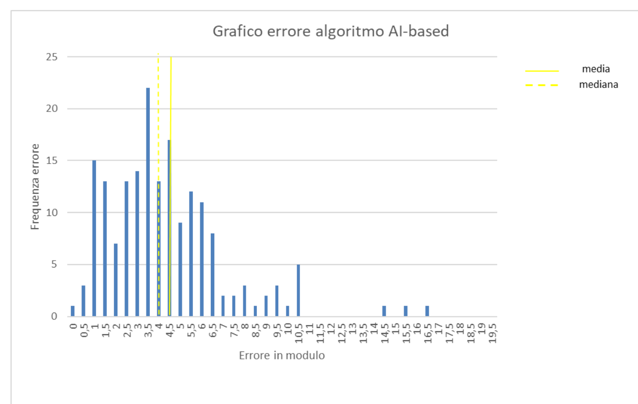


Figura 71: Grafico frequenza errore algoritmo AI-based terzo cartone

La media è pari a 4,55 px, la mediana vale 4 px e la deviazione standard è di 2,73 px.

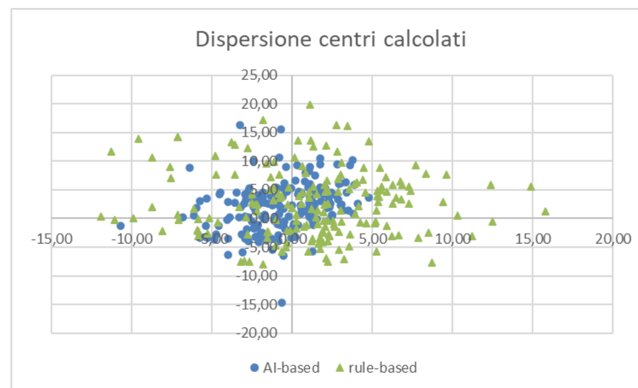


Figura 72: Grafico dispersione centri terzo cartone

Quarta tipologia di cartoni

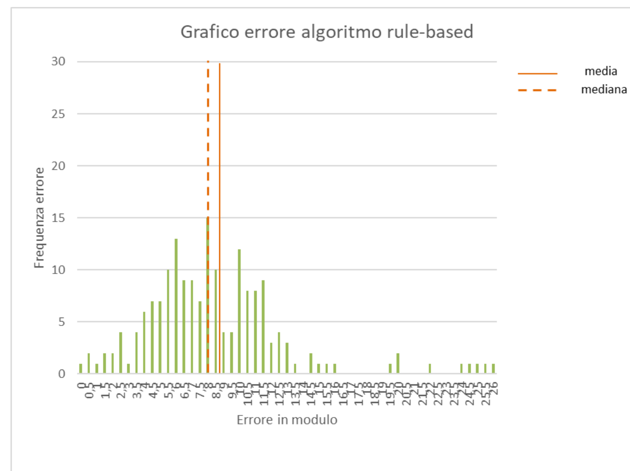


Figura 73: Grafico frequenza errore algoritmo rule-based quarto cartone

La media è pari a 8,73 px, la mediana vale 8 px e la deviazione standard è di 4,6 px.

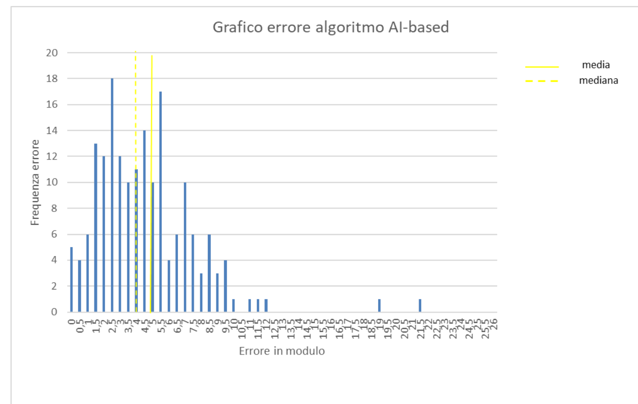


Figura 74: Grafico frequenza errore algoritmo AI-based quarto cartone

La media è pari a 4,8 px, la mediana vale 4 px e la deviazione standard è di 3,05 px.

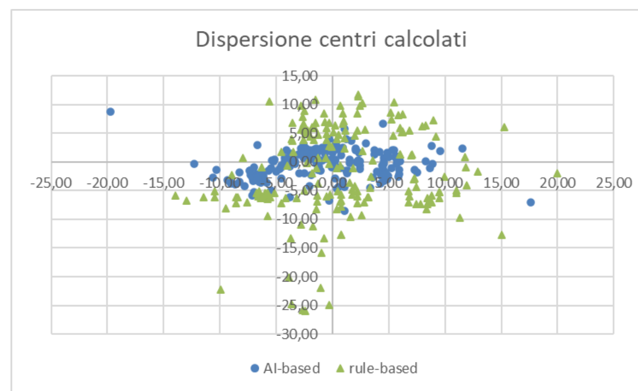


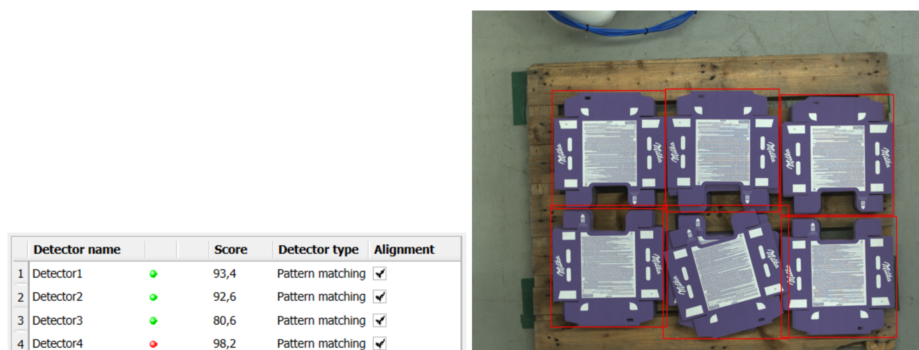
Figura 75: Grafico dispersione centri quarto cartone

Dall'analisi condotta sui grafici, emerge che gli algoritmi basati sul *deep learning* producono risultati superiori rispetto agli approcci *rule-based*. I grafici evidenziano che, nei metodi *rule-based*, l'errore nella determinazione dei centri risulta maggiore in media e si estende su un intervallo di valori più ampio. Inoltre, la dispersione dei centri occupa un'area più ampia nei casi degli algoritmi *rule-based*, evidenziando ulteriormente la maggior accuratezza degli algoritmi *AI-based*.

In particolare, si osserva che l'errore maggiore nel calcolo dei centri mediante l'algoritmo *rule-based* si verifica in presenza di disallineamenti più evidenti: più è grande il disallineamento più risulta impreciso l'algoritmo. Il funzionamento migliore degli algoritmi *AI-based* è dovuto alla loro capacità di apprendere e adattarsi a nuove situazioni.

Sempre nel caso di grandi disallineamenti è importante notare che ci sono delle situazioni in cui l'algoritmo *rule-based* non riconosce determinate acquisizioni, mentre l'algoritmo basato su *deep learning* riesce a rilevarle, riducendo ulteriormente il rischio di errori. Di seguito sono riportati esempi riguardanti questa casistica per ogni tipologia di cartone.

Figura 76: Esempio di immagine test non riconosciuta cartone tipologia uno



	Detector name		Score	Detector type	Alignment
1	Detector1		93,4	Pattern matching	✓
2	Detector2		92,6	Pattern matching	✓
3	Detector3		80,6	Pattern matching	✓
4	Detector4		98,2	Pattern matching	✓

Figura 77: Fallimento algoritmo rule-based cartone tipologia uno

Figura 78: Risultato con algoritmo AI-based cartone tipologia uno

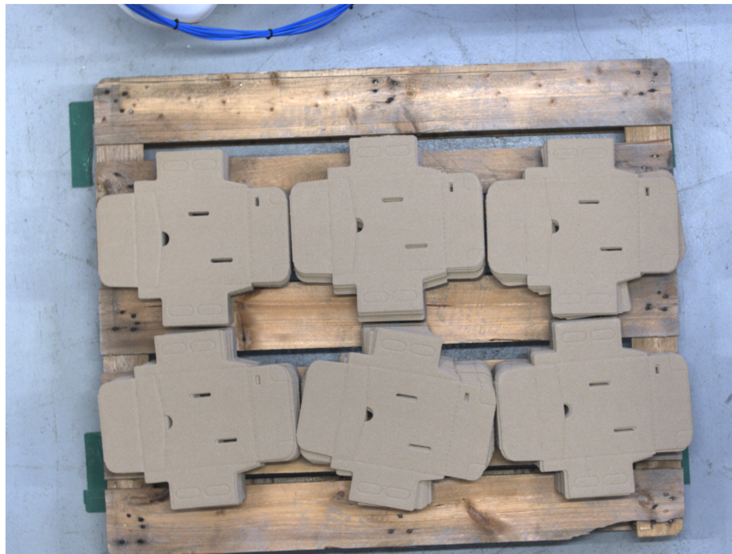


Figura 79: Esempio di immagine test non riconosciuta cartone tipologia due

	Detector name		Score	Detector type	Alignment
1	Detector1		98,0	Contour	✓
2	Detector2		92,4	Contour	✓
3	Detector3		96,7	Contour	✓
4	Detector4		98,4	Contour	✓

Figura 80: Fallimento algoritmo rule-based cartone tipologia due

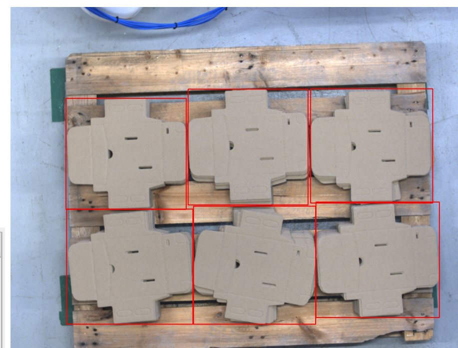


Figura 81: Risultato con algoritmo AI-based cartone tipologia due

Figura 82: Esempio di immagine test non riconosciuta cartone tipologia tre

	Detector name		Score	Detector type	Alignment
5	Detector5		96,3	Pattern matching	✓
6	Detector6		91,8	Pattern matching	✓
7	Detector7		93,3	Pattern matching	✓
8	Detector8		94,0	Pattern matching	✓
9	Discriminatore1		58,1	Pattern matching	✓

Figura 83: Fallimento algoritmo rule-based cartone tipologia tre

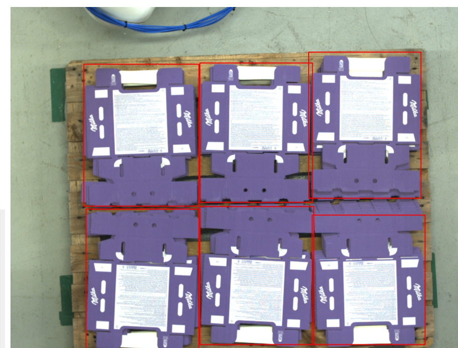


Figura 84: Risultato con algoritmo AI-based cartone tipologia tre

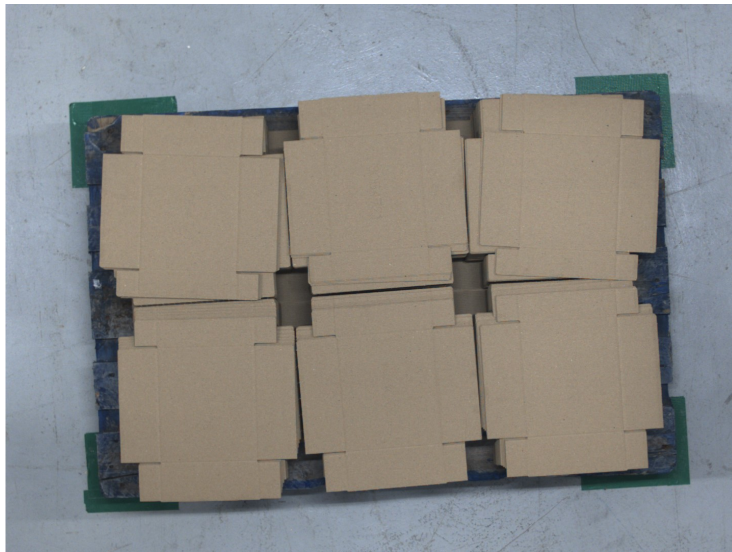


Figura 85: Esempio di immagine test non riconosciuta cartone tipologia quattro

	Detector name		Score	Detector type	Alignment
1	sopra1		89,1	Pattern matching	✓
2	sotto1		85,3	Pattern matching	✓
3	sopra2		88,6	Pattern matching	✓
4	sotto2		93,7	Pattern matching	✓

Figura 86: Fallimento algoritmo rule-based cartone tipologia quattro

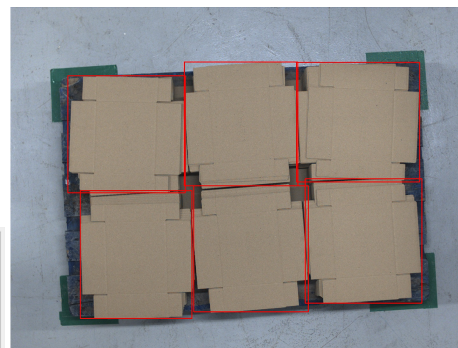


Figura 87: Risultato con algoritmo AI-based cartone tipologia quattro

8 Conclusioni

I risultati emersi dall'analisi dei dati evidenziano che l'algoritmo di *deep learning* offre prestazioni superiori, specialmente in situazioni caratterizzate da significativi disallineamenti. L'utilizzo di un algoritmo *AI-based* comporta tempi di esecuzione leggermente più lunghi rispetto a un approccio *rule-based*. Tuttavia, l'algoritmo *AI-based* è più vantaggioso in quanto è capace di adattarsi dinamicamente a diverse situazioni. Questa flessibilità consente all'algoritmo di essere robusto a cambiamenti nell'ambiente o riposizionamenti dei cartoni senza necessità di essere riconfigurato. Inoltre, gli algoritmi *AI-based* offrono vantaggi in termini di riduzione dei costi. Per implementare un algoritmo *rule-based* su software proprietario è necessario l'uso di una camera smart, al contrario, per l'algoritmo basato su *deep learning* è sufficiente una semplice camera poiché l'elaborazione avviene direttamente sul computer.

È di primaria importanza valutare attentamente quale approccio adottare per ciascuna applicazione di visione per la robotica. Nei contesti strutturati e poco mutevoli l'implementazione di algoritmi *rule-based* è preferibile in quanto garantisce risultati precisi in tempi brevi e richiede minor tempo per essere realizzata. Tuttavia, nei contesti caratterizzati da elevata variabilità è più conveniente utilizzare algoritmi *AI-based* che sono in grado di adattarsi ai cambiamenti e di produrre risultati migliori. Un'altra possibilità consiste nell'adozione di un approccio ibrido in cui algoritmi di *deep learning* e algoritmi *rule-based* collaborano simultaneamente consentendo la realizzazione di applicazioni in grado di affrontare compiti complessi con la massima precisione.

Il prossimo passo fondamentale per completare il progetto consiste nello sviluppo di un sistema di comunicazione tra la telecamera e l'algoritmo con l'obiettivo di massimizzare la velocità e garantire una sincronizzazione precisa tra i due. Questa fase, una volta completata, consentirà l'utilizzo effettivo del sistema per la presa del robot permettendo di valutarne le prestazioni in modo accurato e dettagliato.

Per ottimizzare ulteriormente le prestazioni dell'algoritmo implementato si potrebbero addestrare nuovi modelli di reti neurali al fine di determinare quale si adatti in maniera ottimale al contesto analizzato. In particolare, si confrontano le diverse architetture di reti neurali al fine di individuare quella che soddisfa al meglio i requisiti chiave, tra cui l'accuratezza nella determinazione del *bounding box* e il tempo di esecuzione. Tale analisi permetterebbe di selezionare la rete neurale che meglio si adatta alle esigenze del problema affrontato.

Riferimenti bibliografici

[1] iMAGE S S.p.A *Catalogo IMAGE S* https://www.imagesspa.it/image/catalog/pdf/Catalogo-iMAGE-S-2023.pdf

[2] Libo Yang *Talking About the Industrial Robot Grasping Technology Based on Machine Vision* https://iopscience.iop.org/article/10.1088/1742-6596/1769/1/012075/pdf

[3] Pickit *Camera be fixed or Camera robot-mounted* https://docs.pickit3d.com/en/3.1/optimize-your-application/hardware/faq-which-camera-mount.html

[4] Jianfeng Jiang, Xiao Luo, Qingsheng Luo, Lijun Qiao, Minghao Li *An overview of hand-eye calibration* https://link.springer.com/article/10.1007/s00170-021-08233-6

[5] Sarah Moore *Applications of Machine Vision in Robotics* https://www.azorobotics.com/Article

[6] COGNEX *Cognex Starters Guide* https://www.cognex.com/what-is/deep-learning/deep-learning-vs-machine-vision-and-human-inspection

[7] MVTech,*MVTech documentation* https://www.mvtec.com/technologies/deep-learning/classic-machine-vision-vs-deep-learning

[8] Automate,*Automate editorials* https://www.automate.org/editorials/rule-based-vs-ai-based-machine-vision

[9] Emerson, *Ni documentation* https://www.ni.com/docs/en-US/bundle/ni-vision-concepts-help/page/pattern_matching_techniques

[10] OPTO Engineering, *OPTO Engineering documentation* https://www.opto-e.com/it/basics/extras

[11] COGNEX, *Cognex e-book Deep Learning Machine Vision* https://www.cognex.com/library/media/files/ebook-deep-learning-machine-vision.pdf

[12] MATLAB, *MATLAB documentation* https://it.mathworks.com/discovery/object-detection

[13] Dhillon, A., Verma, G.K., *Convolutional neural network: a review of models, methodologies and applications to object detection. Prog Artif Intell 9, 85–112 (2020)* https://doi.org/10.1007/s13748-019-00203-0

[14] Analytics India Magazine, *10 Evaluation Metrics for Machine Learning Models* https://analyticsindiamag.com/10-evaluation-metrics-for-machine-learning-models/

[15] Ultralytics, *YOLO performance metrics* https://docs.ultralytics.com/guides/yolo-performance-metrics/interpreting-the-output

[16] SensoPart, *Visor Robotic* https://www.sensopart.com/en/products/vision-sensors/visor-robotic/

[17] SensoPart, *Visor Documentation* https://www.sensopart.com/documentation

[18] COGNEX, *Cognex Documentation is2800* https://support.cognex.com/en/documentation/in-sight/is2800

[19] Omron, *Omron S8VK-C12024* https://industrial.omron.it/it/products/S8VK-C12024

[20] Python, *Python 3.9.7 documentation* https://docs.python.org/3/

[21] Visual Studio Code, *Documentation for Visual Studio Code* https://code.visualstudio.com/docs

[22] labelme,*GitHub labelme* https://github.com/wkentaro/labelme

[23] Ultralytics,*Ultralytics documentation* https://docs.ultralytics.com

[24] Jia Shijie, Wang Ping, Jia Peiyi, Hu Siping, Dalian Jiaotong ,*Research on Data Augmentation for Image Classification Based on Convolution Neural Networks*

[25] Meta ,*Detectron2 ModelZoo* https://github.com/facebookresearch/detectron2/

[26] Yuxin Wu, Alexander Kirillov, Francisco Massa, Wan-Yen Lo, Ross Girshick , *Detectron2* https://github.com/facebookresearch/detectron2

[27] Meta , *Detectron2 Documentation* https://colab.research.google.com/drive/

[28] Meta , *Detectron2 Code* https://github.com/computervisioneng/train-object-detector-detectron2