Università degli Studi di Padova

Dipartimento di Tecnica e Gestione dei Sistemi Industriali

Corso di Laurea Magistrale in

Ingegneria Meccatronica

Utilizzo delle librerie MVTec Halcon per sistemi di visione in robotica

Relatore:

Prof. Giovanni Boschetti

Laureando:

Mirko Saggioro

2022411

Anno Accademico 2022/2023

Ringraziamenti

Mi è doveroso dedicare questo spazio della mia tesi a tutte le persone che mi hanno supportato nel mio percorso di crescita universitaria e professionale. Innanzitutto, desidero esprimere la mia gratitudine al Prof. Giovanni Boschetti per i suoi consigli e per avermi concesso l'opportunità di approfondire il mondo dei sistemi di visione e del deep learning, tecnologie che saranno sempre più diffuse nella quotidianità dell'umanità in futuro. Ringrazio poi i miei genitori Stefania e Maurizio, mia sorella Elisa, la mia ragazza Arianna e gli amici di una vita, per avermi sempre supportato in questo meraviglioso percorso!

Indice

Introduzione

L'obiettivo di questo elaborato è l'analisi delle librerie MVTec Halcon, uno dei principali software per l'elaborazione industriale delle immagini (sistemi di visione) in tutto il mondo. Halcon è sviluppato da MVTec Software GmbH, un produttore leader di software per l'elaborazione industriale delle immagini, con sede a Monaco di Baviera, in Germania. I prodotti MVTec sono utilizzati in un'ampia varietà di applicazioni e settori, tra cui semiconduttori, componenti elettrici, intralogistica, alimenti e bevande, apparecchiature di prova, prodotti farmaceutici e ricerca.
Halcon è noto per la sua flessibilità e facilità di sviluppo di applicazioni per l'elaborazione e l'analisi delle immagini industriali e medicali, il che lo rende uno strumento molto completo e versatile. Essendo uno dei software per sistemi di visione più completi al mondo, Halcon riduce i costi e garantisce un migliore time-to-market.
L'elaborato propone una valutazione dettagliata di diverse tecniche di riconoscimento oggetti utilizzando Halcon, tra cui Shape-Based Matching, Blob Analysis, Instance Segmentation e Classification. La tesi si articola in quattro parti principali. Inizialmente viene presentata una panoramica generale sul software Halcon, focalizzando l'attenzione sull'interfaccia e la strumentazione utilizzata per eseguire le varie prove. Nel secondo capitolo sono state confrontate le tecniche di Shape-Based Matching e Blob Analysis per il riconoscimento di figure geometriche, in particolare si è calcolato il loro centroide e l'angolo di inclinazione. Si sono analizzati i tempi di esecuzione, la robustezza a variazioni di illuminazione e la facilità di implementazione. Nel capitolo successivo, sono state testate le librerie di deep learning di Halcon, in particolare la tecnica di Instance Segmentation, sempre con lo scopo di riconoscere e classificare figure geometriche, analizzando i tempi di training e la capacità di rilevare le caratteristiche degli oggetti, come centroide e angolo di inclinazione. Nel capitolo conclusivo si è testato il Deep Learning Tool, un tool di Halcon dedicato al deep learning che è stato utilizzato per riconoscere delle mani nelle immagini utilizzando la tecnica di Classification, in modo da segnalare l'eventuale presenza di un operatore. Per fare ciò, si è importato un dataset di immagini comprensivo sia di mani umane che di altri oggetti di sfondo, così da poter fare una distinzione tra i due. In questa fase di test, si è utilizzata una rete neurale convoluzionale (CNN)

pre-addestrata ed in particolare, si è analizzata la sensibilità della CNN alle variazioni di illuminazione e alle diverse posizioni delle mani nel piano. La tecnica di Classification è utile per distinguere gli oggetti, ma non permette di localizzarli. Per questo motivo si è deciso di implementare la tecnica di Instance Segmentation anche per il riconoscimento delle mani, in modo analogo a quanto fatto per gli oggetti geometrici. In combinazione a questa tecnica si è deciso di implementare anche lo Shape-Based Matching, in modo da riconoscere nello stesso momento sia le mani che gli oggetti presenti nella zona di lavoro. Per quest'ultima prova le immagini sono state acquisite da una videocamera in tempo reale.

Capitolo 1

MVTec Halcon e setup sperimentale

1.1 Software MVTec Halcon

L'implementazione di tecniche di visione artificiale è diventata sempre più importante per molte applicazioni industriali, dal controllo di qualità alla robotica, dall'automazione industriale al monitoraggio di sicurezza. In questo contesto, MVTec Halcon rappresenta una soluzione all'avanguardia per la realizzazione di sistemi di visione artificiale avanzati.
MVTec Halcon è stato sviluppato dalla società tedesca MVTec Software GmbH, che è stata fondata nel 1996 e ha acquisito una vasta esperienza nella fornitura di soluzioni di visione artificiale di alta qualità per una vasta gamma di industrie, tra cui l'automotive, l'elettronica, la logistica, la farmaceutica e molti altri settori.
Il software supporta diverse tecnologie, tra cui la segmentazione dell'immagine, il riconoscimento dei pattern, la classificazione degli oggetti, la rilevazione dei difetti e molto altro ancora. Halcon è un software altamente flessibile e scalabile, che può essere facilmente adattato alle esigenze specifiche di ogni cliente. Inoltre, il software supporta varie tipologie di piattaforme hardware e software, tra cui i sistemi operativi Windows, Linux e macOS. La libreria completa può essere utilizzata da comuni linguaggi di programmazione come C, C++, Python e .NET, come C# o VB.NET.
Halcon garantisce l'indipendenza dall'hardware, fornendo interfacce a centinaia di telecamere e frame grabber industriali, in particolare supportando standard come GenICam, GigE Vision e USB3 Vision. In questo modo, il software è ideale per l'implementazione in sistemi embedded, compresi quelli basati su Arm®-based smart camera e altre piattaforme di visione embedded [15].

1.1.1 Finestra di avvio

La finestra di avvio (riportata in Figura 1.1), fornisce un rapido accesso a varie funzionalità del software Halcon:

1. Halcon mette a disposizione numerosi programmi-esempio commentati, che forniscono una descrizione accurata delle diverse funzionalità e operatori presenti nel software. In questo modo, è possibile avere un'ottima comprensione delle diverse tecniche e dei vari algoritmi utilizzabili per implementare soluzioni per i sistemi di visione in robotica;

2. sono disponibili numerosi webinar e tutorial che forniscono esempi pratici sull'utilizzo del software in contesti reali;

3. è disponibile l'accesso completo alla documentazione di Halcon, che comprende guide dettagliate per l'implementazione di qualsiasi tipo di tecnica nei sistemi di visione, nonché informazioni su tutte le funzioni Halcon disponibili e sul loro utilizzo.

Figura 1.1: Finestra di avvio del software Halcon [6].

1.1.2 Interfaccia Grafica

Di seguito è riportata l'interfaccia grafica (GUI) del software Halcon (Figura 1.2):

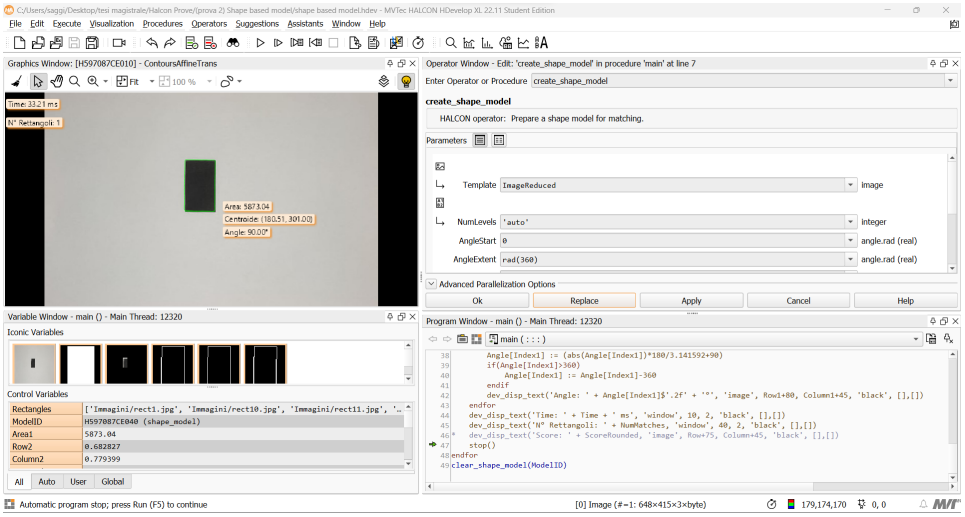


Figura 1.2: GUI del software Halcon.

In alto a sinistra abbiamo la Graphic Window (come riportato in Figura 1.3), ossia una finestra grafica interattiva in cui è possibile visualizzare immagini, regioni, contorni e altri oggetti grafici generati dal programma in esecuzione. È possibile eseguire diverse operazioni di visualizzazione dell'immagine, come ingrandire e ridurre, spostare l'immagine, ripristinare il rapporto di forma e ruotare l'immagine. Inoltre, è possibile zoomare sull'immagine utilizzando la rotellina del mouse.

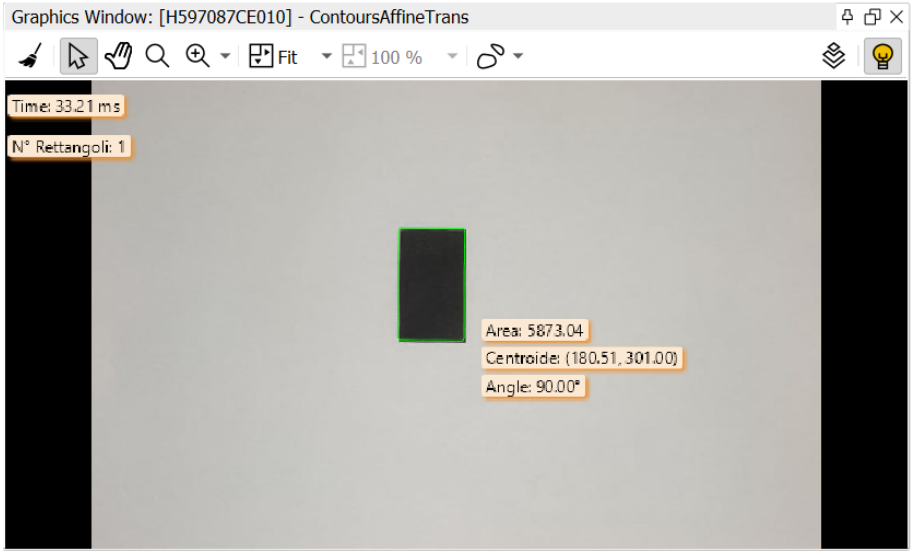


Figura 1.3: Graphics Window del software Halcon.

La finestra di grafica di Halcon è quindi uno strumento utile per l'analisi e

la visualizzazione di immagini e dati grafici generati dal software [6].
In basso a sinistra è invece presente la Variable Window di Halcon (come riportato in Figura 1.4), la quale è una finestra interattiva che mostra l'elenco delle variabili definite nel programma in esecuzione. Le variabili sono divise in due categorie: Iconic Variables e Control Variables.

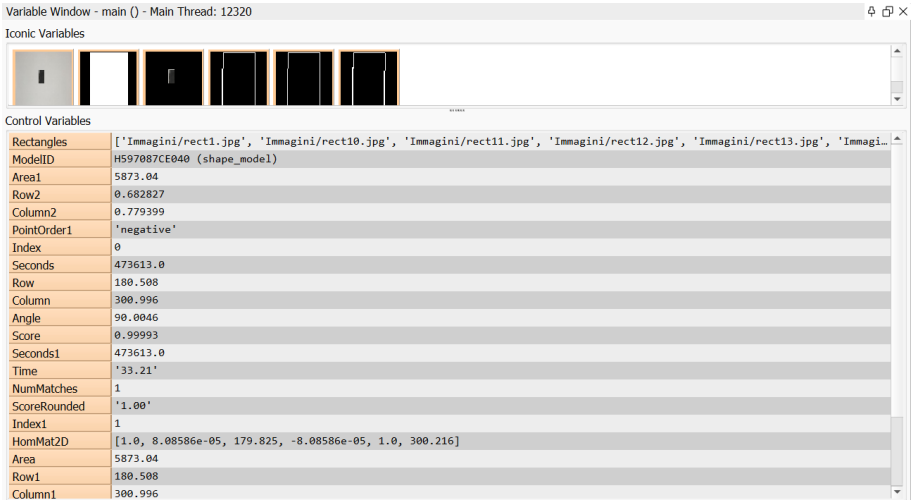


Figura 1.4: Variable Window del software Halcon.

Le Iconic Variables contengono oggetti come immagini, regioni e contorni e vengono visualizzate con un'immagine di anteprima dell'oggetto contenuto nella variabile. Osservando le Iconic Variables, è possibile notare che ogni immagine ha un cosiddetto dominio, che specifica la parte dell'immagine che viene elaborata. Il dominio funge da regione di interesse (ROI) e rappresenta un insieme di pixel. Le Control Variables, invece, contengono numeri, stringhe e handles (riferimenti a strutture dati complesse) e sono visualizzate con il loro valore attuale. Un'altra importante differenza tra le Iconic e Control Variables è che per le Iconic Variables, quando abbiamo serie di dati, si utilizzano gli array e per accedere al primo elemento si utilizza l'indice "1", mentre per le Control Variables si utilizzano le tuple e per accedere al primo elemento si utilizza l'indice "0".
Nella Variable Window, ogni variabile viene visualizzata con il suo nome, tipo, dimensioni e valore attuale. È possibile anche modificare il valore delle variabili direttamente dalla finestra stessa. La finestra delle variabili di Halcon è anche interattiva e permette di visualizzare le immagini e le regioni selezionando la variabile corrispondente e facendo doppio clic su di essa. In questo modo, l'immagine o la regione verrà visualizzata nella finestra grafica di Halcon. Inoltre, è possibile cercare variabili specifiche per nome, semplificando la gestione delle stesse.
La Variable Window di Halcon è uno strumento potente e flessibile che consente agli utenti di gestire e visualizzare facilmente le variabili utilizzate in

un programma Halcon in esecuzione [6].
In basso a destra è presente la Program Window, come riportato in Figura 1.5:

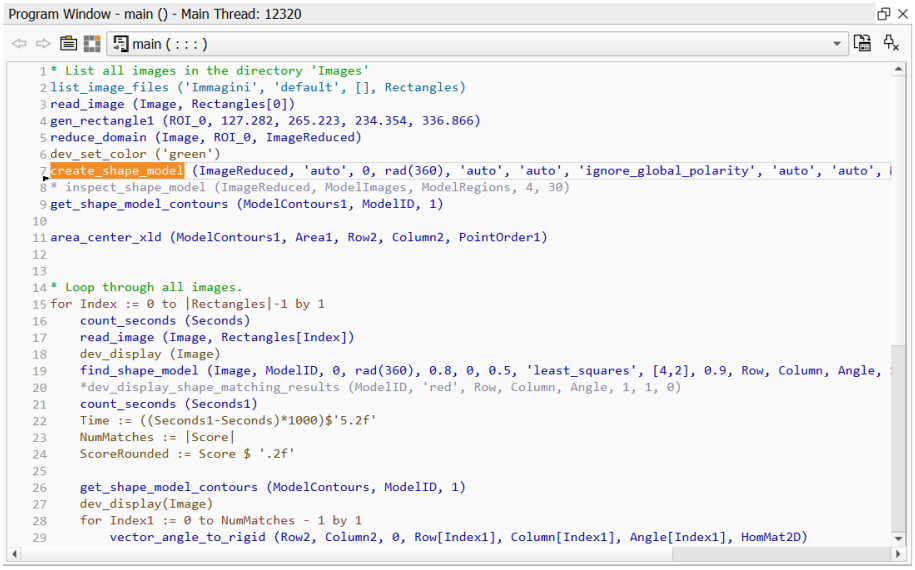


Figura 1.5: Program Window del software Halcon.

La Program Window di Halcon è un'importante interfaccia che permette agli utenti di scrivere, eseguire e debuggare programmi Halcon. Essa consiste in una finestra di testo in cui gli utenti possono inserire istruzioni utilizzando il linguaggio di programmazione Halcon in modo interattivo. Durante l'esecuzione del programma, gli utenti possono visualizzare gli output direttamente nella finestra del programma.
Oltre alla scrittura del codice, la Program Window offre anche accesso alla documentazione e ai file di esempio di Halcon. Inoltre, essa è dotata di funzionalità avanzate come l'autocompletamento del codice e la formattazione automatica, che ne semplificano la scrittura e la lettura.
La Program Window di Halcon offre anche strumenti di debug integrati che permettono agli utenti di individuare e correggere gli errori di programmazione direttamente nella finestra del programma. Questa funzione è particolarmente utile durante lo sviluppo di programmi complessi, in cui possono verificarsi numerosi errori.
Questa finestra è quindi uno strumento essenziale per la scrittura, l'esecuzione e il debug di programmi Halcon, che offre funzionalità avanzate e strumenti di debug integrati per semplificare il processo di sviluppo del software [6].
In alto a destra è presente la Operator Window, come riportato in Figura 1.6:

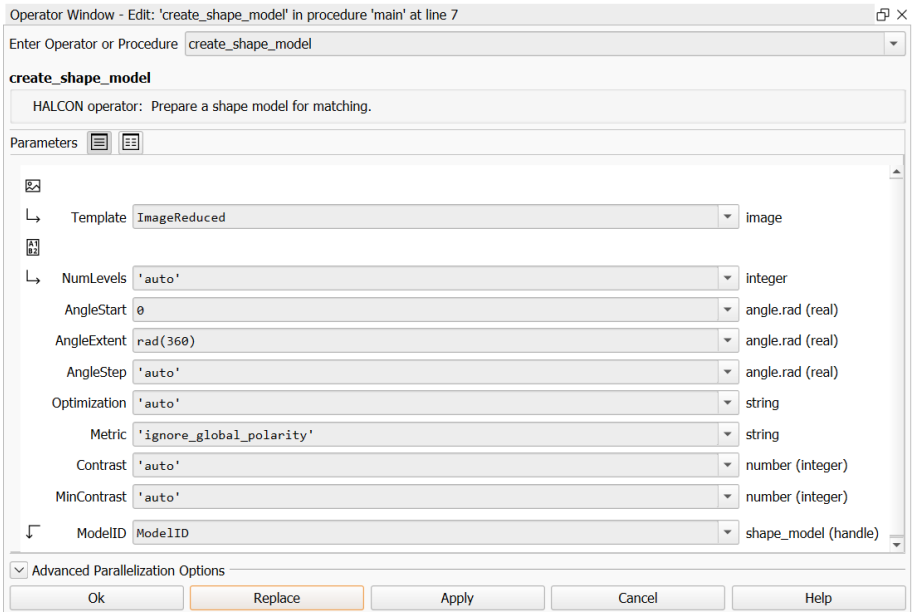


Figura 1.6: Operator Window del software Halcon.

Questa finestra è utilizzata per modificare e visualizzare una chiamata di operatore, insieme a tutti i suoi parametri. Qui è possibile ottenere informazioni sul numero dei parametri dell'operatore, sulla tipologia e sui valori degli stessi. Inoltre, è possibile modificare i valori dei parametri in base alle proprie esigenze di elaborazione delle immagini, utilizzando i valori predefiniti proposti da HDevelop o specificando valori personalizzati. La Operator Window è composta da tre parti principali:

- in alto si trova il campo dove inserire il nome dell'operatore, che consente di selezionare gli operatori desiderati;
- vi è un'ampia area dove si possono visualizzare e modificare i parametri dell'operatore;
- la sezione "Advanced Parallelization Options" consente di chiamare gli operatori come sottothread, per aumentare l'efficienza del sistema.

Notiamo che in questa finestra sono riportati i parametri in input e in output dell'operatore scelto, il loro valore e il loro tipo [6].

1.1.3 Deep Learning in Halcon

Il software MVTec Halcon rappresenta una soluzione avanzata per l'applicazione di tecnologie di deep learning ai sistemi di visione industriali. In particolare, gli utenti hanno la possibilità di addestrare reti neurali convoluzionali

(CNN) basate su algoritmi di deep learning, che possono successivamente essere utilizzate per classificare e riconoscere nuovi oggetti dall'immagine.
L'addestramento di una CNN richiede che gli utenti forniscano ad Halcon immagini di addestramento etichettate, ovvero già pre-assegnate alle rispettive categorie. Una volta fornite le immagini di addestramento, il software analizza automaticamente tali immagini e apprende quali funzionalità possono essere utilizzate per identificare le classi specificate. In tal modo, viene eliminato il tedioso lavoro manuale di creazione di funzionalità.
Per l'etichettatura può venire utilizzato un tool fornito dalla MVTec, chiamato Deep Learning Tool, con il quale si possono facilmente etichettare i dati grazie all'interfaccia utente intuitiva. Questi dati possono essere integrati senza problemi in Halcon per eseguire rilevamento degli oggetti, classificazione e segmentazione basati sul deep learning. Per i progetti di classificazione, è anche possibile addestrare e valutare il modello direttamente nel Deep Learning Tool. L'etichettatura dei dati di addestramento rappresenta il primo passo cruciale per qualsiasi applicazione di deep learning. La qualità di questi dati etichettati gioca un ruolo fondamentale per quanto riguarda le prestazioni, l'accuratezza e la robustezza dell'applicazione.
Una volta che la CNN ha appreso come differenziare le classi specificate, può essere messa al lavoro per classificare nuovi dati di immagine. Gli utenti possono quindi applicare la CNN addestrata a nuovi dati di immagine, la quale poi li classifica utilizzando le informazioni apprese durante l'addestramento. Questo processo, noto come inferenza, è l'aspetto fondamentale nei sistemi di visione industriali.
Va sottolineato come le funzionalità di deep learning di MVTec Halcon non si limitino alla classificazione degli oggetti, ma possano essere estese anche alla classificazione dei difetti. Ad esempio, il software può essere utilizzato per ispezionare le bocche delle bottiglie al fine di individuare eventuali irregolarità, o per verificare se le confezioni di blister per le pillole sono etichettate correttamente. In tal modo, MVTec Halcon rappresenta una soluzione versatile ed efficace per migliorare la qualità e l'efficienza delle attività di visione industriale [15].
Il terzo e il quarto capitolo dell'elaborato forniranno un'analisi dettagliata sull'utilizzo del deep learning in Halcon.

1.2 Setup sperimentale

Oltre al software MVTec Halcon, il setup sperimentale comprende un PC Dell Inspirion 15 7590 e una camera Logitech HD C270 collegata ad un supporto metallico.
Il PC Dell Inspirion 15 7590 ha le seguenti caratteristiche:

- processore Intel(R) Core(TM) i7-9750H di nona generazione, con velocità del clock della CPU di 2.60 GHz;

- scheda grafica dedicata NVIDIA GTX 1050 N17P-G0-K1, con memoria di 3 GB;
- Ram di 16 GB;
- Windows 11 come sistema operativo.

La camera Logitech HD C270 è riportata in Figura 1.7:

Figura 1.7: Camera Logitech HD C270.

Le sue specifiche tecniche sono riportate di seguito:

- risoluzione massima di 720p/30 fps;
- telecamera di 0.9 mega pixel;
- messa a fuoco fissa;
- campo visivo diagonale (dFoV [1]) di 55°.

Inoltre, per testare il riconoscimento degli oggetti con varie tecniche di Halcon, sono state utilizzate figure geometriche realizzate in cartoncino. Queste figure sono state disposte in diverse configurazioni, con l'obiettivo di valutare la capacità del software di identificare e distinguere gli oggetti in diverse situazioni.

[1] Angolo di visualizzazione coperto dalla telecamera quando viene utilizzata per catturare un'immagine o un video.

È importante sottolineare che il computer e la fotocamera utilizzati non sono strumenti specifici per l'ambito industriale. Ciò significa che le analisi eseguite utilizzando le librerie Halcon potrebbero avere prestazioni ancora più elevate di quelle presentate in questo elaborato.

Capitolo 2

Blob Analysis e Shape-Based Matching

In questo capitolo vengono trattate le due tecniche principali per il riconoscimento di oggetti nei sistemi di visione in robotica, ossia Blob Analysis e Shape-Based Matching.
Entrambe le tecniche sono state implementate con il software MVTec Halcon, con lo scopo di riconoscere uno o più rettangoli neri (come riportato in Figura 2.1), da cui si ricaveranno il centroide[2] e l'angolo di inclinazione.

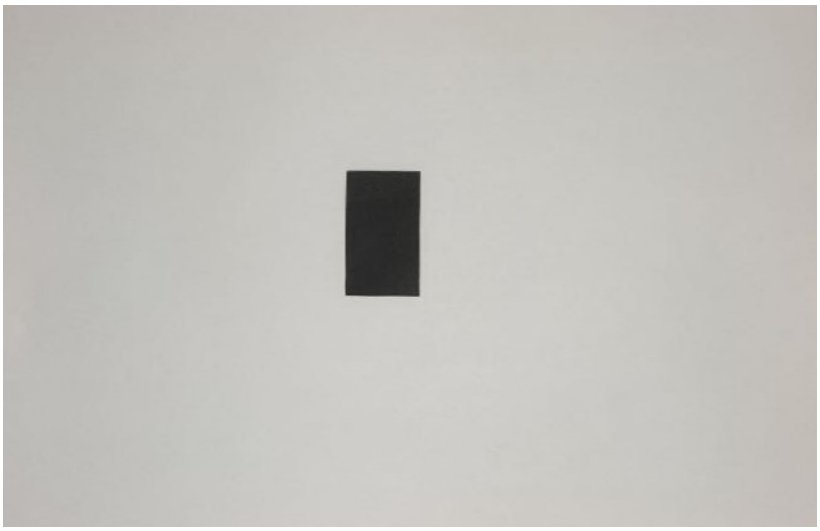


Figura 2.1: Rettangolo di riferimento che dovà essere riconosciuto nelle immagini.

[2]Il centroide di una figura bidimensionale è la "posizione media" di tutti i suoi punti, ovvero la media aritmetica delle posizioni di ciascuno di essi. La definizione si estende a qualunque figura n-dimensionale in uno spazio euclideo n-dimensionale: il suo centroide è la posizione media di tutti i punti in tutte le direzioni coordinate.

Questi dati potrebbero essere poi eventualmente inviati ad un robot per garantire la presa corretta dell'oggetto.
Le tecniche di Blob Analysis e Shape-Based Matching sono state confrontate considerando le seguenti caratteristiche:

- tempi di esecuzione;
- accuratezza nel riconoscimento degli oggetti;
- robustezza a cambi di illuminazione e a disturbi (ombre e motion blur);
- complessità nell'implementazione.

2.1 Blob Analysis

Impostare un valore di soglia desiderato come standard e convertire un'immagine in scala di grigi in valori di 0 e 1 viene chiamato "Processo di binarizzazione". Il metodo per analizzare un'immagine che ha subito un processo di binarizzazione viene chiamato "Blob Analysis". Un "blob" è una regione di un'immagine in cui alcune proprietà sono costanti o approssimativamente costanti; tutti i punti in un blob possono essere considerati in qualche modo simili tra loro [12].
In un'immagine, i pixel degli oggetti rilevanti (anche chiamati fore-ground) possono essere identificati dal loro valore di grigio. Selezionando i pixel luminosi attraverso un'operazione di sogliatura (anche detta Thresholding), si può segmentare l'immagine, ossia partizionarla in regioni di pixel, così da poter individuare l'oggetto [1].
Blob Analysis è il metodo più semplice da implementare per l'elaborazione di immagini per analizzare le caratteristiche di un oggetto, come il numero di oggetti, l'area, la posizione, il centroide e l'angolo di inclinazione.

2.1.1 Workflow della Blob Analysis in Halcon

La Blob Analysis è composta da tre fasi:

- il primo step è l'acquisizione delle immagini. Nel nostro caso si è utilizzato l'operatore Halcon **"read_image"** all'interno di un ciclo for in modo da utilizzare tutte le immagini salvate in una cartella:

```
read_image (Image, Rectangles[Index])
```

- viene poi eseguita la segmentazione dell'immagine, ossia si vuole separare lo sfondo dagli oggetti che si vogliono riconoscere. Per compiere questa operazione si è utilizzata la funzione **"threshold"**, che richiede come parametri di input una soglia inferiore e una superiore, i quali ci permettono di escludere dei valori nella distribuzione dei valori di grigio dell'immagine. Il codice utilizzato è di seguito riportato:

```
threshold (Image, Region, 0, 174)
```

Per impostare correttamente questi parametri si è utilizzato uno strumento di Halcon chiamato Gray Histogram, riportato in Figura 2.2:

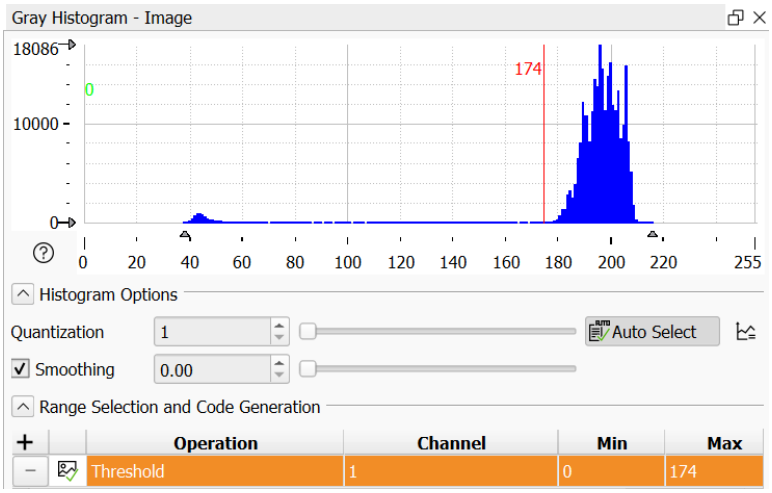


Figura 2.2: Gray Histogram di Halcon.

Utilizzando le due barre verticali (quella verde e quella rossa) si possono impostare correttamente i valori di soglia in modo da escludere lo sfondo, ossia i valori di grigio indicati dal picco in blu.
Si è poi perfezionata la segmentazione dell'immagine con l'operatore **"opening_circle"**, il quale permette di eliminare le regioni piccole (più piccole dell'elemento strutturale circolare di cui si definiscono i parametri) e per levigare i contorni di una regione [5]:

```
opening_circle (Region, RegionOpening, 3.5)
```

Si è utilizzato poi l'operatore **"connection"**, per separare gli oggetti presenti nella stessa regione in modo corretto:

```
connection (RegionOpening, ConnectedRegions)
```

Il nostro obiettivo è riconoscere i rettangoli, quindi per distinguerli dagli altri oggetti con altre forme geometriche si è ricorsi all'utilizzo dell'operatore **"select_shape"**:

```
select_shape (ConnectedRegions, Rect,
'rectangularity', 'and', 0.91405, 1)
select_shape (Rect, Model, 'area',
'and', 5911.27, 6146.03)
```

La funzione **select_shape** consente di separare le regioni in base alle loro caratteristiche. In particolare, la prima operazione è quella che ci permette di discriminare gli oggetti in base alla loro forma. A tal fine, si è notato, consultando la documentazione di Halcon, che il parametro 'rectangularity' può essere utilizzato per riconoscere gli oggetti rettangolari, come mostrato in Figura 2.3:

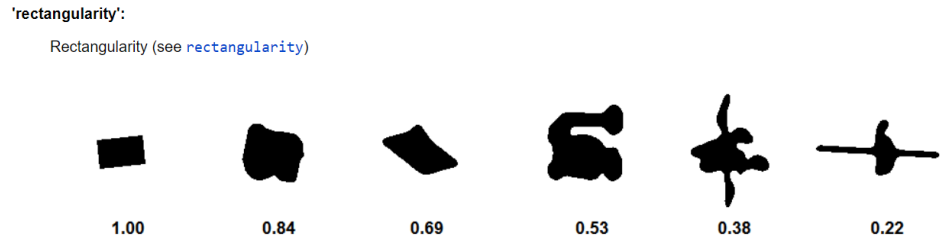


Figura 2.3: 'rectangularity' al variare dei parametri [5].

La seconda operazione invece ci permette di discriminare in base all'area (modificando il parametro 'area'), in modo da escludere i rettangoli troppo piccoli;

- nell'ultima fase, vengono estratti i dati rilevanti dalle regioni, tra cui il centroide, l'angolo di inclinazione e il numero di oggetti identificati, rispettivamente con gli operatori **"area_center"**, **"orientation_region"** e **"count_obj"**:

```
area_center (Model, Area, Row, Column)
orientation_region (Model, Phi)
count_obj (Model, Num_Rect)
```

L'angolo di inclinazione Phi viene determinato utilizzando l'operatore **orientation_region**. Questa funzione si basa sull'operatore **elliptic_axis**, che calcola i raggi Ra e Rb e l'orientamento Phi dell'ellisse avente la stessa orientazione e le stesse proporzioni della regione di input (nel nostro caso Model). Ra rappresenta il raggio principale dell'ellisse, mentre Rb rappresenta il raggio secondario.
L'orientamento dell'asse principale rispetto all'asse x viene espresso con l'angolo Phi in radianti. In altre parole, l'asse principale dell'ellisse coincide con l'asse principale del momento d'inerzia della regione di input [5].

La Figura 2.4 riassume il workflow da seguire per la Blob Analysis in Halcon:

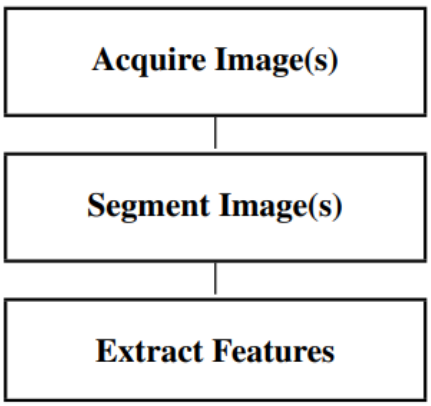


Figura 2.4: Workflow della Blob Analysis in Halcon [1].

2.1.2 Applicazione della Blob Analysis in Halcon

Si è utilizzato un set di 18 immagini in cui sono riportate diverse figure geometriche ed è stata applicata la Blob Analysis. Nelle Figure 2.5, 2.6 e 2.7 sono riportati alcuni dei risultati ottenuti:

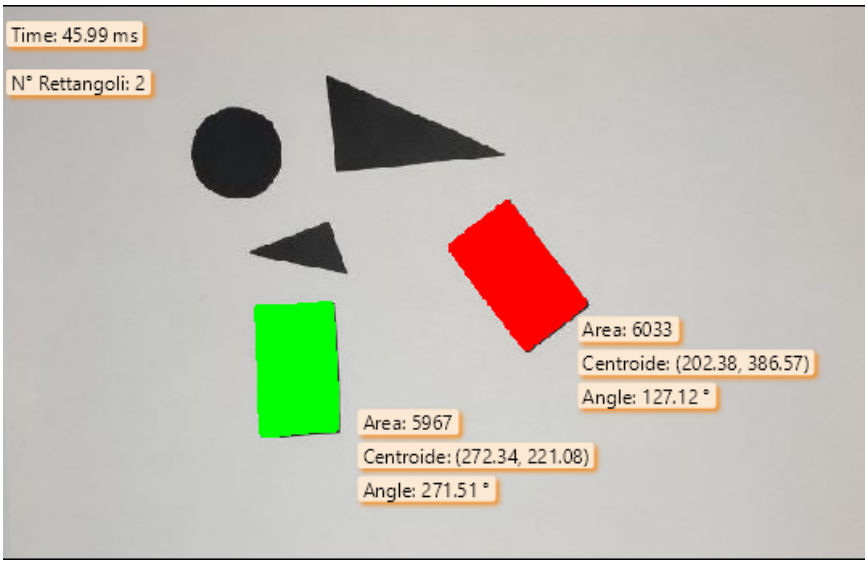


Figura 2.5: Esempio 1 di applicazione della Blob Analysis in Halcon per il riconoscimento di rettangoli.

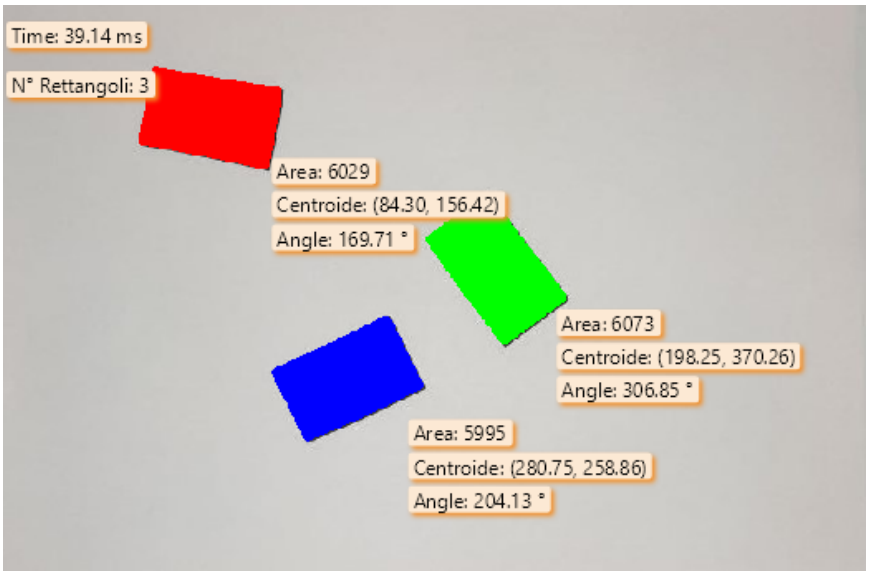


Figura 2.6: Esempio 2 di applicazione della Blob Analysis in Halcon per il riconoscimento di rettangoli.

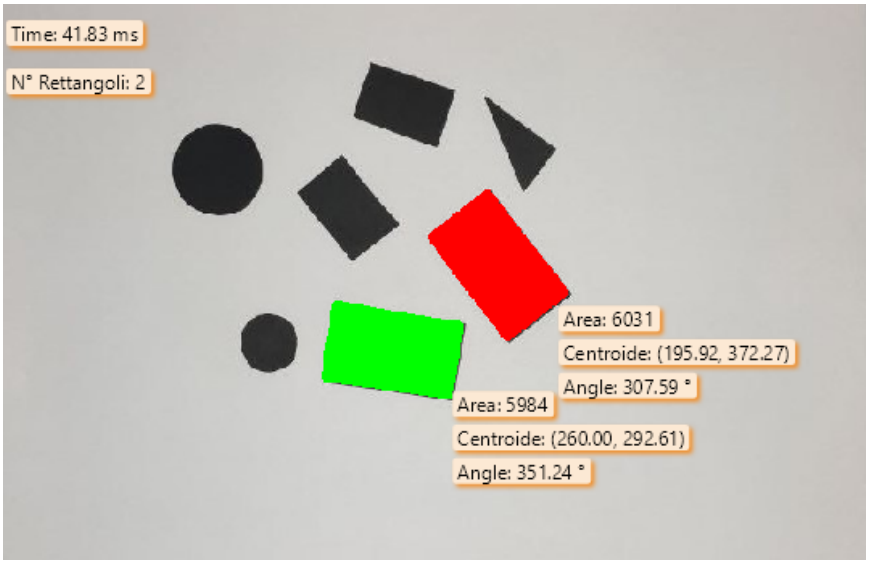


Figura 2.7: Esempio 3 di applicazione della Blob Analysis in Halcon per il riconoscimento di rettangoli.

Quando la foto è nitida, cioè priva di sfocature dovute a movimenti o vibrazioni (motion blur), e quando l'illuminazione è ottimale, si osserva un risultato che soddisfa le aspettative. È interessante notare che il centroide, ovvero il punto geometrico che rappresenta il centro di massa dell'immagine, è espresso come una coppia di valori tra parentesi. Il primo valore indica

la riga, cioè la coordinata verticale dell'immagine, mentre il secondo valore indica la colonna, ovvero la coordinata orizzontale. Entrambi i valori sono espressi in pixel, l'unità di misura tipica delle immagini digitali.
Osserviamo ora il caso in cui vi sia un cambio di illuminazione, come riportato in Figura 2.8:

Figura 2.8: Esempio di applicazione della Blob Analysis nel caso di scarsa illuminazione.

Notiamo che in questo caso, utilizzando la Blob Analysis con gli stessi parametri impostati per gli altri esempi, non si riesce a rilevare nessun rettangolo. Questo è dovuto al **cambio di illuminazione** rispetto alle altre immagini. Questo tipo di variazione può influire significativamente sulla qualità dell'immagine, rendendo più difficile il rilevamento delle forme geometriche. Pertanto, per ottenere risultati più precisi, sarebbe opportuno adattare i parametri della tecnica di analisi in base alle specifiche condizioni di illuminazione dell'immagine.

Un altro aspetto critico relativo alla Blob Analysis è il **motion blur**. Se nella foto sono presenti sfocature dovute a movimenti o vibrazioni, diventa molto complesso poter riconoscere gli oggetti, come riportato in Figura 2.9, dove non si rileva nessun rettangolo. Il motion blur rende difficile distinguere i contorni degli oggetti e può causare la fusione di aree dell'immagine, creando così false associazioni e segmentazioni degli oggetti. Inoltre, può anche ridurre la precisione delle misurazioni fatte sugli oggetti, come l'area e la forma, poiché l'effetto di sfocatura può distorcere le dimensioni e la forma degli oggetti stessi.

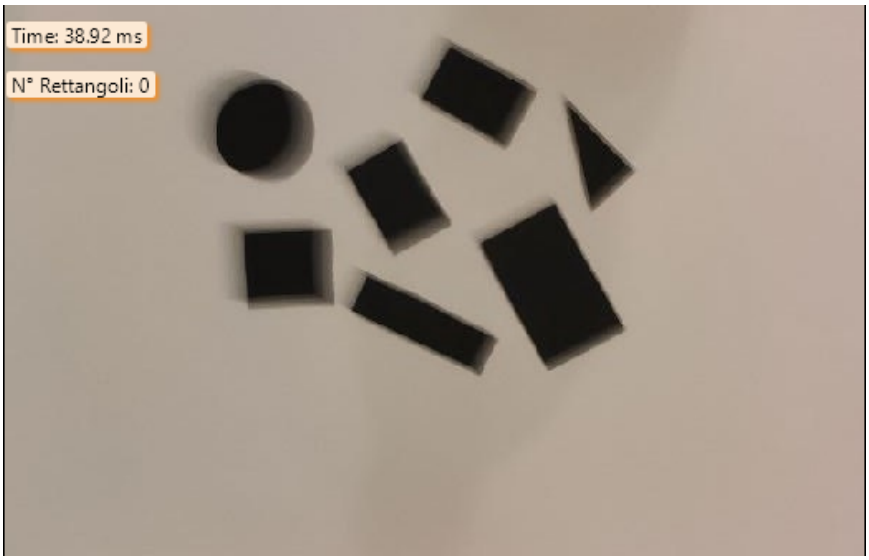


Figura 2.9: Esempio di applicazione della Blob Analysis nel caso di motion blur.

2.1.3 Risultati ottenuti

Analizzando i risultati ottenuti tramite l'applicazione della tecnica di Blob Analysis con Halcon, emergono le seguenti caratteristiche:

- i tempi di esecuzione sono nell'ordine dei 40 ms. Facendo una media di tutti i tempi di esecuzione si è ottenuto che:

$$\text{Tempo di esecuzione medio} = 40.71 \text{ ms}$$

- la tecnica di Blob Analysis implementata in Halcon risulta essere estremamente precisa nell'identificazione degli oggetti in condizioni ambientali ottimali. Tuttavia, è importante sottolineare che ciò vale solo per contesti privi di effetti di motion blur o di variazioni significative di luminosità;

- come accennato in precedenza, la tecnica di Blob Analysis presenta una limitata robustezza nei confronti di alcune variazioni ambientali, come ad esempio le sfocature generate da movimenti o vibrazioni (motion blur) e le variazioni di luminosità. Questi fattori possono infatti pregiudicare l'accuratezza del riconoscimento degli oggetti, tanto da renderlo in alcuni casi impossibile;

- è facilmente implementabile in Halcon.

2.2 Shape-Based Matching

La ricerca efficiente e accurata di oggetti in un'immagine è una delle attività fondamentali dei sistemi di visione in robotica. Lo Shape-Based Matching è un metodo di Template Matching che si concentra sulla forma dell'oggetto anziché sui pixel dell'immagine. Utilizza una misura di similarità robusta che può gestire problemi come occlusioni, rumore e cambiamenti di illuminazione, rendendolo uno dei metodi di matching più affidabili e ampiamente utilizzati per le applicazioni industriali.
Per utilizzare lo Shape-Based Matching, è necessario creare un modello dell'oggetto, definito da un insieme di punti e vettori di direzione associati. Il modello viene generato a partire da un'immagine dell'oggetto (nota come "immagine di modello"), in cui l'oggetto è definito da una regione di interesse (ROI). I punti del modello e i loro vettori di direzione associati vengono estratti dall'immagine di modello mediante l'individuazione dei bordi con una precisione a livello di subpixel e il calcolo dei gradienti di grigio in questi punti.
Durante il processo di matching, il modello trasformato viene confrontato con l'immagine in tempo reale, dove il gradiente di grigio viene calcolato in ogni pixel. Lo Shape-Based Matching utilizza una misura di similarità robusta per comparare i vettori di direzione normalizzati del modello e dell'immagine e restituisce i parametri di trasformazione e un valore di punteggio per ogni istanza dell'oggetto trovata nell'immagine.
Lo Shape-Based Matching è in grado di gestire diverse tipologie di trasformazioni, tra cui traslazioni, trasformazioni rigide, trasformazioni di similarità e trasformazioni affini arbitrarie. Utilizzando le piramidi delle immagini e algoritmi di ricerca sofisticati, questa tecnica può riconoscere gli oggetti in modo estremamente efficiente e preciso [9].

2.2.1 Workflow dello Shape-Based Matching in Halcon

Lo Shape-Based Matching in Halcon si compone di varie fasi:

- il primo passo del processo consiste nell'acquisizione dell'immagine di riferimento che verrà utilizzata per creare il modello per lo Shape-Based Matching. Per effettuare questa operazione, abbiamo utilizzato l'operatore **"read_image"** fornito dalla libreria Halcon, che ci consente di leggere un'immagine da un file e salvarla in una variabile, la stessa operazione fatta in precedenza per la Blob Analysis. Di seguito, il codice utilizzato per effettuare questa operazione:

```
read_image(Image, Rectangles[0])
```

- per riconoscere l'oggetto di riferimento in modo efficace, è necessario creare una "Region of Interest" (ROI), ovvero una regione in cui è con-

tenuto l'oggetto da identificare. Per fare ciò, si utilizza uno strumento presente nella Graphics Window chiamato "Create a new ROI", con cui è possibile disegnare, selezionando la forma desiderata, una regione che includerà l'oggetto di riferimento su cui verrà costruito il modello. Il codice generato a seguito di questa operazione nel nostro caso è il seguente:

```
gen_rectangle1 (ROI_0, 127.282, 265.223, 234.354,
336.866)
reduce_domain (Image, ROI_0, ImageReduced)
```

La funzione **"reduce_domain"** è utile per ridurre il dominio dell'immagine, in modo che venga processata solo la parte di immagine che si trova all'interno della regione da noi indicata e che contiene l'oggetto di interesse.
Durante questa fase, l'operazione più importante consiste nel creare il modello utilizzando l'operatore **"create_shape_model"**, che verrà utilizzato successivamente per effettuare lo Shape-Based Matching:

```
create_shape_model(ImageReduced, 'auto', 0, rad(360)
, 'auto', 'auto', 'ignore_global_polarity', 'auto',
'auto', ModelID)
```

Durante il processo di Shape Based Matching in Halcon, uno dei parametri principali da impostare nella funzione **create_shape_model** è **NumLevels**. Questo perché Halcon utilizza delle piramidi di immagini durante la ricerca dell'oggetto, riducendo ripetutamente la dimensione dell'immagine e del modello come mostrato nella Figura 2.10. La ricerca dell'oggetto inizia con un'immagine a bassa risoluzione, consentendo ricerche veloci, e l'area di ricerca viene successivamente limitata al livello successivo della piramide sulla base dei risultati ottenuti. Ciò consente di risparmiare tempo nella ricerca dell'oggetto e di evitare di dover eseguire ulteriori ricerche. Il parametro **NumLevels** specifica quindi quanti livelli di piramide verranno utilizzati durante la ricerca dell'oggetto nell'immagine. Se il parametro viene impostato su "auto", il numero di livelli verrà determinato automaticamente in base alle dimensioni dell'oggetto. [2]
Un parametro importante è **Metric**: il valore predefinito è "use _polarity", il che significa che se l'oggetto è luminoso e lo sfondo è scuro, saranno trovati solo gli oggetti luminosi su sfondo scuro. Con "ignore_global_polarity", gli oggetti vengono riconosciuti anche se scuri, mentre lo sfondo è luminoso. Questo ha solo un piccolo impatto sul tempo di esecuzione. Infine, con "ignore_local_polarity", è possibile trovare corrispondenze in cui il contrasto cambia localmente. Questa modalità può essere molto utile se si devono gestire cambiamenti di illuminazione, ma aumenta il tempo di esecuzione. Nel nostro caso è

stato sufficiente impostare "ignore_global_polarity".

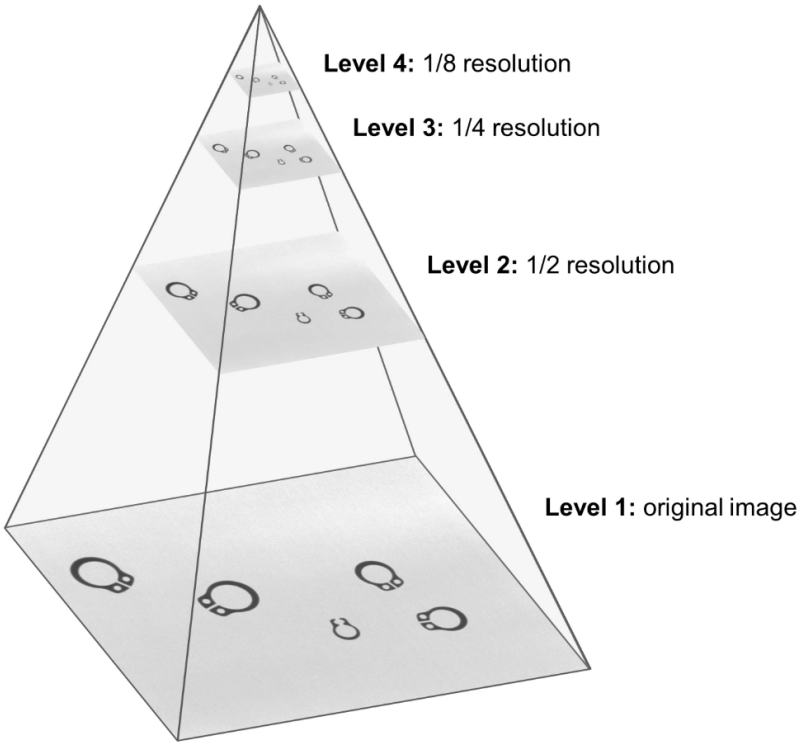


Figura 2.10: Piramide di immagini in base alla risoluzione. [13]

Sempre in **create_shape_model** si possono impostare come parametri anche **AngleStart** e **AngleExtent**, i quali ci permettono di limitare le orientazioni in cui l'oggetto può essere riconosciuto. Imponendo AngleStart = 0 e AngleExtent = rad(360), si abilitano tutte le orientazioni possibili;

- un'altra fase molto importante è quella della ricerca del modello all'interno delle immagini. Questa operazione viene compiuta utilizzando la funzione **"find_shape_model"**:

```
find_shape_model (Image, ModelID, 0, rad(360), 0.8,
0, 0.5, 'least_squares', [4,2], 0.9, Row, Column,
Angle, Score)
```

Anche in questa funzione deve essere impostato il parametro **NumLevels**; se viene scelto troppo alto, alcuni oggetti potrebbero non essere riconoscibili perché l'immagine è a bassa risoluzione, d'altra parte, se si inizia la ricerca a un livello di piramide inferiore, potrebbe richiedere molto tempo a causa dell'alta risoluzione. In alternativa, è possibile

saltare i livelli inferiori nella ricerca del modello, passando alla funzione, per esempio, una tupla di valore [4,2] come parametro. In questo modo la ricerca inizia al quarto livello della piramide e termina al secondo. Ciò può migliorare il tempo di esecuzione, tuttavia la precisione potrebbe diminuire. Nel nostro caso, abbiamo impostato il parametro NumLevels = [4,2] in quanto si è notato un miglioramento nei tempi di esecuzione senza compromettere la precisione.
La funzione **"find_shape_model"** restituirà come output la riga (Row) e la colonna (Column) che indicano il centroide in pixel dell'oggetto rilevato, insieme all'angolo di inclinazione (Angle). Inoltre, la funzione restituirà anche il valore "Score", che rappresenta un parametro numerico che descrive quanto l'oggetto rilevato è simile al modello di riferimento, indicando quindi la precisione dell'identificazione;

- nell'ultima fase occorre "distruggere" il modello creato, per liberare la memoria occupata [2], utilizzando l'operatore **"clear_shape_model"**:

```
clear_shape_model(ModelID)
```

La Figura 2.11 riassume il workflow da seguire per lo Shape-Based Matching in Halcon:

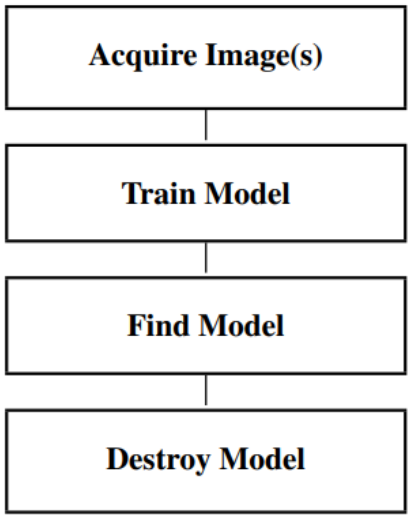


Figura 2.11: Workflow dello Shape-Based Matching in Halcon [1].

2.2.2 Applicazione dello Shape-Based Matching in Halcon

Si è utilizzato lo stesso set di 18 immagini usato per la Blob Analysis, in cui sono riportate diverse figure geometriche.

Nelle Figure 2.5, 2.6 e 2.7 sono riportati alcuni dei risultati ottenuti:

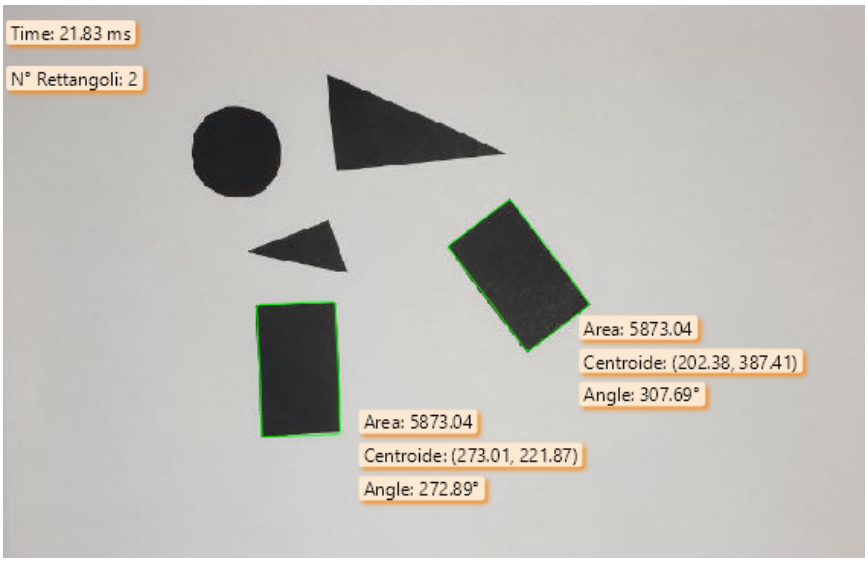


Figura 2.12: Esempio 1 di applicazione dello Shape-Based Matching in Halcon per il riconoscimento di rettangoli.

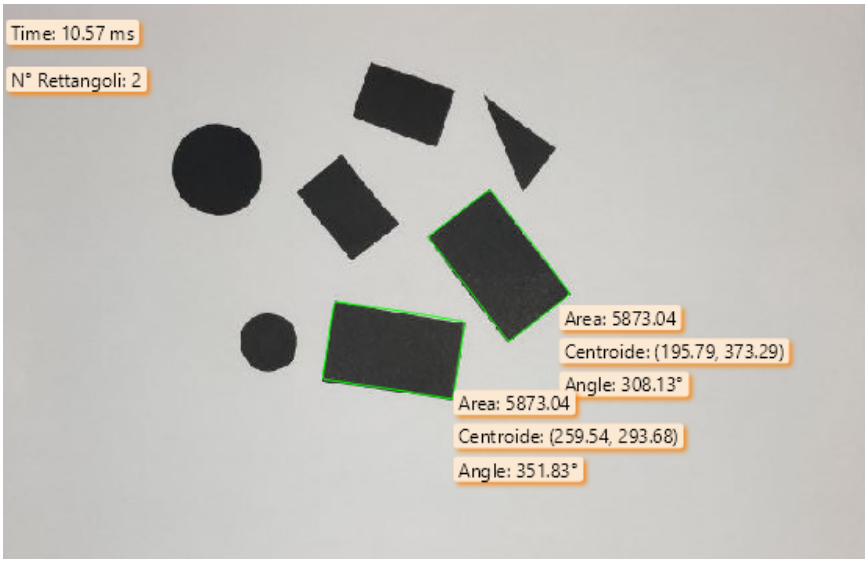


Figura 2.13: Esempio 2 di applicazione della Shape-Based Matching in Halcon per il riconoscimento di rettangoli.

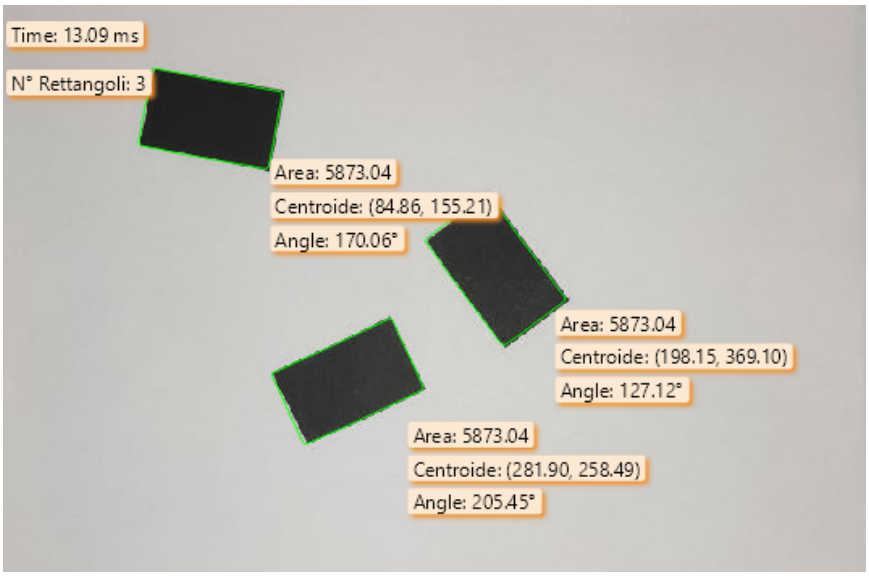


Figura 2.14: Esempio 3 di applicazione della Shape-Based Matching in Halcon per il riconoscimento di rettangoli.

Come nel caso della Blob Analysis, quando la foto è priva di sfocature dovute a movimenti o vibrazioni (motion blur) e l'illuminazione è ottimale, si otterrà un risultato che corrisponde alle aspettative.
Osserviamo ora il caso in cui vi sia un cambio di illuminazione, come riportato in Figura 2.15:

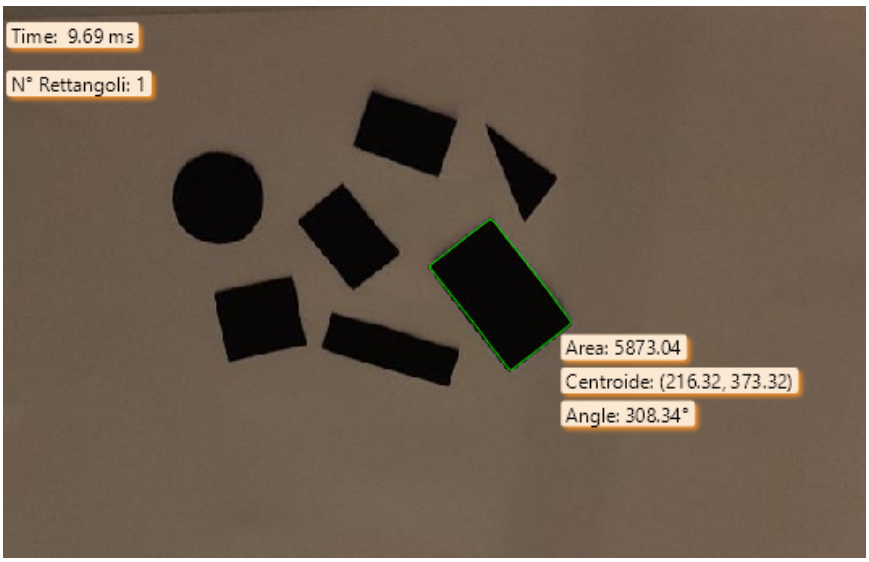


Figura 2.15: Esempio di applicazione dello Shape-Based Matching nel caso di scarsa illuminazione.

È interessante notare che il riconoscimento del rettangolo tramite lo Shape-Based Matching avviene senza problemi anche in presenza di variazioni di illuminazione. Ciò dimostra la notevole robustezza dell'algoritmo alle fluttuazioni di luce e alle ombre, che sono fattori che spesso possono influire negativamente sulle prestazioni di altri metodi di riconoscimento delle forme.
In Figura 2.16 è riportato lo Shape-Based Matching nel caso di motion blur:

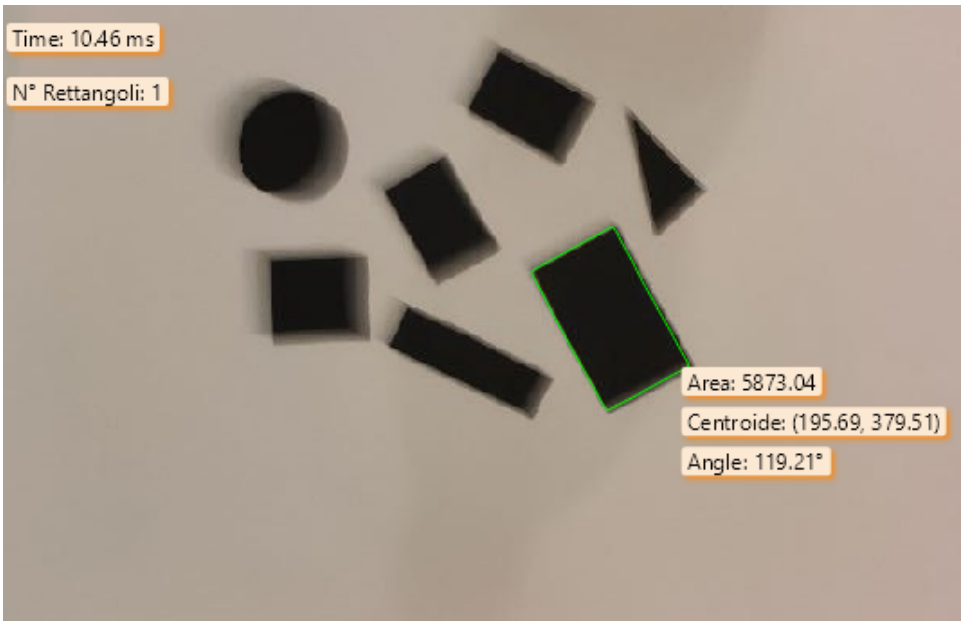


Figura 2.16: Esempio di applicazione dello Shape-Based Matching in Halcon nel caso di motion blur.

Notiamo che il riconoscimento del rettangolo tramite lo Shape-Based Matching avviene senza problemi anche in presenza di sfocature dovute a vibrazioni o movimenti, noti come motion blur. Questo è possibile grazie al fatto che l'algoritmo si basa sui contorni dell'oggetto, piuttosto che sui valori di grigio dell'immagine, rendendolo estremamente robusto alle sfocature (lo stesso vale nel caso delle variazioni di illuminazione), quindi si superano molte delle limitazioni di altri algoritmi basati sulla correlazione di immagini o sulle informazioni di colore.
Alcuni angoli risultano differenti rispetto a quelli ottenuti tramite la Blob Analysis. Ciò è dovuto al fatto che vengono calcolati mediante un metodo distinto. Tuttavia, è possibile notare che tali angoli sono semplicemente ruotati di 180° e pertanto non influenzeranno la presa dell'oggetto in questione da parte del robot.

2.2.3 Risultati ottenuti

Analizzando i risultati ottenuti tramite l'applicazione della tecnica di Shape-Based Matching con Halcon, emergono le seguenti caratteristiche:

- i tempi di esecuzione sono nell'ordine dei 15-20 ms.
  Facendo una media di tutti i tempi di esecuzione rilevati per ogni immagine si è ottenuto che:

  $$\text{Tempo di esecuzione medio} = 15.97 \text{ ms}$$

- la tecnica di Shape-Based Matching implementata in Halcon risulta essere estremamente precisa nell'identificazione degli oggetti in condizioni ambientali ottimali;
- la tecnica di Shape-Based Matching presenta un'elevata robustezza nei confronti delle variazioni ambientali, come le sfocature generate da movimenti o vibrazioni (motion blur) e le variazioni di luminosità. Risulta quindi essere una tecnica molto robusta ed affidabile nella maggior parte delle situazioni;
- è più complessa da implementare in Halcon rispetto alla Blob Analysis.

2.3 Confronto Blob Analysis e Shape-Based Matching

Dai risultati ottenuti è evidente che la tecnica dello Shape-Based Matching implementabile in Halcon è superiore alla Blob Analysis in tutti i campi. In particolare, questa tecnica ha dimostrato una velocità di esecuzione molto più elevata rispetto alla precedente, con un tempo medio di 15.97 ms rispetto ai 40.71 ms della Blob Analysis. Inoltre, la tecnica dello Shape-Based Matching si è dimostrata anche molto più resistente alle variazioni ambientali, grazie alla sua capacità di non subire alcun effetto dai cambiamenti di illuminazione o dalla sfocatura causata da movimentazioni o vibrazioni (motion blur).
Pertanto, per qualsiasi applicazione nell'ambito dei sistemi di visione in robotica, l'utilizzo della tecnica dello Shape-Based Matching rappresenta la scelta ideale, in quanto consente di ottenere risultati precisi e affidabili in tempi notevolmente ridotti rispetto alla Blob Analysis. Questo la rende particolarmente adatta in quei contesti in cui la rapidità di elaborazione è cruciale, come ad esempio nei processi di automazione industriale, dove l'efficienza e la produttività sono fattori di primaria importanza.

Capitolo 3

Deep learning instance segmentation

L'instance segmentation è una tecnica avanzata di computer vision che permette di identificare e separare oggetti all'interno di un'immagine, distinguendo tra diverse istanze dello stesso tipo di oggetto. A differenza della semantic segmentation, che classifica ogni pixel dell'immagine in diverse categorie, l'instance segmentation assegna un'etichetta unica a ciascuna istanza individuale dell'oggetto presente nell'immagine.
L'instance segmentation ha l'obiettivo di sviluppare un algoritmo che performi bene in due ambiti, ovvero la precisione e l'efficienza della segmentazione. Per "precisione" si intende l'accuratezza nel processo di localizzazione e riconoscimento degli oggetti presenti in immagini, con lo scopo di distinguere una vasta gamma di categorie di oggetti in ambienti reali. Permette inoltre di identificare istanze di oggetti appartenenti alla stessa classe, che possono presentare variazioni nell'aspetto all'interno della stessa classe. L'"efficienza" della segmentazione si riferisce al costo computazionale dell'algoritmo, ossia tempi di elaborazione brevi, requisiti di memoria/archiviazione accettabili e un minor carico sui processori.
Nella rilevazione di oggetti attraverso l'instance segmentation, è cruciale rappresentare accuratamente le caratteristiche degli oggetti stessi. I modelli di deep learning, come ad esempio le reti neurali convoluzionali profonde (Deep CNN), sono in grado di apprendere tali caratteristiche da immagini con diversi livelli di astrazione [8][10].

3.1 Rete neurale convoluzionale CNN

Una rete neurale convoluzionale (CNN o ConvNet) è una tipologia di architettura di rete neurale che si basa sul deep learning, la quale permette di apprendere autonomamente dalle informazioni fornite, evitando la necessità di effettuare manualmente la selezione delle caratteristiche di interesse.

Le CNN sono particolarmente utili per riconoscere schemi all'interno di immagini, per il riconoscimento di oggetti, volti, e ambienti. Inoltre, esse possono essere utilizzate efficacemente per la classificazione di dati che non sono immagini, come ad esempio dati audio, serie storiche e segnali.
Le applicazioni che necessitano di funzionalità di riconoscimento di oggetti e di visione artificiale, come ad esempio veicoli autonomi e sistemi di riconoscimento facciale, dipendono ampiamente dall'utilizzo delle CNN.
L'utilizzo delle reti neurali convoluzionali (CNN) è molto diffuso nel campo del deep learning grazie principalmente a tre fattori [16]:

- le CNN eliminano la necessità di estrarre manualmente le caratteristiche (feature) poiché le apprendono direttamente dal dataset di addestramento;
- le CNN producono risultati di riconoscimento altamente precisi grazie alla loro capacità di rappresentare le caratteristiche locali e globali delle immagini;
- le CNN possono essere riaddestrate per nuove attività di riconoscimento, permettendo agli utenti di basarsi su reti preesistenti e ottenere risultati migliori con minori sforzi di addestramento.

Un esempio di workflow di deep learning, dove vengono usate reti CNN è riportato in Figura 3.1:

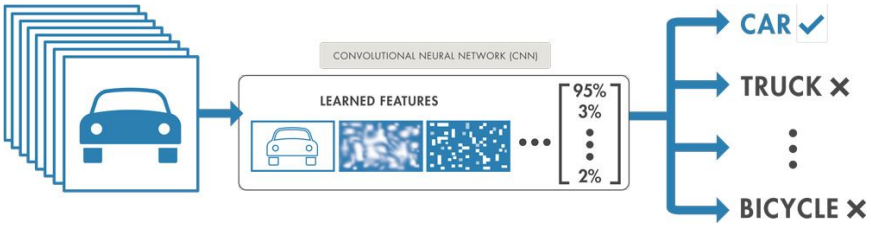


Figura 3.1: Workflow deep learning utilizzando reti neurali convoluzionali [16].

Notiamo che le immagini vengono trasferite alla CNN, la quale apprende automaticamente le caratteristiche (feature) e classifica gli oggetti.
Una rete neurale convoluzionale (CNN) è composta da numerosi strati (decine o centinaia) che apprendono caratteristiche diverse di un'immagine.
Durante l'addestramento, l'immagine viene sottoposta a filtri con diverse risoluzioni e l'output di ogni filtro convoluzionale viene utilizzato come input per il successivo strato. I filtri possono iniziare con caratteristiche molto semplici, come la luminosità o i bordi, e diventare sempre più complessi, includendo feature che definiscono in modo univoco l'oggetto che si sta cercando di riconoscere.

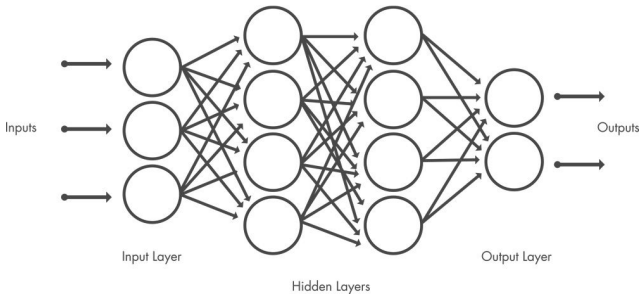


Figura 3.2: Layer di una CNN [16].

Una CNN è costituita da un layer di input, un layer di output e tanti layer intermedi nascosti (come riportato in Figura 3.2).

I layer in questione svolgono operazioni che modificano i dati al fine di imparare le caratteristiche specifiche dei dati stessi. Alcuni dei layer più comuni includono la convoluzione, l'attivazione (o ReLU) e il pooling:

- la convoluzione prevede l'applicazione di una serie di filtri convoluzionali all'input, ognuno dei quali rileva specifiche caratteristiche dalle immagini;

- l'attivazione, o ReLU, accelera e migliora l'addestramento mappando i valori negativi dei neuroni della rete a zero, mantenendo invece quelli positivi. Questa operazione viene a volte chiamata "attivazione", in quanto solo le caratteristiche attivate (neuroni attivi) vengono inviate al layer successivo;

- il pooling semplifica l'output attraverso un downsampling (sottocampionamento) non lineare, riducendo il numero di parametri che la rete deve imparare.

Queste operazioni vengono eseguite su decine o centinaia di layer, ognuno dei quali impara a rilevare diverse caratteristiche.
Come una rete neurale tradizionale, una CNN ha neuroni con pesi e bias che vengono appresi e aggiornati durante l'addestramento. Tuttavia, in una CNN i pesi e i bias sono gli stessi per tutti i neuroni nascosti in uno specifico layer, il che significa che tutti questi neuroni rilevano la stessa feature in diverse parti dell'immagine, come ad esempio i bordi o le macchie. Questo rende la rete in grado di riconoscere oggetti indipendentemente dalla loro posizione nell'immagine. Dopo aver appreso le feature in diversi layer, la CNN passa alla fase di classificazione.
Il penultimo layer è completamente connesso e produce un vettore di dimensione K, dove K rappresenta il numero di classi che la rete è in grado di prevedere. Questo vettore contiene le probabilità per ogni classe di immagini classificate.
L'ultimo layer dell'architettura CNN utilizza una funzione di attivazione soft-

max[3] per fornire l'output della classificazione.
Le reti neurali convoluzionali vengono addestrate utilizzando un vasto set di dati che può comprendere centinaia, migliaia o anche milioni di immagini. Quando si lavora con quantità elevate di dati e architetture di rete complesse, l'elaborazione richiede molto tempo, ma l'utilizzo di GPU può accelerare notevolmente il processo di addestramento del modello [16].

3.2 Workflow del Deep Learning Instance Segmentation

L'applicazione dei modelli di deep learning in Halcon segue principalmente il workflow riportato in Figura 3.3):

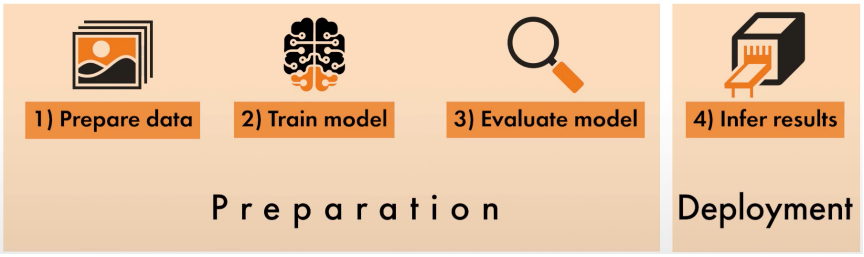


Figura 3.3: Workflow deep learning in Halcon [14]

1) **Prepare data**: il primo passo per creare un modello di deep learning per risolvere un problema di visione artificiale consiste nell'acquisire e preparare i dati. Innanzitutto, occorre acquisire le immagini pertinenti al problema che si intende risolvere. Successivamente, tali immagini vanno etichettate. Poiché i dati devono essere compatibili con i parametri specifici del modello, quali ad esempio la dimensione dell'immagine e la gamma di valori di grigio, è necessario pre-elaborare le immagini.

2) **Train model**: dopo aver preparato i dati, il passo successivo è l'addestramento del modello di deep learning per l'applicazione. In questo

[3]Una funzione softmax, o funzione esponenziale normalizzata, è una generalizzazione di una funzione logistica che mappa un vettore di valori reali arbitrari in un vettore K-dimensionale $\mathbf{z}$ di valori reali arbitrari in un vettore K-dimensionale $\sigma(\mathbf{z})$ di valori compresi in un intervallo (0,1) la cui somma è 1. La funzione è data da:

$$\sigma : \mathbb{R}^K \to \left\{ z \in \mathbb{R}^K \;\middle|\; z_i > 0, \sum_{i=1}^{K} z_i = 1 \right\}$$

$$\sigma(\mathbf{z})_j = \frac{e^{z_j}}{\sum_{k=1}^{K} e^{z_k}} \quad \text{per } j = 1, \ldots, K$$

contesto, Halcon offre diverse reti pre-addestrate che possono essere ulteriormente migliorate attraverso l'uso dei dati etichettati.

3) **Evaluate model**: una volta completato l'addestramento del modello sui propri dati, è necessario valutarne le prestazioni per determinare se sia adeguato all'applicazione in questione.

4) **Infer Results**: una volta ottenuti i risultati desiderati, è possibile utilizzare il modello per analizzare nuove immagini provenienti, ad esempio, da un feed di telecamera in tempo reale. Tale operazione viene definita inferenza.

I primi tre step sono fondamentali per preparare il modello di deep learning da usare per la nostra applicazione. Per questa fase, Halcon mette a disposizione programmi di esempio HDevelop con spiegazioni dettagliate e informazioni di base.

3.2.1 Preparazione dati

L'obiettivo del progetto è quello di riconoscere gli oggetti e la loro forma geometrica, in particolare distinguere se si tratti di un rettangolo o di un triangolo.
La preparazione dei dati si divide in tre fasi:

- lettura e suddivisione dei dati etichettati;
- creazione di un modello di deep learning per il riconoscimento degli oggetti;
- pre-elaborazione dei dati etichettati in preparazione dell'addestramento che utilizzerà il modello appena creato.

Per iniziare dobbiamo leggere un dataset, che è stato realizzato etichettando gli oggetti con il **Deep Learning Tool** di Halcon, utilizzando il codice:

```
read_dict (DatasetFilename, [], [], DLDataset)
```

Deep Learning Tool per l'Instance Segmentation

Il Deep Learning Tool di Halcon rappresenta un'importante risorsa per la creazione di dataset utili alla creazione di modelli di deep learning. Grazie a questo strumento è possibile etichettare (labelling) in modo accurato ogni oggetto, assicurando la qualità del dataset e quindi del modello di deep learning. Infatti, l'etichettatura precisa dei dati di input è fondamentale per garantire la capacità del modello di riconoscere correttamente gli oggetti di interesse.
L'interfaccia grafica del Deep Learning Tool è riporata in Figura 3.4:

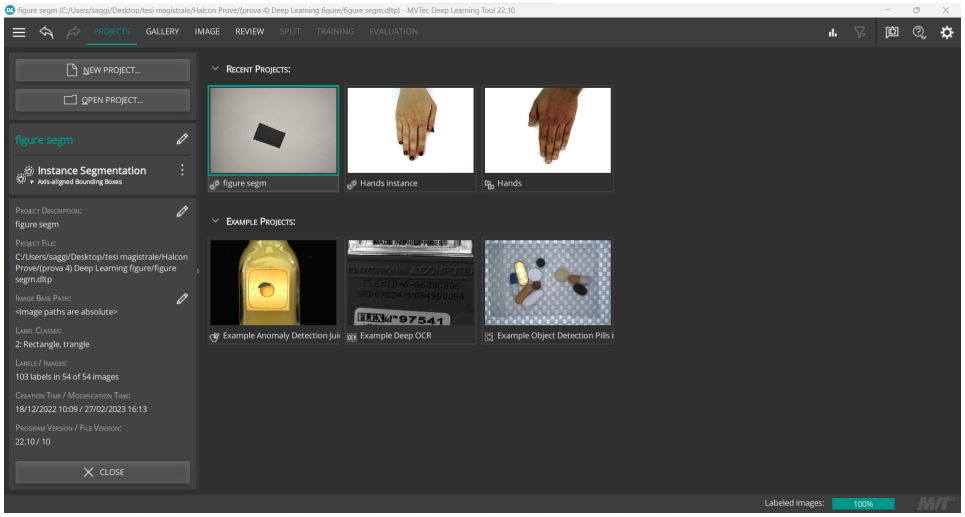


Figura 3.4: GUI del Deep Learning Tool di Halcon.

Per la creazione del dataset da utilizzare per l'addestramento del modello di deep learning sono state utilizzate 54 foto, realizzate con la camera Logitech C270 HD, nelle quali sono presenti rettangoli e triangoli. Utilizzando lo strumento **"Draw a new polygon"**, sono state create per ogni oggetto nelle immagini delle Classi: **Rectangle** e **Triangle**.
Un esempio di questa operazione di labelling è riportato in Figura 3.5:

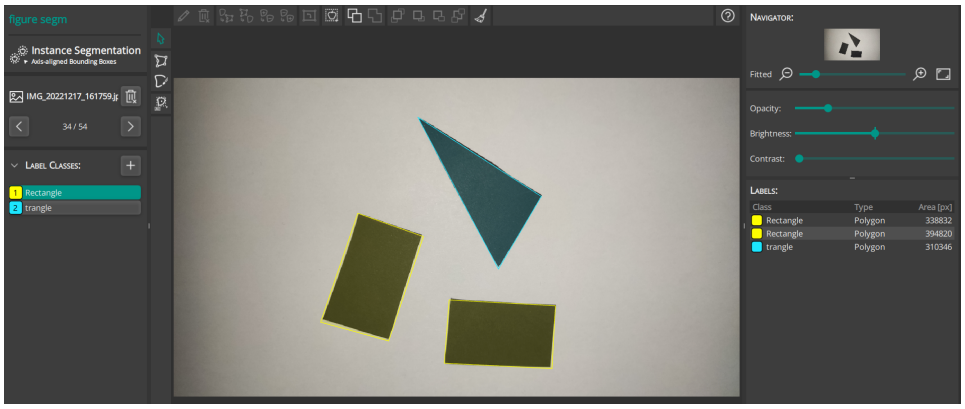


Figura 3.5: GUI del Deep Learning Tool di Halcon.

Dopo aver creato ed etichettato accuratamente il dataset con il Deep Learning Tool di Halcon, quest'ultimo è stato esportato per essere utilizzato nell'addestramento del modello di deep learning, al fine di garantire la precisione e l'affidabilità nel riconoscimento degli oggetti di interesse.

Pre-elaborazione dei dati

Dopo aver letto il dataset, esso verà suddiviso in tre sottoinsiemi, utilizzando l'operatore **"split_dl_dataset"**:

```
split_dl_dataset (DLDataset, SplitPercentageTrain,
SplitPercentageValidation, [])
```

I parametri denominati **SplitPercentageTrain** e **SplitPercentageValidation** consentono di specificare le percentuali di dati che desideriamo utilizzare rispettivamente per il training e per la validazione. In questo caso specifico, tali parametri sono impostati al 70% e al 15%, il che significa che il rimanente 15% dei dati sarà utilizzato per l'operazione di test. I dati di addestramento verranno utilizzati direttamente come input per l'operatore di training, mentre quelli di validazione saranno utilizzati per verificare e visualizzare il progresso del training e adattare gli hyperparameter, ovvero quei parametri che non vengono appresi dal modello ma devono essere specificati dall'utente. Infine, i dati di test vengono utilizzati per una valutazione indipendente dopo il training. La procedura di "split_dl_dataset" suddivide casualmente il dataset utilizzando le percentuali fornite, mantenendo la distribuzione di classe del dataset. E' importante sottolineare che i sottoinsiemi devono essere rappresentativi dell'intero dataset e devono essere indipendenti tra di loro. Notiamo che come parametro in input abbiamo "DLDataset", ossia un dizionario che contiene la directory in cui si trovano le immagini, i nomi delle classi, i loro rispettivi ID e i campioni. Un esempio di DLDataset di Halcon è riportato in Figura 3.6.

Handle Inspect: DLDataset

Keys	Values
DLDataset	H429887CF480 (dict)
'class_ids'	[0, 1]
'class_names'	['Rectangle', 'trangle']
'image_dir'	'C:/Users/saggi/Desktop/Immagini figure deep/'
'samples'	[H429887CF3C0 (dict), H429887CF1E0 (dict), H429887CF3F0 (dict),…
[0]	H429887CF3C0 (dict)
'image_id'	1
'image_file_name'	'1.jpg'
'mask'	iconic: region array (2)
'bbox_label_id'	[0, 1]
'bbox_row1'	[579.457, 460.054]
'bbox_col1'	[866.025, 1841.72]
'bbox_row2'	[1190.12, 1623.38]
'bbox_col2'	[1749.61, 2346.62]
'split'	'validation'
'dlsample_file_name'	'1_dlsample.hdict'
[1]	H429887CF1E0 (dict)
[2]	H429887CF3F0 (dict)
[3]	H429887CF420 (dict)
[4]	H429887CF780 (dict)

Figura 3.6: Esempio di DLDataset di Halcon.

Il dizionario di ogni campione contiene il nome del file dell'immagine individuale e le etichette, che nel nostro caso, consistono nelle coordinate del riquadro di delimitazione (bounding box) del rettangolo o del triangolo e

nell'ID di classe.
Durante la fase di pre-elaborazione vengono creati dei DLSample, i quali contengono direttamente le immagini pre-elaborate. I DLSample vengono scritti su disco e i nomi dei file vengono quindi memorizzati anche nel DL-Dataset [5].
Successivamente, dobbiamo creare il modello di deep learning per il rilevamento degli oggetti. Una delle prime cose da considerare è la dimensione dell'immagine che si vuole utilizzare, poiché è un compromesso tra velocità e prestazioni. Nel nostro caso è stata selezionata una larghezza (ImageWidth) di 512 pixel e un'altezza (ImageHeight) di 384 pixel dell'immagine. In base a questa dimensione, possiamo quindi utilizzare la funzione **"determine_dl_model_detection_param"**, che ci permette di stimare automaticamente alcuni parametri avanzati del modello per il dataset passato come input:

```
determine_dl_model_detection_param(DLDataset, ImageWidth,
ImageHeight, dict{}, DLDetectionModelParam)
```

Questa procedura permette di ottimizzare le prestazioni di training e di inference.
Si è creato poi il modello, utilizzando la funzione **"create_dl_model_detection"**:

```
create_dl_model_detection(Backbone,
|DLDataset.class_ids|, DLDetectionModelParam,
DLModelHandle)
```

Il parametro "Backbone" in Halcon si riferisce ad una rete neurale pre-addestrata utilizzata come estrattore di caratteristiche per una rete neurale personalizzata. Il backbone è il primo strato della rete, che consente di estrarre le caratteristiche dell'immagine. Questo approccio consente poi di utilizzare le caratteristiche estratte per altre attività, come la classificazione o la rilevazione degli oggetti, senza dover addestrare la rete neurale da zero. Il backbone è quindi la base a cui aggiungere ulteriori strati per adattare la rete ai compiti specifici. In Halcon, sono disponibili diverse reti pre-addestrate, e nel nostro caso abbiamo utilizzato la rete compatta "pretrained_dl_classifier_compact.hdl", la quale viene consigliata nel manuale degli operatori di Halcon, poiché è adatta per molte applicazioni [5]. Come ultimo passaggio in questa fase si è pre-processato il dataset. Si procede utilizzando la funzione **"create_dl_preprocess_param_from_model"**, ottenendo così un dizionario con i parametri di preprocessing appropriati:

```
create_dl_preprocess_param_from_model (DLModelHandle,
'none', 'full_domain', [], [], [], DLPreprocessParam)
```

Successivamente si effettua la vera e propria operazione di preprocessing del dataset, utilizzando l'operatore **"preprocess_dl_dataset"**:

```
preprocess_dl_dataset (DLDataset, OutputDirPreprocessing,
DLPreprocessParam, dict{overwrite_files: 'auto'},
DLDatasetFileName)
```

In questo modo vengono letti e scritti molti dati, quindi quest'operazione potrebbe richiedere del tempo, soprattutto lavorando con storage basati su rete.

3.2.2 Addestramento del modello

Panoramica generale della logica di Training

Con il modello appena creato vogliamo rilevare gli oggetti, trovare le diverse istanze in un'immagine e assegnarle ad una classe. È possibile che queste ultime possano anche sovrapporsi parzialmente. Ciò è illustrato nello schema riportato in Figura 3.7:

Figura 3.7: Un possibile esempio per la rilevazione degli oggetti. All'interno dell'immagine di input si trovano tre istanze, le quali vengono assegnate ad una classe [5].

La rilevazione degli oggetti si compone di due operazioni differenti, ossia trovare le istanze e classificarle. Per farlo, utilizziamo una rete combinata composta da tre sezioni principali:

- la prima sezione della rete, denominata "backbone", è costituita da una rete di classificazione pre-addestrata, la cui funzione è generare diverse mappe caratteristiche degli oggetti. In queste mappe, il layer relativo alla classificazione non è presente. Le mappe caratteristiche contengono informazioni codificate di diversi tipi e su scale differenti a seconda della loro profondità all'interno della rete. Pertanto, le mappe caratteristiche con la stessa larghezza e altezza appartengono allo stesso livello;

- nella seconda sezione, vengono prese diverse mappe caratteristiche di livelli diversi e combinate insieme. Ciò consente di ottenere mappe caratteristiche contenenti informazioni sia di livelli inferiori che superiori, le quali saranno utilizzate nella terza sezione. Questa seconda sezione è nota anche come "piramide delle caratteristiche" ed insieme alla prima, costituisce la rete "feature pyramid network";

- la terza sezione è costituita da reti addizionali, le quali utilizzano le mappe caratteristiche selezionate come input per apprendere come localizzare e classificare possibili oggetti. Inoltre, questa sezione si occupa anche della riduzione dell'overlap (sovrapposizione) tra i bounding box.

Una panoramica delle tre sezioni è mostrata nella Figura 3.8:

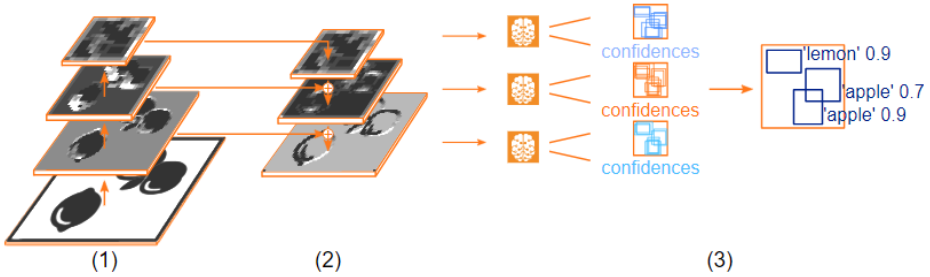


Figura 3.8: Una panoramica schematica delle tre sezioni menzionate: (1) Il backbone. (2) Le mappe caratteristiche del backbone vengono combinate e si generano nuove mappe caratteristiche. (3) Reti aggiuntive, che apprendono come localizzare e classificare potenziali oggetti. La sovrapposizione dei bounding box viene soppressa [5].

Focalizziamoci sulla terza sezione, ovvero la rilevazione degli oggetti. Il primo compito da svolgere consiste nel trovare un bounding box appropriato per ogni singola istanza, poiché questo permette di individuare la posizione dell'oggetto all'interno dell'immagine. Per ottenere ciò, la rete genera dei bounding box di riferimento e impara come modificarli per adattarli al meglio alle istanze presenti nell'immagine. Anche se tutti i bounding box sono di forma rettangolare, possono avere dimensioni e rapporti laterali differenti. Di conseguenza, la rete deve apprendere dove tali bounding box possono essere posizionati e quale forma possono assumere per individuare correttamente le istanze. Nel metodo utilizzato in Halcon, la rete propone, per ogni pixel di ogni mappa caratteristica della "feature pyramid", un insieme di bounding box di riferimento. La rete neurale predice gli spostamenti necessari per modificare i bounding box di riferimento, in modo da ottenere dei riquadri di delimitazione più precisi ed adattabili alle possibili istanze presenti nell'immagine. Questo processo di apprendimento avviene attraverso il confronto tra i bounding box proposti dalla rete e quelli corrispondenti al ground truth, che rappresenta l'informazione precisa sulla posizione delle istanze nell'immagine. L'illustrazione presente nella figura sottostante (3.9) aiuta a comprendere meglio questo concetto. Maggiore è la precisione con cui i bounding box di riferimento rappresentano la forma dei diversi bounding box del ground truth, più facilmente la rete neurale può apprenderli e migliorare la propria capacità di individuare le istanze.

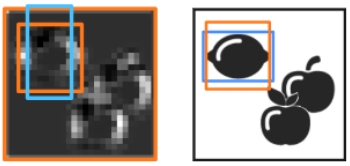


Figura 3.9: (1) La rete modifica il bounding box di riferimento (azzurro) al fine di prevederne uno che si adatta meglio a quello di riferimento (arancione). (2) Durante l'addestramento la bounding box prevista (arancione) viene confrontata con la bounding box del ground truth che ha la sovrapposizione maggiore (blu), così che la rete possa imparare le modifiche [5].

Con i bounding box otteniamo la localizzazione di una possibile istanza, ma questa non è stata ancora classificata. Pertanto, il secondo compito consiste nella classificazione del contenuto della parte dell'immagine all'interno dei bounding box [5]. La classificazione basata sul deep learning è un metodo che assegna a un'immagine un insieme di valori di confidenza, i quali indicano la probabilità che l'immagine appartenga a ciascuna delle classi distinte. Pertanto, la classificazione consiste nell'assegnare a un'immagine una classe specifica da un dato insieme di classi, considerando solamente la previsione migliore. La logica della tecnica di Classification verrà approfondita nel Capitolo 4. Un esempio esplicativo è riportato in Figura 3.10:

Figura 3.10: Un possibile esempio di classificazione, in cui la rete neurale distingue tre classi. L'immagine di input riceve valori di confidenza assegnati per ciascuna delle tre classi distinte: 'mela' 0,85, 'limone' 0,12 e 'arancia' 0,03. La previsione principale ci dice che l'immagine è stata riconosciuta come "mela" [5].

La rete neurale di un classificatore è composta da un numero specifico di strati o filtri, i quali sono disposti e connessi in modo preciso. Ogni livello rappresenta un elemento costituente che svolge attività specifiche, agendo come un contenitore che riceve un input, lo trasforma attraverso una funzione, e restituisce l'output (chiamato anche feature map) al livello successivo. In questo modo, i diversi strati possono svolgere diverse funzioni a seconda del loro tipo. Inoltre, molti livelli o filtri hanno dei pesi, ovvero dei parametri che vengono modificati durante l'addestramento della rete, noti come pesi del filtro. Per addestrare una rete per un compito specifico, viene aggiunta

una funzione di perdita. Esistono diverse funzioni di perdita a seconda del compito, ma tutte funzionano secondo il seguente principio: una funzione di perdita confronta la previsione della rete con le informazioni fornite, ad esempio rileva ciò che dovrebbe essere presente in un'immagine (e, se applicabile, anche dove), e penalizza le deviazioni. Pertanto, addestrando la rete per compiti specifici, ci si sforza di minimizzare la perdita (una funzione di errore) della rete, nella speranza che ciò migliori anche le prestazioni. Questa ottimizzazione viene effettuata calcolando il gradiente e aggiornando di conseguenza i parametri dei diversi strati (pesi del filtro). Tale operazione viene ripetuta iterando più volte sui dati di addestramento. Per addestrare tutti i pesi dei filtri da zero, sono necessarie molte risorse, per cui si può approfittare del fatto che i primi strati rilevano le caratteristiche di basso livello come bordi e curve, mentre la mappa caratteristica dei successivi strati è più piccola, ma rappresenta caratteristiche più complesse. Per una grande rete, infatti, le caratteristiche di basso livello sono abbastanza generali in modo che i pesi dei corrispondenti strati non cambino molto tra le diverse attività. Ciò porta a una tecnica chiamata **transfer learning**: si prende una rete già addestrata e la si riaddestra per una specifica attività, beneficiando di pesi dei filtri già adattati per i livelli inferiori, di conseguenza sono necessarie molte meno risorse. Sebbene in generale la rete dovrebbe essere più affidabile quando addestrata su un set di dati più ampio, la quantità di dati necessari per il riaddestramento dipende anche dalla complessità dell'attività. Uno schema di base che mostra il workflow della tecnica di transfer learning è riportato in Figura 3.11:

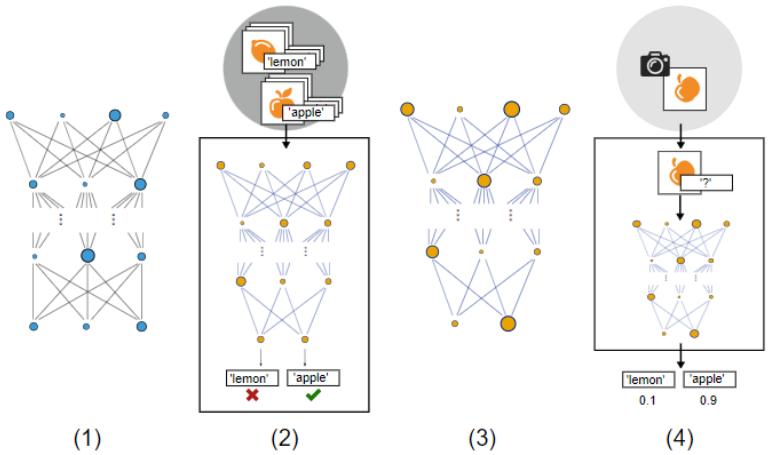


Figura 3.11: Schema base del transfer learning con l'ausilio della classificazione. (1) Viene letto un network pre-addestrato. (2) Fase di addestramento, il network viene addestrato con i dati di addestramento. (3) Il modello addestrato con nuove capacità. (4) Fase di inferenza, il network addestrato fa inferenza su nuove immagini [5].

Vi sono dei parametri aggiuntivi, definiti **hyperparameter**, che influenzano il training, ma che non vengono appresi direttamente durante l'addestramento regolare. Questi parametri devono essere impostati prima di iniziare l'allenamento. Gli hyperparameter più importanti sono il **batch_size, epochs e learning_rate**.
Come già menzionato, addestrare una rete neurale implica modificare i pesi dei filtri (o neuroni) in modo tale che la rete produca risultati migliori per il compito specifico che deve svolgere. Questo processo di aggiornamento dei pesi viene eseguito attraverso un'iterazione di esempi di addestramento, in cui la rete cerca di minimizzare una funzione di perdita, che misura la discrepanza tra le sue previsioni e le risposte attese. Ciò viene fatto attraverso l'aggiornamento dei pesi dei filtri, che avviene in modo graduale e ripetitivo, in base alla quantità di errore della previsione della rete rispetto alla risposta attesa. Per farlo, viene presa una certa quantità di dati dal set di addestramento e per questo sottoinsieme, viene calcolato il gradiente della perdita e la rete viene modificata aggiornando i pesi dei filtri di conseguenza. Questa operazione viene ripetuta con il successivo sottoinsieme di dati fino a quando tutti i dati di addestramento non sono stati elaborati. Questi dati di addestramento sono chiamati batch e la loro dimensione viene detta **"batch_size"**, la quale determina il numero di dati presi in un batch e quindi elaborati insieme.
Un'iterazione completa su tutti i dati di addestramento è chiamata epoca ed è vantaggioso iterare diverse volte sui dati di addestramento. Il numero di iterazioni è definito **"epochs"**, le quali quindi determinano quante volte l'algoritmo compie un'iterazione sul set di addestramento. Come già detto, dopo ogni calcolo del gradiente di perdita, i pesi dei filtri vengono aggiornati (ciò viene fatto per batch e quindi tramite l'algoritmo di discesa stocastica del gradiente SGD). Per questo aggiornamento, si impone un importante hyperparameter: il **"learning_rate"**, che determina il peso del gradiente sugli argomenti aggiornati della funzione di perdita (i pesi dei filtri), ossia quanto vengono aggiornati i parametri del modello durante l'addestramento.
Un metodo per migliorare le prestazioni di una rete è aggiungere dati per l'addestramento, anche se di solito, una piccola frazione supplementare non cambierà sensibilmente le prestazioni totali.
Per monitorare la progressione dell'addestramento di un algoritmo di classificazione è spesso utile visualizzare una misura di convalida, come ad esempio l'errore sui campioni di un lotto. Tuttavia, poiché i campioni possono essere diversi tra loro, la difficoltà dell'attività di classificazione può variare, il che significa che la rete neurale può funzionare meglio o peggio su un particolare lotto rispetto ad un altro. Di conseguenza, la misura di convalida potrebbe non cambiare uniformemente durante le iterazioni, ma in generale dovrebbe migliorare.
La regolazione degli hyperparameter può aiutare a migliorare ulteriormente la misura di convalida [5]. La Figura 3.12 mostra i possibili scenari:

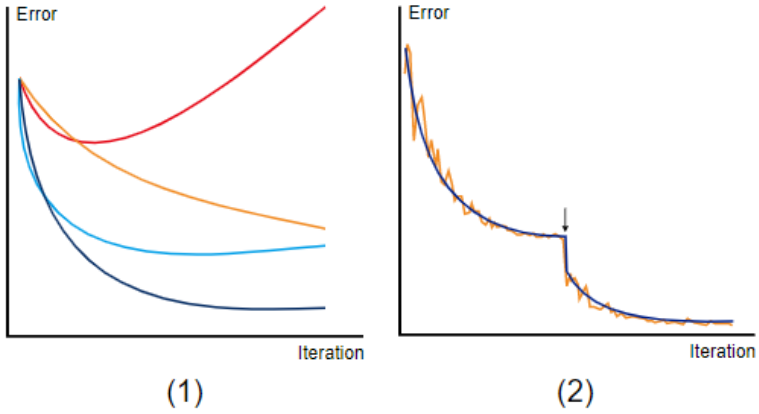


Figura 3.12: Esempio di variazione dei valori di perdita durante l'addestramento. (1) Andamento di diversi valori di perdita (loss) all'aumentare delle iterazioni: in blu scuro abbiamo il learning rate ideale, in rosso un learning rate troppo alto, in azzurro un learning rate alto, in arancione un learning rate basso. (2) Caso ideale in cui il valore di "learning_rate", dopo un determinato numero di iterazioni, viene ridotto. In arancione abbiamo l'errore di addestramento, in blu scuro invece l'errore di validazione. La freccia indica l'iterazione in cui viene ridotto il learning rate [5].

Notiamo che se impostiamo un valore di learning rate iniziale troppo basso, l'addestramento richiederà molto tempo. D'altra parte, se impostiamo il tasso di apprendimento iniziale troppo alto, potrebbe verificarsi un errore. Se la curva sembra buona all'inizio, ma poi si appiattisce, dovremmo provare a diminuire il tasso di apprendimento. In questo modo, il modello potrebbe migliorare ulteriormente.

Workflow della fase di Training in Halcon

Per il nostro studio sono stati imposti inizialmente i seguenti hyperparameter prima di procedere all'operazione di training:

- numero di epoche imposto a 60:

```
epochs := 60
```

- è stata imposta 10 come batch_size, la quale specifica il numero di campioni che sono processati simultaneamente:

```
TrainingBatchSizeMax := 10
```

- Il learning rate è stato imposto a 0.001:

```
Learning_rate := 0.001
```

Va sottolineato che per l'operazione di training è stata utilizzata la scheda grafica dedicata NVIDIA GTX 1050 N17P-G0-K1, con memoria di 3 GB, perché molto più veloce della CPU. Dopo aver definito gli hyperparameter più importanti occorre generare il dizionario per il training e successivamente addestrare il modello. Queste due operazioni vengono fatte rispettivamente con le funzioni **"create_dl_train_param"**:

```
create_dl_train_param (DLModelHandle, epochs, 1, 'true',
42, 'serialize', SerializationStrategy, TrainParam)
```

e **"train_dl_model"**:

```
train_dl_model (DLDataset, DLModelHandle, TrainParam, 0,
TrainResults, TrainInfos, EvaluationInfos)
```

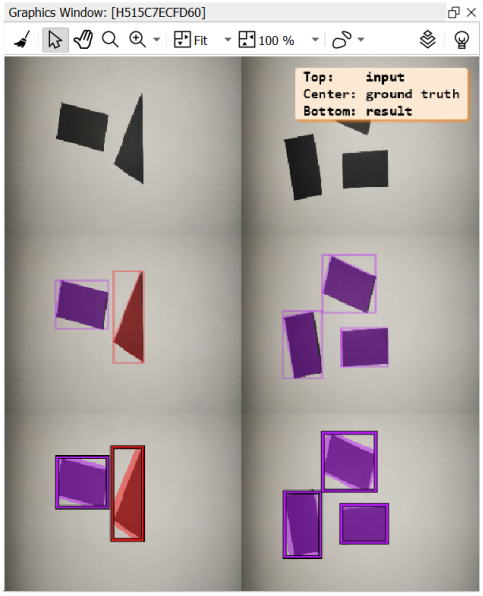


Figura 3.13: Finestra di visualizzazione durante il training.

Dopo aver eseguito tali operazioni, è stato possibile avviare il vero e proprio processo di training. Utilizzando Halcon, verrà visualizzata la finestra seguente, illustrata in Figura 3.13. È importante notare che nella parte superiore (input) sono presenti le immagini pre-elaborate, al centro (ground truth) sono riportati i dati etichettati e nella parte inferiore (result) si possono osservare i risultati attuali del training, in cui il sistema cerca di adattare i bounding box del dataset. Idealmente, è possibile osservare come il modello impari gradualmente ad individuare le istanze.

Vi è poi un'ulteriore finestra di visualizzazione in cui è possibile vedere alcune informazioni sul progresso e sui parametri del training. Vengono di seguito riportati due grafici significativi: l'andamento dei valori di perdita e del mean_ap al variare delle epoche. Il mean_ap, ossia la precisione media, è una misura tra 0 e 1 che rappresenta quanto bene i bounding box predetti si sovrappongono a quelli del ground truth (ossia quelli del dataset). I due grafici ottenuti sono riportati nelle Figure 3.14 e 3.15.

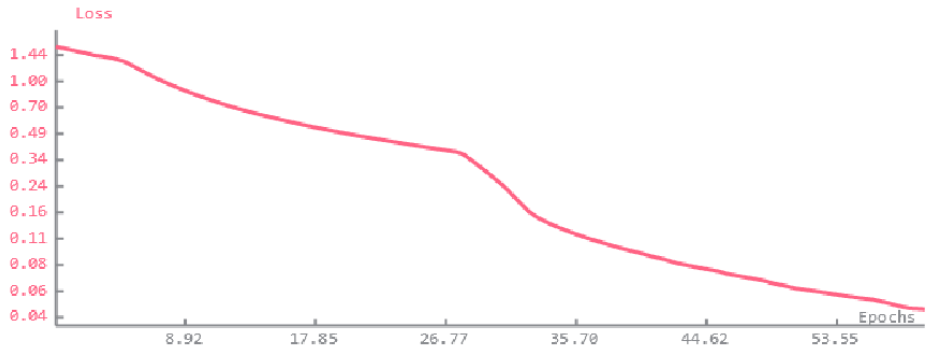


Figura 3.14: Andamento dei valori di perdita al variare delle epoche.

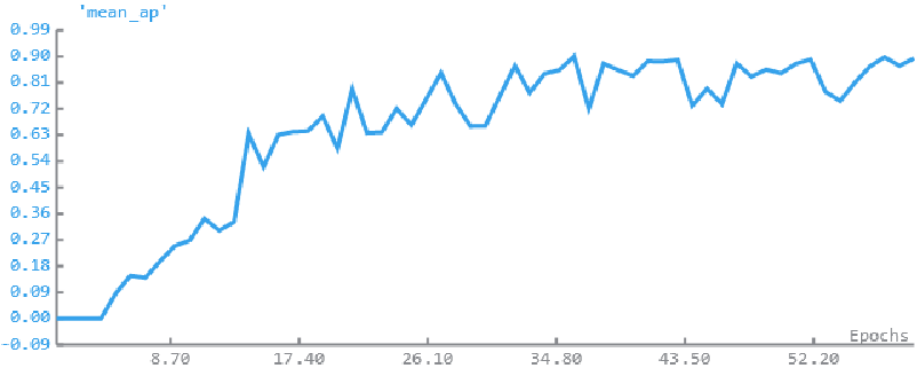


Figura 3.15: Andamento del mean_ap al variare delle epoche.

Durante il training, l'ultimo e il miglior modello, in base alla migliore mean_ap di validazione, vengono scritti sui file in ouput. La durata totale del training con 60 epoche è stata di 216 secondi, con un mean_ap massimo di 0.910.

3.2.3 Miglioramento del modello

Per l'addestramento del modello uno dei parametri più importanti, oltre ai valori di perdita (loss), è il mean_ap. Dalla Figura 3.15 si può notare come il mean_ap si stabilizzi dopo un certo numero di epoche. Di conseguenza, è stata effettuata un'analisi di convergenza per determinare il numero di epoche necessarie per ottenere un modello di deep learning valido in un tempo ridotto. Un'analisi di convergenza è un processo che consiste nel monitorare l'andamento di una grandezza durante l'addestramento del modello stesso, al fine di individuare il momento in cui la grandezza inizia a stabilizzarsi e quindi il modello smette di migliorare significativamente. L'analisi di convergenza è importante in quanto consente di individuare il numero di epoche necessario per addestrare un modello di deep learning in modo efficace ed

efficiente. Se si addestra il modello per un numero eccessivo di epoche, il modello potrebbe andare in overfitting, ovvero adattarsi troppo ai dati di addestramento e perdere la capacità di generalizzare su nuovi dati. Al contrario, se si addestra il modello per un numero troppo ridotto di epoche, il modello potrebbe non essere sufficientemente addestrato e quindi non avere una buona capacità di generalizzazione.
L'analisi a convergenza ha portato al grafico riportato in Figura 3.16:

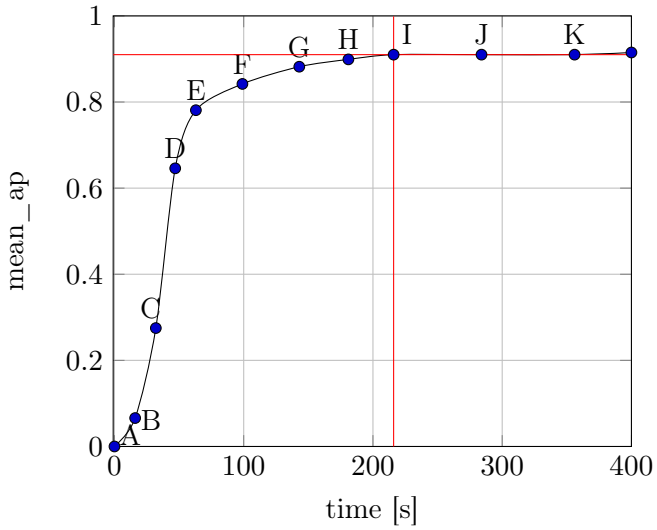


Figura 3.16: Grafico ottenuto con un'analisi di convergenza.

Per ottenere questo grafico si è aumentato progressivamente il numero di epoche (di conseguenza anche il tempo di addestramento) e si è misurato il mean_ap, ossia un valore tra 0 ed 1 che indica la precisione con cui sono stati predetti i bounding box rispetto a quelli del ground truth. I dati ricavati da questa operazione sono riportati in tabella 3.1.
Possiamo osservare che il valore di mean_ap inizia a stabilizzarsi dal nodo I e raggiunge un valore molto vicino a 1. Questo nodo corrisponde a 60 epoche, ovvero ad un tempo di addestramento di 216 secondi. Il numero di epoche ideale, quindi, sembra essere 60, che consente di ottenere un alto valore di mean_ap, senza richiedere un tempo di addestramento troppo lungo.
Per ottenere i dati presenti nella tabella, abbiamo utilizzato la scheda grafica dedicata NVIDIA GTX 1050 N17P-G0-K1 con una memoria di 3 GB. Inoltre, si è deciso di testare le capacità della CPU, che nel nostro caso è un processore Intel(R) Core(TM) i7-9750H di nona generazione, per poter confrontare i tempi di addestramento su 60 epoche. Operando con la CPU l'addestramento è durato 16 minuti (960 secondi) e si è ottenuto un mean_ap di 0.765, quindi l'utilizzo della GPU deve essere sempre preferito rispetto a quello della CPU.

Nodo	Epoche	Tempo [s]	mean_ap
A	0	0	0
B	5	16	0.066
C	10	32	0.275
D	15	47	0.646
E	20	63	0.781
F	30	99	0.842
G	40	143	0.882
H	50	181	0.899
I	60	216	0.910
J	80	284	0.910
K	100	356	0.910

Tabella 3.1: Risultati dell'analisi di convergenza.

3.2.4 Valutazione del modello

Per valutare il modello, ossia per capire se è stato addestrato nel modo corretto ed è in grado di riconoscere e localizzare gli oggetti, si utilizza il comando **"evaluate_dl_model"**:

```
evaluate_dl_model (DLDataset, DLModelHandle, 'split',
'test', GenParamEval, EvaluationResult, EvalParams)
```

Successivamente con **"dev_display_detection_detailed_evaluation"**, visualizziamo la valutazione:

```
dev_display_detection_detailed_evaluation
(EvaluationResult, EvalParams, GenParams, WindowDict)
```

I risultati della validazione vengono riportati in due grafici a torta, come riportato in Figura 3.17:

(1) (2)

Figura 3.17: Grafici a torta in Halcon che rappresentano i risultati della validazione. (1) rappresenta la "pie_charts_precision" e (2) la "pie_charts_recall".

Il grafico a sinistra rappresenta la "pie_charts_precision", in cui vengono mostrati i grafici a torta con i rapporti dei diversi tipi di errore dei falsi positivi per tutte le classi e per ogni classe singolarmente. Per **precision** si intende il rapporto tra tutti i true positive predetti correttamente rispetto a tutti i positivi predetti (quelli veri e quelli falsi). Pertanto, è una misura di quanti risultati predetti come positivi appartengono realmente alla classe selezionata.

$$\textbf{precision} = \frac{TP}{TP + FP} \tag{3.1}$$

Il termine TP rappresenta i true positive (veri positivi), ossia gli oggetti identificati correttamente, mentre FP rappresenta i false positive (falsi positivi). I falsi positivi si hanno ad esempio quando:

- un'istanza viene classificata erroneamente, ovvero viene identificata come un oggetto quando in realtà non lo è;
- viene trovata un'istanza dove c'è solo lo sfondo, ovvero viene identificata come oggetto un'area che in realtà non ne contiene alcuno;
- un'istanza viene localizzata male, ovvero viene individuata ma la posizione indicata non corrisponde a quella reale, il che significa che l'IoU tra l'istanza e il suo ground truth è inferiore alla soglia IoU di valutazione (in Halcon se l'IoU è superiore o uguale a 0.7, allora l'istanza è posizionata in modo corretto). L'IoU, ossia Intersection over Union, rappresenta la percentuale di sovrapposizione tra l'oggetto riconosciuto e quello reale. Per calcolare l'IoU si confronta l'area di sovrapposizione tra l'oggetto riconosciuto e quello reale con l'area totale dei due bounding box. Un esempio visivo è riportato in Figura 3.18.

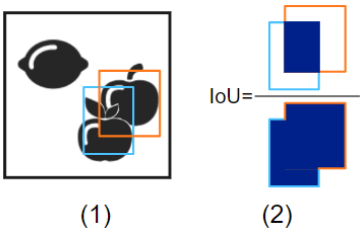


Figura 3.18: (1) L'immagine di input con la bounding box di ground truth (arancione) e la bounding box predetta (azzurro chiaro). (2) L'IoU è il rapporto tra l'area di intersezione e l'area di sovrapposizione [5].

- esistono due o più istanze dell'oggetto presenti nella stessa area [5].

Osservando i risultati notiamo che l'addestramento con 60 epoche è stato eccellente, infatti il modello riconosce alla perfezione tutti gli oggetti e la loro localizzazione, senza nessun falso positivo.

Il grafico di destra rappresenta la "pie_charts_recall", in cui vengono mostrati i grafici a torta del "recall" e dei falsi negativi per tutte le classi e per ogni classe singolarmente. Il **recall**, anche chiamato "tasso di veri positivi", è il rapporto tra tutti i true positive predetti correttamente e tutti i positivi reali. Pertanto, è una misura di quanti campioni appartenenti alla classe selezionata sono stati predetti correttamente come positivi:

$$\textbf{recall} = \frac{TP}{TP + FN} \tag{3.2}$$

Il termine TP indica i veri positivi, ovvero gli oggetti correttamente riconosciuti come positivi dal modello di classificazione. D'altra parte, FN rappresenta i falsi negativi, ovvero gli oggetti erroneamente identificati come negativi dal modello, ma che in realtà sarebbero positivi se classificati correttamente. Un classificatore con alto recall, ma bassa precision individuerà la maggior parte dei membri della classe positiva, al costo però di classificare anche molti negativi come membri della classe. Invece, un classificatore con alta precision, ma basso recall è esattamente il contrario, poiché classifica solo pochi campioni come positivi, ma la maggior parte di queste previsioni sono corrette. Un classificatore ideale con alta precision e alto recall classificherà molti campioni come positivi con alta accuratezza [5]. Oltre ai grafici a torta, per analizzare i risultati della valutazione, è utile visualizzare in Halcon anche la **matrice di confusione** (confusion matrix). Quest'ultima è una tabella utilizzata per valutare le prestazioni di un modello di classificazione, che mostra il numero di volte in cui il modello ha predetto correttamente o erroneamente le classi degli oggetti. La matrice di confusione è organizzata in una griglia quadrata, in cui ogni colonna rappresenta la classe reale dell'etichetta e ogni riga quella prevista dal modello. La diagonale principale della matrice rappresenta le predizioni corrette del modello, mentre le celle al di fuori della diagonale rappresentano le predizioni errate. Questa matrice è utile per calcolare diverse metriche di valutazione del modello, come la precision e il recall. Inoltre, può aiutare a identificare eventuali errori comuni del modello, come ad esempio la confusione tra classi simili o la presenza di un'alta percentuale di falsi positivi o falsi negativi. La matrice di confusione ottenuta nel nostro caso in Halcon è riportata in Figura 3.19:

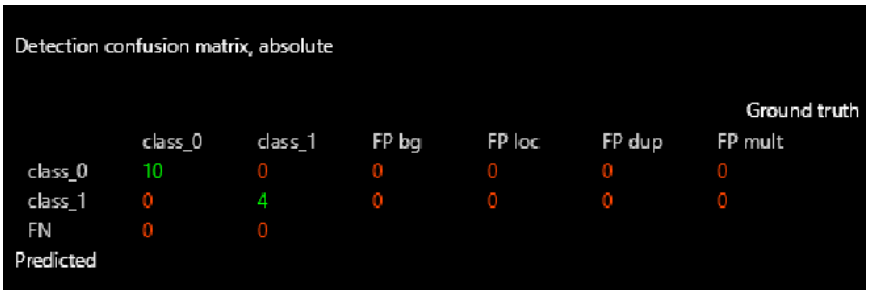


Figura 3.19: Matrice di confusione in Halcon.

Notiamo che nel nostro caso tutti gli elementi si trovano sulla diagonale in verde. Questo è il caso ideale, in cui ogni etichetta di ground truth coincide con la classe reale, per cui il modello è molto preciso.

3.2.5 Risultati di inferenza

Per procedere con l'analisi di inferenza, è necessario inizialmente esaminare dei campioni che rappresentano il 15% del totale dei campioni ottenuti tramite l'operazione di "split_dl_dataset". A questi campioni, sarà applicato il modello di deep learning appena addestrato per identificare e classificare gli oggetti. Nel caso in cui si volessero utilizzare immagini acquisite da una telecamera in tempo reale si userà l'operatore **"grab_image()"**, che verrà approfondito nel Capitolo 4.
Per quanto riguarda la lettura delle immagini si utilizzerà l'operatore **"read_image"**:

```
read_image (Image, DLDataset.image_dir + '/' +
ImageFilesShuffled[IndexInference])
```

Il primo passo da fare è generare un DLSample dall'immagine, utilizzando l'operatore **"gen_dl_samples_from_images"**, il quale permette di memorizzare le immagini fornite in una tupla di dizionari DLSamples:

```
gen_dl_samples_from_images (Image, DLSampleInference)
```

Successivamente, si è pre-processato il DLSample allo stesso modo in cui abbiamo fatto per l'addestramento, utilizzando l'operatore **"preprocess_dl_samples"**:

```
preprocess_dl_samples (DLSampleInference,
DLPreprocessParam)
```

Infine è stato applicato il modello di deep learning addestrato sul campione con la funzione **"apply_dl_model"**:

```
apply_dl_model (DLModelHandle, DLSampleInference, [],
DLResult)
```

L'output restituito è un dizionario chiamato DLResult che contiene le coordinate dei bounding box, l'ID della classe e la relativa confidenza dei risultati. Utilizzando la funzione **"get_dict_object"**, è possibile ottenere le regioni associate agli oggetti riconosciuti e classificati dal modello di deep learning. A partire da queste regioni, è possibile calcolare il centroide, l'area e l'angolo di inclinazione mediante un'operazione simile a quella utilizzata per la Blob Analysis.

```
get_dict_object (Object, DLResult, 'mask')
```

Alcuni esempi dei risultati ottenuti sono riportai nelle Figure 3.20 e 3.21:

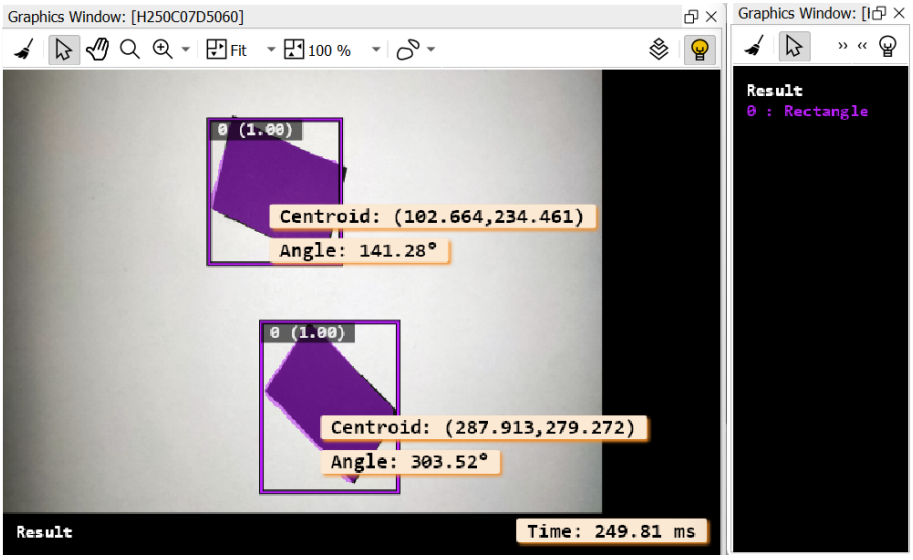


Figura 3.20: Esempio 1 di deep learning instance segmentation in Halcon.

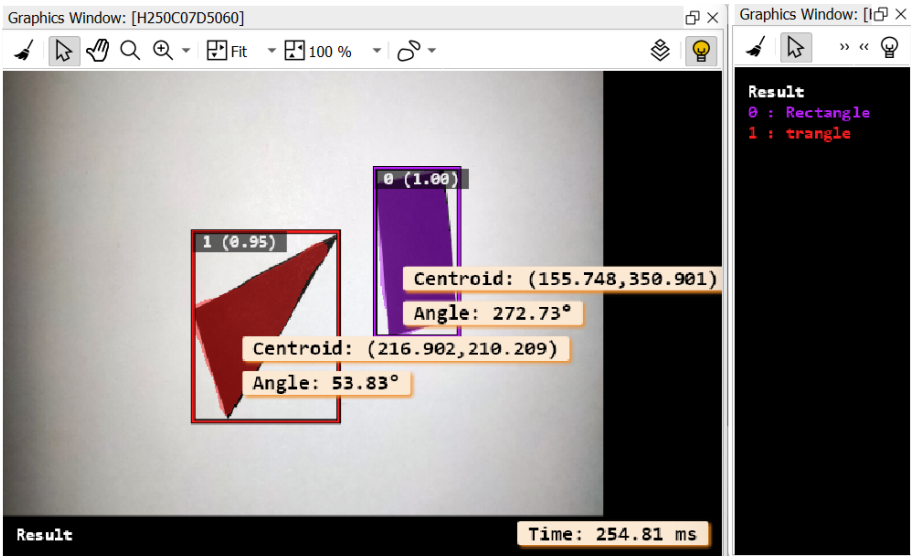


Figura 3.21: Esempio 2 di deep learning instance segmentation in Halcon.

Notiamo che nel riquadro a sinistra sono presenti le immagini con le regioni e i bounding box generati dal modello di deep learning, mentre nel riquadro di destra vengono indicati i nomi delle classi degli oggetti individuati.

3.3 Risultati ottenuti

Analizzando i risultati ottenuti tramite l'applicazione della tecnica di Deep Learning Instance Segmentnation con Halcon, emergono le seguenti caratteristiche:

- i tempi di esecuzione sono nell'ordine dei 400-700 ms. Facendo una media di tutti i tempi di esecuzione rilevati per ogni immagine si è ottenuto che:

  $$\text{Tempo di esecuzione medio} = 563.17 \text{ ms}$$

  Il tempo richiesto dalla tecnica di Instance Segmentation è significativamente maggiore rispetto a quello richiesto dalla Blob Analysis e dallo Shape-Based Matching. Questo può costituire un problema in applicazioni che richiedono tempi di esecuzione rapidi, come ad esempio nell'automazione industriale;

- questa tecnica richiede l'addestramento di un modello di deep learning per il riconoscimento degli oggetti, il quale può essere rapido se si tratta di pochi oggetti con forme semplici, per cui si può utilizzare un dataset con un numero modesto di immagini. Tuttavia, può diventare molto dispendioso in termini di tempo se si ha a che fare con molti oggetti e dalle forme complesse, e quindi il training richiede un dataset con un grande numero di immagini per essere efficace;

- la tecnica di Instance Segmentation, implementata in Halcon, risulta essere molto affidabile nell'identificazione degli oggetti e classificarli, dopo che il modello di deep learning è stato addestrato adeguatamente;

- con la tecnica di Instance Segmentation risulta complesso identificare in modo preciso le caratteristiche degli oggetti riconosciuti, come ad esempio centroide e angolo di inclinazione. Ciò è dovuto al fatto che il modello di deep learning è efficace nel riconoscere e classificare gli oggetti, ma non lo è altrettanto nel generare le regioni associate ad essi, dalle quali poi si ricavano le caratteristiche. Ciò potrebbe portare il robot a non afferrare correttamente l'oggetto;

- è una tecnica complessa da implementare in Halcon rispetto allo Shape-Based Matching.

Alla luce di tali considerazioni, la tecnica da privilegiare utilizzando Halcon per il riconoscimento di oggetti è ancora lo Shape-Based Matching, perché molto più rapida e precisa rispetto alla Deep Learning Instance Segmentation.

Capitolo 4

Riconoscimento di mani utilizzando il deep learning

In questo capitolo analizzeremo due diversi approcci che utilizzano i modelli di deep learning per riconoscere le mani nella zona di lavoro: la Classification e l'Instance Segmentation. La Classification è una tecnica che mira a classificare un'immagine in una o più categorie predefinite. Nel caso in esame verrà addestrato un modello di deep learning per riconoscere se un'immagine contiene una mano o meno e verrà restituita una risposta binaria (sì o no) per ogni immagine. Per l'implementazione della tecnica di Classification, verrà utilizzato inizialmente il Deep Learning Tool di Halcon. Questo strumento consente di eseguire tutti i passaggi necessari senza dover esportare il dataset o scrivere del codice per implementare le varie funzioni in Halcon. Successivamente, i risultati ottenuti saranno confrontati con quelli effettivi del software, dove tutti i passaggi eseguiti dal tool sono stati tradotti in codice.
L'Instance Segmentation, invece, si concentra sull'individuazione e la separazione delle diverse istanze di oggetti presenti nell'immagine: se l'immagine contiene due mani questa tecnica restituirà due maschere distinte, ciascuna delle quali rappresenta una singola mano. Ciò consente di identificare e separare gli oggetti sovrapposti o parzialmente oscurati nell'immagine.
Entrambe le tecniche possono essere utilizzate per riconoscere le mani nella zona di lavoro, ma, come vedremo, l'Instance Segmentation fornisce una maggiore precisione nella localizzazione delle mani rispetto alla Classification.

4.1 Classification

La classificazione delle immagini in Halcon è basata su reti CNN (Convolutional Neural Networks), caratterizzate dalla presenza di almeno un layer di convoluzione nella rete. Le CNN sono ispirate alla corteccia visiva di

umani e animali. Ad esempio, quando vediamo bordi verticali rispondono alcuni neuroni, altri invece si attivano quando osserviamo bordi orizzontali o diagonali. Allo stesso modo, le reti neurali convoluzionali eseguono la classificazione cercando caratteristiche a basso livello, come bordi e curve, costruendo poi concetti più astratti, ad esempio testo, loghi o componenti di macchine. Queste caratteristiche vengono selezionate automaticamente durante l'addestramento. Come già accennato, le CNN utilizzate per i metodi di deep learning hanno molteplici strati. Uno strato è un blocco costruttivo che svolge compiti specifici e può essere visto come un contenitore, che riceve un input, applica un'operazione su di esso e restituisce un output, il quale servirà come input per il prossimo strato.
I layer di input e output sono collegati al dataset, cioè ai pixel dell'immagine o alle etichette, mentre i layer intermedi sono chiamati layer nascosti. Tutti i layer insieme formano una rete, una funzione che mappa i dati di input in classi. Molti di questi layer, chiamati anche filtri, hanno dei pesi, i quali saranno i parametri che verranno ottimizzati durante l'addestramento della rete. Come abbiamo già menzionato nel Capitolo 3, ci sono altri parametri aggiuntivi chiamati hyperparameter che non vengono appresi direttamente durante l'addestramento regolare, i quali hanno valori impostati prima dell'avvio dell'addestramento.
Il classificatore riceve un'immagine in ingresso e, invece di fornire direttamente l'etichetta di classe corrispondente, restituirà i valori di confidenza associati a ciascuna classe, esprimendo la probabilità che l'immagine appartenga a ogni classe distintamente.
Sottolineiamo che il training di una rete non è un problema di pura ottimizzazione, ossia non viene ottimizzata direttamente la funzione di mappatura che identifica le classi. Viene invece introdotta una funzione di perdita che tiene conto della discrepanza tra le classi previste e quelle reali. È proprio quest'ultima funzione che viene ottimizzata, al fine di migliorare la precisione della classificazione delle immagini [4].

4.1.1 Panoramica sui layer

Sono riportati alcuni dei più comuni layer delle reti CNN:

Layer convoluzionale

Il primo hidden layer spesso è un layer convoluzionale, il cui funzionamento consiste nel muovere un filtro, noto anche come **kernel**, su una mappa caratteristica di un altro strato (che può essere considerata come un'immagine). Durante l'operazione di convoluzione, si prende in considerazione una porzione della mappa caratteristica di input che è coperta dal kernel. Su questa porzione viene eseguita un'operazione e il risultato determina il valore della

mappa caratteristica di output, come riportato in Figura 4.1:

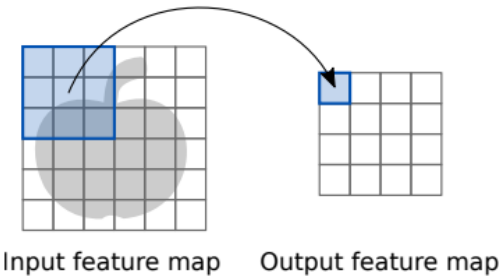


Figura 4.1: Viene applicato uno spostamento del kernel 3x3 su una mappa caratteristica di dimensione 6x6. La prima area selezionata della mappa si trova nell'angolo in alto a sinistra, mentre la seconda area viene selezionata spostandosi con un passo di 1 in verticale e 1 in orizzontale, quindi a partire dal secondo pixel sulla stessa riga. Il risultato dell'operazione è una nuova mappa caratteristica di output di dimensione 4x4 [4].

Il **passo** indica di quanti pixel il kernel si sposterà verso destra sulla mappa caratteristica. Una volta raggiunta la fine della larghezza della mappa caratteristica, il kernel si sposterà verso il basso di un numero di pixel determinato dallo stesso parametro, così da iniziare la scansione della riga successiva. In questo modo, il passo controlla la procedura con cui il kernel attraversa l'intera mappa caratteristica durante la convoluzione. Il kernel è una matrice predefinita di dimensioni specifiche che contiene un insieme di numeri, chiamati pesi del filtro. Durante il processo di addestramento, questi pesi vengono appresi dal modello. Ogni peso del filtro specifica l'importanza di un particolare pixel nella mappa caratteristica di input durante la convoluzione.
Nei layer convoluzionali l'operazione eseguita è un prodotto di Hadamard[4]: i valori dei pixel della sezione della mappa caratteristica vengono moltiplicati elemento per elemento con i pesi del filtro e sommati. Un esempio di questa operazione è riportato in Figura 4.2. Poiché questa operazione rappresenta una convoluzione, il nome di questo strato è "layer convoluzionale". Il procedimento viene applicato all'intera mappa caratteristica di input per produrre una mappa di output che indica varie caratteristiche dell'immagine di partenza, in base ai filtri utilizzati. Di conseguenza, vengono prodotte mappe bidimensionali, che sono poi accorpate lungo la terza dimensione per generare l'output (come riportato in Figura 4.4).

[4]Il prodotto di Hadamard (noto anche come prodotto per elemento, prodotto per ingresso) è un'operazione binaria che avviene tra due matrici delle stesse dimensioni e produce un'altra matrice del stessa dimensione degli operandi, dove ogni elemento i, j è il prodotto degli elementi i, j delle due matrici originarie.

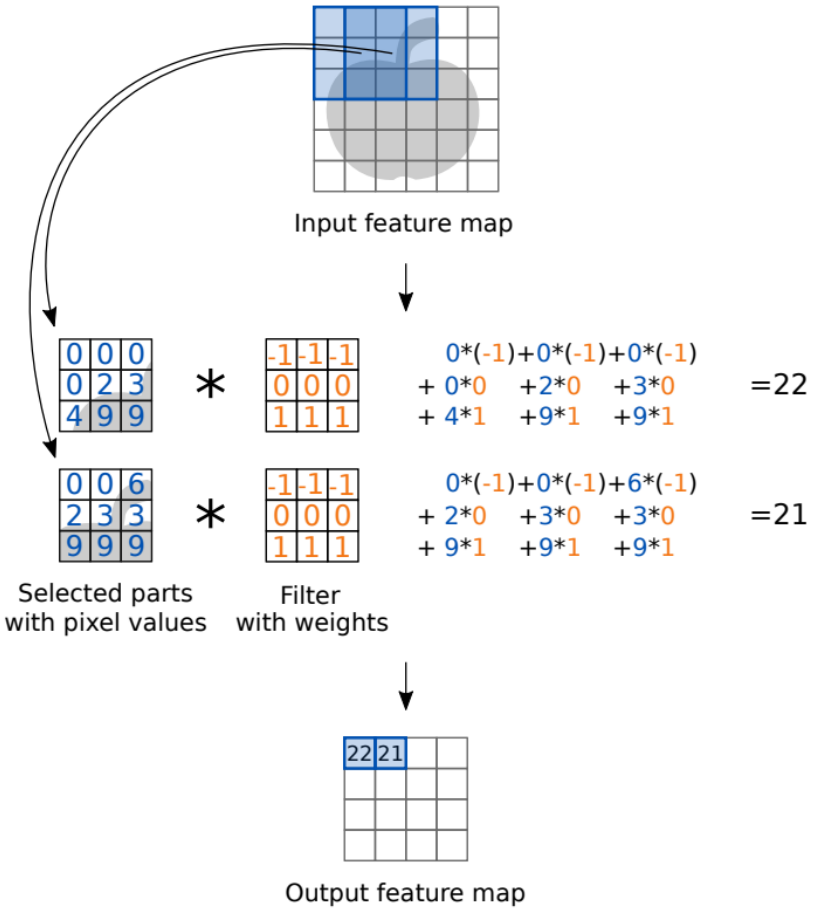


Figura 4.2: Viene spostato un kernel 3x3 su una mappa caratteristica 6x6 con uno passo di 1,1. L'operazione riportata è un prodotto di Hadamard tra le due sezioni evidenziate in azzurro [4].

Pooling layer

Il Pooling è una tecnica di sottocampionamento molto utilizzata per ridurre la dimensione delle mappe caratteristiche, che rappresentano l'output di uno o più strati di una rete neurale convoluzionale. Questa riduzione dimensionale è importante per vari motivi:

- ridurre il numero di parametri della rete: meno parametri significa avere una rete più semplice, che richiede meno memoria e meno tempo per l'addestramento;

- ridurre il rischio di overfitting: una rete che ha troppe unità può imparare a memoria il set di addestramento, senza generalizzare bene su

nuovi esempi. La riduzione delle dimensioni delle mappe caratteristiche può aiutare a ridurre questo rischio;

- rendere la rete meno sensibile alle piccole variazioni nell'input: riducendo la dimensione delle mappe caratteristiche, si rende la rete meno sensibile a variazioni piccole dell'input, migliorando così la robustezza della rete.

Il Pooling funziona selezionando una regione della mappa caratteristica di dimensione fissa (ossia il kernel), e riducendo questa regione a un unico valore. Il modo in cui viene selezionato questo valore può variare, ad esempio scegliendo il massimo valore presente nella regione (**max pooling**) o calcolando la media di tutti i valori presenti nella regione (**average pooling**).
Il kernel viene poi fatto scorrere lungo la mappa a intervalli di lunghezza fissa, ovvero il passo, e la stessa operazione viene ripetuta su ogni regione selezionata. In questo modo, la mappa caratteristica originale viene ridotta di dimensione, ma conserva comunque le informazioni più importanti [4].
Un esempio di questa operazione è riportato in Figura 4.3:

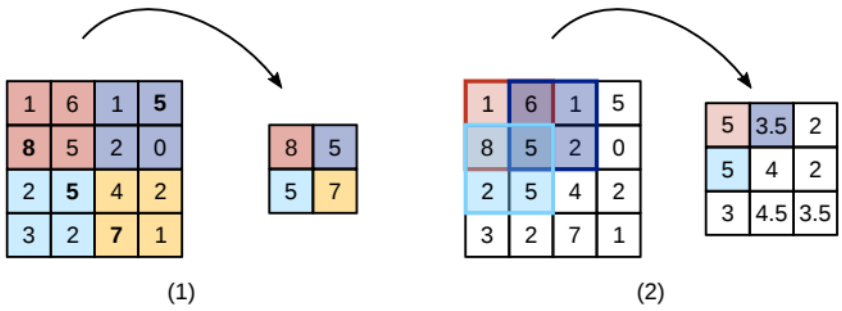


Figura 4.3: Due esempi di pooling: (1) il **max pooling** con dimensione del kernel di 2,2 e passo di 2,2 suddivide la mappa caratteristica 4x4 di input in 4 rettangoli non sovrapposti e restituisce il massimo di ciascun rettangolo. (2) L'**average pooling** con dimensione del kernel di 2,2 e passo di 1,1 suddivide la stessa mappa caratteristica 4x4 di input in 9 rettangoli, in alcuni casi sovrapposti, e restituisce la media di ciascun rettangolo [4].

Layer non lineare: Rectified Linear Unit (ReLU) layer

Una CNN deve essere in grado di approssimare funzioni non lineari, pertanto ha bisogno di almeno uno strato non lineare.
Gli strati ReLU (Rectified Linear Unit) sono un tipo di strato non lineare che è diventato comune nelle CNN. Questo layer applica una funzione non lineare semplice all'output del livello di convoluzione [4]. In particolare, l'output è

la funzione massima tra 0 e il valore di input:

$$f(x) = \max\ (0, x) \tag{4.1}$$

Questa funzione è molto semplice da calcolare ed efficiente in termini di tempo di esecuzione.

Fully connected layer

Nelle CNN, l'ultimo strato è spesso costituito da uno strato completamente connesso (fully connected layer). Questo strato riceve in input le mappe caratteristiche di alto livello generate dallo strato precedente e, sulla base di queste, seleziona una classe appropriata. Ad esempio, se l'immagine rappresenta una mano (classe: mano), le mappe caratteristiche potrebbero rappresentare parti di essa come le dita, il palmo e le unghie.
Queste mappe sono generate automaticamente durante il processo di apprendimento della CNN [4].

Configurazione di base di una rete CNN

Un esempio illustrativo di una CNN completa è mostrato in Figura 4.4, dove viene mostrata una semplice rete utilizzata per la classificazione:

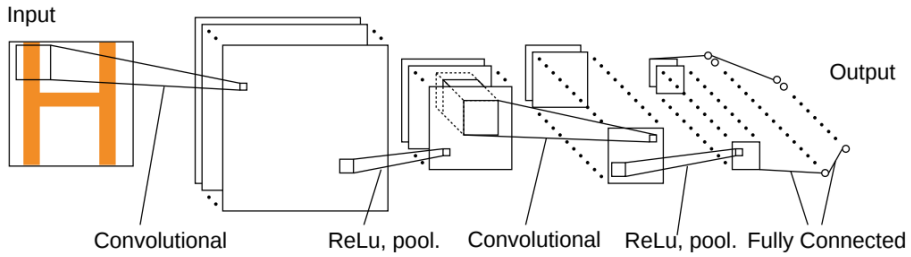


Figura 4.4: Rete CNN utilizzata per la classificazione. Gli hidden layer sono: Convolutional → ReLU, Pooling → Convolutional → ReLU, Pooling → Fully Connected [4].

La rete neurale utilizza una metodologia di elaborazione graduale, dove ogni strato sfrutta l'output del precedente per elaborare informazioni sempre più complesse. I primi hidden layer della rete sono in grado di individuare le caratteristiche a basso livello dell'immagine, come i bordi e le curve. Gli strati successivi descrivono le caratteristiche a livello superiore, utilizzando mappe sempre più sofisticate. Inoltre, i filtri più profondi della rete sono in grado di percepire le informazioni da un'area sempre più ampia dell'immagine originale, fornendo alla rete una conoscenza sempre più completa del contesto. Infine, uno strato fully connected viene utilizzato per calcolare l'output finale della rete, consentendo alla rete di prevedere le classi degli

oggetti presenti nell'immagine con maggiore precisione.
Grazie a questo processo di elaborazione graduale, la rete neurale diventa sempre più efficace nell'individuazione delle caratteristiche delle immagini e nell'assegnazione delle corrispondenti etichette di classificazione [4].

4.1.2 Deep Learning Tool

Per l'implementazione della tecnica di Classification, è stato utilizzato in primo luogo il Deep Learning Tool di Halcon. Il dataset utilizzato per fare il training della rete è composto di 476 immagini, le quali sono state acquisite con la videocamera Logitech HD C270 e sfruttando una selezione del dataset di 11076 mani realizzato da Afifi, M. (2019)[11], utilizzato per la creazione di un modello che fosse in grado di riconoscere il genere di una persona a partire da tali dati. Le mani rappresentate nelle immagini sono posizionate in modo variabile e presentano diversi livelli di luminosità. Alcuni esempi del dataset utilizzato sono riportati in Figura 4.5:

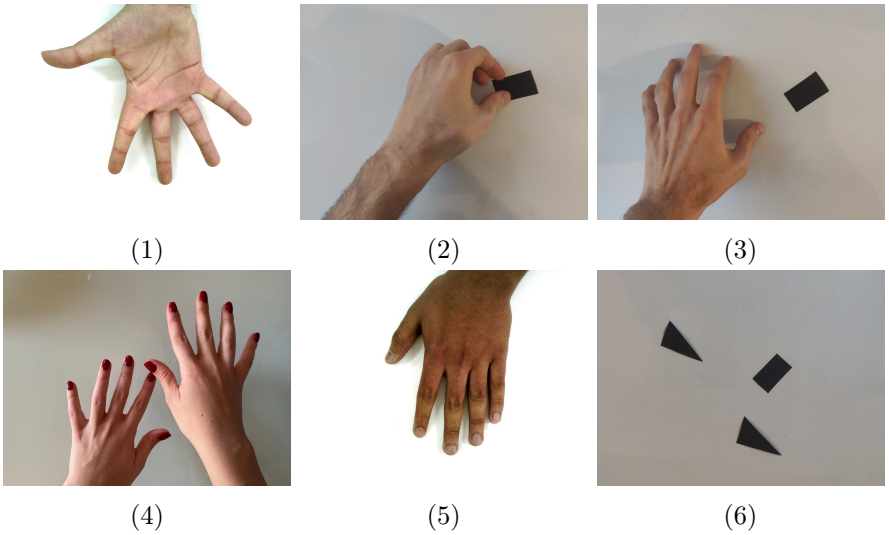


Figura 4.5: Le figure (1) e (5) fanno parte del dataset realizzato da Afifi, M. (2019) [11], mentre (2), (3) e (4) sono immagini acquisite con la videocamera Logitech HD C270. La figura (6) fa comunque parte del dataset, perché per la classificazione sono necessarie anche immagini in cui non sono presenti delle mani, per poter fare la distinzione tra "Mani" e "No mani".

Il Deep Learning Tool consente di generare etichette per le immagini, che permettono di specificare la classe degli oggetti presenti in esse. Nel nostro caso, abbiamo creato due classi per distinguere le immagini in cui sono presenti mani da quelle in cui non lo sono, denominandole "Mani" e "No mani". Queste classi sono chiaramente evidenziate negli esempi riportati nella Figura 4.5.

Dopo avere eseguito il labelling di tutte le immagini, si è potuto suddividere il dataset etichettato in tre diversi sottoinsiemi, una procedura già vista nel Capitolo 3 per l'Instance Segmentation. Questi tre sottoinsiemi sono:

- **Training Images**, che nel nostro caso è stato impostato al 70% (corrisponde a 333 immagini). Questa percentuale di dati verrà utilizzata per il training del modello di deep learning;
- **Validation Images**, che nel nostro caso è stato impostato al 15% (corrisponde a 71 immagini). Questa percentuale di dati verrà utilizzata per la validazione del modello;
- **Test Images**, che nel nostro caso è stato impostato al 15% (corrisponde a 72 immagini). Questa percentuale di dati verrà utilizzata per verificare che il modello funzioni correttamente (test).

La finestra che ci permette di selezionare queste percentuali è riportata in Figura 4.6:

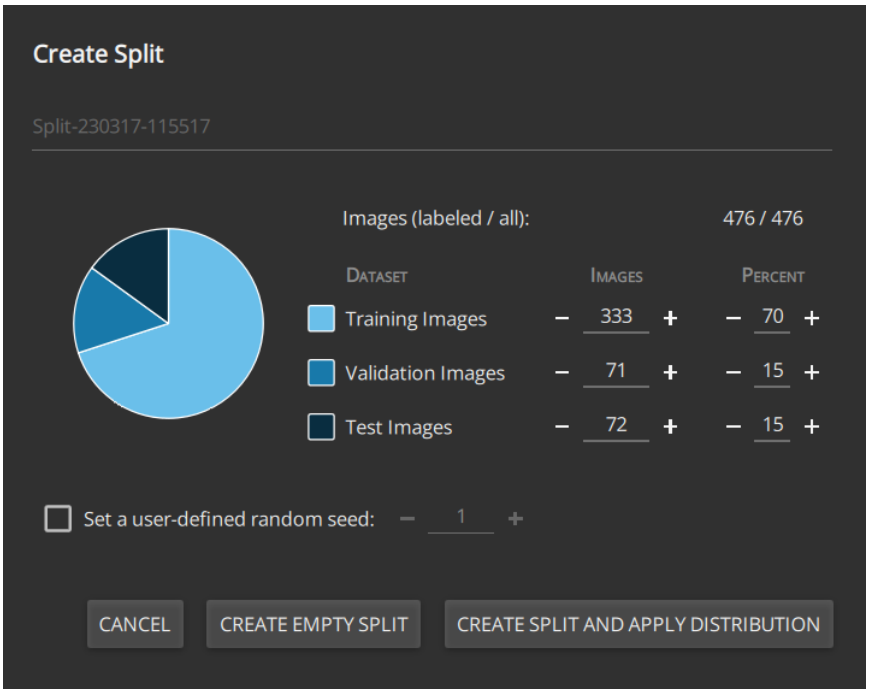


Figura 4.6: Finestra del Deep Learning Tool che permette di generare i sottoinsiemi del dataset.

Cliccando su "Create split and apply distribution", è possibile applicare la distribuzione dei dati desiderata per ogni sottoinsieme. Successivamente, utilizzando il Deep Learning Tool, è possibile creare il modello e configurare gli hyperparameter necessari per il training, come riportato in Figura 4.7.

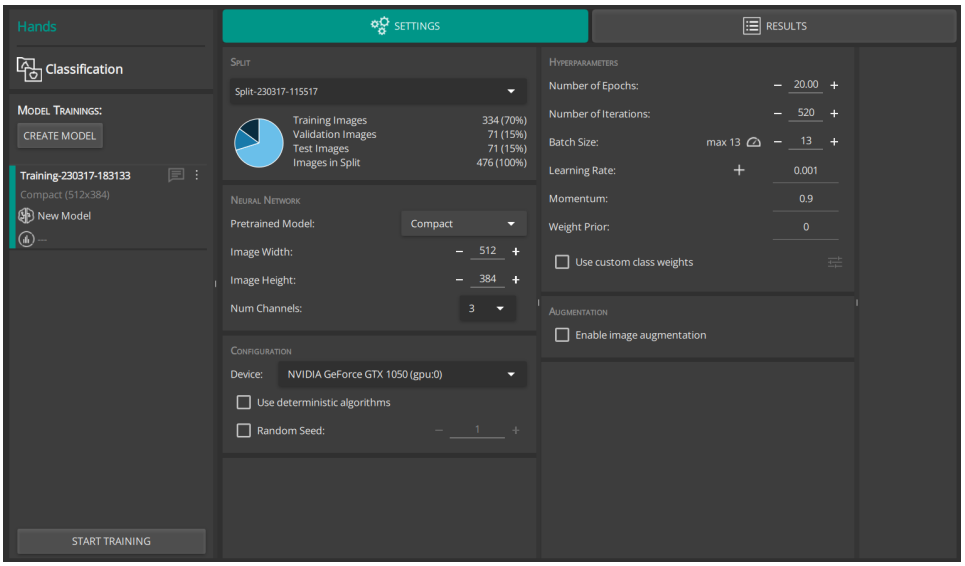


Figura 4.7: Finestra del Deep Learning Tool che permette di impostare gli hyperparameter.

Si può notare che è possibile impostare la dimensione dell'immagine e, nel nostro caso, abbiamo deciso di imporre una larghezza di 512 pixel e un'altezza di 384 pixel. Si possono poi scegliere i valori dei tre hyperparameter fondamentali, ossia:

- **Number of Epochs**, ossia il numero delle epoche, che nel nostro caso è stato impostato a 20;
- **Batch Size**, ossia il numero di immagini di input (ed etichette corrispondenti) che vengono trasferite contemporaneamente alla memoria del dispositivo e quindi elaborate nello stesso momento. Questo valore dipende dal dispositivo utilizzato, ma il Deep Learning Tool consente di selezionare automaticamente il batch size massimo consentito [7] e nel nostro caso è stato impostato a 13;
- **Learning Rate**, ossia la dimensione del passo che l'algoritmo di ottimizzazione compie durante la fase di apprendimento per cercare di minimizzare la funzione di costo. Nel nostro caso è stato impostato a 0.001.

La scelta del dispositivo per l'addestramento del modello è di cruciale importanza. Si può optare per la GPU (NVIDIA GeForce GTX 1050) o la CPU (Intel(R) Core(TM) i7-9750H @2.60GHz). Nel nostro caso, abbiamo optato per la GPU poiché nel Capitolo 3 è stata dimostrata la sua maggiore velocità in termini di tempi di esecuzione.

Per ottenere un training efficiente del modello, è fondamentale scegliere accuratamente la rete pre-addestrata fornita da Halcon, in quanto questa permette di ridurre significativamente i tempi di esecuzione. È possibile scegliere tra cinque reti diverse [7]:

- **Compact**: questa rete neurale è progettata per essere efficiente in termini di memoria e tempo di esecuzione. La rete non contiene alcun layer completamente connesso. La larghezza e l'altezza dell'immagine non devono essere inferiori a 15 pixel;

- **Enhanced**: questa rete neurale ha più hidden layer rispetto alla "Compact" e si presume quindi che sia più adatta per attività di classificazione più complesse. Ciò ha il costo di richiedere più tempo e memoria. La larghezza e l'altezza dell'immagine non devono essere inferiori a 47 pixel. Non esiste una dimensione massima dell'immagine, ma immagini di grandi dimensioni aumenteranno notevolmente la richiesta di memoria e il tempo di esecuzione;

- **ResNet-50**: è un tipo di classificatore basato su una rete neurale convoluzionale, che utilizza una rete residua con 50 strati convoluzionali. È simile alla rete "Enhanced", ma offre il vantaggio di rendere l'addestramento più stabile e di garantire una maggiore robustezza interna, risulta quindi particolarmente indicato per svolgere compiti più complessi.
  La larghezza e l'altezza dell'immagine non devono essere inferiori a 32 pixel. Non esiste una dimensione massima dell'immagine, ma immagini di grandi dimensioni aumenteranno notevolmente, anche in questo caso, la richiesta di memoria e il tempo di esecuzione;

- **MobileNetV2**: è un modello di classificazione leggero e a basso consumo energetico, ideale per applicazioni di visione mobile ed embedded. Larghezza e altezza dell'immagine non dovrebbero essere inferiori a 32 pixel. Non esiste un limite massimo definito per le dimensioni dell'immagine, tuttavia, maggiore è la dimensione e maggiore sarà la richiesta di memoria e il tempo di esecuzione necessari per elaborarla in modo significativo;

- **AlexNet**: è una rete neurale progettata per compiti di classificazione semplici. Si distingue per i suoi kernel nei primi layer di convoluzione, che sono più grandi rispetto ad altre reti con una performance di classificazione comparabile (ad esempio, la rete "Compact").
  La larghezza e l'altezza dell'immagine non dovrebbero essere inferiori a 29 pixel. Come negli altri casi, non esiste una dimensione massima dell'immagine, tuttavia le dimensioni più grandi aumenteranno significativamente la richiesta di memoria e il tempo di esecuzione.

Per la nostra applicazione si è scelta come rete pre-addestrata la "Compact". Il training del modello ha portato ai risultati riportati in Figura 4.8:

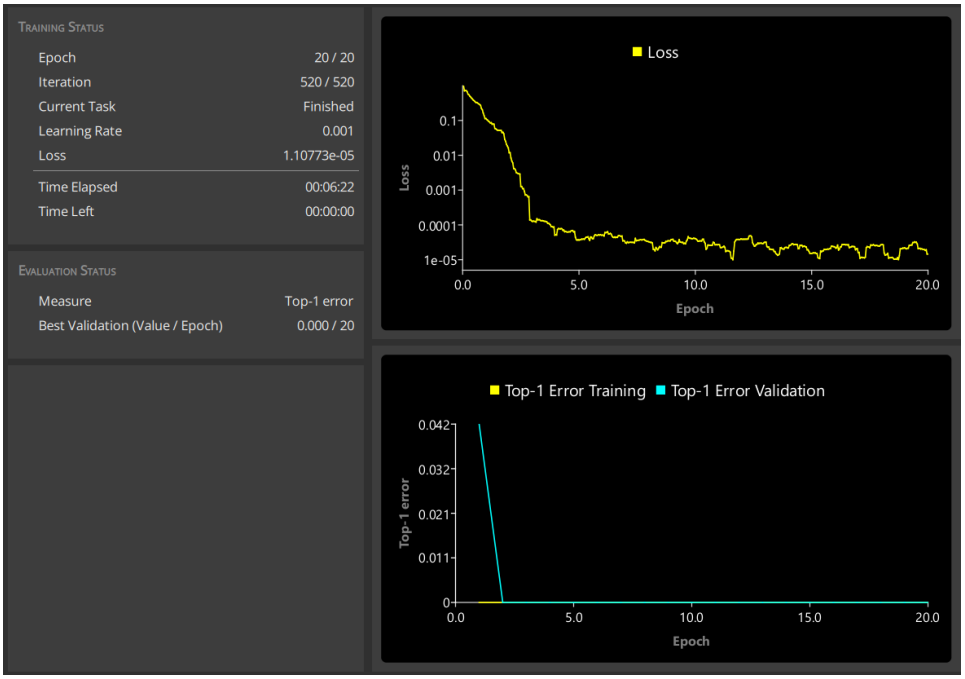


Figura 4.8: Risultati del training riportati dal Deep Learning Tool.

Notiamo che il tempo necessario per il training è stato di 6 minuti e 22 secondi. I due grafici che vengono riportati sono fondamentali per la valutazione del modello di deep learning addestrato: in alto viene riportato il valore di perdita (**Loss**), quello in basso invece riporta il **Top-1 Error**, entrambi in funzione del numero di epoche. Abbiamo già analizzato approfonditamente il valore di perdita nei capitoli precedenti, quindi ora focalizziamoci sul Top-1 Error. Questo parametro indica il numero di immagini la cui classe è stata erroneamente predetta dal modello. Per un addestramento efficace, il valore del Top-1 Error dovrebbe diminuire man mano che le epoche aumentano, fino a raggiungere valori prossimi allo zero. Nel nostro caso, possiamo affermare che l'addestramento è stato eccellente poiché il Top-1 Error si avvicina a zero molto prima di raggiungere le 20 epoche.
Arrivati a questo punto il Deep Learning Tool ci permette di valutare il modello addestrato, riportando una serie di indicatori e alcuni grafici a torta che ci permettono di esaminare i risultati del training. Tali informazioni sono mostrate nella Figura 4.9. Un parametro di fondamentale importanza è **Inference time per image**, il quale indica il tempo necessario per effettuare una previsione una volta che il modello è stato addestrato. Questo valore viene calcolato su un campione casuale di immagini, impostando il batch size ad 1 e utilizzando la mediana, al fine di eliminare eventuali valori

anomali. Tuttavia, va tenuto presente che questo valore riflette soltanto lo stato attuale del sistema utilizzato (CPU/GPU, carico di lavoro del processore, ecc.) e, di conseguenza, può fornire soltanto una stima approssimativa [7].

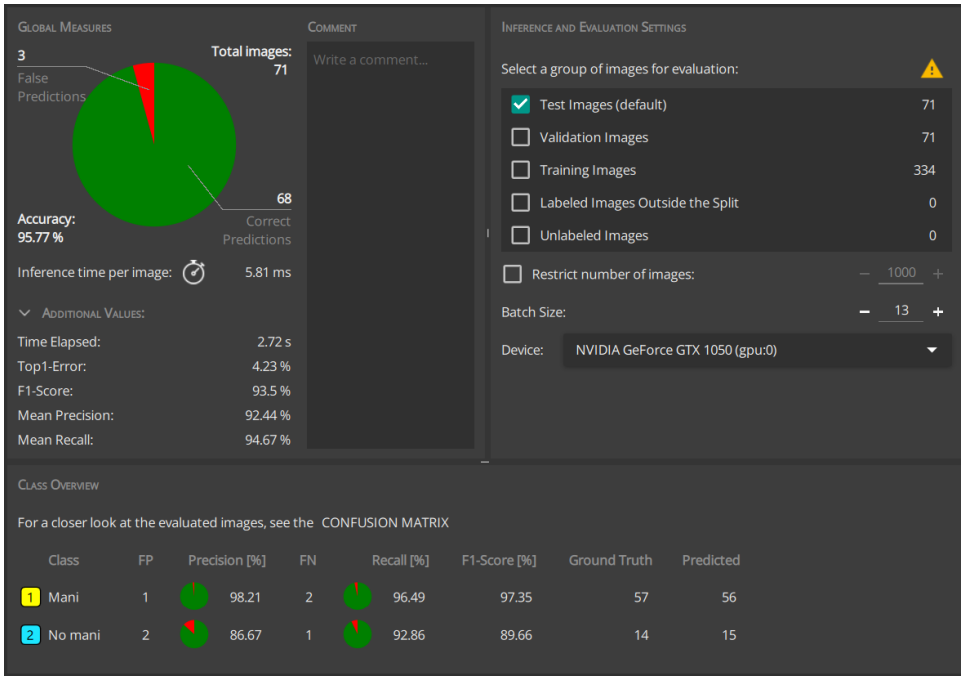


Figura 4.9: Valutazione del training nel Deep Learning Tool.

Notiamo che viene riportata anche l'**Accuracy** (A), ossia l'accuratezza, la quale indica il rapporto tra il numero di previsioni corrette e il numero totale di previsioni:

$$A = \frac{TP + TN}{TP + TN + FP + FN} \tag{4.2}$$

Dove TP: "veri positivi", TN: "veri negativi", FP: "falsi positivi", FN: "falsi negativi". L'accuratezza descrive come il modello si comporta in tutte le classi. Altri parametri utili per la valutazione del modello sono quelli riportati in **Additional Values**:

- **Time Elapsed**: indica la misura del tempo totale in secondi necessario per la valutazione. Nel nostro caso è di 2.72 secondi.

- **Top-1 Error**: rapporto tra le immagini per cui la classe predetta più probabile è errata (cioè non è la classe corretta) rispetto alla classe reale. Nel nostro caso è del 4.23%.

- **Mean Precision**: proporzione di tutti i positivi predetti correttamente rispetto a tutti i positivi predetti:

$$P = \frac{TP}{TP + FP} \tag{4.3}$$

La precisione riflette l'affidabilità del modello rispetto al rilevamento di campioni positivi. Nel nostro caso è del 92.44%.

- **Mean Recall**: Proporzione di tutti i positivi predetti correttamente rispetto a tutti i positivi reali:

$$R = \frac{TP}{TP + FN} \tag{4.4}$$

Il recall misura la capacità del modello di rilevare campioni positivi. Nel nostro caso è del 94.67%.

- **F1-Score**: è la media armonica di precisione e recall:

$$F_1 = \frac{2}{P^{-1} + R^{-1}} = 2\frac{P \cdot R}{P + R} = \frac{TP}{TP + \frac{1}{2}(FP + FN)} \tag{4.5}$$

Nel nostro caso è del 93.5%.

Osservando il grafico a torta in Figura 4.9 notiamo che il training ci ha portato ad un ottimo modello, perché su 71 immagini, ha predetto la classe errata solo in tre occasioni, quindi con un'accuratezza del 95.77%.
Per aiutare nella valutazione del modello il Deep Learning Tool riporta anche la matrice di confusione (Figura 4.10), nella quale vengono indicati anche i falsi positivi (FP) e i falsi negativi (FN) per ogni classe.

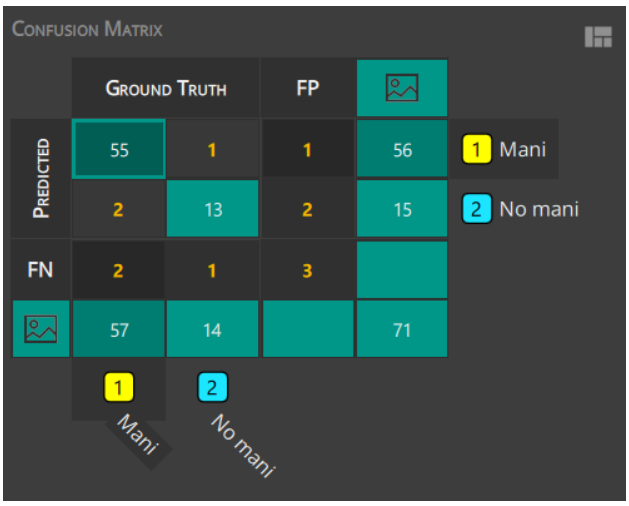


Figura 4.10: Matrice di confusione nel Deep Learning Tool.

Una funzionalità molto interessante che mette a disposizione il Deep Learning Tool è la **Heatmap**, ossia un'immagine che evidenzia le regioni dell'immagine che hanno avuto maggior peso nell'elaborazione e nell'output del modello di deep learning. Questa immagine viene generata utilizzando una rappresentazione in falsi colori, dove le regioni dell'immagine che hanno un peso maggiore vengono rappresentate con colori più caldi, come il rosso o il giallo, mentre le regioni con un peso minore vengono rappresentate con colori più freddi, come il blu o il verde. In tal modo, una heatmap offre approfondimenti sui meccanismi interni, altrimenti nascosti, dell'algoritmo di deep learning. Alcune delle heatmap ottenute sono riportate nelle Figure 4.11 e 4.12:

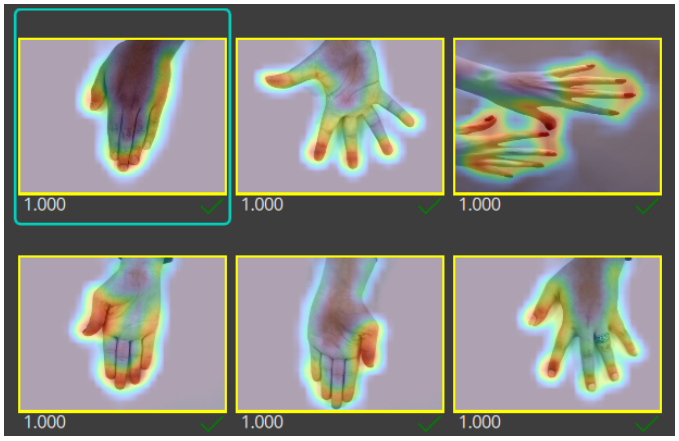


Figura 4.11: Heatmap della classe "Mani".

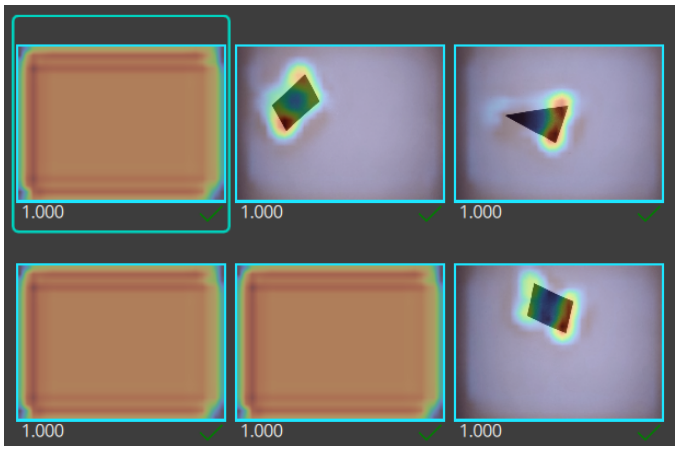


Figura 4.12: Heatmap della classe "No mani".

Osservando queste heatmap notiamo come l'algoritmo di deep learning, per riconoscere le mani, si concentri molto sulle dita e quindi sulle estremità.

Questo è probabilmente dovuto al fatto che le dita sono una caratteristica distintiva e facilmente riconoscibile delle mani umane, e quindi rappresentano un'informazione chiave per l'algoritmo nel processo di identificazione.
In conclusione, il Deep Learning Tool rappresenta uno strumento estremamente utile poiché ci consente di creare un modello di deep learning in pochi passaggi, grazie a un'interfaccia user-friendly che non richiede la scrittura di alcuna riga di codice.

4.1.3 Classification in Halcon

Si è deciso di implementare la tecnica di Classification anche in Halcon, al fine di valutare tutte le differenze con il Deep Learning Tool, anche in termini di prestazioni. L'obiettivo è quindi lo stesso, ovvero classificare le immagini in ingresso con le etichette "Mani" nel caso in cui contengano delle mani e con "No mani" se non sono presenti. Il dataset utilizzato per il training è uguale a quello precedente. Come già visto nel Capitolo 3 per l'Instance Segmentation, anche in questo caso il workflow da seguire è composto di 4 passaggi:

- **prepare data**;
- **train model**;
- **evaluate model**;
- **infer results**.

Analizziamo in modo approfondito passo per passo.

Preparazione dei dati

Per prima cosa occorre leggere un dataset di dati. Per la classificazione, è sufficiente mettere le immagini in diverse cartelle in base alle loro classi. Questa procedura interpreta quindi la struttura delle cartelle e assegna le etichette alle immagini, le quali poi verranno lette con il comando **"read_dl_dataset_classification"**. In alternativa, è possibile utilizzare **"read_dict"** per leggere un set di dati che si è etichettato con il Deep Learning Tool, come nel nostro caso:

```
read_dict (DatasetFilename, [], [], DLDataset)
```

Il set di dati viene quindi diviso in tre sottoinsiemi, come già fatto per l'Instance Segmentation. I dati di addestramento verranno utilizzati direttamente come input per l'addestramento, quelli di validazione vengono utilizzati per controllare e visualizzare il progresso dell'addestramento, in modo che gli hyperparameter possano essere adattati se necessario. Infine, i dati

di test vengono utilizzati per una valutazione indipendente dopo l'addestramento. Il comando che ci consente di suddividere casualmente il set di dati utilizzando delle percentuali specificate, mantenendo la distribuzione delle classi, è **"split_dl_data_set"**. Si noti che i sottoinsiemi devono essere indipendenti e ciascuno di essi deve rappresentare l'intero set di dati.

```
split_dl_dataset (DLDataset, 70, 15, [])
```

Nel nostro caso si è scelto di riservare il 70% dei dati del dataset per il training, il 15% per la validazione del modello e il restante per l'inferenza dei risultati.
Un altro aspetto fondamentale è la pre-elaborazione del dataset. La funzione **"create_dl_preprocess_param_from_model"** crea un dizionario con i parametri di pre-elaborazione basati su un determinato modello di apprendimento:

```
create_dl_preprocess_param_from_model (DLModelHandle,
'none', 'full_domain', [], [], [], DLPreprocessParam)
```

Il dizionario risultante può quindi essere utilizzato come input per la procedura **"preprocess_dl_dataset"**:

```
preprocess_dl_dataset (DLDataset, OutputDir,
DLPreprocessParam, PreprocessSettings, DLDatasetFileName)
```

Poiché vengono letti e scritti molti dati, questa operazione potrebbe richiedere del tempo. Come ultimo passaggio di questa fase, si seleziona la rete pre-addestrata su cui basare il training del modello. Nel nostro caso, abbiamo optato per la rete **compact**, la quale è stata caricata utilizzando il comando **read_dl_model**, così da poter effettuare un confronto con i risultati ottenuti tramite il Deep Learning Tool:

```
read_dl_model ('pretrained_dl_classifier_compact.hdl',
DLModelHandle)
```

Training del modello

Prima di iniziare sono stati definiti tutti gli hyperparameter necessari per il training, i quali sono stati impostati con la funzione **"set_dl_model_param"**. Si è deciso di utilizzare la GPU per realizzare l'addestramento, in modo da avere delle prestazioni più veloci:

```
DLDevice := DLDeviceHandles[0]
set_dl_model_param_max_gpu_batch_size (DLModelHandle, 13)
```

Nel nostro caso DLDeviceHandles[0] corrisponde proprio alla GPU e con il comando **"set_dl_model_param_max_gpu_batch_size"** impostiamo la batch size massima, ossia il numero massimo di campioni che vengono

utilizzati in una singola iterazione di addestramento per aggiornare i pesi del modello. Per garantire il confronto con i risultati ottenuti con il Deep Learning Tool è stata impostata a 13. Si è poi definito il learning rate e il numero di epoche:

```
set_dl_model_param(DLModelHandle, 'learning_rate', 0.001)
epochs := 20
```

I valori di questi due parametri sono identici a quelli del caso precedente, rispettivamente 0.001 e 20.
Dopo aver definito gli hyperparameter occorre generare il dizionario per il training e successivamente addestrare il modello. Queste due operazioni vengono fatte con le funzioni **"create_dl_train_param"**:

```
create_dl_train_param (DLModelHandle, epochs, 1, 'true',
42, [], [], TrainParam)
```

e **"train_dl_model"**:

```
train_dl_model (DLDataset, DLModelHandle, TrainParam, 0,
TrainResults, TrainInfos, EvaluationInfos)
```

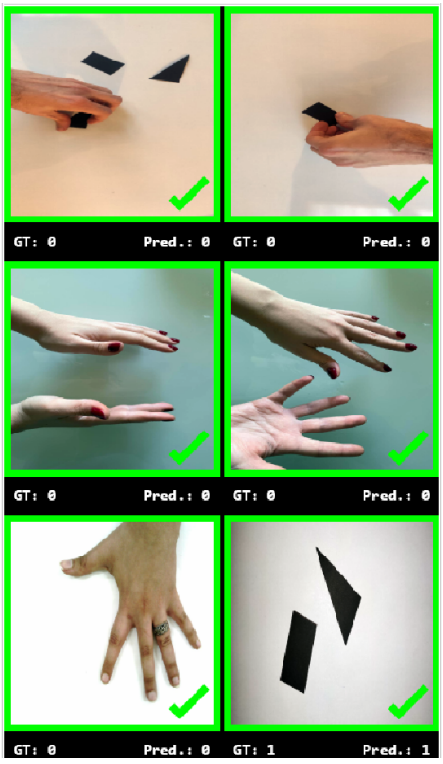


Figura 4.13: Finestra di visualizzazione durante il training di Classification.

Dopo aver eseguito tali operazioni, è stato possibile avviare il vero e proprio processo di training. Utilizzando Halcon, verrà visualizzata la finestra seguente, illustrata in Figura 4.13. È importante notare che sotto ogni immagine vi è il label originale a sinistra, ossia il ground truth (GT), e la classe predetta a destra (Pred). Il numero 0 corrisponde alla classe "Mani", mentre 1 alla classe "No mani". Se GT = Pred allora il riconoscimento della classe è corretto. Idealmente, è possibile osservare come il modello impari gradualmente ad individuare le istanze.

Vi è poi un'ulteriore finestra di visualizzazione in cui è possibile vedere alcune informazioni sul progresso e sui parametri del training, come avveniva nel Deep Learning Tool. Vengono di seguito riportati due grafici significativi: l'andamento dei valori di perdita e del Top-1 Error al variare delle epoche. I due grafici ottenuti sono riportati nelle Figure 4.14 e 4.15. Notiamo che l'andamento di Top-1 Error è quello desiderato, perché tende a zero per tutta la durata dell'addestramento. Alla fine del training, l'ultimo e il miglior modello vengono salvati su file per poi essere utilizzati nella validazione.

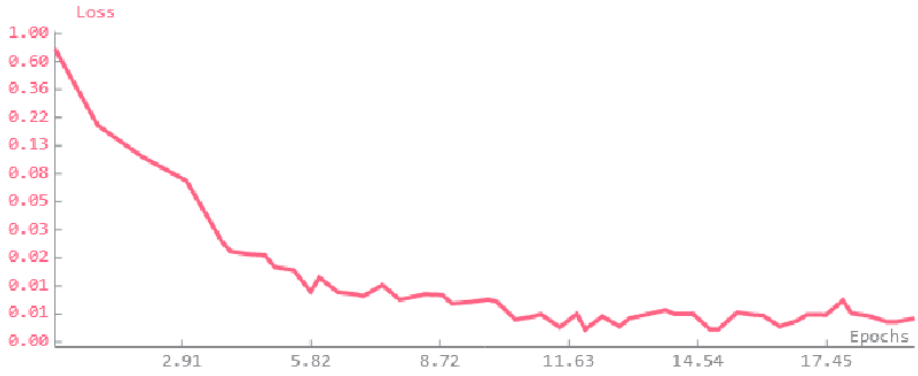


Figura 4.14: Andamento dei valori di perdita al variare delle epoche.

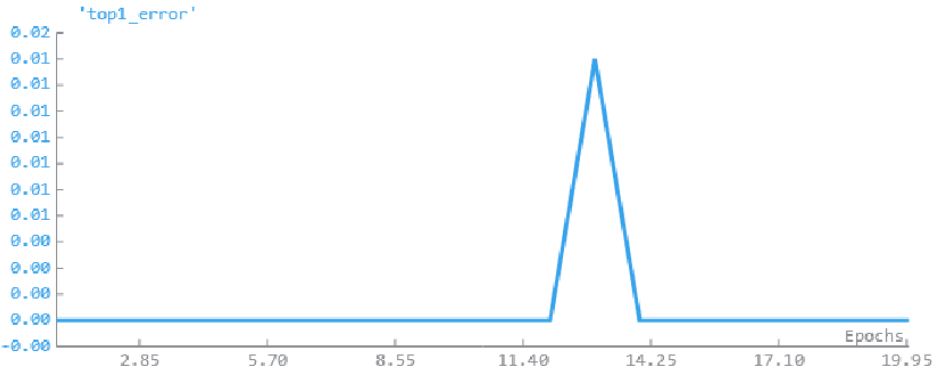


Figura 4.15: Andamento del Top-1 Error al variare delle epoche.

La durata totale del training con 20 epoche è di 1 minuto e 44 secondi, quindi molto inferiore di quella che si era ottenuta, imponendo gli stessi parametri, con il Deep Learning Tool.

Valutazione del modello

Per valutare il modello, ossia per capire se è in grado di classificare correttamente le immagini in cui sono presenti delle mani, si utilizza il comando

"evaluate_dl_model":

```
evaluate_dl_model (DLDataset, DLModelHandle, 'split',
'test', GenParamEval, EvaluationResult, EvalParams)
```

Successivamente con **"dev_display_classification_evaluation"**, visualizziamo la valutazione:

```
dev_display_classification_evaluation (EvaluationResult,
EvalParams, EvalDisplayMode, WindowDict)
```

I risultati della validazione vengono riportati in due grafici a torta, come riportato in Figura 4.16:

(1) (2)

Figura 4.16: Grafici a torta in Halcon che rappresentano i risultati della validazione. (1) rappresenta la "pie_charts_precision" e (2) la "pie_charts_recall".

Il grafico a sinistra rappresenta la "pie_charts_precision", dove "precision" fa riferimento alla proporzione di tutti i positivi predetti correttamente rispetto a tutti i positivi predetti (veri e falsi). Pertanto, è una misura di quante previsioni positive appartengono realmente alla classe selezionata. In quello a destra è invece riportata la "pie_charts_recall", dove "recall" fa invece riferimento al "tasso di veri positivi", ossia alla proporzione di tutti i positivi predetti in modo corretto rispetto a tutti i positivi reali. Pertanto, è una misura di quanti campioni appartenenti alla classe selezionata sono stati previsti correttamente come positivi [5]. Per poter confrontare i risultati ottenuti con Halcon e con il Deep Learning Tool, vengono riportati dei parametri utili per la valutazione del modello, che si sono ottenuti in questo caso:

- Top-1 Error = 0.27%
- Mean precision = 98.3%

- Mean recall = 93.3%
- F1-Score = 95.6%

Osserviamo che tali valori sono leggermente più elevati di quelli ottenuti con il Deep Learning Tool. Come abbiamo già visto, oltre ai grafici a torta è utile visualizzare in Halcon anche la matrice di confusione (confusion matrix), riportata in Figura 4.17:

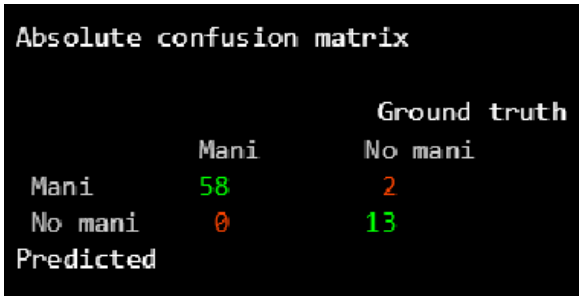


Figura 4.17: Matrice di confusione in Halcon.

Notiamo che solo in due casi le immagini sono state identificate nella classe "Mani", quando invece non lo erano, quindi abbiamo due falsi positivi (FP). Nel complesso quindi si può dire che il nostro modello è stato addestrato in modo sufficientemente preciso.

Risultati di inferenza

Nell'analisi di inferenza è necessario inizialmente esaminare i campioni che rappresentano il 15% del totale ottenuti tramite l'operazione di "split_dl_dataset". A questi campioni, sarà applicato il modello di deep learning appena addestrato per classificare le immagini e assegnare quindi un label. I passaggi da seguire in questa fase sono analoghi a quelli compiuti per l'Instance Segmentation.
Per prima cosa occorre leggere le immagini, per cui si utilizzerà l'operatore **"read_image"**. Il passo successivo sarà quello di generare un DLSample dall'immagine, utilizzando l'operatore **"gen_dl_samples_from_images"**, il quale ci permette di memorizzare le immagini fornite in una tupla di dizionari DLSamples:

```
gen_dl_samples_from_images (Image, DLSample)
```

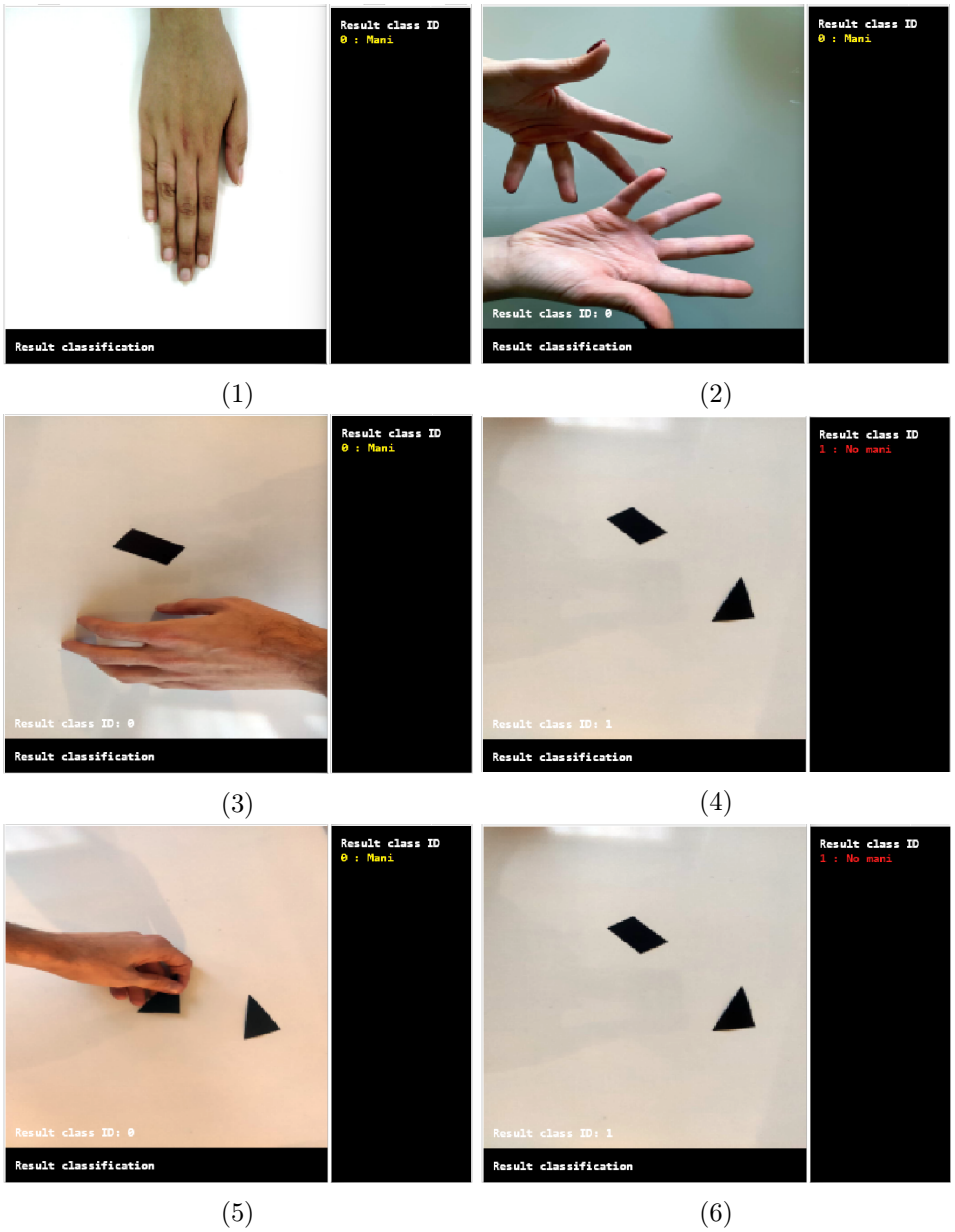
Successivamente, si è pre-processato il DLSample utilizzando l'operatore **"preprocess_dl_samples"**:

```
preprocess_dl_samples (DLSample, DLPreprocessParam)
```

È stato poi applicato il modello di deep learning addestrato sul campione con la funzione **"apply_dl_model"**:

```
apply_dl_model (DLModelHandle, DLSample, [], DLResult)
```

Infine, è possibile visualizzare i risultati della classificazione. Nella Figura 4.18 sono riportati alcuni esempi:



(1) (2)

(3) (4)

(5) (6)

Figura 4.18: Nelle figure (1), (2), (3) e (5) sono state individuate delle mani, quindi sono state etichettate con la classe "Mani" (in giallo). La (4) e la (6) invece, sono state etichettate con "No mani" (in rosso).

4.1.4 Confronto Classification tra Halcon e Deep Learning Tool

I risultati ottenuti utilizzando i due metodi sono simili in termini di precisione nella classificazione. Tuttavia, vi è una differenza significativa in termini di tempo impiegato: il Deep Learning Tool richiede un tempo maggiore rispetto alla stessa operazione eseguita con Halcon. Questo può essere dovuto ad una diversa ottimizzazione dei due software. Il Deep Learning Tool rimane comunque uno strumento utilissimo per la creazione dei dataset e dei modelli di deep learning, senza la necessità di scrivere alcuna riga di codice.

4.2 Instance Segmentation

Si è notato come l'obiettivo della classificazione sia quello di assegnare etichette alle immagini in base alla loro classe, ma non si occupa di localizzare gli oggetti all'interno dell'immagine stessa. Per questo motivo, è stata adottata la tecnica di Instance Segmentation, che consente non solo di separare le istanze e restituire due maschere distinte se sono presenti due mani, ma anche di definire la loro posizione all'interno dell'area di lavoro.
Questo approccio risulta particolarmente utile nel caso in cui si voglia, ad esempio, bloccare un cobot se la mano dell'operatore si avvicina troppo ad esso.

4.2.1 Workflow Instance Segmentation

Le fasi necessarie per implementare la tecnica di Instance Segmentation sono analoghe a quelle descritte nel Capitolo 3, verrà quindi posta particolare attenzione sui risultati ottenuti.

Preparazione dei dati

Uno dei passaggi fondamentali consiste nella creazione del dataset con il Deep Learning Tool, per il quale sono state impiegate 234 immagini raffiguranti delle mani. Per la generazione delle maschere, che in questo caso non sono più rettangolari ma hanno la forma delle mani, si è utilizzato lo **Smart Label Tool**, uno strumento del Deep Learning Tool che ci permette di disegnare dei bounding box, a partire dai quali viene creata automaticamente un'etichetta con la forma della mano. Alcuni esempi sono riportati in Figura 4.19. È importante notare come siano state generate delle maschere verdi che aderiscono perfettamente alla forma della mano. Quest'ultime svolgeranno un ruolo fondamentale durante la fase di training del modello di deep learning, perché ci garantiranno il riconoscimento delle caratteristiche della mano e la loro localizzazione. Successivamente, è stato possibile importare il dataset e iniziare la fase di training del modello di deep learning.

(1) (2) (3)

Figura 4.19: Maschere in verde generate dal Deep Learning Tool che identificano delle mani.

Training del modello

Prima di iniziare il training si sono impostati i seguenti hyperparameter: batch size = 13, learning rate = 0.001, numero di epoche = 30, larghezza dell'immagine = 512 pixel, altezza dell'immagine = 512 pixel. Si è selezionata la rete pre-addestrata **"compact"**, ovvero quella utilizzata nei precedenti esempi. Inoltre si è deciso di operare con la GPU, in modo da ottenere prestazioni più elevate.

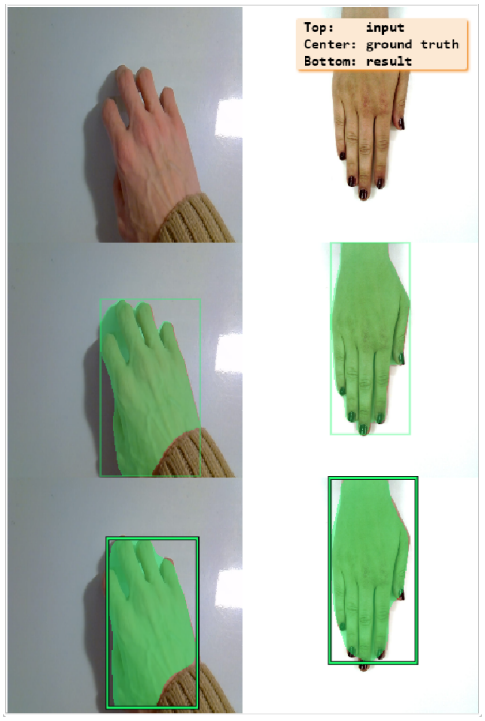


Figura 4.20: Finestra di visualizzazione durante il training di Instance Segmentation.

Durante il training in Halcon è possibile visualizzare come il modello impari a riconoscere le mani, creando delle etichette sempre più simili a quelle originali. Un esempio è riportato in Figura 4.20.
Notiamo che nella parte superiore (input) sono presenti le immagini pre-elaborate, al centro (ground truth) sono riportati i dati etichettati e nella parte inferiore (result) si possono osservare i risultati attuali del modello di deep learning, in cui il sistema cerca di adattare le maschere e i bounding box del dataset.

Nelle Figure 4.21 e 4.22, sono riportati rispettivamente il valore di perdita e

il mean_ap, al variare delle epoche.

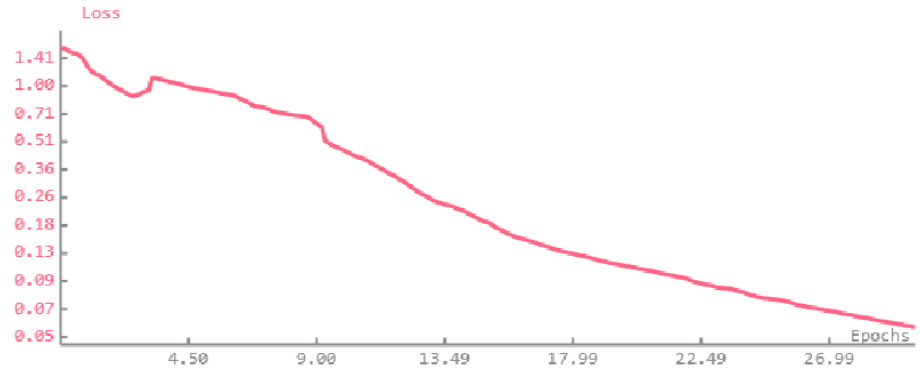


Figura 4.21: Andamento dei valori di perdita al variare delle epoche.

Figura 4.22: Andamento del mean_ap al variare delle epoche.

Nel caso in esame, è possibile notare come il valore di mean average precision (mean_ap) oscilli tra valori elevati e valori molto bassi. Questo potrebbe essere causato dalla presenza di posizioni delle mani molto diverse tra loro nel dataset, il che rende difficile per il modello riconoscere forme che non ha ancora incontrato. Tuttavia, è importante sottolineare che solo il modello con il valore di mean_ap più elevato (best) sarà utilizzato per i successivi passaggi.

Miglioramento del modello

Abbiamo già osservato come per l'addestramento del modello, uno dei parametri più importanti è il mean_ap, per cui è stata effettuata, anche in questo caso, un'analisi di convergenza per determinare il numero di epoche necessarie per ottenere un modello di deep learning valido in un tempo ridotto.

L'analisi a convergenza ha portato al grafico riportato in Figura 4.23:

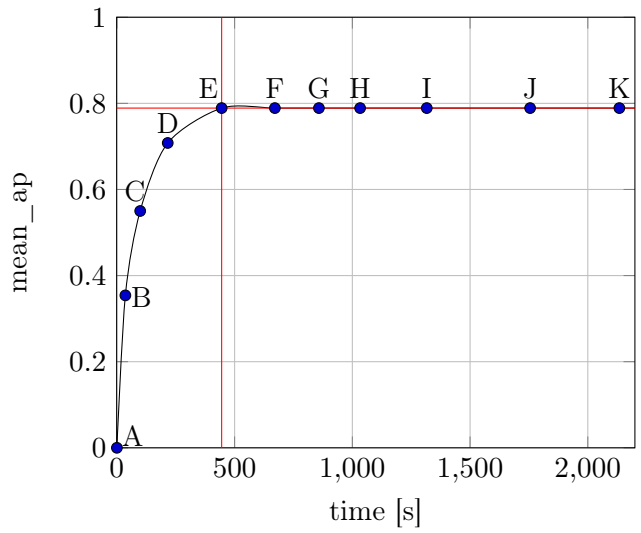


Figura 4.23: Grafico ottenuto con un'analisi di convergenza.

Per ottenere questo grafico si è aumentato progressivamente il numero di epoche (di conseguenza anche il tempo di addestramento) e si è misurato il mean_ap, ossia un valore tra 0 ed 1 che indica la precisione con cui sono stati predetti i bounding box rispetto a quelli del ground truth.
I dati ricavati da questa operazione sono riportati in tabella 4.1:

Nodo	Epoche	Tempo [s]	mean_ap
A	0	0	0
B	2	36	0.066
C	5	99	0.275
D	10	216	0.646
E	20	445	0.789
F	30	671	0.789
G	40	858	0.789
H	50	1033	0.789
I	60	1316	0.789
J	80	1755	0.789
K	100	2134	0.789

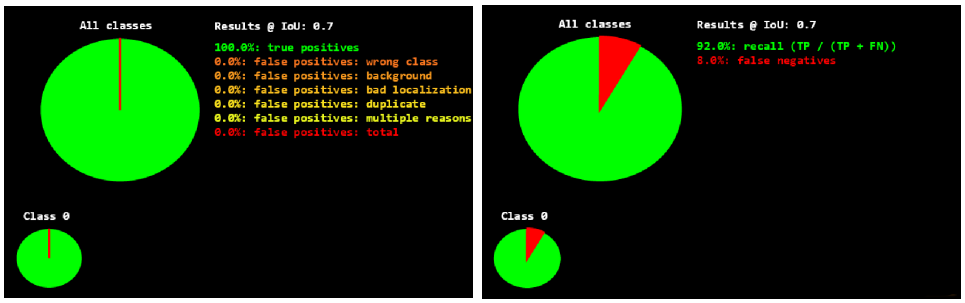
Tabella 4.1: Risultati dell'analisi di convergenza.

Possiamo osservare che il valore di mean_ap inizia a stabilizzarsi dal nodo E al valore di 0.789, che per la nostra applicazione è considerato accettabile. Questo nodo corrisponde a 20 epoche, ovvero ad un tempo di addestramento di 445 secondi.

Il numero di epoche ideale, quindi, sembra essere 20, che consente di ottenere un alto valore di mean_ap, senza richiedere un tempo di addestramento troppo lungo.

Valutazione del modello

Per la valutazione del modello, come nei casi precedenti, si fa riferimento ai grafici a torta pie_charts_precision e pie_charts_recall, riportati in Figura 4.24:



(1) (2)

Figura 4.24: Grafici a torta in Halcon che rappresentano i risultati della validazione. (1) rappresenta la "pie_charts_precision" e (2) la "pie_charts_recall".

Si può notare che nella situazione analizzata non sono stati riscontrati falsi positivi, ovvero non è stato identificato alcun oggetto come mano quando in realtà non lo era. Tuttavia, è stato riscontrato un tasso del 8% di falsi negativi, ossia alcune mani non sono state riconosciute come tali. Questi risultati sono riassunti dalla matrice di confusione illustrata nella Figura 4.25:

Figura 4.25: Matrice di confusione in Halcon.

I risultati così ottenuti sono accettabili per la nostra applicazione, quindi si è proceduto all'analisi di inferenza.

4.2.2 Acquisizione immagini in Halcon

Per l'analisi di inferenza, invece che utilizzare il 15% dei campioni ricavati con l'operazione di **split_dl_dataset**, si è deciso di ricavare le immagini da una telecamera in tempo reale. Il modo più semplice per acquisire delle immagini è quello di utilizzare l'**"Image Acquisition Assistant"**, la cui interfaccia è riportata in Figura 4.26:

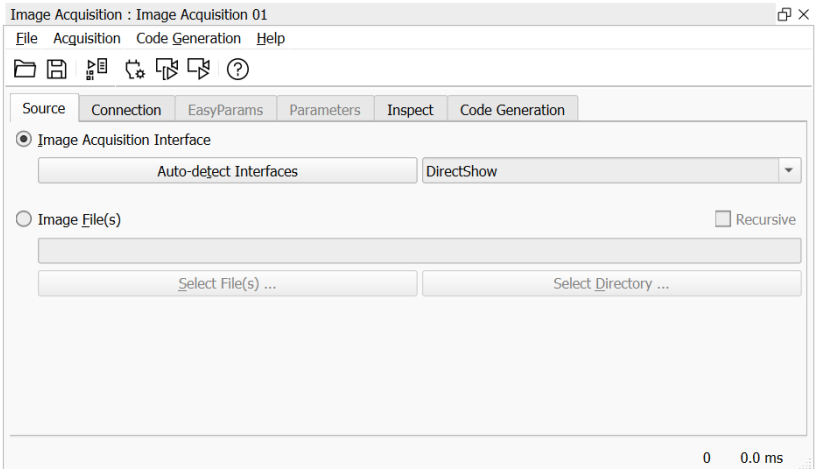


Figura 4.26: Selezione dell'interfaccia nell'Image Acquisition Assistant in Halcon.

Cliccando su **"Auto-detect Interfaces"** possiamo selezionare da una lista l'interfaccia del sistema di acquisizione immagini desiderata. In Halcon vi sono diverse interfacce standardizzate di acquisizione immagini che sono incluse nell'installazione, come ad esempio GenICam, GigE Vision, OpenNI, Video4Linux, DCAM, DirectFile, DirectShow e Twain. Un'interfaccia standardizzata di acquisizione immagini è adatta per diversi dispositivi di acquisizione immagini che non necessariamente hanno lo stesso set di parametri da regolare durante l'acquisizione dell'immagine. Vi sono interfacce definite "generiche", in cui sono consentiti parametri arbitrari. Esempi di interfacce generiche sono l'interfaccia GenICamTL, che segue il modulo GenTL dello standard GenICam, e l'interfaccia GigEVision che segue lo standard GigE Vision. Poiché i parametri per un'interfaccia generica possono essere arbitrari, le informazioni sui parametri specifici del dispositivo non vengono fornite con la descrizione dell'interfaccia ma devono essere richieste dal dispositivo. Ad esempio, se l'interfaccia segue lo standard GenICam, le informazioni necessarie sono disponibili sotto forma di un file xml che è tipicamente memorizzato sul dispositivo [3].
Nel nostro caso è stata scelta l'interfaccia **DirectShow**. Successivamente si può selezionare la telecamera che andrà a connettersi con il software Halcon, cliccando **"Connect"** nella finestra riportata in Figura 4.27:

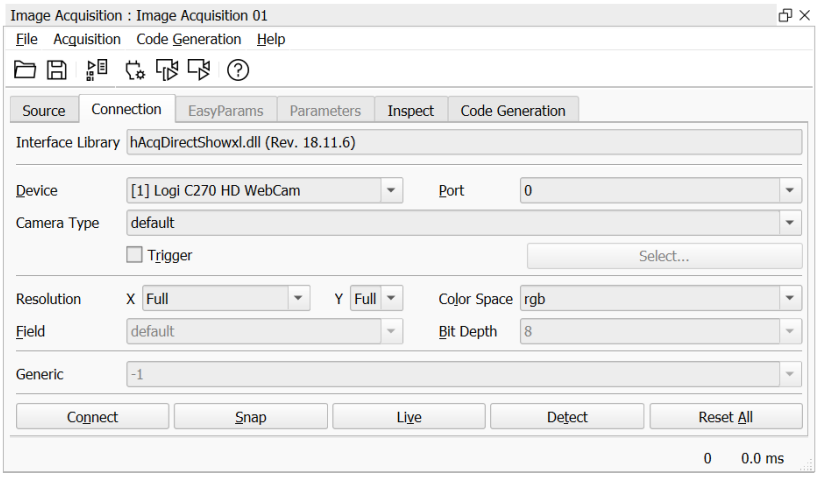


Figura 4.27: Selezione telecamera nell'Image Acquisition Assistant.

Nella finestra denominata "Parameters" riportata in Figura 4.28 è possibile configurare l'acquisizione delle immagini provenienti dalla fotocamera.

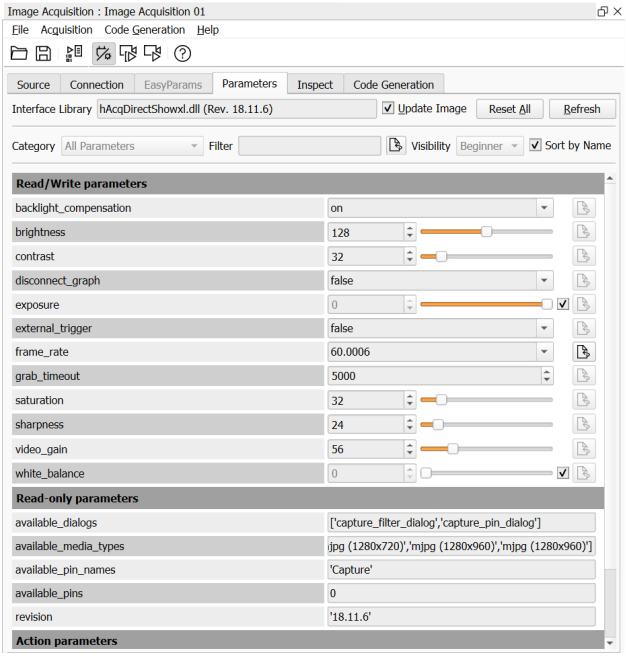


Figura 4.28: Configurazione parametri nell'Image Acquisition Assistant.

Osserviamo che sono disponibili molteplici opzioni per la regolazione dei parametri, tra cui la luminosità (brightness), il contrasto (contrast), e molti altri. Uno dei parametri chiave è il frame rate, ovvero la frequenza di acquisizione delle immagini per secondo. Nel nostro caso il valore è stato impostato

a 60, al fine di ottenere una rappresentazione fluida e continua. Qualsiasi modifica effettuata sui parametri in questa finestra verrà successivamente inserita nel codice automaticamente (con l'operatore **"set_framegrabber_param"**), sfruttando la funzionalità dell'Image Acquisition Assistant, come mostrato nella Figura 4.29:

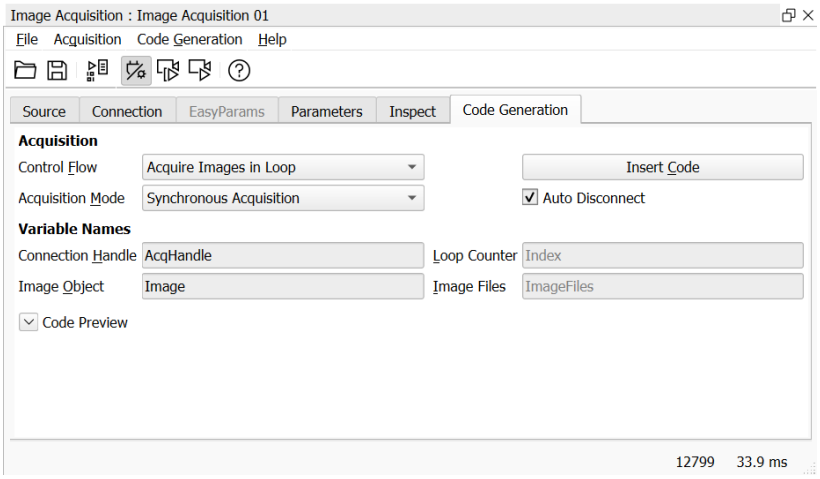


Figura 4.29: Generazione codice nell'Image Acquisition Assistant.

Si è impostata un'acquisizione delle immagini sincrona (**"Synchronous Acquisition"**), in modo che ogni immagine venga catturata ogni volta che viene chiamata la funzione **"grab_image"**. Cliccando su **"Insert Code"**, possiamo generare il codice in Halcon che ci permette di acquisire le immagini in tempo reale:

```
open_framegrabber ('DirectShow', 1, 1, 0, 0, 0, 0,
'default', 8, 'rgb', -1, 'false','default', '[1] Logi
C270 HD WebCam', 0, -1, AcqHandle)

set_framegrabber_param (AcqHandle, 'frame_rate', 60.0002)

* Loop through all images.
while(true)
    grab_image (Image, AcqHandle)
    *
    * Application deep learning model
    *
endwhile
```

La funzione **"open_framegrabber"** ci permette di connettere il dispositivo di acquisizione dell'immagine e impostare i parametri generali, ad esempio, il tipo di telecamera utilizzata o la porta a cui è collegata. Le immagini poi sono acquisite in un ciclo (while), con l'operatore **"grab_image"** [3].

4.2.3 Risultati ottenuti

Giunti a questo punto, si è deciso di utilizzare non solo il modello di deep learning in grado di riconoscere e localizzare le mani, ma di integrare anche il software di Shape-Based Matching, il quale è in grado di identificare i rettangoli, insieme ai loro centroidi e angoli di inclinazione. Questo utilizzo congiunto ci consente di simulare ciò che potrebbe effettivamente accadere in un'area di lavoro, in cui l'operatore introduce le mani mentre il cobot sta spostando dei pezzi. Quando una mano viene rilevata, il cobot si ferma immediatamente. I risultati ottenuti sono riportati in Figura 4.30:

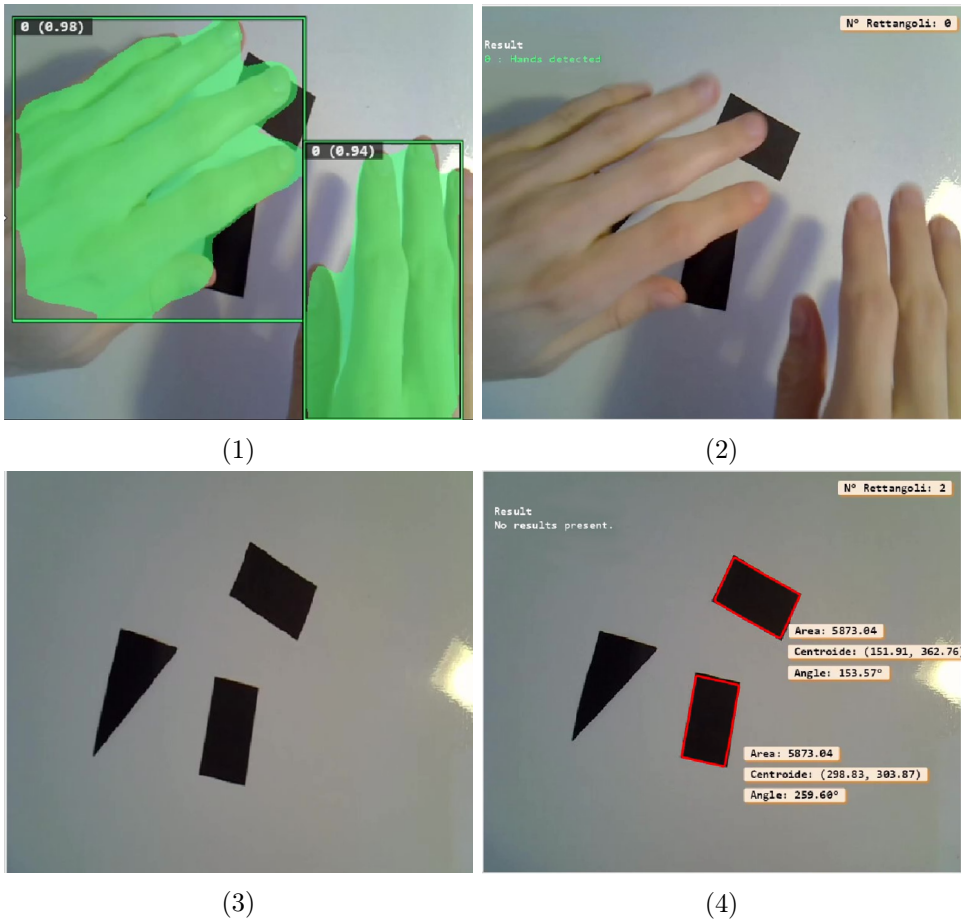


Figura 4.30: (1) e (2) presentano il caso in cui delle mani entrano nella zona di lavoro. In (3) e (4) non vi sono mani, quindi è possibile riconoscere i rettangoli, con i loro centroidi e angoli di inclinazione.

Notiamo come nel caso in cui vi siano delle mani (Figura 4.30 (1) e (2)), nella graphic window a sinistra vengano generati i bounding box e le maschere che seguono le forme delle mani in verde, invece in quella a destra

viene generato l'avviso "Hands detected". Inoltre, è interessante notare che, poiché le mani coprono i rettangoli, quest'ultimi non vengono riconosciuti e quindi il numero di oggetti rilevati è zero. Nella Figura 4.30 (3) e (4) le mani non sono presenti, quindi nella graphic window a destra viene generato il messaggio "No results present.", per cui non vi sono problemi nel riconoscere i rettangoli e le loro caratteristiche.
È importante evidenziare che, avendo utilizzato la tecnica di Instance Segmentation per riconoscere le mani, possiamo anche identificare il loro centroide e quindi, in maniera approssimata, potremmo calcolare la loro posizione rispetto al robot. Questo amplia notevolmente i possibili utilizzi di questo software, poiché si potrebbe decidere di non bloccare sempre il cobot in caso di mani nella zona di lavoro, ma di farlo solo se queste si avvicinano ad una distanza preimpostata.

Conclusione

Utilizzando le librerie Halcon, sono state implementate numerose tecniche di computer vision per il riconoscimento e la localizzazione di oggetti. La tecnica di Blob Analysis, che può essere facilmente implementata in Halcon, ha presentato un tempo medio di esecuzione di 40.71 ms, ma ha mostrato grandi limitazioni nel caso di variazioni di illuminazione e motion blur, tanto da rendere impossibile il riconoscimento degli oggetti. D'altra parte, la tecnica di Shape-Based Matching è risultata essere molto più robusta alle variazioni ambientali, rendendo possibile l'identificazione delle figure geometriche anche in caso di variazioni di luce e sfocature. Inoltre, il tempo medio di esecuzione è stato di soli 15.97 ms, pertanto è consigliato sempre l'utilizzo della tecnica di Shape-Based Matching in Halcon, rispetto alla tecnica di Blob Analysis.
Nei capitoli conclusivi è stata valutata l'efficacia delle librerie di deep learning di Halcon per il riconoscimento di figure geometriche. L'obiettivo era quello di valutare le prestazioni della tecnica di Instance Segmentation, la cui applicazione ha generato un tempo medio di esecuzione pari a 563.17 ms, quindi è risultato significativamente superiore a quello conseguito con lo Shape-Based Matching. Inoltre, con Instance Segmentation, risulta complesso identificare le caratteristiche degli oggetti riconosciuti, come ad esempio il centroide e l'angolo di inclinazione. Ciò è dovuto al fatto che il modello di deep learning è ottimo nel classificare gli oggetti, ma non altrettanto nel generare le regioni associate ad essi, dalle quali si ricavano le caratteristiche. Di conseguenza, lo Shape-Based Matching rimane la tecnica consigliata per il riconoscimento degli oggetti, poiché offre una maggiore precisione nella ricostruzione delle caratteristiche degli oggetti riconosciuti.
Le librerie di deep learning sono state impiegate per il riconoscimento di mani umane, adottando due tecniche: la Classification e la Instance Segmentation. In particolare, grazie alla Classification e all'utilizzo del Deep Learning Tool di Halcon, si sono potute identificare le mani nella zona di lavoro e, di conseguenza, segnalarne la presenza. Tuttavia, poiché la tecnica di classificazione non consente di localizzare gli oggetti, si è deciso di testare anche la tecnica di Instance Segmentation, al fine di individuare la posizione delle mani. Quest'ultima potrebbe trovare applicazione nel riconoscimento della presenza di un operatore nella zona di lavoro di un cobot, permettendo

a quest'ultimo di bloccarsi automaticamente al rilevamento di una mano e prevenendo eventuali collisioni.
Alla luce di tali considerazioni, il software MVTec Halcon si è dimostrato affidabile e preciso nel riconoscimento degli oggetti. Tale strumento consente quindi, in modo accurato e rapido, lo sviluppo di sistemi di visione avanzati per la robotica e l'automazione industriale.

Bibliografia

[1] Halcon: the Power of Machine Vision, *Solution Guide I - Basics*, 2008

[2] Halcon: the Power of Machine Vision, *Solution Guide II - Matching*, 2015

[3] Halcon: the Power of Machine Vision, *Solution Guide II-A - Image Acquisition*, 2016

[4] Halcon: the Power of Machine Vision, *Solution Guide II-D - Classification*, 2022

[5] Halcon Operator Reference, *21.05.0.0 Copyright © MVTec Software GmbH*, 1996-2021

[6] Halcon: a product of MVTec, *HDevelop User's Guide*, 1997-2022

[7] MVTec Deep Learning Tool - *Training Guide*,© Copyright 2019-2022 - MVTec Software GmbH

[8] Richard Szeliski, *Computer Vision: Algorithms and Applications, 2nd ed.* © 2022 Springer, The University of Washington

[9] Ulrich, Markus, Follmann, Patrick and Neudeck, Jan-Hendrik. *"A comparison of shape-based matching with deep-learning-based object detection."* tm - Technisches Messen, vol. 86, no. 11, 2019, pp. 685-698. `https://doi.org/10.1515/teme-2019-0076`

[10] Hafiz, A.M., Bhat, G.M. *"A survey on instance segmentation: state of the art."* Int J Multimed Info Retr 9, 171–189 (2020). `https://doi.org/10.1007/s13735-020-00195-x`

[11] Afifi, M. *"11K Hands: Gender recognition and biometric identification using a large dataset of hand images."* Multimedia Tools and Applications 78, 20835–20854 (2019). `https://doi.org/10.1007/s11042-019-7424-8`

[12] Blob Analysis - Visco Technologies USA, Inc. `https://www.visco-tech.com/usa/technical/direction-presence/blob/`

[13] Shape-based matching with MVTec HALCON: speedup vs. robustness, advanced parameters https://www.mvtec.com/services-support/videos-tutorials/single-video/shape-based-matching-with-mvtec-halcon

[14] How to start with deep learning https://www.mvtec.com/technologies/deep-learning/how-to-start-with-deep-learning

[15] Halcon - The powerful software for your machine vision application https://www.mvtec.com/products/halcon

[16] Reti neurali convoluzionali - MathWorks. https://it.mathworks.com/discovery/convolutional-neural-network-matlab.html