ALMA MATER STUDIORUM – UNIVERSITÀ DI BOLOGNA

Scuola di Ingegneria ed Architteture
Corso di Laurea Magistrale in Ingegneria Informatica

STRUMENTO DI REALTÀ AUMENTATA SU DISPOSITIVI MOBILI PER LABELING DI IMMAGINI SEMI-AUTOMATICO

Relazione finale in
COMPUTER VISION AND IMAGE PROCESSING

Relatore
Prof. LUIGI DI STEFANO

Candidato
LORENZO COTTIGNOLI

Correlatore
PhD. DANIELE DE GREGORIO

Terza Sessione di Laurea
Anno Accademico 2017 – 2018

KEYWORDS

Computer Vision

Augmented Reality

Object Detection

Semi-automatic Image Labeling

Deep Learning

Indice

Introduzione

In questa tesi verrà proposta l'implementazione di un sistema innovativo di realtà aumentata su dispositivi mobili per il labeling semi-automatico di immagini.
Grazie al progresso degli ultimi anni nell'ambito dell'apprendimento automatico, è possibile creare modelli complessi per il rilevamento di oggetti nelle immagini (*Object Detection*), indipendentemente dalle caratteristiche di tali oggetti. Questi modelli, solitamente, corrispondono a reti neurali, le quali necessitano di una grande quantità di esempi per apprendere il comportamento desiderato. Attualmente nell'ambito dell'Object Detection, la collezione di tali esempi avviene manualmente e questo comporta un notevole dispendio di tempo e risorse.
L'obiettivo di questa tesi consiste nel fornire uno strumento di supporto semplice ed intuitivo, in grado di ridurre drasticamente i tempi di acquisizione degli esempi, necessari per l'addestramento delle reti neurali per l'Object Detection.
La tesi è strutturata in tre capitoli: nel primo verrà fornito il background teorico per una maggiore comprensione dell'importanza dell'oggetto di questa tesi e delle tematiche trattate; nel secondo capitolo verranno descritti nel dettaglio il sistema proposto e le relative scelte implementative; nell'ultimo capitolo verranno mostrati e valutati diversi dataset generati grazie all'utilizzo del sistema sviluppato.

Capitolo 1

Background

In questo capitolo verranno introdotti argomenti teorici per permettere di comprendere appieno il funzionamento e l'importanza dello strumento di realtà aumentata su dispositivi mobili per labeling di immagini semi-automatico, oggetto di questa tesi.

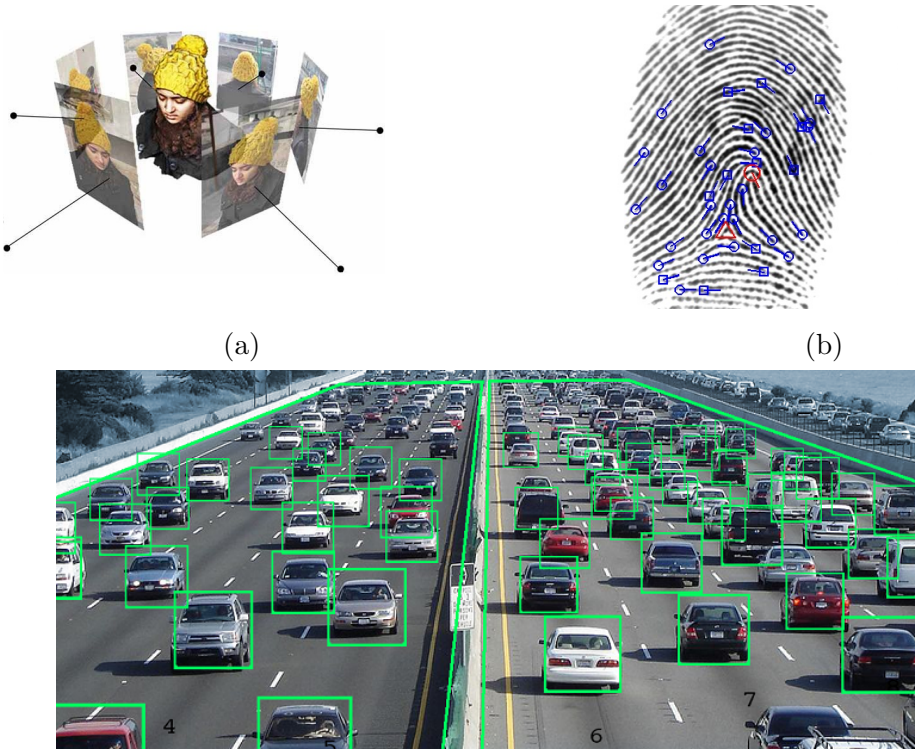
1.1 Computer Vision

Come umani, noi siamo in grado di percepire la struttura tridimensionale del mondo che ci circonda ed estrarre delle informazioni da essa con estrema facilità [1]. Nel caso guardassimo un vaso di fiori vicino a noi, saremmo in grado di riconoscere facilmente la forma di ogni petalo anche se la superficie è soggetta a zone di luce ed ombra e saremmo in grado di segmentare, senza alcuno sforzo, ogni fiore dallo sfondo. Se guardiamo uno scaffale al supermercato siamo in grado di identificare facilmente i prodotti che ci interessano e la loro posizione.
Il campo scientifico che cerca di simulare ed automatizzare attività che il sistema visivo umano può svolgere, tra cui gli esempi sopracitati, è la Visione Artificiale (Computer Vision). La *Computer Vision* è un campo interdisciplinare, in quanto trova le proprie basi nella fisica, matematica e nella neurobiologia, specialmente negli studi sul sistema di visione biologica. La Computer Vision comprende metodi di acquisizione, elaborazione, analisi e comprensione di immagini e video digitali allo scopo di estrarre informazioni di alto livello dal mondo reale da convertire in informazioni numeriche.

La Computer Vision viene utilizzata attualmente in un'ampia varietà di applicazioni del mondo reale, quali:

- **Riconoscimento ottico dei caratteri (Optical Character recognition, OCR)**: e.g. utilizzato per la lettura dei codici postali scritti a mano, il riconoscimento automatico della targa, rilevamento del testo contenuti in un documento scannerizzato;
- **Medical imaging**: e.g. il rilevamento automatico di possibili malattie negli organi o nelle ossa per migliorare le diagnosi, permettere di eseguire studi a lungo termine sulla morfologia cerebrale delle persone;
- **Rilevazione eventi**: e.g. monitoraggio di intrusi, controllo delle piscine per prevenire annegamenti, analisi del traffico nelle strade (vedi figura 1.1c);
- **Navigazione autonoma**: e.g. rilevamento di ostacoli o pedoni per la guida automatica di veicoli o robot;
- **Rilevazione di errori**: e.g. monitoraggio delle macchine di produzione automatizzate per eliminare possibili oggetti difettosi;
- **Modellazione 3D**: e.g. conversione di più immagini 2D di un oggetto o persona in un modello 3D (vedi figura 1.1a);
- **Biometria**: e.g. per applicazioni forensi, per autenticazioni automatiche tramite retina o impronta digitale (vedi figura 1.1b).

Come è possibile notare negli esempi precedenti, una parte molto importante della Computer Vision è il rilevamento di oggetti specifici (Object Detection) all'interno di immagini digitali.



(a) (b)

(c)

Figura 1.1: Applicazioni reali di Computer Vision: (a)Modellazione 3D a partire da immagini 2D; (b)Riconoscimento impronte digitali; (c)Sorveglianza automatica del traffico.

1.2 Object Detection

La *Object Detection* può essere definita nel seguente modo: data un'immagine di riferimento (model image) dell'oggetto da ricercare, determinare se l'oggetto è presente o meno nell'immagine sotto analisi (target image) e, nel caso di rilevamento, stimarne la posa. Con il termine posa si definisce la rotazione e la traslazione di un oggetto rispetto ad un sistema di riferimento.
La Object Detection è un problema intrisicamente complesso, poiché un oggetto all'interno della *target image* può risultare molto diverso da quello presente nella *model image*. Questo può essere dovuto a diversi motivi (figura 1.2), quali: l'esposizione dell'oggetto a fonti di illuminazione differenti, la bassa risoluzione delle immagini, rotazioni, occlusioni, cambiamenti di scala o sfocature dovute

al movimento del soggetto.

Figura 1.2: In queste immagini vi sono le possibili trasformazioni a cui è sottoposto un oggetto:(a)cambiamenti di illuminazione; (b)rotazione; (c)presenza di occlusioni; (d)sfocatura dovuta al movimento; (e)scale diverse; (f)immagini a bassa qualità.

Nell'ambito della Object Detection, gli oggetti appartenenti ad un'immagine possono essere rilevati a diversi livelli di astrazione:

(a) **Image classification**: data una foto si ottiene l'elenco degli elementi presenti al suo interno, ma non vi è alcuna informazione relativamente alla loro posizione;

(b) **Object localization**: gli algoritmi di questo tipo hanno l'obiettivo di definire, per ogni oggetto nell'immagine, una *bounding box* e un'etichetta (*label*) che ne definisca la classe di appartenenza. Con il termine *bounding box* si fa riferimento al più piccolo poligono, solitamente un quadrilatero, in grado di delimitare al suo interno tutti i punti dell'immagine appartenenti ad un oggetto;

(c) **Semantic segmentation**: in questo livello di astrazione ad ogni pixel dell'immagine viene associato un valore in base alla classe di appartenenza. Tutti i pixel dello sfondo appartengono ad una classe particolare, detta *Background*;

(d) **Instance segmentation**: risulta simile alla Semantic segmentation ma il valore che viene assegnato al pixel è diverso per ogni istanza dell'oggetto presente nell'immagine.

Nella figura 1.3 è possibile notare le differenze tra i vari livelli di astrazione appena descritti.

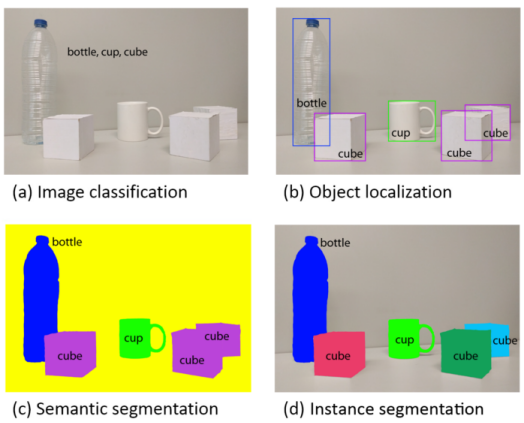


Figura 1.3: Rilevamento degli oggetti a diversi livelli di astrazione

In questa tesi, con il termine *Object Detection*, si fa riferimento al secondo livello di astrazione appena descritto, ovvero alla *object localization* (bounding

box + label).
Il problema del rilevamento degli oggetti ha ottenuto un alto grado di interesse nel corso degli anni, per questo motivo sono stati proposti diversi tipi di algoritmi per risolverlo. Tra gli approcci classici, quello in grado di fornire risultati migliori è il *Feature-based Matching*. Nell'ultimo decennio, grazie ai molteplici studi sulle reti neurali, il rilevamento degli oggetti si è concentrato principalmente sull'utilizzo di modelli di *Deep Learning*, i quali raggiungono le performance migliori in questo ambito.

1.2.1 Feature-based Matching

Per il rilevamento di oggetti in un'immagine, questi algoritmi suddividono il flusso di lavoro in più fasi: estrazione e matching di feature locali, validazione geometrica ed infine stima della posa dell'oggetto.
Una *feature locale* consiste in un *pattern* dell'immagine che si differenzia da ciò che le si trova attorno. Solitamente è associata a un cambiamento di una proprietà dell'immagine o di più proprietà contemporaneamente. Le proprietà di un'immagine più comunemente considerate sono l'intensità, il colore e la trama. Per estrarre le feature vi sono diversi algoritmi, i più utilizzati sono:

1. **SIFT (Scale-Invariant Feature Transformation)** [2] il suo rilevatore utilizza la differenza tra gaussiane (DoG) per approssimare il Laplaciano (LoG) ed infine effettua una ricerca dei massimi nello spazio di scala.

2. **SURF (Speeded Up Robust Features)** [3] fa uso di *integral image* per calcolare in modo efficiente un'approssimazione grezza della matrice hessiana.

3. **FAST (Features from Accelerated Segment Test)** [4] valuta solo l'intensità di un numero limitato di particolari pixel usando gli alberi decisionali.

Nella figura 1.4, sono mostrate le feature estratte da una *model image* (sinistra) e una *target image* (destra) tramite l'algoritmo SIFT.

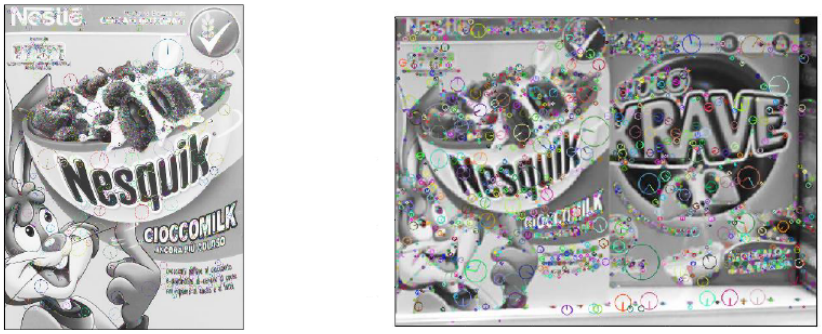


Figura 1.4: Estrazione di feature tramite SIFT.

Una volta ottenuti le feature e i rispettivi descrittori, è necessario ricercare le corrispondenze (*matches*) tra le due immagini. L'idea principale è quella di confrontare le varie feature tra le due immagini e prendere quelle più simili. Questo tipo di problema può essere risolto con la tecnica *Nearest Neighbours Search*.
Una volta effettuata la fase di matching, è necessario valutare le corrispondenze per eliminare quelle non valide (vedi figura 1.5). Secondo G. Lowe et. al [5] un match è valido, se:

$$\frac{d_{NN}}{d_{2-NN}} \leq T$$

dove d_{NN} è la distanza tra il feature point della scena e il suo primo NN (Nearest Neighbour), d_{2-NN} invece è la distanza rispetto al suo secondo NN. Lowe, inoltre, ha dimostrato che con T = 0.8 è possibile rigettare il 90% di match errati e perdere solo il 5% di quelli corretti.

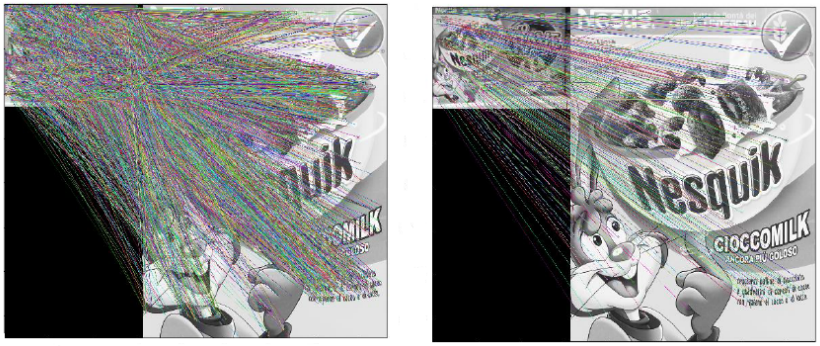


Figura 1.5: A sinistra è possibile vedere tutti i match tra le due immagini; a destra, invece, sono mostrati solo i match validi.

Successivamente, è necessario effettuare la validazione geometrica. Un algoritmo possibile per questa fase è GHT (Generalized Hough Transform) [6], la quale permette, ad esempio di rilevare possibili baricentri dell'oggetto da rilevare nella target image (vedi figura 1.6).

Figura 1.6: Rilevazione dei possibili baricentri nella target image.

Per identificare l'esatta posizione del baricentro dell'oggetto, è necessario effettuare una ricerca dei massimi locali. Ogni massimo trovato corrisponde ad una possibile istanza del modello nella target image.
L'ultima fase è quella di stima della posa degli oggetti grazie alle coppie di feature model-target image. Una volta ottenuta la posa è sufficiente calcolare la bounding box dei vari oggetti (vedi figura 1.7).

Figura 1.7: Rilevazione multipla della model image (destra) nella target image (sinistra).

1.3 Deep Learning

Il *Deep Learning* è un ramo del *Machine Learning* che si basa su algoritmi di apprendimento a diversi livelli di rappresentazione in modo da modellare complesse relazioni tra i dati. Il Deep Learning è un approccio emergente ed è stato ampiamente applicato nei domini di intelligenza artificiale tradizionali [7] [8].
Inoltre, il Deep Learning ha dimostrato di aver successo nelle attività di visione artificiale perché in grado di estrarre feature appropriate. Nelle più recenti competizioni di riconoscimento visivo su vasta scala di ImageNet [9] (ILSVRC, *ImageNet Large Scale Visual Recognition Challenge*), infatti, i metodi di Deep Learning sono stati ampiamente adottati da diversi ricercatori e hanno raggiunto valori di precisione molto elevati.
I modelli utilizzati nell'ambito del Deep Learning sono le *Deep Neural Network* (DNN), cioè reti neurali con una struttura molto complessa. In questa tesi verranno trattate le reti neurali convoluzionali (CNN), le quali sono una tipologia di DNN che utilizzano filtri convoluzionali e quindi adatte a segnali 2D come le immagini. Per una maggiore comprensione di tali DNN, è necessario definire il concetto di rete neurale.
Le reti neurali, dette anche *reti neurali artificiali* (ANN), sono sistemi di elaborazione ispirati alle reti neurali biologiche che costituiscono il cervello animale.

Questi sistemi apprendono, migliorando progressivamente la loro abilità, ad eseguire determinate attività basandosi su degli esempi, generalmente senza una programmazione specifica per l'attività richiesta. Una ANN è basata su un insieme di unità connesse tra loro, note come *neuroni artificiali* (vedi figura 1.8). Ogni connessione (sinapsi) tra neuroni permette la trasmissione del segnale ad un altro neurone. I neuroni associano ad ogni ingresso un peso, il quale viene utilizzato nella fase di propagazione del segnale per effettuare una media pesata dei valori in ingresso. Il risultato ottenuto viene passato ad una funzione non lineare (per esempio *ReLU*, Retified Logic Unit) che genera l'output del neurone. I neuroni sono organizzati in strati (*layer*). Inoltre, i neuroni appartenenti allo stesso strato non sono connessi tra loro, ma sono connessi con tutti i neuroni del layer precedente, da cui ottengono i valori di input, e con tutti quelli del layer successivo, a cui forniscono l'output.
Come è mostrato in figura 1.8 ogni rete neurale è formata da più strati, i quali sono suddivisi in:

- **input layer**: questo strato riceve i segnali esterni sotto forma di array monodimensionale e, tramite i propri neuroni, li trasmette agli strati più interni.

- **hidden layer**: questi strati formano la parte interna della rete, ovvero dove avviene l'elaborazione dei dati in ingresso. In ogni rete, in base alla tipologia, sono presenti uno o più di questi strati.

- **output layer**: questo strato fornisce all'esterno il risultato ottenuto.

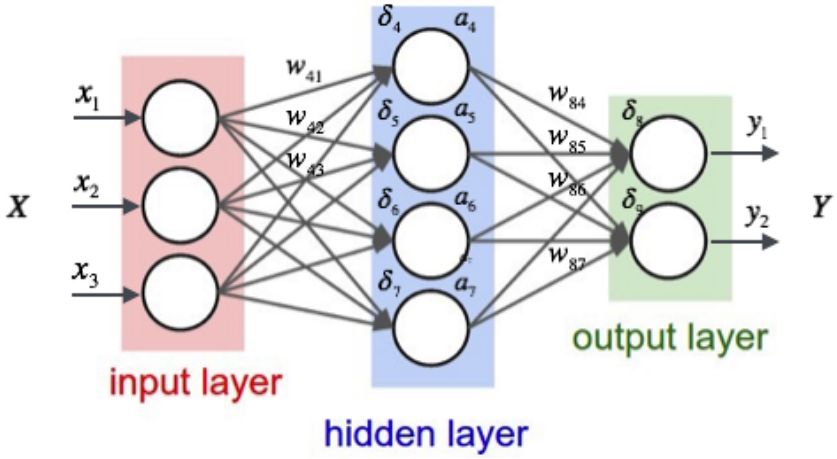


Figura 1.8: Struttura di una rete neurale artificiale.

Per comprendere meglio il motivo per il quale le ANN non sono utilizzate nell'elaborazione di immagini intere, consideriamo il seguente esempio: dato un dataset di immagini 32x32x3 (altezza, larghezza e numero di canali per i colori) e un singolo nodo (completamente connesso) del primo hidden layer di una ANN, tale nodo avrà $32 * 32 * 3 = 3072$ parametri. Nel caso utilizzassimo immagini un po' più grandi, 200x200x3, avremmo $200x200x3 = 120000$ parametri. Inoltre, vi saranno molti neuroni in una ANN, di conseguenza il numero di parametri cresce molto rapidamente. Per questo motivo, una ANN risulta molto dispendiosa e l'enorme quantità di parametri porterebbe rapidamente all'overfitting [10]. Inoltre, le ANN non modellano la relazione spaziale che c'è tra i pixel di una immagine, perché la trasformano in un array monodimensionale e quindi perdono la struttura 2D.
Uno dei modelli di Deep Learning più importanti ed attualmente utilizzati in ambito di visione artificiale sono le reti neurali convoluzionali (CNN). Il modello delle CNN è ispirato al meccanismo di percezione visiva naturale delle creature viventi. Queste reti neurali, infatti, sono progettate per elaborare dati che si presentano sotto forma di array multidimensionali, ad esempio un'immagine a colori composta da tre array bidimensionali contenenti ognuno l'intensità dei pixel nei tre canali di colore RGB. Generalmente, una CNN è formata da tre tipi di strati:

- **convoluzionale (convolutional layer)**: negli strati convoluzionali, una CNN utilizza diversi kernel per convolvere sia l'intera immagine che

le mappe di feature intermedie, generando così varie mappe di feature (vedi figura 1.9). Ci sono tre principali vantaggi nell'utilizzo dell'operazione di convoluzione:

1. il meccanismo di condivisione del peso nella stessa mappa delle feature riduce il numero di parametri;
2. la connettività locale impara le correlazioni tra pixel vicini;
3. l'invarianza rispetto alla posizione dell'oggetto.

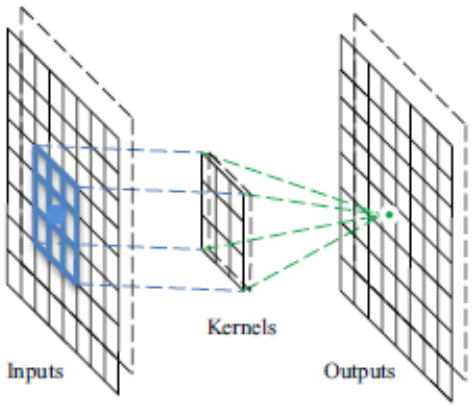


Figura 1.9: Struttura di un convolutional layer.

- **di raggruppamento (pooling layer)**: generalmente, uno strato di raggruppamento segue uno strato di convoluzione ed è utilizzato per ridurre le dimensioni della mappa delle feature e dei parametri della rete (vedi figura 1.10). Questi layer sono invarianti alla traslazione poiché tengono in considerazione i pixel vicini. Le strategie più comunemente utilizzate sono il raggruppamento medio (average pooling layer) e il raggruppamento massimo (max pooling layer).

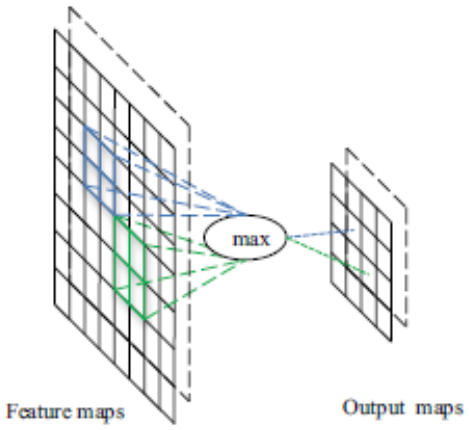


Figura 1.10: Struttura di un max pooling layer.

- **completamente connesso (fully connected layer)**: questo strato segue l'ultimo pooling layer della rete e solitamente ce ne sono diversi per convertire le mappe delle feature 2D in un vettore di feature 1D, per ottenere un'ulteriore rappresentazione delle feature. I fully connected layer si comportano come una rete neurale tradizionale e contengono circa il 90% dei parametri in una CNN (vedi figura 1.11).

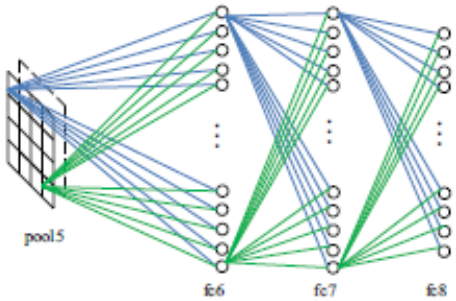


Figura 1.11: Struttura di un fully connected layer.

Solitamente, una CNN è una rete neurale gerarchica in cui i primi strati sono un alternarsi di convolutional layer e pooling layer seguiti, infine, da alcuni fully connected layer.
L'addestramento di una rete si suddivide in due fasi: la prima di *forward* e

la seconda di *backward*. Nella prima fase l'obiettivo è quello di rappresentare l'immagine di input con i parametri correnti (pesi e bias) in ogni strato. Successivamente, la predizione in uscita, insieme alle etichette contenenti la conoscenza base dell'immagine in esame, è utilizzata per calcolare la funzione di loss. A questo punto, basandosi sulla funzione di loss, nella fase di backward si calcola il gradiente che modifica i parametri in modo tale da minimizzare l'errore. Infine, tutti i parametri vengono aggiornati rispetto ai gradienti calcolati e viene ripetuto il procedimento. Dopo un numero sufficiente di iterazioni delle due fasi, il procedimento di addestramento della rete può essere interrotto.
Nel corso degli anni sono state proposte diverse tipologie di CNN che risolvessero il problema della Object detection e le più utilizzate attualmente sono *YOLO* [11](You Only Look Once) e *SSD* [12] (Single Shot multibox Detector).

1.3.1 YOLO: You Only Look Once

YOLO è un nuovo approccio per la rilevazione di oggetti proposta da Joseph Redmon et. al [11]. Tale approccio, infatti, fornisce una rete neurale in grado di predire le bounding box e le probabilità di classe associate, valutando l'intera immagine una sola volta. Questa rete usa feature dell'intera immagine per predire le bounding box. Inoltre, identifica contemporaneamente tutte le bounding box di tutte le classi di oggetti presenti in un'immagine. Il design di YOLO consente l'allenamento end-to-end e di raggiungere velocità realtime mantenendo una precisione media molto elevata.
Il sistema divide l'immagine di input in una griglia $S \times S$. Se il centro di un oggetto cade in una cella della griglia, allora quella cella è responsabile della rilevazione di quell'oggetto (vedi figura 1.12).

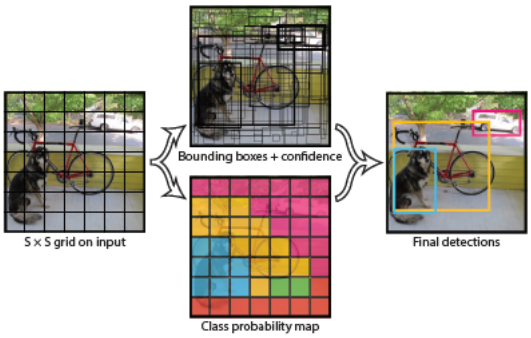


Figura 1.12: Rilevazione oggetti tramite l'approccio YOLO.

Ogni cella della griglia predice B bounding box e i valori di confidenza di tali celle. Questi valori di confidenza riflettono quanto il modello è sicuro che la box contenga un oggetto e anche quanto accurata sia la box che ha predetto. Formalmente si definisce la confidenza come $Pr(Object) * IOU_{pred}^{truth}$. Se nessun oggetto esiste in quella cella, i punteggi di confidenza saranno zero. Altrimenti il punteggio di confidenza deve essere uguale all'intersezione sull'unione (IOU) tra la box prevista e la ground truth.
Ogni bounding box predetta consiste in 5 valori: x, y, w, h e la confidenza. Le coordinate (x,y) rappresentano il centro della box relative ai limiti della cella della griglia, mentre la larghezza (w) e l'altezza (h) sono relative all'intera immagine. Infine, la confidenza, come detto in precedenza, rappresenta l'IOU tra la box prevista e qualsiasi box di verità.
Ogni cella della griglia predice anche C probabilità di classe condizionale, $Pr(Class_i|Object)$. Queste probabilità sono condizionate dalla presenza o meno di un oggetto nella cella della griglia. Viene predetto solo un insieme di probabilità di classe per ogni cella, indipendentemente dal numero box B.
In fase di test vengono moltiplicate le probabilità di classe condizionale e predizioni di confidenza delle singole box, $Pr(Class_i|Object) * Pr(Object) * IOU_{pred}^{truth} = Pr(Class_i) * IOU_{pred}^{truth}$ così da fornire punteggi di confidenza specifici per classe per ogni box. Questi valori codificano sia la probabilità che quella classe appaia nella box sia quanto bene la box predetta si adatti all'oggetto.
Il modello implementato consiste in una rete neurale convoluzionale e l'architettura di tale rete è ispirata a GoogleNet, un modello per la classificazione di

immagini (vedi figura 1.13).

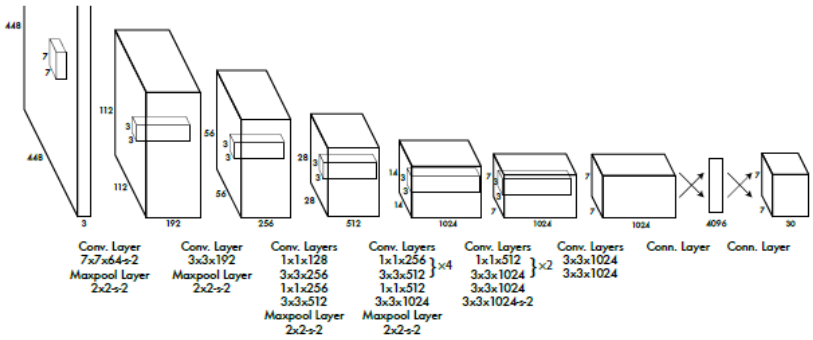


Figura 1.13: Architettura della rete neurale convoluzionale YOLO.

1.3.2 SSD: Single Shot multibox Detector

L'approccio SSD [12], come YOLO, si basa su una rete convoluzionale feed-forward che produce una collezione a dimensione fissa di bounding box e di punteggi per la presenza di istanze di oggetto in tali box, seguita da una fase di soppressione dei non massimi per produrre i rilevamenti finali. La struttura di questa rete (vedi figura 1.14) è studiata per produrre rilevamenti con le seguenti caratteristiche chiave:

- **mappe di feature multi-scala per il rilevamento**: vengono aggiunti convolutional feature layer alla della rete di base. Questi strati diminuiscono in dimensioni progressivamente e consentono previsioni di rilevamento a diversi livelli di scala. Il modello convoluzionale per la predizione dei rilevamenti è diverso per ogni feature layer;

- **Predittori convoluzionali per il rilevamento**: ogni feature layer aggiunto può produrre un insieme fisso di predizioni di rilevamento utilizzando una serie di filtri convoluzionali. Per un feature layer di dimensione $m \times n$ con p canali, l'elemento base per la predizione dei parametri di un potenziale rilevamento è un piccolo *kernel* $3x3xp$. Tale kernel produce un punteggio per una categoria o uno spostamento rispetto alle coordinate di default della box. In ciascuna delle $m \times n$ posizioni in cui viene applicato il kernel, viene prodotto un valore di output. I valori di output dell'offset della bounding box vengono misurati in relazione a una

posizione predefinita della box relativa a ciascuna posizione della mappa delle feature.

- **Box e proporzioni predefinite**: sono associate una serie di bounding box predefinite con ciascuna cella della mappa delle feature, per più mappe delle feature nella parte parte superiore della rete. Le box predefinite affiancano la mappa delle feature in modo convoluzionale, affinché la posizione di ogni box rispetto alla cella corrispondente sia fissa. In ciascuna cella della mappa delle feature, vengono predetti gli offset relativi alle forme della box predefinita nella cella, nonché i punteggi per classe che indicano la presenza di un'istanza di classe in ogni di queste box. In particolare, per ogni k box in una certa posizione, vengono calcolati c punteggi di classe e i 4 offset rispetto alla forma della box di default originale. Ciò comporta un totale di $(c+4)k$ filtri che sono applicati intorno ad ogni posizione nella mappa delle feature, producendo $(c+4)kmn$ output per una mappa di feature di dimensioni $m \times n$. Consentendo diverse forme di box predefinite in svariate mappe di feature, è possibile discretizzare in modo efficiente lo spazio delle possibili forme delle box di output.

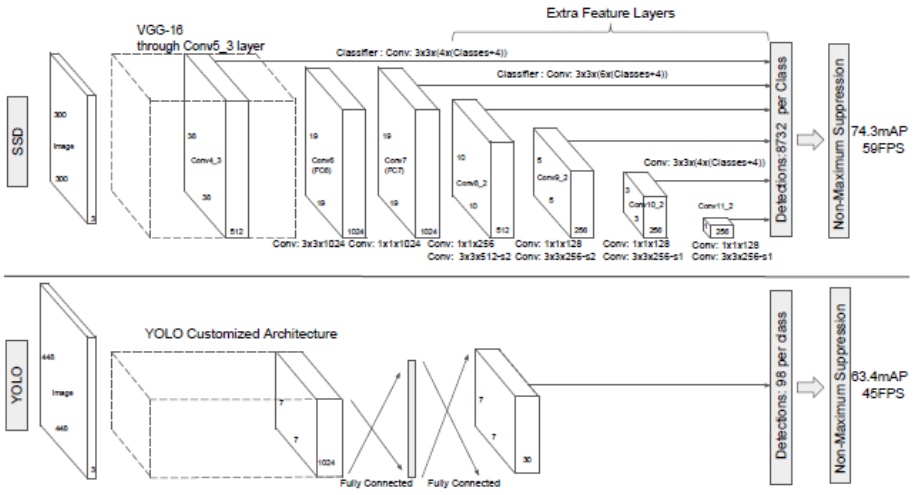


Figura 1.14: Confronto tra i due modelli di rilevamento a single shot: SSD e YOLO

1.4 Object Detection Datasets

Le reti neurali, come detto nei capitoli precedenti, stanno prendendo il sopravvento sulle tecniche di visione artificiale tradizionali, ma essendo modelli *Data-Driven*, cioè basati sui dati, necessitano di una grande mole di immagini di training per essere addestrate. Infatti, per permettere alla rete di poter imparare il comportamento desiderato, la fase di addestramento utilizza una grande quantità di esempi. Inoltre, è fondamentale che a tali immagini vi siano associate informazioni, quali la classe e la posizione degli oggetti presenti al loro interno.
Poiché collezionare grandi quantità di dati risulta difficile e molto oneroso, sono stati creati dei dataset di pubblico dominio per favorire la ricerca e lo sviluppo di algoritmi innovativi. Per quanto riguarda l'ambito della visione artificiale, i principali dataset disponibili sono: *ImageNet* [9], *MS-COCO* [13], *Open Images* [14] e *THE PASCAL VOC* [15].
ImageNet è costruito sopra la struttura gerarchica fornita da WordNet, il database semantico-lessicale di lingua inglese concepito per definire e descrivere in concetti espressi dai vocabolari. Ogni concetto importante in WordNet, possibilmente descritto da diverse parole o frasi, è definito insieme di sinonimi, detti *synset* (vedi figura 1.15). Ci sono più di 100000 synset in WordNet e la maggior parte sono sostantivi (80000+). ImageNet punta a contenere in media 1000 immagini per illustrare ognuno di questi synset. La qualità delle immagini di ogni concetto è controllata e annotata senza ausilio di strumenti automatici.

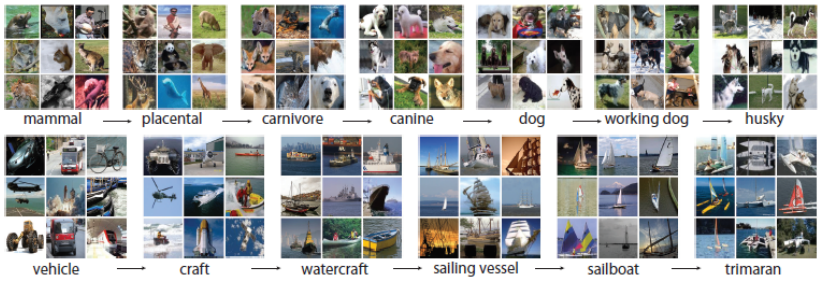


Figura 1.15: Due esempi di attraversamento della struttura gerarchica fornita da ImageNet. Dal concetto generale (sinistra) a quello specifico (destra).

Microsoft Common Objects in Context (MS-COCO) è un dataset di grandi dimensioni utilizzato per affrontare tre problemi di ricerca di base nella comprensione di una scena: rilevamento di viste non iconiche (o prospettive non canoniche) di oggetti, ragionamento contestuale tra oggetti e rilevamento 2D di precisione di oggetti (vedi figura 1.16). Il dataset contiene 91 categorie di oggetti comuni, 82 dei quali hanno più di 5000 istanze etichettate. In totale il dataset conta 2500000 istanze etichettate su 328000 immagini. A differenza di ImageNet, COCO ha meno categorie ma più istanze per categoria. Questo può favorire l'apprendimento di modelli di oggetti dettagliati nel caso di localizzazione 2D di precisione.

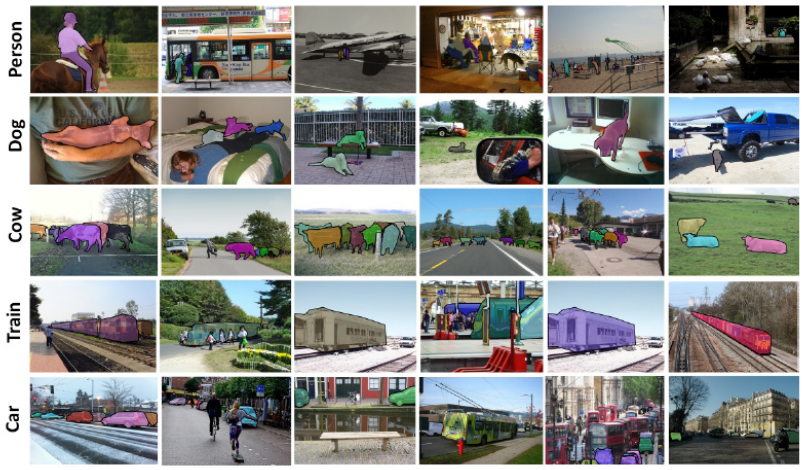


Figura 1.16: Immagini presenti nel dataset MS-COCO.

THE PASCAL VOC è un dataset di immagini fornito per il THE PASCAL Visual Object Classes Challenge, una competizione annuale nata nel 2006. La sfida consiste in due componenti: il dataset, compreso di un software per la valutazione dei risultati ottenuti e la competizione relativamente agli ambiti di Object Classification, Detection e Segmentation. Essendo creato per questa sfida, il dataset è diviso in due parti, quali i dati addestramento/validazione e quelli di test. Inoltre per convenienza, il primo gruppo è diviso ulteriormente nell'insieme di training e quello di validazione. Il dataset fornisce una completa annotazione per venti classi, cioè tutte le immagini sono annotate con un bounding box per ogni istanza delle venti classi (vedi figura 1.17). In aggiunta

alle bounding box per ogni oggetto sono specificati attributi del tipo: *'orientation'*, *'occluded'*, *'truncate'*, ecc. Le annotazioni sul l'insieme di test non sono rilasciate pubblicamente.

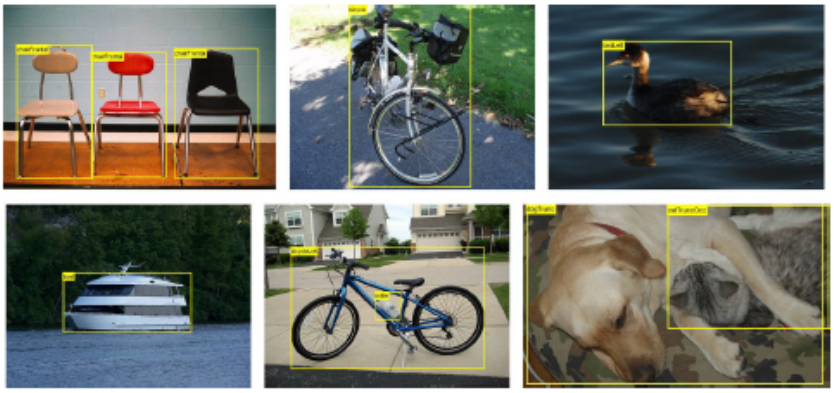


Figura 1.17: Immagini presenti nel dataset PASCAL VOC per le sfide di Object Classification e Detection.

Il problema principale di questi dataset è che non trovano utilità nelle applicazioni dove gli oggetti di interesse sono diversi da quelli da essi forniti. Per questo motivo, molti problemi reali di Object Detection necessitano di creare dataset ad hoc e ciò implica un dispendio elevato di tempo nella fase di etichettatura delle immagini.

1.5 Labeling Automatico

Dato il continuo aumento degli ambiti reali in cui vengono utilizzati approcci di Deep Learning, la creazione di nuovi dataset specifici risulta fondamentale. Per questa ragione sono stati proposti vari approcci per semplificare e ridurre i tempi di creazione di un dataset etichettato.
Un approccio per evitare l'etichettatura manuale, che sta ottenendo sempre più popolarità, consiste nell'utilizzo di dataset sintetici generati grazie al rendering di modelli CAD 3D di oggetti da diversi punti di vista. Una grande sfida nell'utilizzo di dati sintetici è l'intrinseca differenza tra gli esempi virtuali usati per il training e i dati reali della fase di valutazione. Per questo motivo, vi è

un notevole interesse nello studio dell'impatto della trama, dell'illuminazione e della forma per affrontare questa disparità. Un altro problema con le immagini sintetiche generate dai sistemi di rendering è che visualizzano oggetti in pose che non necessariamente sono fisicamente realistiche. Inoltre, le occlusioni vengono solitamente trattate in modo piuttosto ingenuo, cioè applicando il ritaglio o incollando patch rettangolari, che si traducono nuovamente in scene non realistiche.
Chaitanya Mitash et. al [16] hanno proposto un sistema in grado eseguire la simulazione fisica per posizionare oggetti in configurazioni realistiche e renderizzare immagini di scene per generare un dataset sintetico per l'addestramento di rilevatori di oggetti. Per configurare questo sistema vengono utilizzate informazioni dalla calibrazione della camera, dai modelli degli oggetti e dalla localizzazione di superfici. Il sistema in questione inizialmente genera e carica i modelli 3D degli oggetti in un ambiente calibrato nel simulatore. Una volta caricati tutti i modelli, viene scelto un sottoinsieme di essi per generare una scena. A questo punto, gli oggetti vengono posizionati sulla superficie e interpretati da diverse pose della camera predefinite. Infine, vengono calcolate le bounding box 2D per ogni oggetto grazie alla proiezione prospettica. Nella figura 1.18 è mostrata la pipeline del sistema descritto.

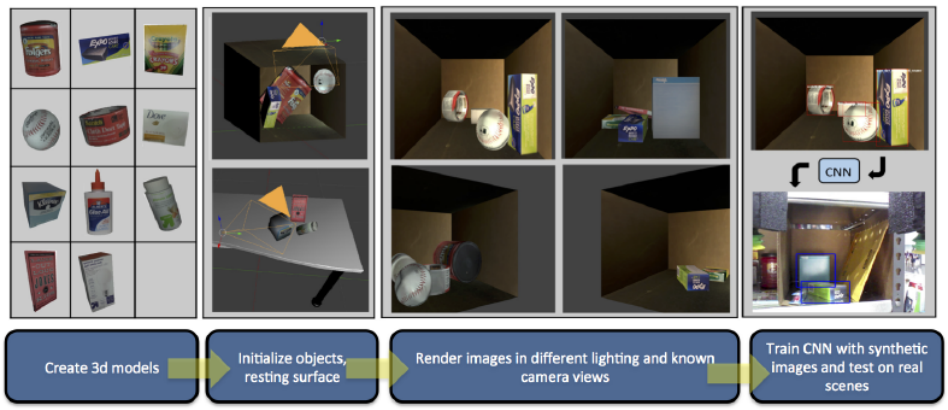


Figura 1.18: Pipeline del sistema di simulazione fisica.

Un altro approccio possibile è quello proposto da Duc Thanh Nguyen et. al [17]. Il tool in questione è in grado di generare un dataset etichettato partendo da scene indoor. Il tool è formato da quattro fasi (vedi figura 1.19):

ricostruzione della scena, segmentazione 3D automatica, raffinazione e annotazione interattiva e segmentazione 2D. Nella prima fase, il sistema prende una sequenza di frames RGB-D e ricostruisce una mesh, chiamata 3D scene mesh. Dopo la ricostruzione, vengono calcolati e memorizzati le corrispondenze tra i vertici 3D nella scena ricostruita e i pixel 2D su tutti i fotogrammi di input. Ciò consente il passaggio senza interruzioni tra la segmentazione in 3D e 2D in fasi successive. Nella seconda fase, la 3D scene mesh è segmentata automaticamente. Si parte raggruppando le mesh dei vertici in supervertici e successivamente in regioni. La terza parte del sistema è progettata per permettere agli utenti di interagire con esso, tramite metodi di raffinamento ed infine permette l'annotazione semantica degli oggetti. La quarta ed ultima parte del framework è progettata per la segmentazione di frames 2D. In questa fase, viene utilizzato un algoritmo che utilizza il risultato della segmentazione 3D come partenza per la segmentazione 2D e che si basi sulla corrispondenza dei contorni.

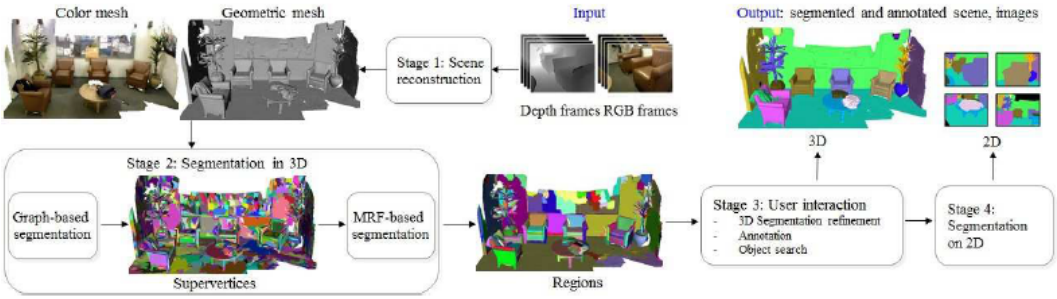


Figura 1.19: Pipeline del sistema di segmentazione di scene indoor.

Entrambi gli approcci sono molto validi ma hanno diverse limitazioni, come per esempio l'utilizzo di dati sintetici nel primo caso, mentre nel secondo l'utilizzo del tool è limitato ad ambienti indoor. Per questo motivo, il sistema oggetto della tesi utilizza un ambiente flessibile come quello dei dispositivi mobili e applica tecniche di realtà aumentata in modo tale da utilizzare immagini reali.

Capitolo 2

Progetto

Nel capitolo precedente è stata spiegata l'importanza della creazione di dataset nelle applicazioni reali di Object Detection. Inoltre, come detto in precedenza, tale procedimento richiede un grande dispendio di risorse e tempo.
Per questo motivo, il sistema proposto ha lo scopo di generare in modo automatico dataset di Object Detection, riducendo drasticamente i tempi di creazione. Infine, deve semplificare ed automatizzare l'addestramento di una rete neurale convoluzionale sui i dataset creati. Di conseguenza, è possibile suddividere il sistema in due macro operazioni: *creazione dataset* e *addestramento di CNN*. La prima macro operazione verrà eseguita tramite smartphone, mentre la seconda, necessitando di elevate capacità di calcolo, verrà svolta su desktop.
In questo capitolo verranno descritti il comportamento e la struttura del sistema proposto. Il sistema si suddivide in due moduli: *mobile* e *desktop* (vedi figura 2.1). Il primo ha come scopo la generazione effettiva del dataset tramite l'utilizzo di smartphone. Il secondo, invece, permette di convertire il dataset precedente in un formato diverso e di utilizzarlo per eseguire l'addestramento di una rete neurale convoluzionale. In questo capitolo, con il termine *AR dataset*, si farà riferimento alla collezione di dati creati dal modulo mobile, mentre il *Training dataset* corrisponde all'insieme di dati generati dalla conversione del modulo desktop.

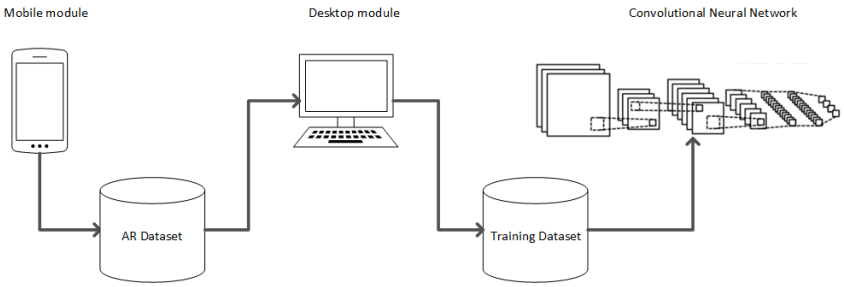


Figura 2.1: Workflow del sistema oggetto della tesi.

L'approccio utilizzato, per permettere la generazione automatica di immagini etichettate, si basa su tecniche di *realtà aumentata*.
La *realtà aumentata* (AR) è una variante della realtà virtuale [18]. Nei sistemi di realtà virtuale, l'utente è immerso in un ambiente fittizio, impedendogli così di vedere il mondo reale attorno. I sistemi di realtà aumentata, invece, consentono la coesione o la sovrapposizione degli oggetti virtuali con il mondo reale. Perciò, l'AR integra la realtà piuttosto che sostituirla. In figura 2.2 è possibile vedere un esempio di un'applicazione che utilizza tecniche di AR.

Figura 2.2: Esempio di un'applicazione di realtà aumentata.

L'idea generale dell'approccio utilizzato consiste nel creare oggetti virtuali, di cui si conoscono le dimensioni, e di posizionarli nella scena reale, in modo tale da inglobare gli oggetti fisici su cui si desidera generare il dataset. Una

volta posizionati tali oggetti nella scena ed aver salvato le loro pose, è necessario generare diversi fotogrammi della scena da vari punti di vista. Inoltre, ad ogni fotogramma viene associata la relativa posa della camera. Grazie alle informazioni precedenti, è possibile stabilire la posizione di un oggetto all'interno dei vari fotogrammi. Di conseguenza, per effettuare l'etichettatura degli oggetti in ogni singola foto, è sufficiente associare a priori un'etichetta ad ogni oggetto virtuale.
Dopo aver collezionato le informazioni sopracitate da diverse scene, è possibile creare il proprio AR dataset. Il modulo desktop del sistema, successivamente, permette di convertire il dataset in diversi formati e di utilizzarlo per l'addestramento della rete neurale convoluzionale scelta.
Nei capitoli seguenti verranno analizzati nel dettaglio i due moduli che compongono il sistema in questione.

2.1 Modulo Mobile

Il modulo Mobile permette la creazione vera e propria del dataset, il tutto tramite l'uso di uno smartphone. L'insieme di dati in questione, come già anticipato, si compone di una lista di immagini, una lista di pose della camera ed una lista di informazioni sugli oggetti virtuali, quali la posa, la scala e le dimensioni.
Utilizzando uno smartphone, grazie alla fotocamera integrata, risulta semplice ottenere foto da diversi punti di vista della scena in esame. I dati relativi alle pose, invece, risultano più complessi da recuperare. Il dispositivo, infatti, deve essere in grado di comprendere l'ambiente che lo circonda, la sua posizione rispetto ad esso e di monitorare i suoi spostamenti all'interno della scena. Gli algoritmi in grado di risolvere questa problematica sono definiti come SLAM (*Simultaneous Localization And Mapping*).
In figura 2.3 è mostrata l'applicazione oggetto di questo modulo in funzione.

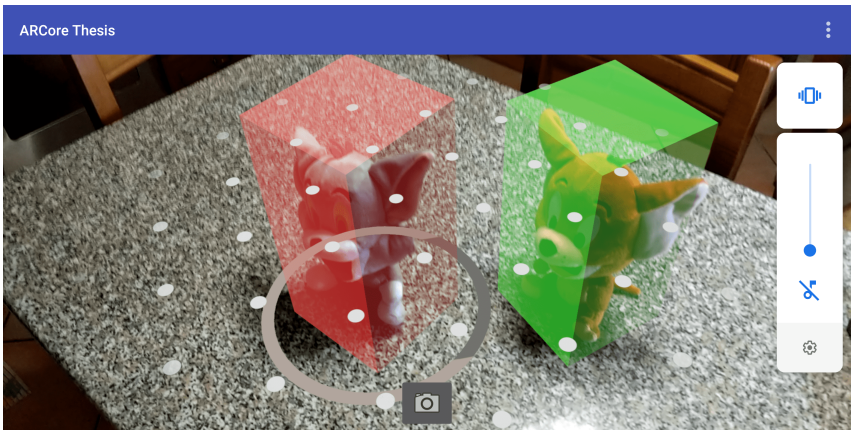


Figura 2.3: Screenshot dell'applicazione in funzione.

2.1.1 Simultaneous Localization And Mapping (SLAM)

La *Simultaneous Localization And Mapping* (SLAM) [19], conosciuta anche come *Concurrent Mapping and Localization* (CML), è il processo attraverso il quale un agente (robot, veicolo o persino un dispositivo mobile trasportato da una persona) ha la capacità di costruire una mappa globale dell'ambiente e, allo stesso tempo, di utilizzare tale mappa per dedurre la sua posa in qualsiasi momento. Per generare una mappa dall'ambiente, l'entità deve possedere sensori che le permettano di percepire e ottenere misurazioni degli elementi dal mondo circostante. Questi sensori sono classificati in esterocettivi e propriocettivi.
Tra i sensori esterocettivi è possibile trovare il sonar, laser scanner, camere e sistemi di posizionamento globale (GPS). Tuttavia questi sensori hanno capacità limitate e introducono molti errori.
I sensori propriocettivi permettono all'entità di ottenere misurazioni come la velocità, il cambiamento di posizione e l'accelerazione. Alcuni esempi di questo tipo di sensori sono: accelerometro, giroscopio e bussola. Questi sensori permettono di ottenere una stima incrementale dei movimenti dell'entità mediante un metodo di navigazione *dead-reckoning* [20]. A causa del rumore intrinseco di questa tipologia di sensori, essi non sono sufficienti per avere una stima accurata della posizione dell'entità per tutto il tempo.
Per mantenere una stima accurata e robusta della posizione dell'entità, poiché ogni sensore introduce del rumore, è necessario utilizzare la fusione di infor-

mazioni provenienti da più sensori di percezione (*Sensor Fusion*). Un esempio comune di sistemi che sfruttano la Sensor Fusion, come è possibile notare in figura 2.4, sono gli smartphone, i quali possiedono un gran numero di sensori differenti.

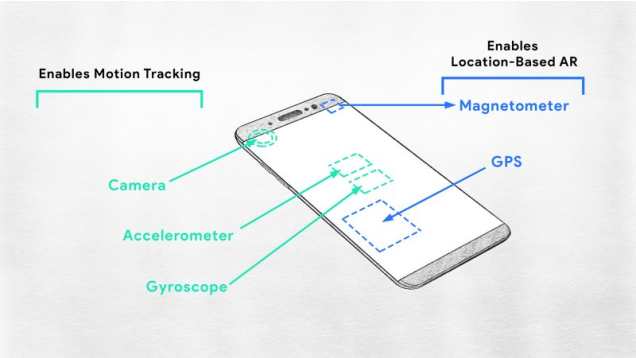


Figura 2.4: Insieme dei sensori disponibili su uno smartphone.

Il sensore maggiormente utilizzato negli approcci più moderni è la camera. Il principale motivo di questa tendenza è attribuita alle capacità delle tecniche di Computer Vision. I sistemi che fanno uso di queste tecniche, infatti, sono in grado di ottenere informazioni sulla distanza e di recuperare l'aspetto, il colore e la trama dell'ambiente.
I sistemi che fanno uso della camera per effettuare la SLAM, si basano sul concetto di *salient feature*, cioè punti, regioni o bordi della scena facilmente riconoscibili. In questo modo il sistema, tramite il matching di queste zone caratteristiche tra diversi fotogrammi, è in grado di generare in maniera incrementale la conoscenza dell'ambiente.
Per sfruttare appieno queste tecniche nella fase progettuale, è stata scelta la libreria *ARCore* fornita da Google. Questa libreria permette e semplifica lo sviluppo di applicazioni mobile di realtà aumentata.

2.1.2 Google ARCore

ARCore [21], come già anticipato, è una piattaforma utilizzata per la creazione di applicazioni di realtà aumentata. Utilizzando diverse API, ARCore consente al telefono di rilevare l'ambiente circostante, comprendere il mondo e

interagirvi tramite informazioni. Nella figura 2.5, è possibile vedere un esempio di app che utilizza questa libreria.

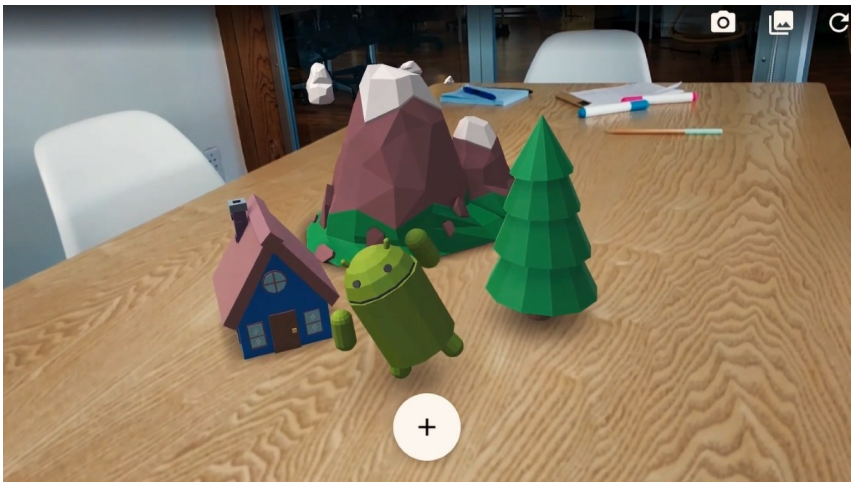


Figura 2.5: App mobile che utilizza Google ARCore.

Questa libreria utilizza tre funzionalità chiave per integrare il contenuto virtuale con il mondo reale visto attraverso la fotocamera del telefono:

- **Tracciamento del movimento (Motion tracking)**: mentre lo smartphone si muove attraverso l'ambiente, ARCore utilizza un processo per capire dove si trova il dispositivo relativamente al mondo circostante. ARCore rileva i feature point nell'immagine catturata dalla fotocamera e li utilizza per calcolare il suo cambiamento di posizione. Le informazioni visive sono combinate con misurazioni inerziali fornite dall'IMU (*Inertial Measurement Unit*) dello smartphone per stimare la posa relativa della fotocamera (posizione e orientamento) rispetto al mondo. Allineando la posa della camera virtuale che renderizza il contenuto 3D con la posa della fotocamera del dispositivo fornita da ARCore, gli sviluppatori sono in grado di visualizzare i contenuti virtuali dalla prospettiva corretta. L'immagine virtuale renderizzata può essere sovrapposta all'immagine ottenuta dalla fotocamera del dispositivo, facendo apparire così il contenuto virtuale come se fosse parte del mondo reale.

- **Comprensione ambientale (Environmental understanding)**: ARCore migliora costantemente la sua comprensione dell'ambiente reale ri-

levando feature point e piani. ARCore, infatti, cerca gruppi di feature point che sembrano giacere su superfici orizzontali o verticali comuni, come tavoli o muri, e rende queste superfici e le relative informazioni disponibili all'applicazione. Grazie alle informazioni fornite, è possibile posizionare oggetti virtuali appoggiati sulle superfici rilevate. Poiché ARCore utilizza i feature point per la rilevazione dei piani, le superfici senza trama, come un muro bianco, potrebbero non essere rilevate correttamente.

- **Stima della luce (Light estimation)**: ARCore è in grado di rilevare informazioni sull'illuminazione dell'ambiente circostante e di fornire l'intensità media e la correzione del colore di un determinato fotogramma. In questo modo è possibile illuminare gli oggetti virtuali all'interno di un'applicazione nelle stesse condizioni dell'ambiente reale.

Data la natura iterativa della comprensione dell'ambiente circostante, è possibile che le pose vengano aggiornate. Per questo motivo, quando si desidera posizionare un oggetto virtuale, è necessario definire una *anchor* per garantire che ARCore tenga traccia della posizione dell'oggetto nel tempo. Per evitare che ciò accada anche agli oggetti ambientali, quali feature point e piani, è stata definita una tipologia di oggetti a cui appartengono, chiamati *trackable*. Gli oggetti appartenenti a questa classe vengono tracciati nel tempo. Inoltre, è possibile legare una anchor ad un trackable, in modo tale da mantenere stabile la relazione tra oggetto virtuale e trackable anche quando il dispositivo si muove.
La piattaforma ARCore, infine, fornisce all'utente un sistema di interazione con l'ambiente virtuale basato sulla proiezione di raggi. Infatti, data una coordinata relativa allo schermo dello smartphone, viene proiettato un raggio virtuale e qualsiasi piano o feature point che esso interseca viene restituito.
In conclusione, questo strumento ci permette di demandare alla libreria di ARCore la generazione e gestione dell'ambiente virtuale, in modo tale da concentrare la nostra attenzione sulla progettazione della struttura e delle funzionalità del modulo mobile.

2.1.3 Struttura

In questo sottocapitolo verranno analizzate le scelte progettuali e la struttura del modulo mobile.
Il modulo software in questione consiste in un'applicazione Android in grado di gestire un ambiente di realtà virtuale, di fornire diverse interazioni tra l'utente e la scena e, infine, di creare un AR dataset partendo dalla scena.
Durante la fase di analisi dei requisiti sono stati rilevati i seguenti casi d'uso (vedi figura 2.6):

1. **Rilevamento di una superficie**. Questa fase è fondamentale poiché ARCore, come detto in precedenza, necessita di rilevare una superficie per permettere il tracking degli oggetti. Tramite il rilevamento della superficie, ARCore definisce il sistema di riferimento mondo. Una volta rilevata una superficie, è possibile posizionare gli oggetti all'interno dell'ambiente virtuale.

2. **Inserimento di un elemento virtuale nella scena**. Questo caso d'uso rappresenta una delle caratteristiche fondamentali dell'applicazione, ovvero la necessità di inserire modelli di oggetti virtuali 3D all'interno della scena in esame. Essi verranno posti sopra la superficie rilevata in precedenza.

3. **Eliminazione di un oggetto virtuale dalla scena**. Questa operazione è complementare all'inserimento e permette di modificare continuamente il numero di elementi virtuali presenti nella scena.

4. **Modifica della scala di un oggetto virtuale presente nella scena**. Poiché le dimensioni dei modelli 3D sono definite a priori, risulta utile permettere di ingrandire o rimpicciolire i singoli oggetti applicandogli un fattore di scala.

5. **Modifica della posizione di un oggetto virtuale all'interno della scena**. Un'altra caratteristica fondamentale per il nostro modulo mobile, è quella di manipolare la posizione dei vari elementi.

6. **Modifica della rotazione di un oggetto virtuale presente nella scena**. Poiché gli elementi verranno posti rispetto alla superficie rilevata, la rotazione degli oggetti sarà disponibile solo lungo l'asse delle y.

7. **Salvataggio fotogrammi ed informazioni relative alla scena**. Una volta inseriti i vari oggetti nella scena, risulta fondamentale, per la generazione del AR dataset, generare i fotogrammi da più punti di vista e i relativi dati associati. Per questo motivo, verrà fornita la possibilità di eseguire una videoregistrazione fittizia della scena. Durante questa registrazione, invece di generare un file video, verranno salvati in memoria i dati necessari alla creazione dell'AR dataset. Una volta iniziata la registrazione, il salvataggio avviene periodicamente fino alla sua interruzione.

8. **Definizione impostazioni di salvataggio**. Deve essere possibile definire il numero di fotogrammi salvati per secondo.

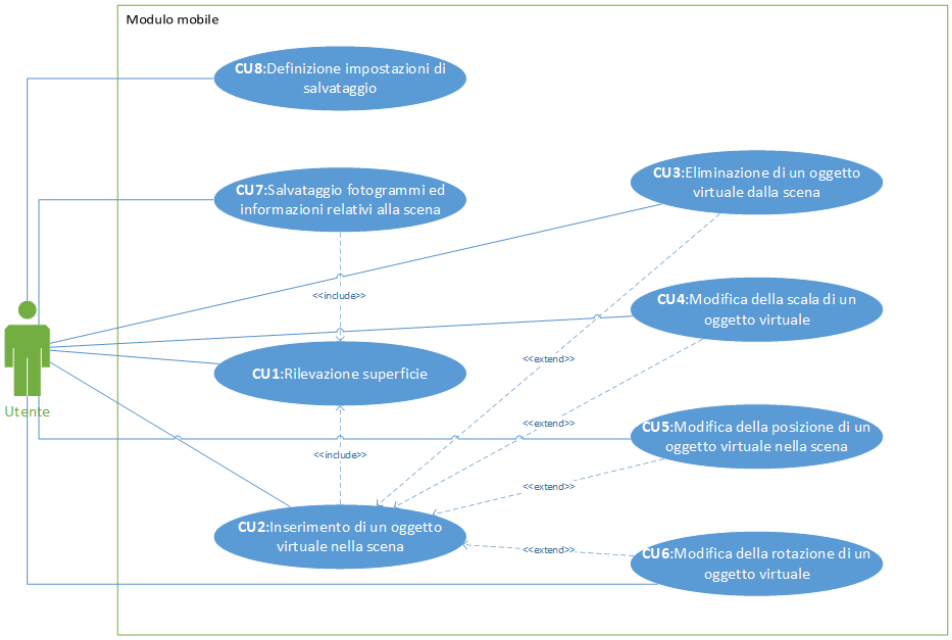


Figura 2.6: Diagramma UML dei casi d'uso del modulo mobile.

Dopo aver eseguito la fase di analisi dei requisiti è necessario progettare la struttura del codice. Gli obiettivi principali della fase progettuale è di

realizzare un'applicazione in grado di soddisfare i casi d'uso appena descritti e di ottenere una struttura modulare, riutilizzabile e facilmente modificabile. La struttura principale è stata definita seguendo il pattern architetturale *Model-View-Controller*(MVC, vedi figura 2.7). Grazie a questa scelta architetturale, è possibile effettuare modifiche ad una delle tre entità effettuando aggiustamenti minimi o nulli alle altre.

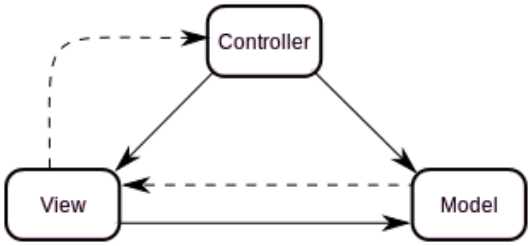


Figura 2.7: Schema esplicativo del pattern Model-View-Controller.

Nel progetto in esame, il *Model* consiste nella struttura dati contenente informazioni importanti sulla scena, come le pose degli oggetti virtuali, il modello 3D associato ad ogni elemento, ecc. Il *Controller* rappresenta la logica dell'applicazione: recupera gli input dell'utente tramite la View, li passa al Model che li elabora e restituisce il risultato alla View. Il Controller quindi fa uso delle API fornite da ARCore per la realizzazione dei casi d'uso descritti in precedenza. La *View*, invece, corrisponde all'interfaccia utente. Poiché il progetto consiste in una applicazione Android, la View è definita da una serie di file xml che definiscono i layout delle varie schermate.
Lo scopo del Model è quello di modellare le informazioni relative agli oggetti virtuali presenti nella scena. Le informazioni principali sono la posizione, la rotazione e la scala. Ogni oggetto virtuale, inoltre, possiede un id che ne identifichi l'istanza ed un id che definisce la classe di appartenenza dell'oggetto reale da esso inglobato. L'id della classe è fondamentale per la creazione del dataset e viene associato al modello 3D visualizzato nella scena. Infine, il Model contiene anche informazioni relativi alla camera in uso, ovvero i valori intrinseci della camera. Un'altra informazione necessaria per la creazione del dataset, è la lista delle pose della camera ottenute durante l'ultima operazione di videoregistrazione.
Il Model è stato strutturato in modo tale che sia il più riusabile possibile (vedi

figura 2.8). Di seguito è fornita una breve descrizione delle interfacce utilizzate e il relativo diagramma UML delle classi.

- **Node**: rappresenta un elemento all'interno della scena e fornisce la sua posa (posizione + rotazione) e la scala.
- **Pose**: definisce la posa di un oggetto, cioè posizione e rotazione.
- **Box**: interfaccia utilizzata per modellare le informazioni relative ai modelli 3D caricati nella scena. Tali informazioni sono il tipo di modello, le dimensioni e l'id della classe.
- **CameraIntrinsics**: utilizzata per strutturare i valori intrinseci della camera in uso.
- **Model**: permette la creazione e l'eliminazione di Node e Box. Inoltre, associa ad ogni Node una ed una sola Box. La stessa Box, invece, può essere associato a più Node, perché in una scena possono essere presenti più oggetti reali della stessa classe di appartenenza. Inoltre, come già anticipato, contiene gli intrinseci della camera e la lista delle pose della camera ottenute dall'ultima videoregistrazione.

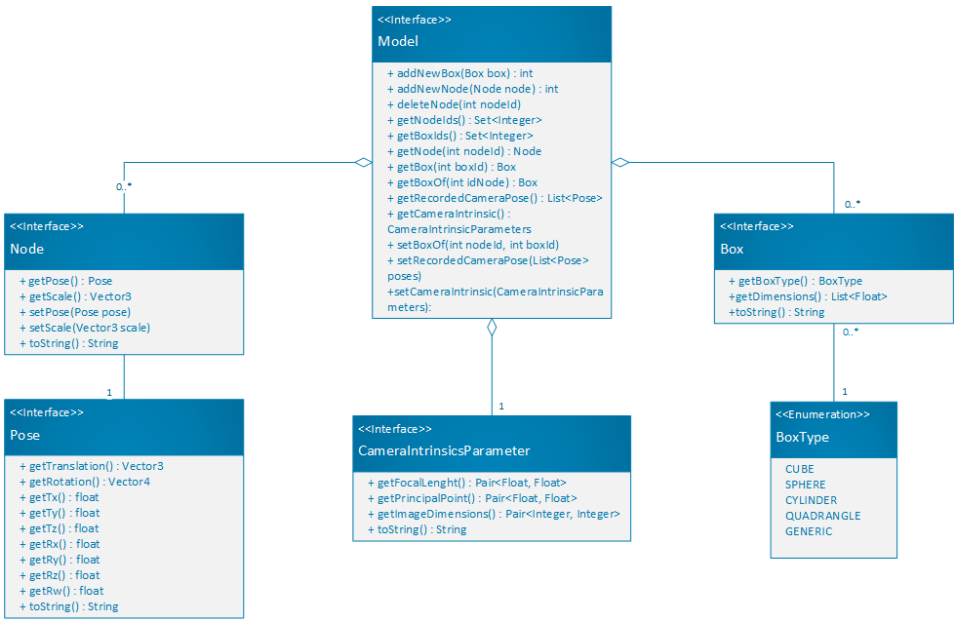


Figura 2.8: Diagramma UML delle classi del Model.

Avendo definito la struttura del Model tramite interfacce, è possibile definire diverse implementazione mantenendo il sistema funzionante.
Una parte molto importante dell'applicazione è il salvataggio della scena, infatti tale procedura permette la generazioni delle scene che costituiscono un AR dataset. Le informazioni necessarie sono i fotogrammi e i dati presenti nel Model appena descritto. Per questo motivo, una volta lanciata la registrazione, vengono salvati periodicamente la posa della camera in una lista e il fotogramma nella memoria esterna dello smartphone. Una volta terminata questa fase, la lista di pose viene passata al Model e anch'esso viene salvato nella memoria esterna. La struttura della cartella associata ad ogni scena è la seguente:

- **lista di immagini .jpg**: le immagini corrispondono ai fotogrammi salvati durante la fase di registrazione e viene mantenuto l'ordine di acquisizione tramite il nome del file.

- **camera_pose.txt**: questo file contiene in ogni riga una posa della camera. La posa nell'i-esima riga è associata all'i-esima immagine. Le pose sono nel formato:

  $$x\ y\ z\ rx\ ry\ rz\ rw$$

  Dove i primi 3 numeri rappresentano la posizione e gli ultimi 4 la rotazione nel formato quaternione.

- **intrinsics.txt**: questo file contiene i dati intrinseci della camera. Il formato è il seguente:

  $$fl_x\ fl_y\ image\_width\ image\_height\ pp_x\ pp_y$$

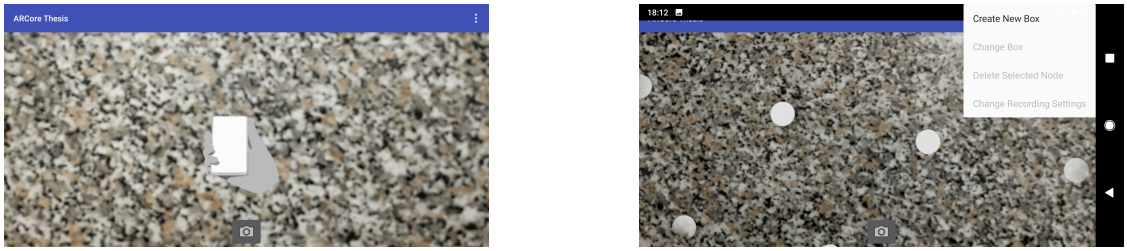
  Dove fl indica la lunghezza focale, mentre pp consiste nel punto principale.

- **nodes.txt**: questo file contiene una riga per ogni oggetto virtuale presente nella scena. In ogni riga sono definite le informazioni relative a tali oggetti, quali posizione, rotazione, id classe, scala, dimensioni. Il formato è il seguente:

  $$x\ y\ z\ rx\ ry\ rz\ rw\ id_{class}\ scale_x\ scale_y\ scale_z\ boxtype\ dim_1\ dim_2 *\ dim_3 *$$

Le ultime due dimensioni sono opzionali, poiché nel caso di un cubo è sufficiente una sola dimensione.

L'applicazione realizzata risulta molto intuitiva nel suo utilizzo. Infatti, una volta aperta l'applicazione, verrà mostrata un'icona che indica all'utente di inquadrare la superficie da utilizzare (figura 2.9a). Una volta rilevata la superficie il software permette, tramite il menu laterale, di inserire modelli 3D sul piano (figura 2.9b).



(a) (b)

Figura 2.9: (a) Screenshot dell'applicazione appena avviata. (b) Screenshot dell'applicazione in cui è mostrato il menù laterale.

Inoltre, il sistema fornisce la possibilità di interagire con i modelli inseriti tramite le gesture standard:

- **tap**: permette di selezionare l'oggetto virtuale che interseca il raggio proiettato dal punto selezionato.
- **swipe**: permette di muovere l'oggetto selezionato nella superficie di riferimento.
- **pinch/zoom**: permettono di diminuire o aumentare le dimensioni dell'oggetto selezionato applicandogli un fattore di scala.
- **rotate**: permette la rotazione dell'elemento virtuale rispetto all'asse delle y.

Il menu laterale, inoltre, permette di eliminare l'oggetto selezionato, di sostituire il modello 3D dell'oggetto selezionato con un altro e di definire la frequenza di cattura dei fotogrammi durante la fase di registrazione (2.9b).

Per eseguire la registrazione è presente un pulsante nella parte bassa dello schermo. Una volta che il salvataggio dei fotogrammi è iniziato, è sufficiente ripremere il tasto per terminare la registrazione.
Infine, per recuperare le scene generate, è sufficiente collegare il cellulare ad un PC e copiare il contenuto della cartella ARCoreThesis.

2.2 Modulo Desktop

Il modulo Desktop permette di effettuare, in maniera automatica, la conversione del AR dataset in diversi formati e l'addestramento automatico di una rete neurale convoluzionale. La rete neurale in questione verrà addestrata ed utilizzata per la rilevazione degli oggetti presenti nel dataset.
Poiché in questo modulo è necessario l'addestramento di reti neurali convoluzionali, si è scelto di utilizzare *Tensorflow* come libreria di supporto.

2.2.1 Tensorflow

Tensorflow [22] è una libreria software open source per elaborazioni numeriche ad alte prestazioni. La sua architettura flessibile consente l'implementazione di calcoli complessi su diverse piattaforme (CPU, GPU, TPU) ed architetture, quali desktop, cluster di server e dispositivi mobili. Questa libreria fu originariamente sviluppata da ricercatori ed ingegneri del team Google Brain all'interno dell'organizzazione di intelligenza artificiale di Google. Per questo motivo, Tensorflow fornisce un forte supporto al machine learning e al deep learning. Inoltre, il core di elaborazione numerica, grazie alla sua flessibilità, viene utilizzato in molti altri ambiti scientifici.
Per lo sviluppo di questo modulo si è fatto uso della API di Tensorflow per l'object detection. Questa API è un framework open source costruito su Tensorflow, il quale semplifica la creazione, l'addestramento e l'utilizzo di modelli per l'object detection.
Per eseguire l'addestramento di una rete neurale convoluzionale è necessario il file di configurazione e il dataset. Il file di configurazione contiene i parametri di modello, di allenamento e di valutazione. Il dataset, invece, consiste in due o più file, suddivisi in training set ed evaluation set, nei quali vi è una lista di

immagini ad ognuna delle quali vi sono associate diverse informazioni. I dati associati ad ogni immagine sono: *altezza*, *larghezza*, *id e nome dell'immagine* e tre liste contenenti informazioni relative agli oggetti presenti, quali *bounding box 2D*, *id e nome classe* dell'oggetto. Le bounding box sono definite secondo il formato:

$$xmin, xman, ymin, ymax$$

Come spiegato in precedenza, il modulo mobile genera un dataset contenente informazioni sugli oggetti virtuali 3D presenti nelle varie scene, mentre il modulo Desktop necessita di bounding box 2D. Per questo motivo risulta necessario effettuare una trasformazioni partendo dalle pose originali in 3D per ottenere i dati 2D richiesti.

2.2.2 Conversione da 3D a 2D

In questo capitolo verrà fornito il metodo e le basi teoriche per ottenere delle bounding box 2D partendo da pose 3D.
I dati in nostro possesso consistono in una lista di immagini, ognuna delle quali ha associato una posa della camera rispetto al sistema di coordinate mondo, e la lista di informazioni degli oggetti virtuali nella scena. Le informazioni per ogni oggetto sono le seguenti:

- posa 3D dell'oggetto rispetto al sistema di coordinate mondo, cioè posizione (x,y,z) e rotazione (nel formato quaternione, qx qy qz qw).
- id relativo alla classe di appartenenza.
- tipo di modello utilizzato e relative dimensioni.
- fattore di scala applicato alle dimensioni del modello.

Di conseguenza, la situazione che questi dati ricostruiscono è similare a quella rappresentata nella figura 2.10.

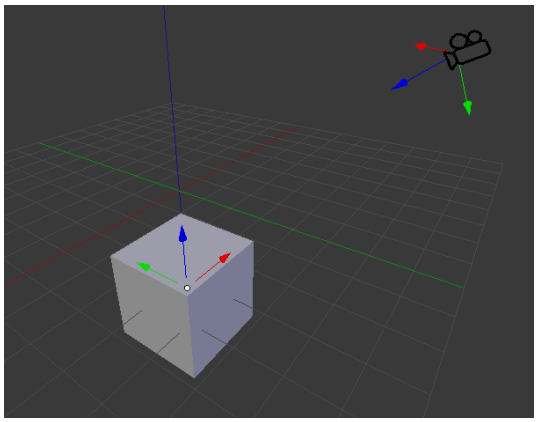


Figura 2.10: Rappresentazione di una scena 3D con un cubo, una camera e i relativi sistemi di riferimento.

Il sistema di riferimento mondo è il sistema di coordinate 3D definito dalla libreria ARCore nella fase di riconoscimento dell'ambiente circostante. Di conseguenza, tutte le pose degli oggetti presenti all'interno della scena (camera e oggetti virtuali) sono da intendersi rispetto a tale sistema.
Ogni elemento 3D, come possiamo vedere nell'immagine, possiede un proprio sistema di riferimento, quindi anche la camera e gli oggetti virtuali. Infatti, tramite la posa 3D di un oggetto, è possibile ottenere la posizione di un punto rispetto al sistema mondo partendo dalle coordinate di un punto nel sistema di dell'oggetto.
Per esempio, nel caso dell'immagine precedente, sappiamo che la posizione $[x, y, z]$ del cubo rispetto al mondo fa riferimento al punto medio della base del cubo. Di conseguenza, ponendo tale punto all'origine del sistema di riferimento del cubo ($[0, 0, 0]$), siamo in grado di ottenere le coordinate di uno degli spigoli ($[-\frac{lato}{2}, 0, \frac{lato}{2}]$). Data la posa P, sia R una matrice 3×3 ottenuta dalla rotazione e T una matrice 3×1 data dalla traslazione, e le coordinate dello spigolo $pt = [x_1, y_1, z_1]^T$ nel sistema di riferimento del cubo. Le coordinate dello spigolo nel sistema mondo, $pt' = [x'_1, y'_1, z'_1]^T$, sono date da:

$$pt' = R \cdot pt + T$$

$$\text{con } R = \begin{bmatrix} r_{11} & r_{12} & r_{13} \\ r_{21} & r_{22} & r_{23} \\ r_{31} & r_{32} & r_{33} \end{bmatrix}, T = \begin{bmatrix} t_x \\ t_y \\ t_z \end{bmatrix}$$

Inoltre, è possibile eseguire il medesimo calcolo tramite il prodotto della matrice di trasformazione ${}^{world}T_{cube} = \begin{bmatrix} R & T \\ 0 & 1 \end{bmatrix}$ e le relative coordinate omogenee del punto.
Nel caso in cui si voglia effettuare il passaggio opposto, ovvero trovare le coordinate di un punto rispetto al cubo dato la sua posizione rispetto al mondo, è sufficiente effettuare la moltiplicazione delle coordinate omogenee del punto con la matrice inversa $({}^{world}T_{cube})^{-1}$, che può essere indicata come ${}^{cube}T_{world}$.
Per calcolare la bounding box dell'oggetto 3D, è necessario ottenere la posizione 3D dell'oggetto rispetto al sistema di riferimento camera ed infine effettuare la conversione da 3D a 2D tramite proiezione prospettica. Grazie alla spiegazione sul cambio di sistemi di riferimento, siamo in grado di ottenere facilmente un punto dell'oggetto nel sistema di riferimento camera. Siano $[x, y, z, 1]^T$ le coordinate omogenee dello spigolo del cubo rispetto al sistema cubo, le relative coordinate 3D $[\tilde{x}, \tilde{y}, \tilde{z}]$ nel sistema di riferimento camera sono date dal prodotto matriciale seguente:

$$\tilde{p} = A \cdot {}^{camera}T_{world} \cdot {}^{world}T_{cube} \cdot [x, y, z, 1]^T$$

dove ${}^{camera}T_{world}$ corrisponde alla matrice di trasformazione inversa ottenuta dalla posa della camera in un dato fotogramma, mentre A è la matrice dei parametri intrinseci della camera ed è definita come:

$$A = \begin{bmatrix} f_x & 0 & u_0 & 0 \\ 0 & f_y & v_0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix}$$

dove f_x e f_y rappresentano la lunghezza focale della camera rispetto i due assi, mentre u_0 e v_0 definiscono le coordinate del punto principale o di perforazione. La lunghezza focale coincide con lo spazio misurato tra il centro ottico di una lente sottile ed il punto di focalizzazione dei raggi paralleli entranti, emessi da un punto luce posto all'infinito. Il punto principale, invece, corrisponde al punto di intersezione tra l'asse ottico e il piano dell'immagine. Nella figura 2.11, in cui è mostrata la struttura di una camera generica, la lunghezza focale è indicata come f e il punto principale come c.

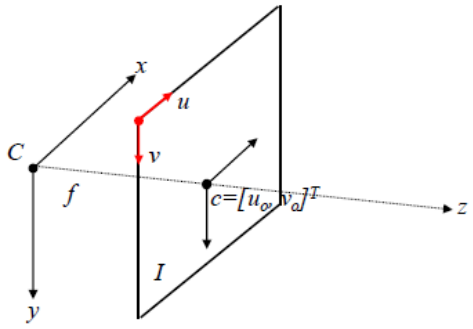


Figura 2.11: Rappresentazione della struttura di una camera generica.

A questo punto, per ottenere il corrispettivo punto 2D $[u, v]$, è sufficiente effettuare le seguenti divisioni:

$$u = \frac{\tilde{x}}{\tilde{z}} \, v = \frac{\tilde{y}}{\tilde{z}}$$

Per calcolare la bounding box 2D del cubo di esempio, è quindi necessario effettuare il procedimento appena descritto per ogni spigolo. Infine, una volta ottenuti tutti i punti 2D, vengono scelti $u_{min}, u_{max}, v_{min}, v_{max}$. Nella figura 2.12 è mostrata un'immagine, nella quale sono evidenziate le bounding box generate tramite il sistema proposto.

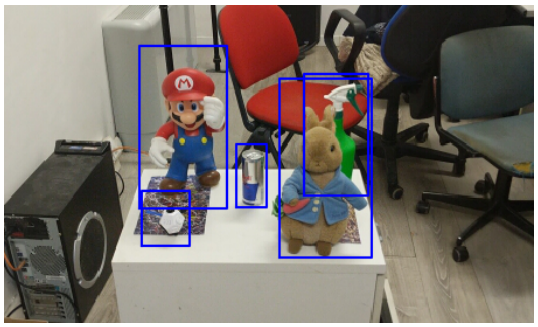


Figura 2.12: Bounding box 2D generate dal sistema proposto.

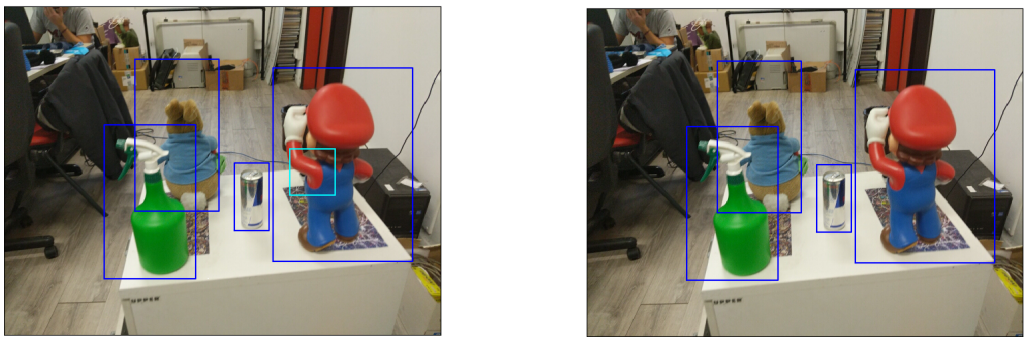
Nel caso vi siano più oggetti, è possibile che guardando la scena da una particolare posizione un oggetto risulti coperto da un altro. Per questo motivo, è necessario che la bounding box dell'oggetto coperto non venga rilevata dal nostro sistema. Per evitare l'utilizzo di bounding box 2D non desiderate, per ogni fotogramma della scena viene effettuato un controllo tra tutte le coppie di bounding box possibili (vedi algoritmo 2.1).

Input : Lista delle bounding box 2D dei singoli oggetti presenti nella scena.
Output: Lista delle bounding box 2D valide.

foreach *bb*1 *in Bounding_box_list* **do**
  include_bb ← *vero*
  foreach *bb*2 *in Bounding_box_list* **do**
    if *bb*1 ≠ *bb*2 *AND Area di intersezione tra bb*1 *e bb*2 > *s AND* *distanza[bb*1*]* > *distanza[bb*2*]* **then**
      include_bb ← *falso*
    end
  end
  if *include*$_b$*b* = *vero* **then**
    aggiungere *bb*1 alle bounding box valide
  end
end

Algorithm 2.1: Algoritmo per il filtraggio delle bounding box.

Nella figura 2.13 è possibile vedere un caso pratico in cui un oggetto all'interno della scena è coperto da un altro. Nella figura 2.13a sono definite tutte le bounding box generate grazie alle pose degli oggetti nella scena. Nella figura 2.13b, invece, sono evidenziate solo le bounding box valide, ovvero solo quelle degli oggetti visibili in questo fotogramma.



(a) (b)

Figura 2.13: (a) Generazione di tutte le bounding box possibili. (b) Generazione delle bounding box valide.

2.2.3 Struttura

In questo sottocapitolo verranno analizzate le scelte progettuali e la struttura del modulo desktop.
Il modulo software in questione ha come scopo l'automatizzazione del processo di addestramento di una rete neurale a partire da un dataset, di conseguenza l'utente ha un impatto minimo su questa parte del sistema. Infatti, dopo una rapida fase di analisi dei requisiti, i casi d'uso rilevati sono i seguenti (vedi figura 2.14):

1. **Conversione del AR dataset nel formato tfrecord dataset**. Dato il percorso della cartella in cui si trova un dataset fornito dal modulo mobile, vengono generati i relativi file tfrecord. Tali file sono fondamentali per esecuzione dell'addestramento tramite Tensorflow.

2. **Addestramento automatico della rete neurale convoluzionale dato un tfrecord dataset**. Dato il percorso della cartella contenente i file tfrecord, eseguire un addestramento.

3. **Addestramento automatico della rete neurale convoluzionale dato un AR dataset**. Questa situazione corrisponde all'esecuzione in sequenza dei due casi d'uso precedenti.

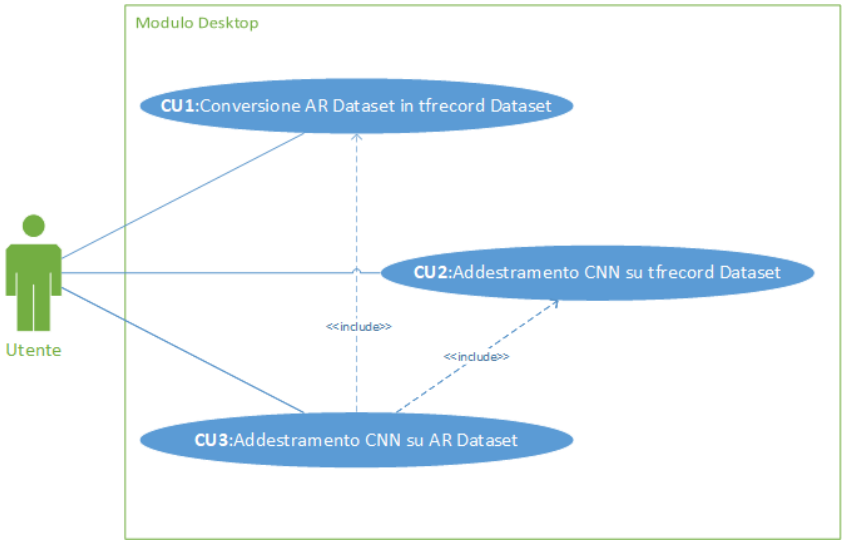


Figura 2.14: Diagramma UML dei casi d'uso del modulo desktop.

Considerando la natura del modulo e le tre diverse modalità di esecuzione, si è deciso di rendere l'applicazione uno script python. Lo script può essere eseguito utilizzando due sottocomandi differenti, di seguito è mostrata la sintassi:

- **python script.py a -d "AR dataset dir path" -e "evaluation AR scene dir path"** [**-n NETWORK_TYPE -s num_train_steps -o "Tensorflow object detection dir path"**] . Tramite il sotto comando *a*, nel caso non si specifichino i parametri *-n* e *-o*, si può effettuare la conversione del AR dataset in tfrecord, oppure in caso contrario si esegue la conversione e successivamente l'addestramento. Il significato dei parametri in ingresso è il seguente:
  - **d**: corrisponde al percorso della directory in cui si trova l'AR dataset.
  - **e**: definisce il percorso della directory contenente la scena che verrà utilizzata per la fase di valutazione dell'addestramento.
  - **n**: corrisponde al tipo di rete neurale convoluzionale da utilizzare per l'addestramento, le tipologie disponibili sono definite a priori.
  - **s**: definisce il numero di *train step* che deve eseguire la rete nella fase di addestramento.
  - **o**: definisce il percorso della directory object_detection appartenente al repository di Tensorflow. Questo parametro è fondamentale per l'addestramento della rete.
- **python script.py t -d "tfrecord dataset dir path" -n NETWORK_TYPE -s num_train_steps -o "Tensorflow object detection dir path"**. Grazie al sottocomando *t* è possibile eseguire l'addestramento utilizzando un dataset in formato tfrecord già creato in precedenza. Il parametro *-d* in questo caso rappresenta il percorso della directory contenente i file tfrecord per l'addestramento.

Dopo aver analizzato l'interazione con l'utente, è necessario definire la struttura del codice, ovvero effettuare un'analisi progettuale.
Il modulo desktop può essere rappresentato da alcune classi di oggetti che comunicano tra di loro (vedi figura 2.15), quelle principali sono:

- **ar_dataset**: questa entità rappresenta l'AR dataset generato dal modulo mobile. Dato il percorso della directory nella quale si trova il dataset e di quella della scena per la fase di evaluation, è possibile creare un'istanza di questa classe. Essa avrà il compito di recuperare le informazioni presenti nella collezione di dati, organizzarli in una struttura dati e renderli disponibili tramite i propri metodi. Essendo un AR dataset formato da più AR scene, è possibile modellare i due concetti con due entità differenti. Di conseguenza, ogni AR scene avrà il compito di convertire le pose 3D degli oggetti in bounding box 2D.

- **tf_dataset**: questa classe modella il concetto di tfrecord dataset. Date le informazioni fornite dall'ar_dataset, è possibile definire un oggetto appartenente a questa classe. Tale istanza ha lo scopo di eseguire la conversione del dataset e di creare in memoria i relativi file tfrecord. Inoltre, questa classe deve poter fornire il percorso della directory nella quale tali file sono stati generati, in modo tale che possano poi essere utilizzati per l'addestramento.

- **training**: lo scopo di questa classe è di definire i parametri relativi al dataset scelto, di salvarli nel file di configurazione utilizzato dalla rete neurale ed infine di eseguire l'addestramento.

- **converter**: questa entità ha il solo scopo di utilizzare i file generati dall'addestramento (checkpoint) e il file di configurazione per generare un modello della rete utilizzabile. I formati definiti sono *"*.pb"* e *"*.tflite"*, il quale può essere utilizzato per eseguire la rete addestrata sui dispositivi mobili.

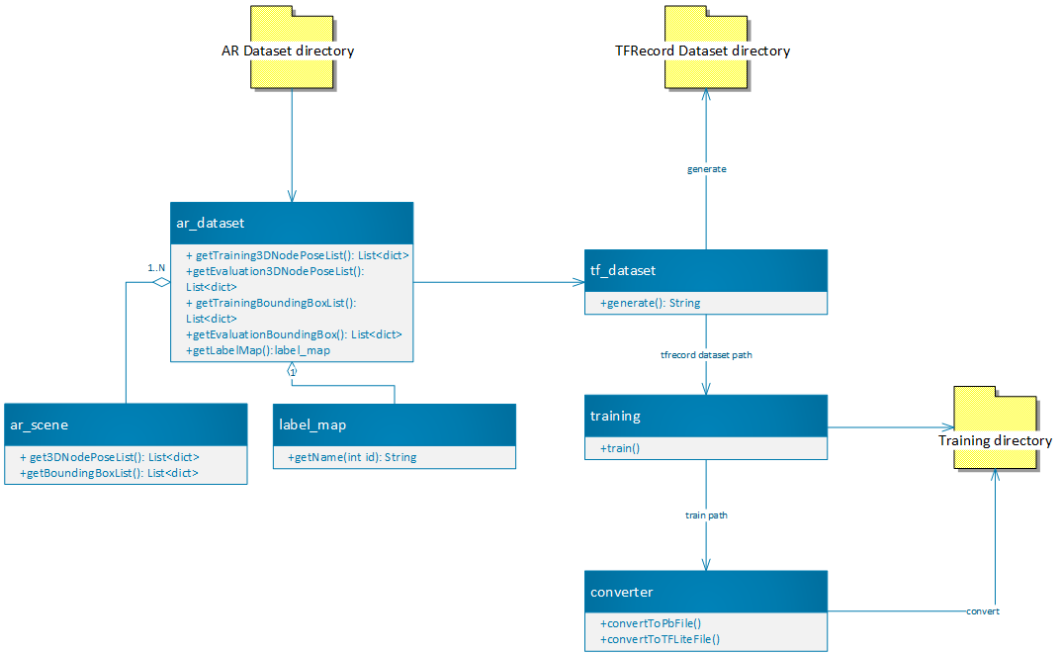


Figura 2.15: Diagramma UML delle classi del modulo desktop.

La label map è un oggetto che modella un file, al cui interno è salvato l'id e il relativo nome di ogni classe di oggetti presente nel dataset. Nella cartella dello script verranno generate due cartelle, una chiamata *"datasets"* e una *"trainings"*. All'interno della prima verranno create delle cartelle (*TFRecord Dataset directory*) contenenti i dataset in formato tfrecord, mentre nella seconda vi saranno delle cartelle (*Training Directory*) al cui interno vi sono i dati generati dall'addestramento della rete su un particolare dataset (come config file, checkpoing, ecc.).
Dopo aver analizzato entrambi i moduli del sistema proposto in questa tesi, è possibile visionare, nel prossimo capitolo, i risultati sperimentali ottenuti.

Capitolo 3

Risultati Sperimentali

In questo capitolo verranno mostrati le analisi qualitative effettuate sul sistema proposto e i risultati ottenuti. Per svolgere tali analisi, grazie all'applicazione del modulo mobile, sono stati generati due dataset: *COB* (Cvlab Office Boxes) e *CORO* (Cvlab Office Random Objects).
Il dataset *COB* contiene 5 classi di oggetti ognuna delle quali corrisponde ad una particolare scatola di vari prodotti. Nella figura 3.1 è possibile vedere i diversi tipi di scatola utilizzate per la generazione di questo dataset e le etichette associate. La scelta di tali oggetti è data dalla loro forma regolare, infatti i modelli virtuali 3D utilizzati dal modulo mobile del sistema durante la creazione del dataset, inglobano gli oggetti reali in maniera perfetta.

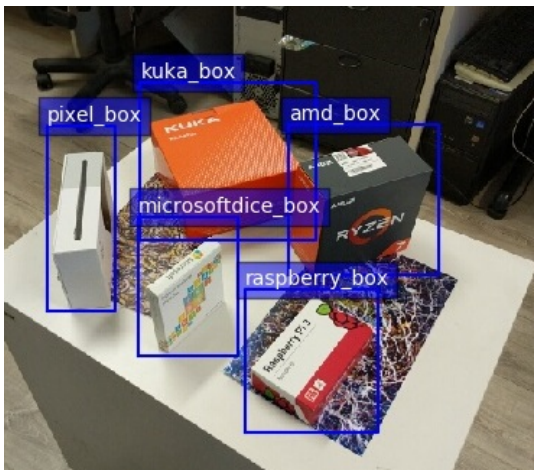


Figura 3.1: Insieme degli oggetti utilizzati per realizzare il dataset COB.

Il dataset *CORO* contiene 5 classi di oggetti e, a differenza del precedente, è stato generato utilizzando oggetti aventi dimensioni e forme molto diverse. In questo modo, è possibile valutare le performance del sistema in esame su oggetti con forme irregolari. Nella figura 3.2 sono mostrati i diversi oggetti utilizzati nella creazione di questo dataset e le etichette associate.

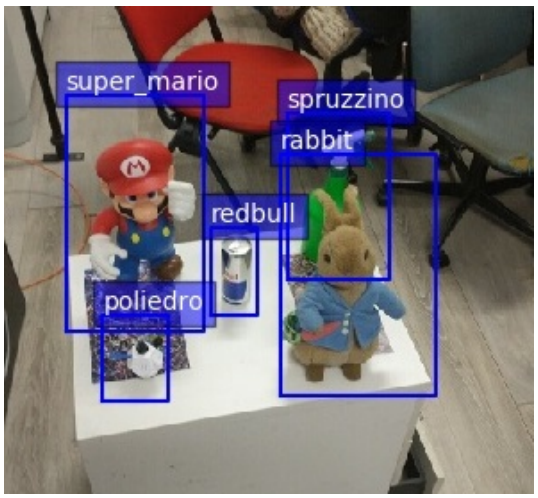


Figura 3.2: Insieme degli oggetti utilizzati per realizzare il dataset CORO.

Infine, grazie al modulo desktop, sono stati utilizzati tali dataset per addestrare due reti neurali convoluzionali di tipo *SSD Inception-v2* [23]. Una volta terminato l'addestramento, le reti neurali sono state valutate nel rilevamento degli oggetti appartenenti ai dataset utilizzati.
Nei seguenti sottocapitoli verranno mostrati i parametri scelti per l'addestramento delle *CNN* e i risultati ottenuti.

3.1 COB Dataset

La CNN utilizzata è del tipo SSD Inception-v2 e tale modello è pre-addestrato sul dataset MS-COCO. I parametri utilizzati sono i seguenti:

- **Numero epoche**: 100000.
- **Learning rate iniziale**: 0,004.
- **Decay steps**: 20000.

- **Decay factor**: 0,75.

Nelle figure 3.3 - 3.5 sono mostrati i valori *mean Average Precision* della rete addestrata, mentre nelle figure 3.6 e 3.7 i valori di *Average Recall*.

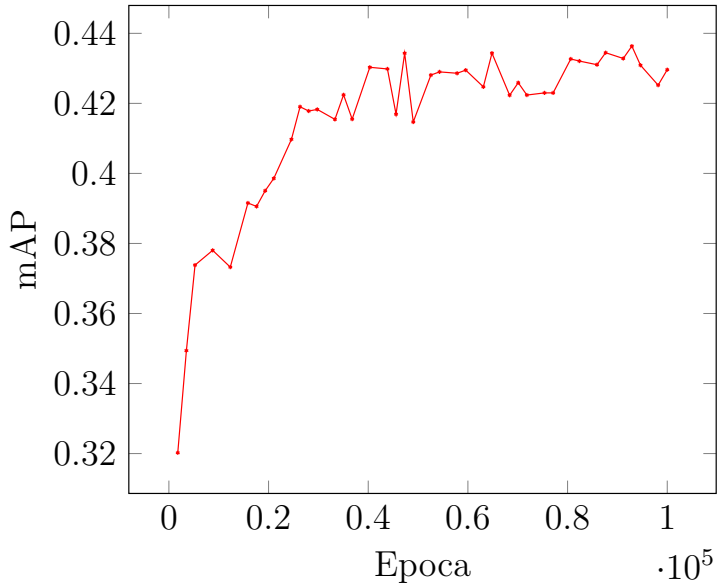


Figura 3.3: Dataset COB mean Average Precision.

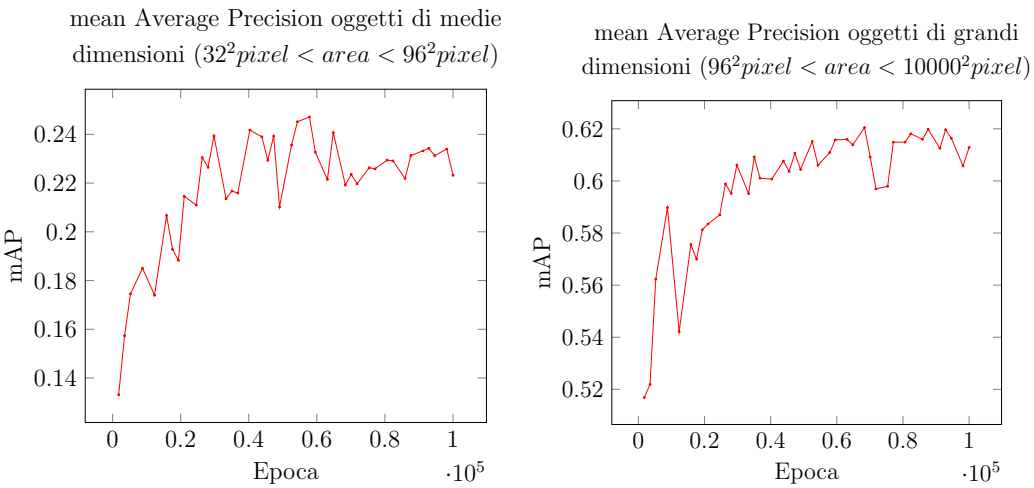


Figura 3.4: Dataset COB mean Average Precision per classi di dimensioni degli oggetti.

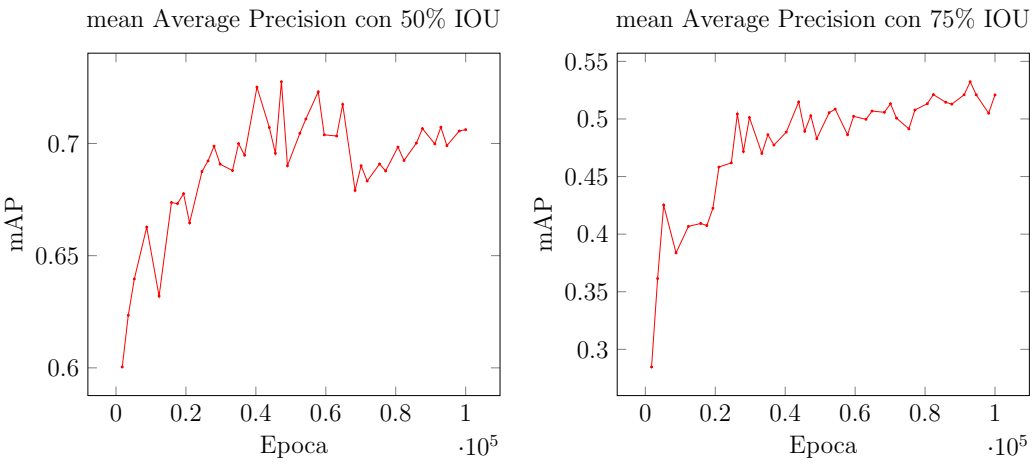


Figura 3.5: Dataset COB mean Average Precision Intersection Over Union.

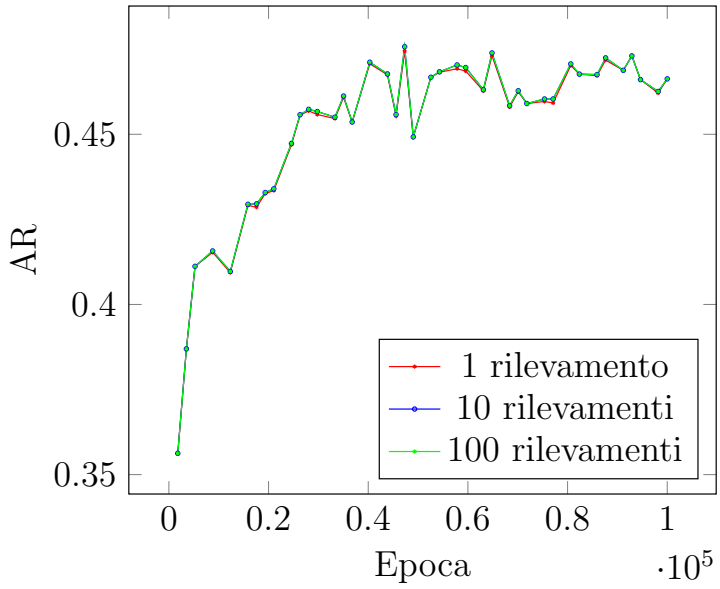


Figura 3.6: Dataset COB Average Recall.

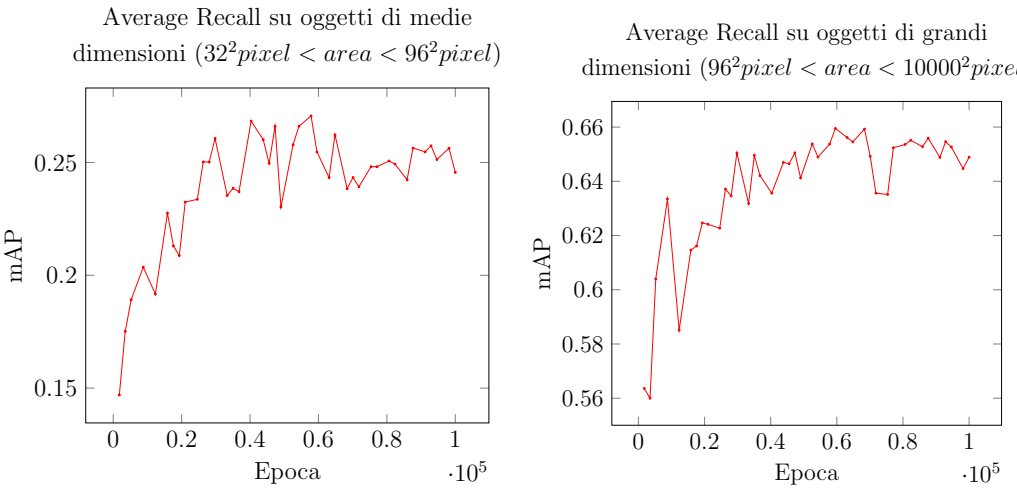


Figura 3.7: Dataset COB Average Recall per classi di dimensioni degli oggetti.

3.2 CORO Dataset

La CNN utilizzata è del tipo SSD Inception-v2 e tale modello è pre-addestrato sul dataset MS-COCO. I parametri utilizzati sono i seguenti:

- **Numero epoche**: 100000.
- **Learning rate iniziale**: 0,004.
- **Decay steps**: 20000.
- **Decay factor**: 0,75.

Nelle figure 3.8 - 3.10 sono mostrati i valori *mean Average Precision* della rete addestrata, mentre nelle figure 3.11 e 3.12 i valori di *Average Recall*.

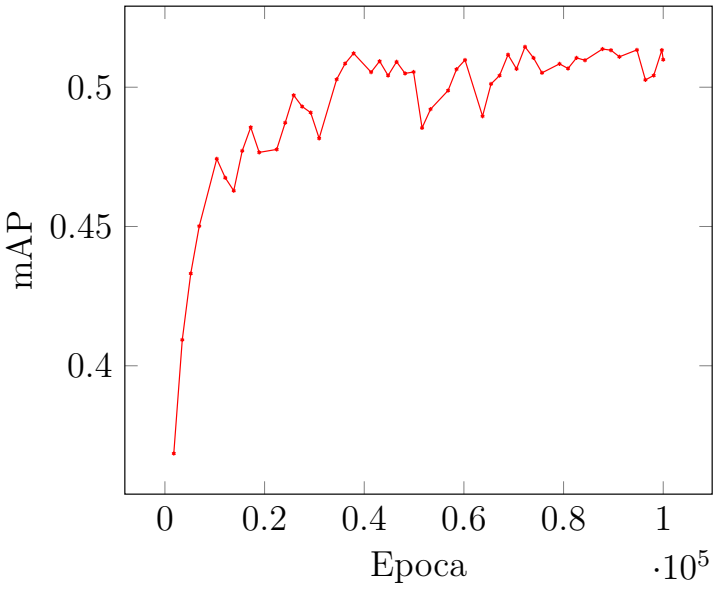


Figura 3.8: Dataset CORO mean Average Precision.

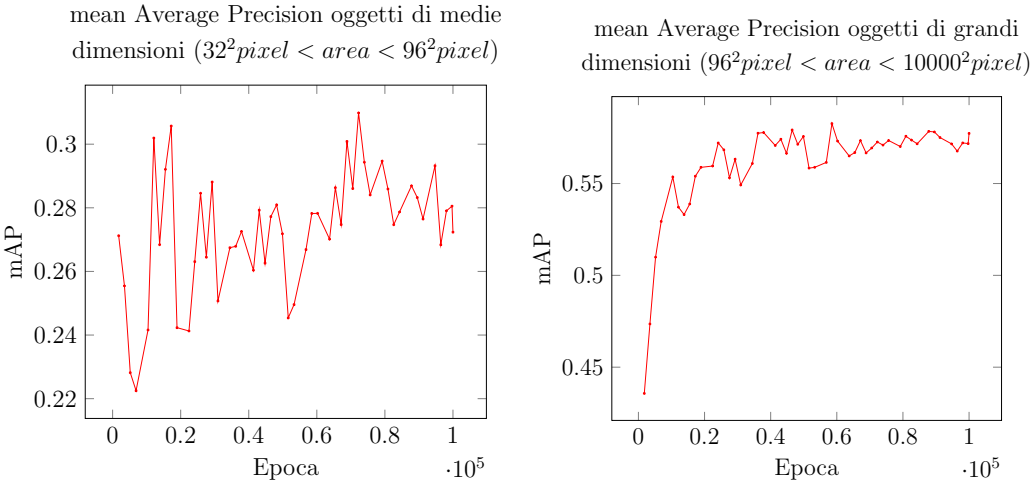


Figura 3.9: Dataset CORO mean Average Precision per classi di dimensioni degli oggetti.

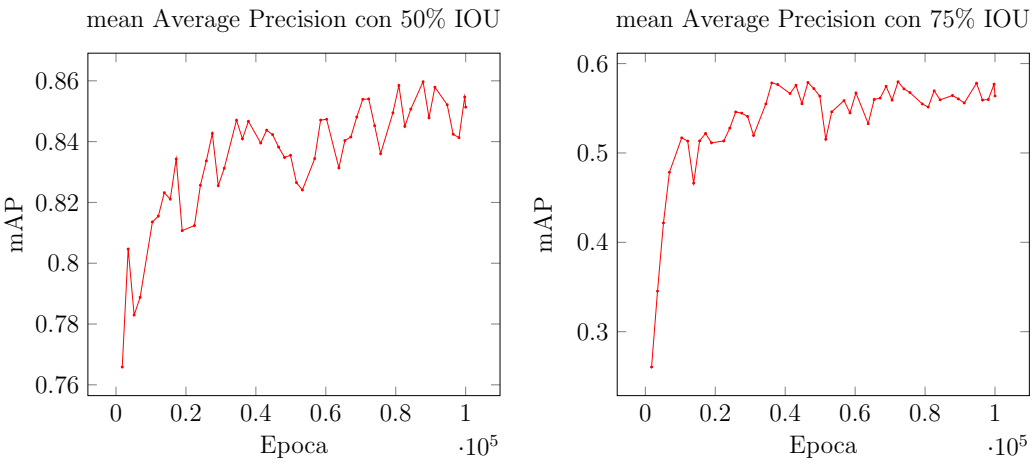


Figura 3.10: Dataset CORO mean Average Precision Intersection Over Union.

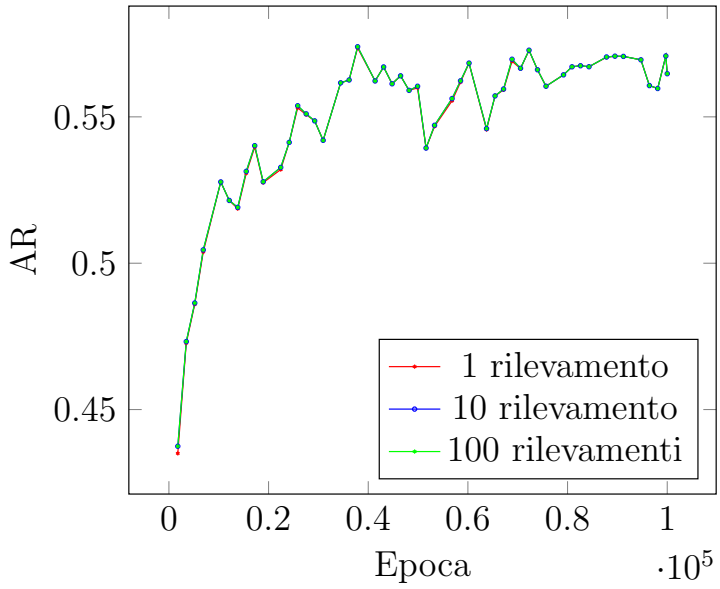


Figura 3.11: Dataset CORO Average Recall.

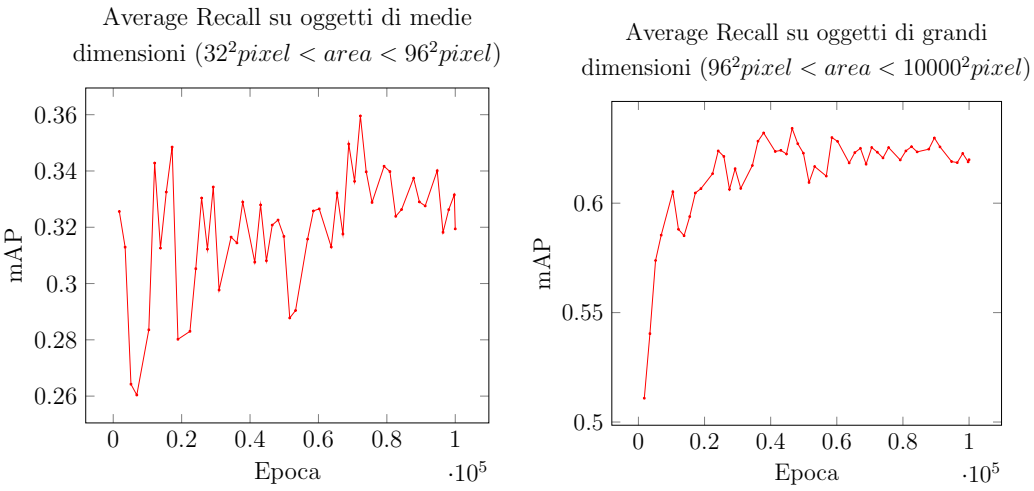


Figura 3.12: Dataset CORO Average Recall per classi di dimensioni degli oggetti.

3.3 TFLite

Il sistema proposto, come già anticipato in precedenza, dopo aver addestrato la rete neurale converte il modello nel formato *tflite*. Tale formato è stato definito da Tensorflow nella sua versione Lite. *Tensorflow Lite* [24] consiste in una versione più leggera di Tensorflow adatta per i dispositivi mobile ed embedded. Tensorflow Lite, infatti, permette l'esecuzione di diversi modelli di Machine Learning su tali dispositivi, mantenendo una bassa latenza e la dimensione del modello ridotta.
Per questo motivo, si è deciso di effettuare un'ulteriore analisi qualitativa. Tale analisi consiste nell'utilizzo del modello *tflite* generato, allo scopo di effettuare rilevazioni in real-time su smartphone degli oggetti su cui è stata addestrata la rete neurale. Nella figura 3.13 sono mostrati alcuni fotogrammi ottenuti durante le rilevazioni di oggetti appartenenti al dataset COB, mentre nella figura 3.14 quelli del dataset CORO.

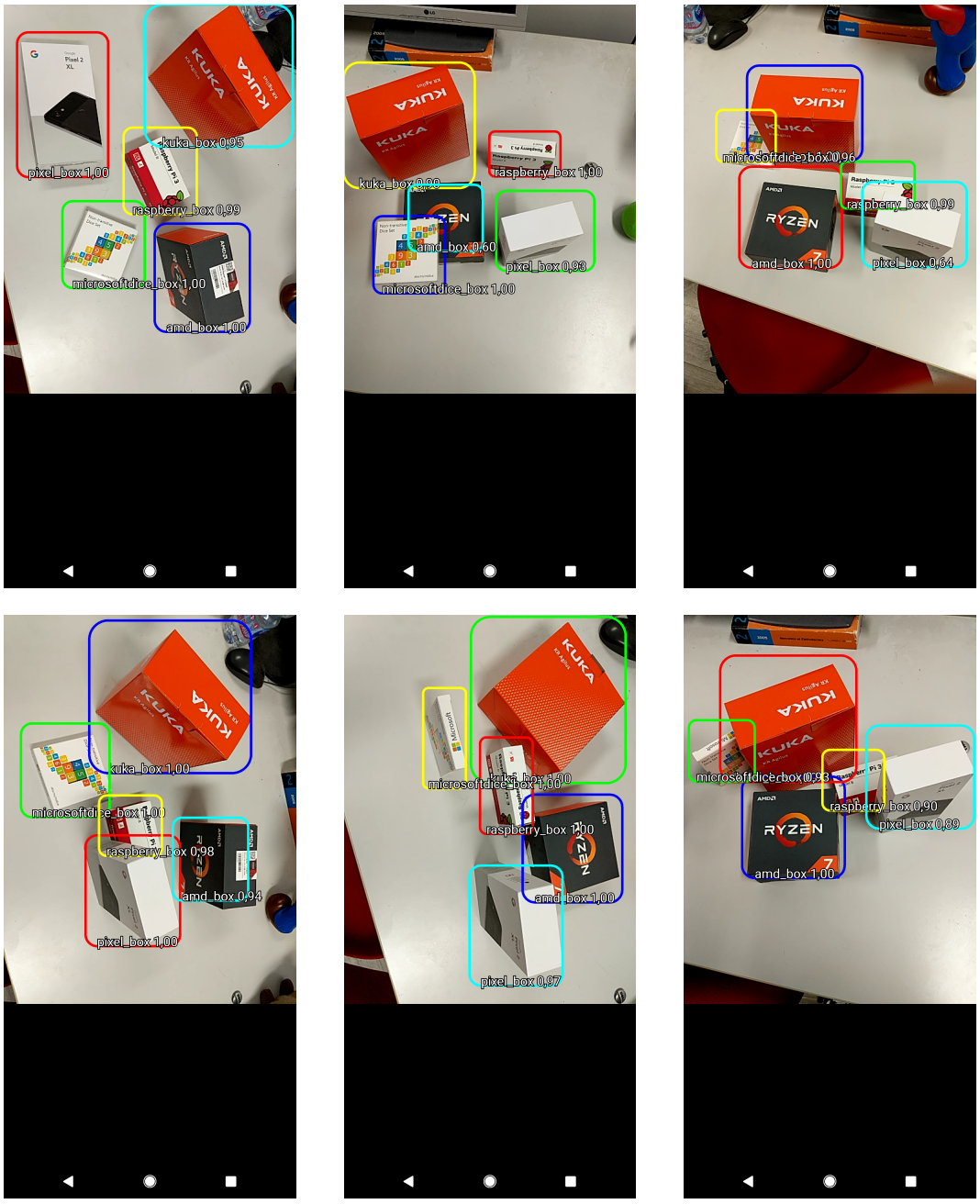


Figura 3.13: Insieme di fotogrammi ottenuti durante il rilevamento in real-time di oggetti appartenenti al dataset COB.

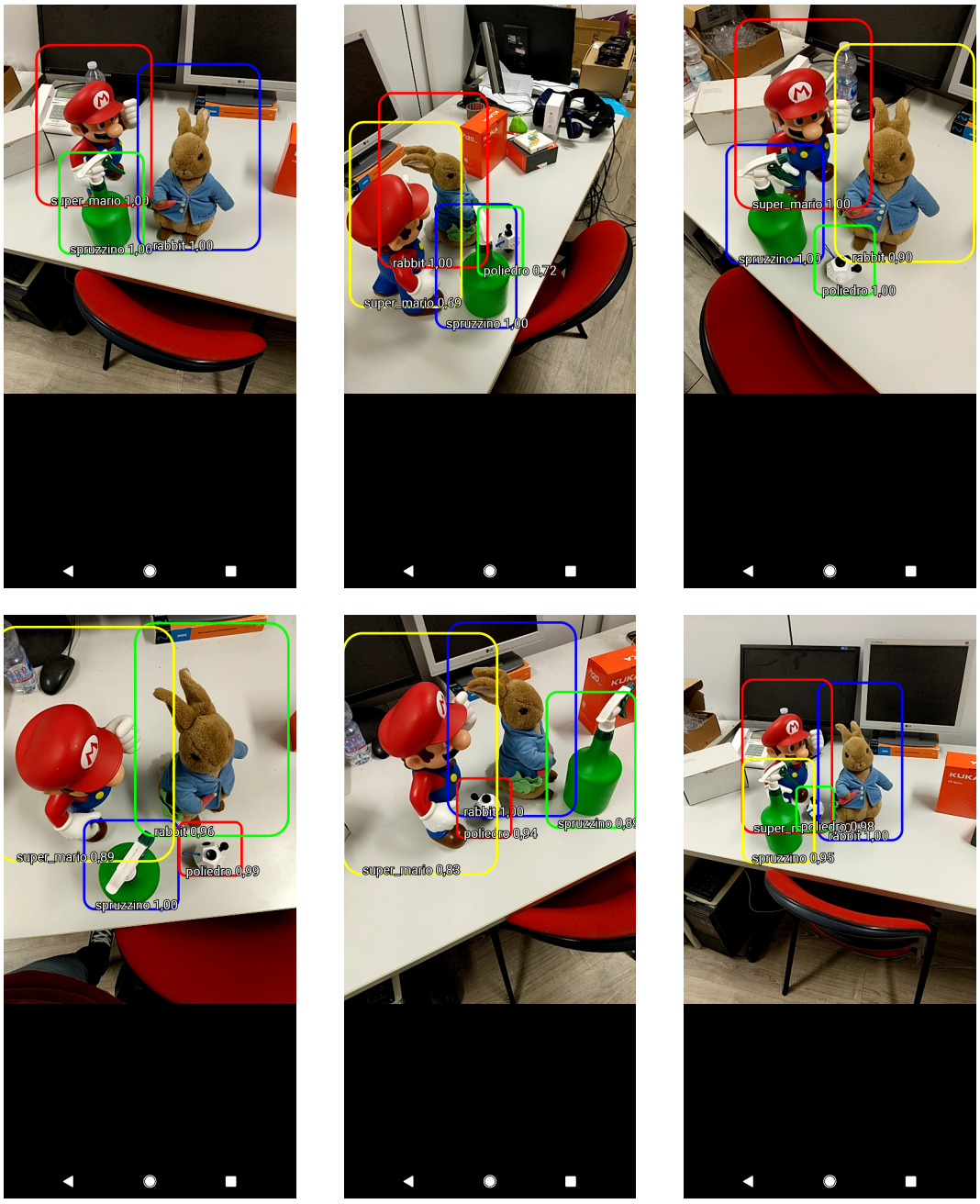


Figura 3.14: Insieme di fotogrammi ottenuti durante il rilevamento in real-time di oggetti appartenenti al dataset CORO.

Conclusioni

L'obiettivo della tesi era quello di fornire uno strumento di realtà aumentata su dispositivi mobili per labeling di immagini semi-automatico.
Dopo un'attenta fase di progettazione è stato definito un sistema software formato da due moduli: *mobile* e *desktop*. Il primo consiste in un'applicazione per dispositivi mobili, la quale, grazie a tecniche di realtà aumentata, permette la generazione di dataset contenenti immagini labellate. Il secondo, invece, permette di utilizzare i dati generati dall'applicazione mobile per l'addestramento di reti neurali convoluzionali per il rilevamento di oggetti.
Infine, è stata effettuata una serie di analisi per valutare l'efficacia e le performance del sistema, i quali sono stati esposti nel capitolo 3. I risultati ottenuti sottolineano una buona capacità della rete addestrata di rilevare gli oggetti del dataset. Per questo motivo e poiché permette una riduzione drastica dei tempi di creazione dei dataset, il sistema fornito risulta soddisfare tutti i requisiti predefiniti.
In conclusione, è possibile affermare che il sistema proposto fornisce un approccio unico ed innovativo per la generazione semi-automatica di dataset contenenti immagini labellate.

Sviluppi futuri

Il sistema software proposto, avendo dimostrato le sue ottime capacità e performance, apre le porte a diversi sviluppi futuri. Un possibile miglioramento potrebbe consistere nell'introduzione di nuove funzionalità nell'applicazione mobile, e.g. la possibilità di caricare da file i modelli 3D da utilizzare. Un possibile sviluppo futuro molto importante è quello di permettere l'addestramento di altre tipologie di DNN nell'ambito della Computer Vision, e.g. DNN

in grado di stimare la posa 3D dell'oggetto rilevato. Questo è possibile poiché il sistema oltre a fornire informazioni 2D sugli oggetti presenti nelle immagini, è in grado di fornire anche la posa 3D di tali elementi.

Ringraziamenti

Vorrei iniziare col ringraziare il Professor Luigi Di Stefano per l'opportunità offertami di svolgere una tesi nell'ambito della Visione Artificiale, la quale mi ha sempre affascinato.
Un ringraziamento particolare va al Dottore di Ricerca Daniele De Gregorio per la sua disponibilità e il tempo dedicatomi: oltre ad avermi coadiuvato nella stesura finale della tesi, mi ha permesso di approfondire le mie conoscenze relative agli argomenti trattati e supportato nell'utilizzo di nuove tecnologie.
In ultimo, vorrei ringraziare tutte le persone che mi hanno accompagnato durante questo percorso di studi, supportandomi nei momenti di difficoltà e condividendo con me la gioia degli obiettivi raggiunti.

Elenco delle figure

Bibliografia

[1] R. Szeliski, *Computer vision: algorithms and applications*. Springer Science & Business Media, 2010.

[2] D. Lowe, "Object recognition from local scale-invariant features," vol. 2, pp. 1150 – 1157 vol.2, 02 1999.

[3] H. Bay, T. Tuytelaars, and L. Van Gool, "Surf: Speeded up robust features," in *Computer Vision – ECCV 2006* (A. Leonardis, H. Bischof, and A. Pinz, eds.), (Berlin, Heidelberg), pp. 404–417, Springer Berlin Heidelberg, 2006.

[4] E. Rosten and T. Drummond, "Machine learning for high-speed corner detection," in *Computer Vision – ECCV 2006* (A. Leonardis, H. Bischof, and A. Pinz, eds.), (Berlin, Heidelberg), pp. 430–443, Springer Berlin Heidelberg, 2006.

[5] D. G. Lowe, "Distinctive image features from scale-invariant keypoints," *International journal of computer vision*, vol. 60, no. 2, pp. 91–110, 2004.

[6] D. H. Ballard, "Generalizing the hough transform to detect arbitrary shapes," *Pattern recognition*, vol. 13, no. 2, pp. 111–122, 1981.

[7] Y. Bengio *et al.*, "Learning deep architectures for ai," *Foundations and trends® in Machine Learning*, vol. 2, no. 1, pp. 1–127, 2009.

[8] G. Hinton, L. Deng, D. Yu, G. E. Dahl, A.-r. Mohamed, N. Jaitly, A. Senior, V. Vanhoucke, P. Nguyen, T. N. Sainath, *et al.*, "Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups," *IEEE Signal processing magazine*, vol. 29, no. 6, pp. 82–97, 2012.

[9] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "Imagenet: A large-scale hierarchical image database," in *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on*, pp. 248–255, Ieee, 2009.

[10] D. M. Hawkins, "The problem of overfitting," *Journal of chemical information and computer sciences*, vol. 44, no. 1, pp. 1–12, 2004.

[11] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You only look once: Unified, real-time object detection," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 779–788, 2016.

[12] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. Reed, C.-Y. Fu, and A. C. Berg, "Ssd: Single shot multibox detector," in *European conference on computer vision*, pp. 21–37, Springer, 2016.

[13] T.-Y. Lin, M. Maire, S. Belongie, J. Hays, P. Perona, D. Ramanan, P. Dollár, and C. L. Zitnick, "Microsoft coco: Common objects in context," in *European conference on computer vision*, pp. 740–755, Springer, 2014.

[14] A. Kuznetsova, H. Rom, N. Alldrin, J. Uijlings, I. Krasin, J. Pont-Tuset, S. Kamali, S. Popov, M. Malloci, T. Duerig, *et al.*, "The open images dataset v4: Unified image classification, object detection, and visual relationship detection at scale," *arXiv preprint arXiv:1811.00982*, 2018.

[15] M. Everingham, S. A. Eslami, L. Van Gool, C. K. Williams, J. Winn, and A. Zisserman, "The pascal visual object classes challenge: A retrospective," *International journal of computer vision*, vol. 111, no. 1, pp. 98–136, 2015.

[16] C. Mitash, K. E. Bekris, and A. Boularias, "A self-supervised learning system for object detection using physics simulation and multi-view pose estimation," in *Intelligent Robots and Systems (IROS), 2017 IEEE/RSJ International Conference on*, pp. 545–551, IEEE, 2017.

[17] D. T. Nguyen, B.-S. Hua, L.-F. Yu, and S.-K. Yeung, "A robust 3d-2d interactive tool for scene segmentation and annotation," *IEEE transactions*

on visualization and computer graphics, vol. 24, no. 12, pp. 3005–3018, 2018.

[18] R. T. Azuma, "A survey of augmented reality," *Presence: Teleoperators & Virtual Environments*, vol. 6, no. 4, pp. 355–385, 1997.

[19] J. Fuentes-Pacheco, J. Ruiz-Ascencio, and J. M. Rendón-Mancha, "Visual simultaneous localization and mapping: a survey," *Artificial Intelligence Review*, vol. 43, no. 1, pp. 55–81, 2015.

[20] G. P. Roston and E. P. Krotkov, "Dead reckoning navigation for walking robots," tech. rep., CARNEGIE-MELLON UNIV PITTSBURGH PA ROBOTICS INST, 1991.

[21] "Google arcore." `https://developers.google.com/ar/`. Visitato: 28-02-2019.

[22] "Tensorflow." `https://www.tensorflow.org/`. Visitato: 28-02-2019.

[23] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna, "Rethinking the inception architecture for computer vision," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 2818–2826, 2016.

[24] "Tensorflow lite." `https://www.tensorflow.org/lite`. Visitato: 28-02-2019.