Alma Mater Studiorum - Università di Bologna

SCUOLA DI INGEGNERIA E ARCHITETTURA
Dipartimento Informatica - Scienza e Ingegneria
Corso di Laurea Magistrale in Ingegneria informatica

Sistema di visione artificiale "chiavi in mano" composto da sistema stereo e rete neurale per applicazioni di guida robot industriali

Tesi di laurea in Computer Vision and Image Processing

Presentata da:
Lauria Davide

Relatore:
Chiar.mo Prof.
Di Stefano Luigi

Correlatore:
Dott.
Casadio Giuseppe

Sessione IV
Anno Accademico 2019-2020

Ringrazio...il professore Di Stefano Luigi per la disponibilità e la pazienza.
Il team di Specialvideo che mi ha accolto e supportato in tutto nonostante il momento storico anomalo, in particolare nelle persone del Dottor Casadio Giuseppe e del Dottor Ballardini Valeriano.
Infine un ringraziamento speciale a tutta la mia famiglia e alle persone importanti che mi sono state vicino in questo lungo percorso.

Indice

Capitolo 1

Introduzione

Quando è stata introdotta l'intelligenza artificiale, gli scienziati/ricercatori informatici sognavano di sviluppare macchine in grado di vedere e capire il mondo come fanno gli esseri umani. Gli sforzi compiuti hanno portato all'avvento della visione artificiale, che è un vasto sottocampo dell'intelligenza artificiale che si occupa dell'elaborazione del contenuto di dati/informazioni visive. Negli ultimi tempi, la tecnologia di visione artificiale ha compiuto grandi passi avanti grazie ai miglioramenti nell'apprendimento profondo e nelle reti neurali artificiali [1]. Il deep learning è una suddivisione dell'intelligenza artificiale che è eccezionalmente efficace nel processare dati non strutturati come video e immagini.
I progressi hanno fornito la via per aumentare l'uso della visione artificiale nei domini esistenti e introdurne di nuovi. Nella maggior parte dei casi, gli algoritmi di visione artificiale sono diventati una componente essenziale delle applicazioni che utilizziamo ogni giorno.
Con l'evolversi delle tecnologie e dei bisogni delle imprese la visione artificiale è diventata sempre più fondamentale anche in ambito industriale e nei processi che ne fanno parte.

Questo elaborato nasce da un progetto propostomi dal team di Specialvideo, una realtà del bolognese che dal 1993 opera nel settore della visione artificiale, che riguarda una commessa affidatagli da un'azienda produttrice di prosciutti.
Questo prosciuttificio aveva la necessità di automatizzare il procedimento di "sugnatura" dei prosciutti ("spennellamento" di una salsa speziata sulla carne) durante il processo industriale. Per questo scopo ho articolato il progetto in 3 fasi: calibrazione e impostazione di un sistema stereo di telecamere adibito a identificare la scena tramite immagini, training di una rete neurale con le immagini raccolte per identificare la parte del prosciutto da "sugnare" e infine ricostruzione della scena 3D in modo da poter passare queste informazioni a un robot che sarà addestrato per svolgere l'operazione di "sugnatura". Di seguito vediamo come ho organizzato il documento. Nel capitolo 2 viene introdotto il concetto di visione artificiale e le sue tipiche applicazioni in ambito industriale, nel capitolo 3 ho tenuto a spiegare un po' di storia riguardo Specialvideo e qualche suo case history degno di nota. Nel capitolo 4 entriamo nel vivo del progetto, tratta l'impostazione del sistema stereo e di come sono arrivato al calcolo della mappa di disparità, nel capitolo 5 invece si parla di deep learning, un approfondimento generale su ciò che tratta e delle sue applicazioni legate alla visione artificiale, in particolare la segmentazione. I capitoli 6 e 7 trattano

della parte finale del progetto, ovvero la parte di segmentazione delle immagini dei prosciutti tramite rete neurale e quella finale di ricostruzione della scena 3D. Nel capitolo 8 infine ho riportato i test effettuati sulle risorse hardware a mia disposizione per poter capire l'impatto dei vari programmi utilizzati. Il capitolo 9 è riservato alle conclusioni e a qualche mia considerazione finale.

Capitolo 2

Background: cos'è la Computer Vision?

La computer vision, o visione artificiale, è quel campo di studi interdisciplinare che si occupa di capire come i computer possano riprodurre processi e funzioni dell'apparato visivo umano. Non solo, quindi, acquisire le immagini statiche o in movimento, identificarle, riconoscerle ed estrarne le informazioni utili per prendere decisioni: l'elaborazione digitale delle immagini, *digital image processing*, è infatti solo una parte della computer vision. Una parte che rientra, precisamente, nella *early vision*, l'elaborazione "a basso livello di astrazione" su cui, dagli anni Sessanta in poi, sono stati fatti numerosi passi avanti. La sfida più ambiziosa della computer vision riguarda la visione **high level**, "ad alto livello di astrazione e comprensione", che dall'immagine in 2D riesce a elaborare, ricostruire ed analizzare l'intero contesto in 3D in cui l'immagine è inserita.

A fine Ottocento, il pittore George Seurat cominciò a dipingere senza mescolare i colori sulla tavolozza, ma giustapponendo direttamente sulla tela bianca piccoli tocchi di colori puri complementari, "scomponendoli in puntini": sarebbe stato l'occhio di chi osserva alla giusta distanza a mescolarli e vedere sul quadro sfumature e tinte uniformi. Allo stesso modo, ogni immagine digitale può essere scomposta in puntini, i pixel, le unità minime della superficie di una immagine, disposti in modo diverso su una griglia, una matrice in 2D (scala di grigi) o 3D (a colori): la dimensione dell'immagine dipende dal numero di righe e di colonne della matrice, il colore dell'immagine dal numero o dalla terna di numeri associata ad ogni pixel. Tra zero-nero e 255-bianco nel caso di immagini in scala di grigi, invece una terna di numeri con l'intensità di rosso, verde e blu per le immagini a colori (RGB).
In un sistema di visione artificiale, l'immagine viene acquisita a partire da un sensore (fotosensibile, tomografico, radar) che produce un segnale in uscita e lo invia al calcolatore, che lo digitalizza e memorizza. L'immagine viene quindi immagazzinata e, a seconda degli scopi di analisi per cui è stato programmato il sistema, viene prima *pre-processata* dal software, ovvero resa "idonea" all'analisi, poi elaborata in senso stretto, attraverso l'estrazione delle caratteristiche di interesse, *feature extraction*, e/o la selezione di particolari aree per un'ulteriore elaborazione. Il risultato dell'analisi verrà quindi confrontato con il modello di riferimento, classificato e utilizzato per prendere decisioni.

Uno dei fattori trainanti alla base del miglioramento della visione artificiale in

tempi recenti è la quantità di dati che generiamo quotidianamente (3 milioni di immagini condivise quotidianamente), utili per allenare e migliorare i sistemi di visione artificiale. La dimensione prevista per il mercato della visione artificiale dovrebbe raggiungere i 48,6 miliardi di dollari entro il 2022.[2]
In meno di un decennio, le applicazioni informatiche odierne legate alla visione artificiale hanno raggiunto una precisione del **99%** (partendo da circa un 50%), rendendole più precise degli umani nel rispondere rapidamente a input visivi[2].

2.1 Applicazioni della Computer vision

La visione artificiale presenta molteplici applicazioni pratiche. Tra le principali: il riconoscimento degli oggetti; il telerilevamento (remote sensing) ovvero il monitoraggio ambientale a distanza che serve a rilevare e classificare le condizioni del pianeta; la scansione dei codici in QrCode; il riconoscimento ottico dei caratteri (OCR), che converte qualunque tipo di testo, sia esso scansionato, scritto a mano, all'interno di una immagine; il restauro di immagini e di opere d'arte grazie a un'evoluta analisi delle superfici sia dal punto di vista geometrico che dei materiali; i robot che ispezionano e manipolano gli oggetti; il visual servoing, o asservimento visivo basato sull'immagine, che controlla il movimento dei robot grazie alle informazioni rilasciate dai sensori visivi; la modellazione 3D; il tracciamento dei movimenti e l'analisi diagnostica in telemedicina; l'indicizzazione dei database di immagini; i veicoli a guida autonoma, capaci di muoversi nell'ambiente circostante e di "riconoscere" la strada, i segnali, i pedoni; la manutenzione predittiva, il controllo dei processi produttivi e il supporto alla sicurezza negli impianti industriali.

Ciascuna delle aree di applicazione sopra descritte è possibile grazie a una serie di task di visione artificiale, problemi di misurazione o di elaborazione più o meno ben definiti, che possono essere risolti utilizzando diverse metodologie. Di seguito vengono presentati alcuni esempi di attività tipiche di visione artificiale.
Le attività di visione artificiale includono metodi per acquisire, elaborare, analizzare e comprendere immagini digitali e l'estrazione di dati di grosse dimensioni dal mondo reale al fine di produrre informazioni numeriche o simboliche, ad esempio sotto forma di decisioni. La comprensione, in questo contesto, significa la trasformazione delle immagini visive in descrizioni del mondo, che possono interfacciarsi con altri processi di pensiero e sollecitare un'azione appropriata. Questa comprensione dell'immagine può essere vista come la comprensione delle informazioni simboliche derivanti dai dati dell'immagine utilizzando modelli costruiti con l'ausilio di geometria, fisica, statistica e teoria dell'apprendimento.
Riconoscimento degli oggetti, il problema classico nella visione artificiale, nell'elaborazione delle immagini e nella visione artificiale è quello di determinare se i dati dell'immagine contengono o meno un oggetto, una caratteristica o un'attività specifica.
Tramite questa tecnica è possibile riconoscere uno o più oggetti o classi di oggetti pre-specificati o appresi, di solito insieme alle loro posizioni 2D nell'immagine o pose 3D nella scena; vediamo un esempio di riconoscimento real time in figura 2.1.
Identificazione, viene riconosciuta una singola istanza di un oggetto. Gli esempi includono l'identificazione del volto o dell'impronta digitale di una persona specifica, l'identificazione di cifre scritte a mano o l'identificazione di un veicolo specifico.

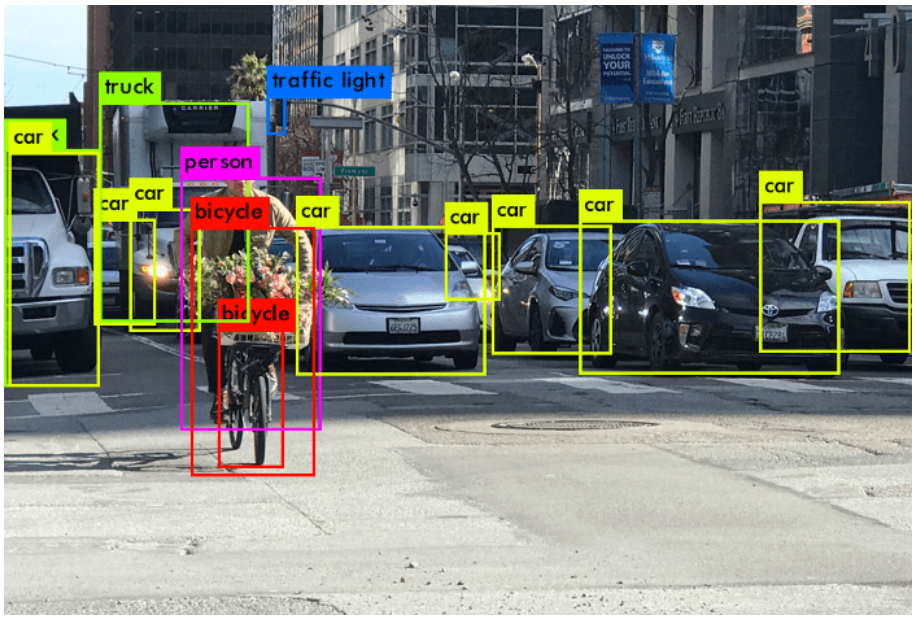


Figura 2.1: Riconoscimento di oggetti in real time, tramite algoritmo YOLO.[3]

Rilevamento, i dati dell'immagine vengono scansionati per rilevare una condizione specifica. Gli esempi includono il rilevamento di possibili cellule o tessuti anormali nelle immagini mediche o il rilevamento di un veicolo in un sistema di pedaggio stradale automatico. Il rilevamento basato su calcoli relativamente semplici e veloci viene talvolta utilizzato per trovare regioni più piccole di dati di immagine interessanti che possono essere ulteriormente analizzati da tecniche più esigenti dal punto di vista computazionale per produrre un'interpretazione corretta.
Attualmente, i migliori algoritmi per tali compiti sono basati su reti neurali convoluzionali. I migliori algoritmi continuano però ancora a faticare con il riconoscimento di oggetti piccoli o sottili, come una piccola formica sullo stelo di un fiore o una persona che tiene una penna in mano. Hanno anche problemi con le immagini che sono state distorte con l'utilizzo di filtri (un fenomeno sempre più comune con le moderne fotocamere digitali). Al contrario, questo tipo di immagini raramente disturba il sistema visivo umano. Gli esseri umani, tuttavia, tendono ad avere problemi in altre situazioni, ad esempio, non sono efficaci nel classificare oggetti in classi a grana fine, come la particolare razza di cane o specie di uccelli, mentre le reti neurali convoluzionali gestiscono questi casi con più facilità.
Esistono diversi compiti specializzati basati sul riconoscimento, come:

- *Recupero di immagini basato sul contenuto:* ricerca di tutte le immagini in un insieme più ampio di immagini con un contenuto specifico. Il contenuto può essere specificato in diversi modi, ad esempio in termini di somiglianza relativa a un'immagine di target (dammi tutte le immagini simili all'immagine X), o in termini di criteri di ricerca di alto livello forniti come input di testo (dammi tutte le immagini che contengono molte case);

- *Stima della posa:* stima della posizione o dell'orientamento di un oggetto specifico rispetto alla telecamera. Un'applicazione di esempio per questa tecnica è

sicuramente la realtà aumentata, che tipicamente richiede la stima della posa degli oggetti nello spazio;

- *Riconoscimento ottico dei caratteri:* identificazione dei caratteri nelle immagini di testo stampato o scritto a mano, solitamente al fine di codificare il testo in un formato modificabile o indicizzabile (ad esempio ASCII);

- *Lettura di codici 2D:* lettura di codici 2D come i codici QR;

- *Riconoscimento facciale*;

- *Tecnologia di riconoscimento delle forme:* nei sistemi di riconoscimento delle persone e differenziazione tra esseri umani e oggetti.

Analisi del movimento, diverse attività riguardano la stima del movimento dove una sequenza di immagini viene elaborata per produrre una stima della velocità in ogni punto dell'immagine o della scena 3D, o anche della telecamera che produce le immagini. Esempi di tali attività sono:

- *Egomotion:* determinazione del movimento rigido 3D (rotazione e traslazione) della telecamera da una sequenza di immagini prodotta dalla stessa;

- *Tracking:* seguire i movimenti di un insieme più piccolo di punti di interesse o oggetti (ad esempio, veicoli, esseri umani o altri organismi) in una sequenza di immagini;

- *Optical Flow:* determinare, per ogni punto dell'immagine, il suo movimento rispetto al piano dell'immagine, cioè il suo movimento apparente. Questo movimento è il risultato sia di come si muove il punto 3D corrispondente nella scena, sia di come si muove la telecamera rispetto alla scena.

Ricostruzione della scena, data una o più immagini di una scena, o di un video, la ricostruzione della scena mira a calcolare un modello 3D della scena. Nel caso più semplice il modello può essere un insieme di punti 3D. Metodi più sofisticati producono un modello di superficie 3D completo. L'avvento dell'imaging 3D che non richiede movimento o scansione e gli algoritmi di elaborazione correlati sta consentendo rapidi progressi in questo campo. Il rilevamento 3D basato su griglie può essere utilizzato per acquisire immagini 3D da più angolazioni.
Restauro dell'immagine, lo scopo del restauro dell'immagine è la rimozione del rumore (rumore del sensore, sfocatura di movimento, ecc.), vediamo un esempio in figura 2.2. L'approccio più semplice possibile per la rimozione del rumore è costituito da vari tipi di filtri come filtri passa-basso o filtri mediani. Metodi più sofisticati presuppongono un modello di come appaiono le strutture dell'immagine, per distinguerle dal rumore. Analizzando prima i dati dell'immagine in termini di strutture, come linee o bordi, e quindi controllando il filtraggio sulla base delle informazioni derivanti dalla fase di analisi, si ottiene solitamente un livello migliore di rimozione del rumore rispetto agli approcci più semplici. Un esempio in questo campo è la pittura.

Figura 2.2: Esempio di denoising di un'immagine.

2.1.1 Applicazioni industriali

Quando si parla di computer vision tipicamente si tende a pensare solo ad applicazioni di riconoscimento/classificazione di oggetti o di videosorveglianza, ma in ambito industriale i casi d'uso reali sono molti di più. Di seguito si riporta un elenco.

Manutenzione predittiva

Qualsiasi business manifatturiero ha bisogno di regolari periodi di manutenzione delle macchine e di tutte le apparecchiature su cui esso si basa, in maniera da prevenire guasti di uno o più componenti che inevitabilmente costringerebbero a uno stop non pianificato della produzione. La manutenzione predittiva è un processo basato su machine learning o deep learning e sensori montati su macchinari/dispositivi che, attraverso la raccolta di dati da questi ultimi, deve identificare segnali che preludono la rottura di uno o piu' componenti e suggerire azioni corretive prima che il danno accada. Per capire meglio l'importanza di questa tecnica basta pensare ad alcuni settori quali, per esempio, l'automotive o il farmaceutico, dove anche un solo minuto di stop imprevisto della produzione può costare all'azienda decine di migliaia di euro. Alcuni moderni sistemi di manutenzione predittiva sono basati su computer vision e utilizzano non solo immagini, ma anche i metadati a esse associati.
Ad esempio, il modello di intelligenza artificiale può indicare che gli attuali programmi o macchinari sono ideali e non richiedono modifiche, oppure che è urgente evitare un guasto o, ancora, che è possibile rimandare un intervento dispendioso perché quella determinata attrezzatura non ne ha realmente necessità. In aggiunta, più dati vengono acquisiti dai sensori e più gli algoritmi di machine learning o deep learning possono apprendere sulla storia e sulle condizioni delle macchine, migliorando di continuo le metodologie di manutenzione.

Controllo qualità

Qualsiasi azienda manifatturiera ha come obiettivo primario la creazione di componenti o prodotti idealmente privi di difetti. Una tecnologia basata su machine vision (a supporto degli operatori o anche completamente automatizzata) di sicuro aiuta in questa direzione, quantomeno riducendo drasticamente la probabilità di difetti.

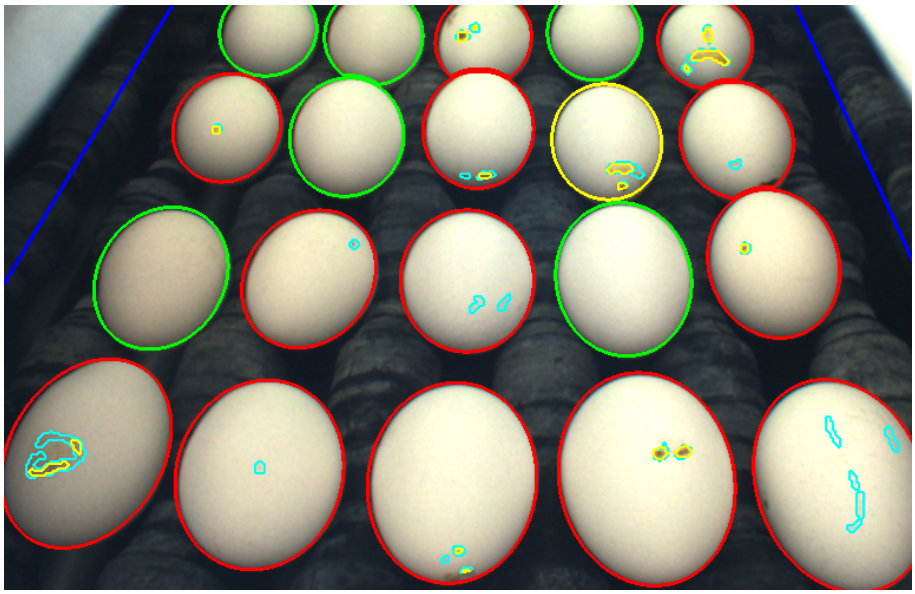


Figura 2.3: Esempio di identificazione di difetti tramite computer vision per un azienda agricola, nel caso specifico durante il controllo sulle uova.

Essa consente anche di identificare diverse categorie di difetti e di assegnare loro dei pesi specifici in modo da impostare regole di stop automatico della produzione in base ai tipi e rispettive percentuali riscontrati. Così come per la manutenzione predittiva, anche in questi casi sono utilizzati sia immagini che relativi metadati, ne vediamo un esempio in figura 2.3.

Ispezione di package

In alcuni settori, come l'industria farmaceutica per esempio, l'ispezione di pillole o capsule prima che queste vengano inserite nei contenitori di destinazione è un'operazione di vitale importanza. Gran parte delle soluzioni per questo tipo di controlli è attualmente basata su computer vision. Man mano che le pillole proseguono il loro percorso all'interno del loro processo produttivo, tramite la raccolta e analisi di immagini, è possibile, per esempio, stabilire quali siano conformi agli standard di qualità e quali invece presentino difetti di diversa natura o contare la quantità che sta per essere inserita nel package di destinazione. In questo modo è possibile il rigetto automatico delle confezioni difettose, in modo da prevenire la loro distribuzione sul mercato.

Lettura di codici a barre e QR codes

La lettura massiva di centinaia o migliaia di codici a barre o QR codes per giorno è un'attività che gli esseri umani non possono completare agevolmente in tempi accettabili e senza commettere errori. Un esempio pratico lo possiamo trovare nella produzione di dispositivi mobili, dove si fa largo uso di circuiti stampati sempre più piccoli. Per soddisfare richieste sempre più crescenti, i produttori fanno ricorso alla cosiddetta panelization[4], una tecnica che consente di connettere insieme in un singolo array un numero elevato di piccoli circuiti stampati, in modo da facilitarne

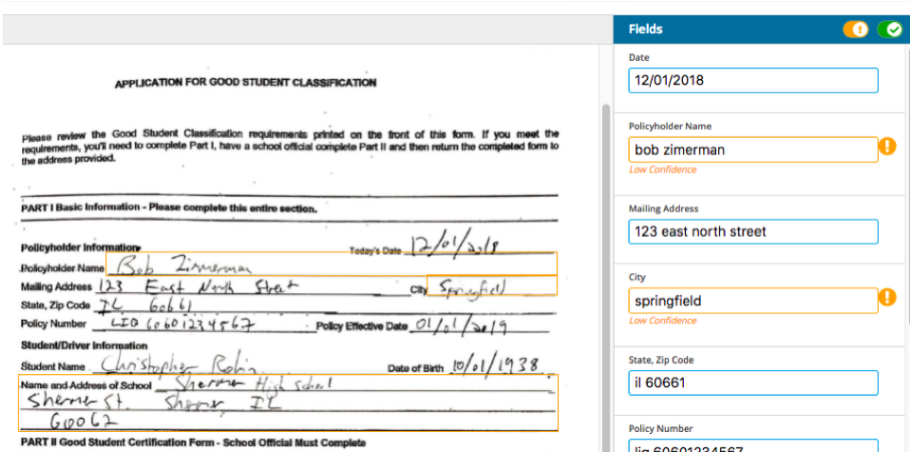


Figura 2.4: Esempio di riconoscimento della grafia in un documento.

i passaggi attraverso le varie fasi della catena di assemblaggio. Al termine del processo produttivo, per eseguire i test finali, è impensabile oramai ricorrere a tecniche manuali per identificare e separare migliaia di circuiti stampati tutti uguali: esistono già soluzioni basate su machine vision in grado di leggere e identificare con precisione un numero elevato di codici a barre o QR codes (le uniche cose che permettono di identificare un circuito stampato in maniera univoca) in tempi brevissimi.

Lettura di testi e analisi di documenti scritti a mano

Tutte le aziende manifatturiere seguono una serie di best practice standard per ottimizzare e, se non eliminare, ridurre il più possibile gli errori. Per tali scopi, ogni passo eseguito da tutti gli attori coinvolti necessita di essere documentato in tempo reale su appositi moduli. Per svariate ragioni, che possono andare dalla sicurezza alla parziale o nulla digitalizzazione di tutta la documentazione di produzione, in moltissimi casi una gran parte dei documenti è ancora cartacea e compilata a mano. Sistemi basati su deep learning e addestrati a riconoscere testi scritti a mano e trasferire le informazioni in formato digitale sono già disponibili[4], un esempio lo vediamo in figura 2.4. La loro adozione nel manifatturiero non è ancora diffusa come in altre aree di business, ma c'è da aspettarsi che questo trend sia destinato ad aumentare nel brevissimo termine.

Assemblaggio di componenti e prodotti

Tutti gli impianti di produzione devono aderire a standard di qualità e sicurezza. Questo aspetto diventa molto più critico in particolare in impianti altamente performanti, come nel caso dell'industria farmaceutica. Controlli quali ad esempio la corretta chiusura dei tappi, la presenza di impurità nei liquidi o crepe nei flaconi, etc. sono nella quasi totalità dei casi effettuati da sistemi basati su computer vision che, esaminando immagini da multiple fotocamere e dati provenienti da altri sensori, forniscono agli operatori anche suggerimenti circa la potenziale causa che ha provocato i difetti riscontrati.

Figura 2.5: Software che tramite computer vision riesce a valutare se un ambiente di lavoro è sicuro o meno.

Sicurezza

I casi illustrati finora riguardano principalmente macchinari e prodotti, ma la computer vision negli ultimi anni inizia a vedere le prime applicazioni anche nella prevenzione degli incidenti sul lavoro, almeno nel grande manufatturiero. Tra le principali applicazioni, sistemi in grado di verificare che gli addetti ai lavori indossino il loro PPE (Personal Protective Equipment), come vediamo in figura 2.5, e che impediscono loro l'accesso in caso di inadempienza.

Tracciamento di prodotti

Alcuni settori, come ad esempio quello farmaceutico, devono sottostare a regolamentazioni stringenti specifiche per le diverse aree geografiche in cui i prodotti transitano. Da qui la necessità di tracciarli dalla produzione fino ai consumatori finali. Le aziende sono obbligate a stampare sui contenitori dei prodotti una serie di dettagli, quali ad esempio date di scadenza, numeri seriali, altri dati di produzione, etc. Nella maggior parte dei casi, un identificatore univoco, chiamato GTIN (Global Trade Item Number) è usato per tracciare prodotti in transito attraverso diversi Paesi. I produttori sono dotati di sistemi che generano automaticamente i GTIN in un database principale e che vengono successivamente utilizzati nei processi produttivi e stampati nelle confezioni, in modo da poter rendere i singoli prodotti tracciabili e verificabili. Tradizionalmente la verifica avveniva tramite sistemi OCR in grado di leggere codici a barre, inviare i codici letti al database principale e poterli validare. Stanno prendendo piede sistemi basati su computer vision in grado di riconoscere e interpretare anche altre forme di testo stampato ed etichette.

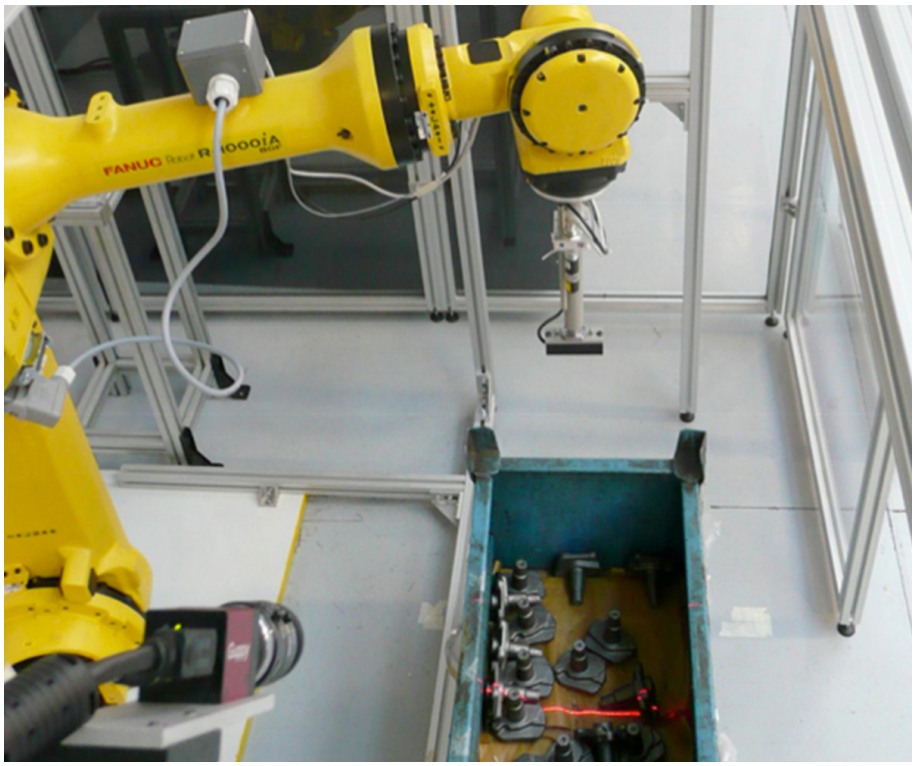


Figura 2.6: Esempio di sistema di visione per guida robot.

Inventario

L'attività di inventario di magazzino nel manufatturiero è una attività estremamente costosa in termini di tempo necessario per completarla. Nel caso di aziende produttive grandi e medio-grandi l'ispezione e il conteggio di materiali/componenti/prodotti può durare giorni e necessitare di centinaia di ore lavorative. Un settore che sta prendendo piede di recente è quello dell'utilizzo di sistemi basati su diverse tecniche di IA, che includono anche machine vision, questi ultimi in particolare basati sull'utilizzo di droni a supporto dell'attività umana, il cui obbiettivo è quello di ridurre tempi e costi di inventario.

Guida robot

In ambito industriale i sistemi di guida robot forniscono l'abilità di vedere gli oggetti, quindi permettono di prelevare i pezzi in posizione casuale oppure di eseguire un percorso di lavorazione non predefinito. L'impiego di robot industriali nelle isole robotizzate consente di realizzare impianti di automazione estremamente versatili e con tempi di attrezzaggio ridotti. Le applicazioni principali riguardano il prelievo di pezzi sul nastro di trasporto (2D), disposti in strati sui bancali (2D multistrato), o pezzi disposti completamente alla rinfusa dove è necessario l'impiego di tecniche tridimensionali (3D), come in figura 2.6.
Ne parlerò dettagliatamente più avanti.

Capitolo 3

L'azienda e il progetto

3.1 Specialvideo s.r.l.: esperienza nel settore

Specialvideo opera dal 1993 nel settore della visione artificiale, è specializzata nella progettazione di sistemi di visione industriali per l'individuazione automatica di difetti, le misure senza contatto e la guida robot [5]. Fornisce soluzioni tecnologicamente avanzate, progettate su misura per il cliente e fornite "chiavi in mano" offre anche un'assistenza continua, per garantire il corretto funzionamento del sistema e della linea produttiva. Dal 2004 ha instaurato una partership con Datalogic Automation, per creare sinergie e collaborare insieme all'evoluzione dei prodotti della gamma vision sensor. Il team è costituito da professionisti altamente qualificati, impiegati nello sviluppo di nuove applicazioni. La mission dell'azienda da 28 anni a questa parte è quella di realizzare progetti ad hoc per ogni cliente, per essere in grado di soddisfare i requisiti richiesti con la massima efficienza [5].
I sistemi vengono progettati in modo da fornire le dovute prestazioni non solo teoriche, ma anche nell'applicazione reale installata in ambito produttivo. La scelta e l'ottimizzazione degli algoritmi garantisce una velocità di calcolo sufficiente tale da non interferire con il normale processo produttivo.

3.1.1 Tecnologie

Specialvideo s.r.l. cerca di rimanere sempre al passo coi tempi in ambito tecnologico sia hardware che software, di seguito vediamo una lista delle principali tecnologie utilizzate per sistemi di visione artificiale.

Telecamera matriciale

La telecamera matriciale è un dispositivo optoelettronico in grado di produrre un'immagine bidimensionale della scena inquadrata, ne vediamo uno schema del funzionamento in figura 3.1; i modelli utilizzati oggi in ambiente industriale sono costituite da un sensore allo stato solido, cioè mediate un chip di silicio. Le tecnologie più diffuse sono quelle dei sensori CCD (Charge-Couple Devices) e CMOS (Complementary metal–oxide–semiconductor):

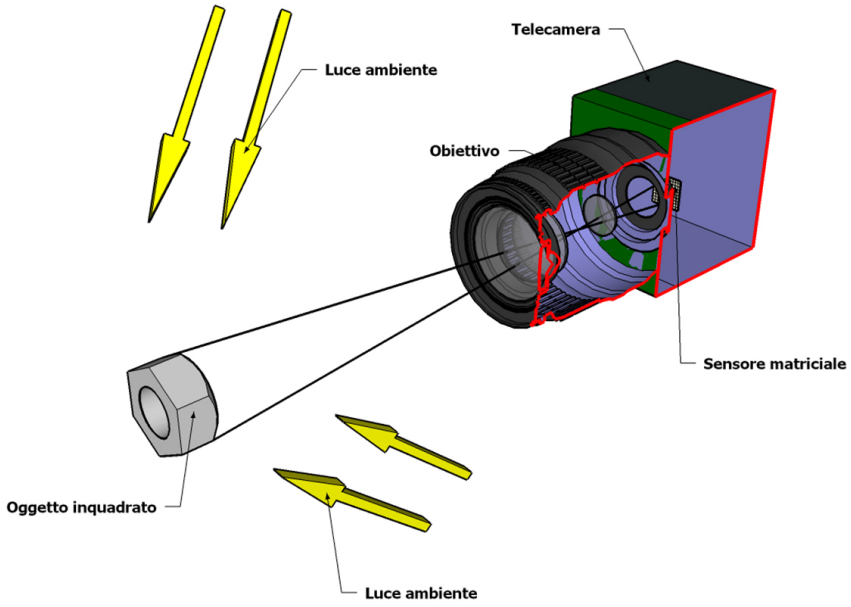


Figura 3.1: Schematizzazione del funzionamento di una telecamera matriciale.

- *la tecnologia CCD* offre una migliore qualità dell'immagine e continua ad essere ampiamente impiegata nel settore industriale quando si richiede un elevato rapporto segnale/rumore;
- *la tecnologia CMOS* è più economica e può fornire sensori ad elevatissima risoluzione a costi accessibili.

Le due tecnologie restano comunque complementari e la scelta di una o dell'altra dipende dall'applicazione e dalla reperibilità dei componenti sul mercato: esistono telecamere con sensori CMOS ad alte prestazioni e costi maggiori di telecamere a CCD con prestazioni mediocri. Il sensore è costituito da un insieme di elementi fotosensibili (detti pixel), in grado di trasformare la luce incidente in un segnale elettrico. Nella configurazione più comune, i pixel sono disposti secondo una matrice rettangolare, le cui dimensioni possono variare da pochi millimetri fino oltre 35 mm di diagonale.

Telecamera lineare

La telecamera lineare produce un'immagine costituita da una sola linea di pixel. L'immagine completa viene costruita a bordo dell'elaboratore, sfruttando il movimento relativo tra l'oggetto e la telecamera. Il movimento dell'oggetto viene solitamente sincronizzato per mezzo di un encoder.
Le telecamere lineari, come quella mostrata in figura 3.2, vengono utilizzate perché possono fornire un'elevata risoluzione spaziale (linea da 1024, 2048, ..., 8000 pixel), risultato difficilmente ottenibile con le telecamere matriciali. Le applicazioni tipiche

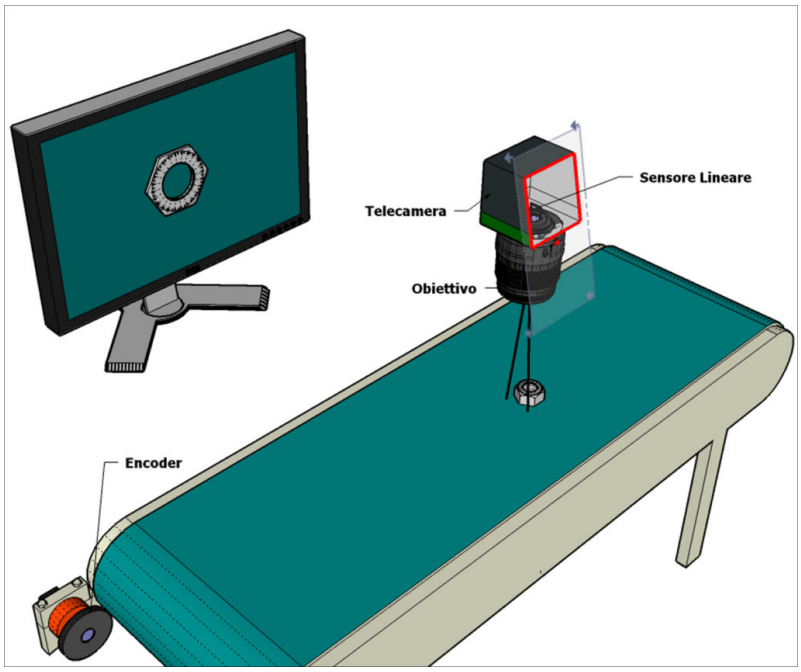


Figura 3.2: Schematizzazione del funzionamento di una telecamera lineare.

riguardano il controllo di produzioni continue come nastri di tessuto o di PVC. Le immagini ottenute con questa tecnologia possono essere elaborate nello stesso modo di quelle ottenute dalle telecamere matriciali e ad esempio possono essere impiegate per il controllo di qualità, la presenza di corpi estranei, l'allineamento di stampe e così via.

Smart camera

Si può definire Smart Camera (figura 3.3) un sistema in cui i componenti di digitalizzazione e di elaborazione delle immagini sono racchiuse in un unico contenitore. Il termine viene normalmente associato a dispositivi impiegati in applicazioni di visione artificiale. Questa architettura si pone come alternativa ad altri sistemi in cui i componenti risiedono in diversi dispositivi come per esempio quelli basati su telecamere, frame grabber e PC. Una Smart Camera, al pari degli altri sistemi di visione, esegue un programma applicativo in grado di elaborare le immagini e comunicare le informazioni estratte al mondo esterno attraverso le interfacce di cui è dotata. Spesso i programmi applicativi installati sul dispositivo sono configurabili e facili da utilizzare, anche da parte di utenti poco esperti. In questo senso la differenza tra le Smart Camera ed i sistemi di visione tradizionali risiede essenzialmente nelle dimensioni e nelle limitate risorse hardware disponibili. Tuttavia non è raro trovare delle Smart Camera in grado di svolgere alcuni compiti specifici in modo più veloce ed efficiente dei sistemi basati su PC. Un altro vantaggio di questa tipo di tecnologia è che sono completamente digitali ed il trasferimento delle immagini avviene direttamente nella memoria a bordo del dispositivo, in modo rapido e senza l'ausilio di canali esterni di comunicazione.

Figura 3.3: Un esempio di Smart Camera.

Tecniche 3D

Le tecniche 3D vengono utilizzate in svariati ambiti, dal controllo della qualità alla guida robot; troviamo un riassunto delle principali tecniche in tabella 3.1. Per quanto concerne la guida robot è necessario non confondere le necessità di tempo di calcolo del sistema di visione con le necessità del robot (anche quando il calcolo richiede tempo in genere viene effettuato in anticipo sugli spostamenti del robot). Un vantaggio generale del 3D è la sostanziale indipendenza dal colore, anche se nei dettagli varia a seconda della tecnica. La necessità di avere l'oggetto fermo o in movimento può essere sia un vantaggio che uno svantaggio (ad es. è un vantaggio necessitarlo in movimento se si trova già su un nastro trasportatore). Le tecniche che richiedono una singola immagine (snap) prescindono dal movimento o meno dell'oggetto. In questo campo d'azione le tecniche utilizzate fanno capo a due branche principali:

- *TOF:* time of flight della luce, basato su dei sensori;
- *Triangolazione:* basato su una o più telecamere ed eventualmente illuminatori.

Per la triangolazione ci sono svariate tecniche utilizzabili.
Lama di luce laser, un laser lineare illumina una sezione dell'oggetto. Conoscendo la posizione della telecamera e del laser, è possibile ricavare le coordinate dei vari punti illuminati dal laser. Per effettuare la scansione dell'oggetto è necessario che il laser o l'oggetto si muova. Si ottiene un'immagine "densa", cioè si hanno buoni dati su tutti i punti della superficie. Questa tecnica è, anche più delle altre, indipendente dai colori; d'altra parte è impossibile ottenerli anche se fossero necessari. Viene

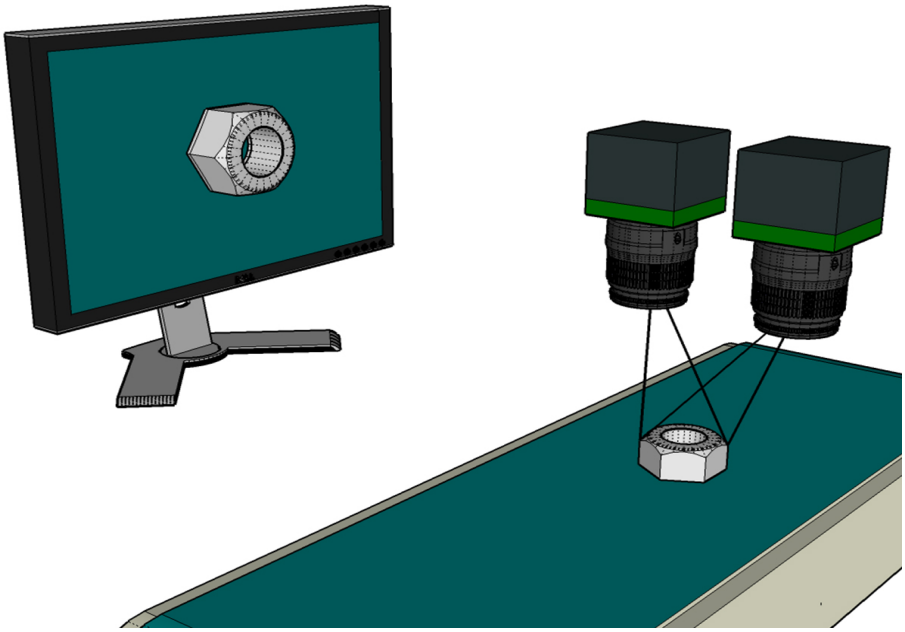


Figura 3.4: Schematizzazione di un sistema stereo.

utilizzata per svariate applicazioni, dalla guida robot al controllo della qualità.
Stereoscopia, si utilizzano due telecamere, che riprendendo l'oggetto da angoli diversi, permettono una visione stereoscopica dei punti inquadrati da entrambe (similmente alla visione umana), esempio in figura 3.4. È sufficiente una singola immagine (snap) ed è quindi possibile applicare questa tecnica sia su oggetti fermi, che in movimento. L'immagine risulta poco densa, cioè non si avranno tutti i punti dell'oggetto, ma dai punti ottenuti si avranno comunque buoni dati. Volendo, è possibile ottenere anche i colori dell'oggetto.
Omografia, richiede che la parte che si vuole studiare dell'oggetto sia tutta su una stessa superficie, ovvero un piano nello spazio. Considerando la dimensione e la deformazione prospettica si ottiene la posizione dell'oggetto nello spazio. Lavora su singola immagine (snap). Generalmente non necessita di un apposito illuminatore, è sufficiente una telecamera.
Fringe, si necessita di un oggetto statico e si fanno una decina di immagini da angolazioni differenti. E' una tecnica lenta ma precisa. Utilizzabile per misure e per reverse engineering, ma non per la guida robot.
Viste multiple, svariate inquadrature di un singolo oggetto fatte da una telecamera. E' necessario che si vedano bene i bordi, quindi ad esempio è adatto per un cubo ma non per una sfera. E' necessario il confronto con un modello Cad completo dell'oggetto, non essendo sufficiente il confronto con qualche immagine di riferimento. È un metodo lento, è sufficiente una telecamera e in genere non è richiesto nemmeno un illuminatore apposito. È adatto alle applicazioni di guida robot per cui sia sufficiente una bassa precisione.

Tecnica di Triangolazione	Condizioni	Vantaggi	Svantaggi	Applicazioni tipiche
Laser	Movimento	Denso	No colori	Tutto (guida robot, controllo qualità)
Stereoscopia	Snap	Anche colori	Poco denso	Guida robot
Omografia	Snap	Molto veloce, generalmente no illuminatore	Tutta superficie su unico piano, non preciso	Inseguimento oggetti
Fringe	Statico	Preciso, una sola telecamera	Lento	Reverse engineering, misure
Viste multiple	Snap	Una sola telecamera, no illuminatore	Lento, necessità bordi e modello cad	Guida robot con bassa precisione

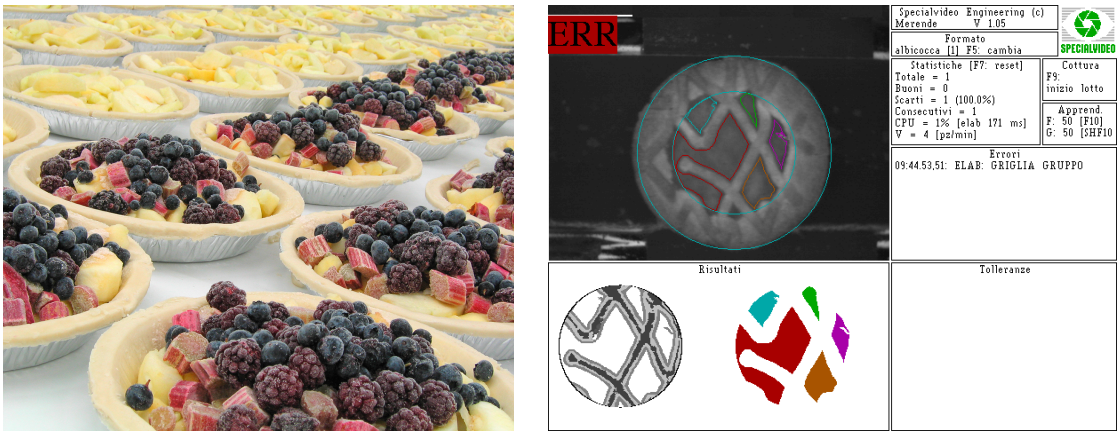
Tabella 3.1: Riassunto delle varie tecniche 3D.

3.1.2 Stato dell'arte

Tra le applicazioni in ambito industriale in cui la computer vision viene coinvolta negli ultimi anni citate nel capitolo 2.1.1 le più frequenti all'interno delle commesse di Specialvideo s.r.l. si concentrano in 4 ambiti principali: controllo qualità, guida robot, misure senza contatto, riconoscimento colore. Di seguito vedremo un esempio di commessa portata a termine con successo per ognuno dei 4 ambiti.

Controllo qualità: Crostatine

In questo caso la commessa prevedeva di garantire che le crostatine prodotte industrialmente oltre che buone fossero anche belle; Specialvideo ha realizzato un sistema di visione in grado di giudicare l'aspetto delle crostatine, volutamente mai uguali tra loro, per scartare quelle eccessivamente irregolari, come ad esempio quella in figura 3.5.
Il sistema di visione effettua il controllo qualità al 100% in linea di produzione, a monte dell'imballaggio in flow-pack, con una cadenza massima dei prodotti di 12000 pz/h, ed ha permesso di garantire, a differenza dell'ispezione visiva umana, oggettività, affidabilità e costanza nel tempo del controllo.
Il sistema è stato realizzato in modo da tenere conto della disomogeneità della produzione, tollerando le variazioni dovute all'aspetto "rustico" e scartando le irregolarità che rendono il prodotto poco invitante all'occhio del consumatore. Il controllo più complesso consiste nella verifica della completezza e della regolarità della griglia interna. Per la natura stessa del processo di produzione, le crostatine non sono mai perfettamente uguali tra loro, ed il concetto di "regolarità" è da tradursi in questo caso con un giudizio sulla "bellezza" del prodotto, concetto difficile da implementare in una macchina. Per questo scopo sono stati realizzati particolari algoritmi software in grado valutare la regolarità della griglia e quindi la bellezza del prodotto. Oltre



(a) Esempio di crostatina a cui è stato sottoposto il controllo qualità.

(b) Esempio di crostatina scartata perchè non conforme agli standard di qualità.

Figura 3.5: Controllo qualità su crostatine[5].

alla regolarità della griglia interna, il sistema verifica il corretto livello di cottura della crostatina e controlla che non vi siano briciole di pasta sulla confettura, tracce di confettura sulla pasta del bordo, punti neri o corpi estranei. Se la crostatina analizzata non è conforme alle specifiche qualitative impostate, il sistema di visione attiva l'apposito dispositivo di espulsione con due diverse procedure di scarto secondo il tipo di difetto.
La mancanza totale di confettura o crostatine non integre sono considerate come scarto per utilizzo zootecnico, mentre tutti gli altri difetti classificano il prodotto come seconda scelta.
La progettazione dei dispositivi di illuminazione e delle ottiche da impiegare è uno dei passi più importanti nella realizzazione di un sistema di visione, in quanto ottenere una buona immagine è fondamentale per costruire un buon sistema.
Una delle esigenze più stringenti di tutti i sistemi produttivi è la flessibilità, che richiede la veloce capacità di adattamento della linea a prodotti sempre diversi ed in costante evoluzione. L'inserimento di un nuovo prodotto avviene tramite una procedura di autoapprendimento: è sufficiente che l'operatore, durante questa fase, controlli che i primi prodotti analizzati non siano difettosi. Il sistema di visione apprende cosi' da solo i nuovi parametri su cui effettuare il controllo ed i limiti di accettabilità del nuovo prodotto. In particolare, il sistema autoapprende le caratteristiche geometriche della zona centrale delle crostatine "buone" e si costruisce una base di dati statistici basata sui prodotti analizzati durante la fase di auto apprendimento. Una volta appreso il nuovo formato, questo viene salvato in modo permanente sull'elaboratore ed associato ad un codice mnemonico. Il richiamo di un formato precedentemente memorizzato può essere fatto o in automatico tramite segnali di I/O o manualmente da tastiera. Dopo la fase di autoapprendimento, l'utente provvisto dell'apposita password ha a disposizione un parametro per aumentare o diminuire la selettività del sistema secondo le esigenze della produzione.

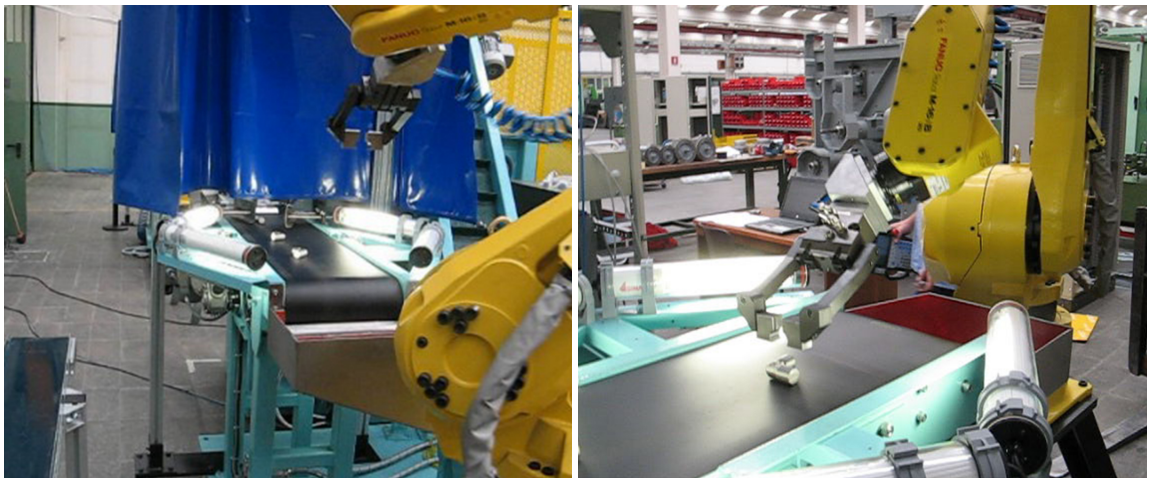


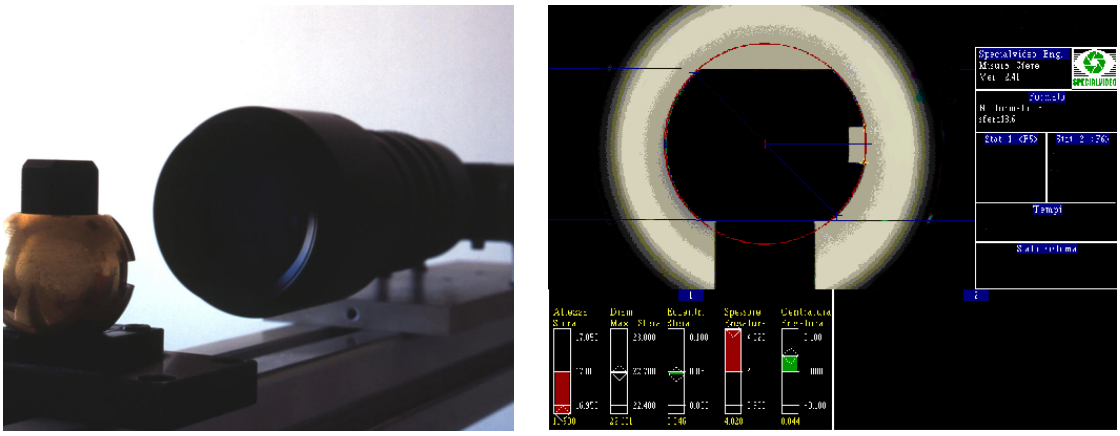
Figura 3.6: Sistema di guida robot per prelevare pezzi dal nastro trasportatore[5].

Guida robot: Rubinetti

La guida robot è un sistema di visione artificiale in grado di guidare il robot ad eseguire un'azione mediante l'uso di una telecamera. In ambito industriale i sistemi di guida robot forniscono l'abilità di vedere gli oggetti, quindi permettono di prelevare i pezzi in posizione casuale oppure di eseguire un percorso di lavorazione non predefinito. L'impiego di robot industriali nelle isole robotizzate consente di realizzare impianti di automazione estremamente versatili e con tempi di attrezzaggio ridotti. I sistemi di guida robot che realizza Specialvideo s.r.l. possono essere configurati per lavorare con pezzi fermi o in movimento, con pezzi a contatto fra loro e parzialmente occlusi. E' inoltre possibile corredarli di funzioni per il controllo di qualità, per rilevare la presenza di parti difettose e gestire di conseguenza lo scarto dei pezzi. Le applicazioni principali riguardano il prelievo di pezzi sul nastro di trasporto (2D), disposti in strati sui bancali (2D multistrato), o pezzi disposti completamente alla rinfusa dove è necessario l'impiego di tecniche tridimensionali (3D).
Specialvideo s.r.l. ha realizzato un sistema di visione per la guida robot nella produzione di componenti di rubinetti, in questo caso i pezzi vengono prelevati dal nastro trasportatore per procedere alle successive fasi della lavorazione, come vediamo in figura 3.6. Per questo tipo di applicazioni si è rivelato affidabile un tradizionale approccio 2D. Il posizionamento delle telecamere e l'illuminazione sono stati studiati appositamente per l'applicazione.

Misure senza contatto: Sfere di ottone

Per verificare tolleranze di lavorazione, Specialvideo ha realizzato un sistema di visione in grado di effettuare misure di elevata precisione di sfere in ottone.
La stazione di misura è posta subito a valle della lavorazione; le sfere sono posizionate da un robot comandato dal PLC della macchina. Il sistema di visione controlla, mediante speciali algoritmi sub-pixel, diverse quote, quali ad esempio l'altezza della sfera, lo spessore e la posizione della fresatura rispetto al centro "ideale" della sfera. La forma ideale del pezzo, in realtà, non è una sfera perfetta, ma è più simile ad un ellissoide di rotazione. E' stato quindi necessario realizzare particolari



(a) Obiettivo utilizzato per il sistema di visione.

(b) Esempio di videata visibile all'operatore.

Figura 3.7: Sistema senza contatto per la misura delle sfere di ottone[5].

algoritmi di interpolazione del bordo del pezzo per verificare che l'eccentricità sia sempre contenuta entro i limiti previsti dal costruttore. Con un sistema di visione è possibile effettuare anche misure fuori standard, non possibili con calibri o altri dispositivi manuali. In questo caso, ad esempio, viene calcolato il massimo diametro orizzontale, misura fisicamente non possibile per la presenza della fresatura. Oltre a comandare lo scarto dei pezzi difettosi, i dati delle misurazioni vengono elaborati sotto forma di media mobile, in modo da fornire indicazioni sulla degenerazione del processo produttivo ed intervenire quindi sulle cause che stanno pregiudicando la qualità del prodotto, prevenendo la creazione di pezzi difettosi.
La precisione raggiunta da questo sistema è di +/- 0.2 mm. Per raggiungere elevate precisioni può risultare critico anche il fattore temperatura. Il coefficiente di dilatazione termico dell'ottone è di circa 18 ppm/° e con sbalzi di alcune decide di gradi la dilatazione può diventare significativa. Il sistema di visione controlla infatti la sfera subito a valle del processo di tornitura, quando il pezzo è ancora abbastanza caldo a causa delle lavorazioni subite. Una speciale funzione di calibrazione permette di impostare i rapporti mm/pixel sui due assi; tale funzione è stata realizzata in modo da tenere conto anche della dilatazione termica.
Il cuore del sistema è costituito dal software, interamente progettato e realizzato presso i laboratori interni. Le funzioni e gli algoritmi di base sono prelevati da librerie frutto dell'esperienza specifica nel campo della visione artificiale, mentre le nuove funzioni, specifiche per questa applicazione, sono state realizzate appositamente in modo da fornire un prodotto "chiavi in mano". Gli altri dispositivi necessari ad ottenere le misure con le precisioni richieste sono:

- Telecamera, ad alta risoluzione e basso rumore elettrico;
- Obiettivo, di tipo telecentrico che permette di minimizzare gli errori dovuti all'effetto grandangolare dell'ottica;
- Illuminazione, uno speciale dispositivo di retro-illuminazione che permette di ottenere un ottimo contrasto tra il pezzo e lo sfondo, e di essere insensibile alla particolare superficie riflettente della sfera di ottone;

- Scheda acquisizione immagini, ad alta risoluzione e basso jitter;
- L'unità di elaborazione, costituito da un PC che fornisce la necessaria potenza di calcolo per l'elaborazione delle immagini.

Durante il controllo automatico, il sistema visualizza sul monitor l'immagine dell'ultima sfera controllata ed i risultati delle misure. Per rendere immediata ed intuitiva la lettura delle misurazioni, i risultati possono essere visualizzati in forma di trend o di carte X-R, come mostrato in figura 3.7b. In tal modo l'operatore può rilevare per tempo una lenta degenerazione del processo produttivo, ed intervenire quindi sulle cause che stanno pregiudicando la qualità del prodotto, prevenendo la creazione di pezzi difettosi. Il sistema è dotato di numerosi parametri di funzionamento, in modo da potersi adattare, entro certi limiti, a diversi tipi di prodotto e di condizioni di funzionamento. La misura dimensionale di un oggetto mediante la sua immagine può sembrare a prima vista equivalente al contare il numero di pixel attribuiti all'oggetto. Ma per le misure di precisione non è così perché:

- I bordi dell'oggetto sono normalmente sfumati, cioè coinvolgono diversi pixel (da 2 a 5 normalmente), quindi nell'immagine non esiste un punto "vero" di bordo;
- Determinare il punto di confine mediante una soglia di binarizzazione vincola il sistema alla stabilità dell'illuminazione stessa, perché la misura viene a dipendere dal livello di illuminazione (un oggetto scuro in campo chiaro "diventa" sempre più piccolo al crescere della luce di sfondo);
- Se la distanza è variabile, altrettanto variabile risulta la misura.

Le tecniche per avere misure indipendenti dal livello di illuminazione utilizzano l'analisi delle derivate o, analogamente, le trasformazioni prodotte da filtri opportuni. In altre parole si cerca nell'immagine il luogo più plausibile della transizione dall'oggetto allo sfondo, utilizzando tutti i pixel coinvolti nella transizione stessa. Il successivo importante passo è la conversione della misura da pixel a millimetri o, più in generale, dal sistema di riferimento della telecamera a quello dello spazio in cui si trova l'oggetto. In entrambi i casi, il modo usuale per calcolare i parametri della conversione da pixel a millimetri è tramite una funzione di calibrazione in cui si trovano quei parametri (per esempio il fattore di scala) che trasformano "al meglio" le misure da pixel a millimetri.

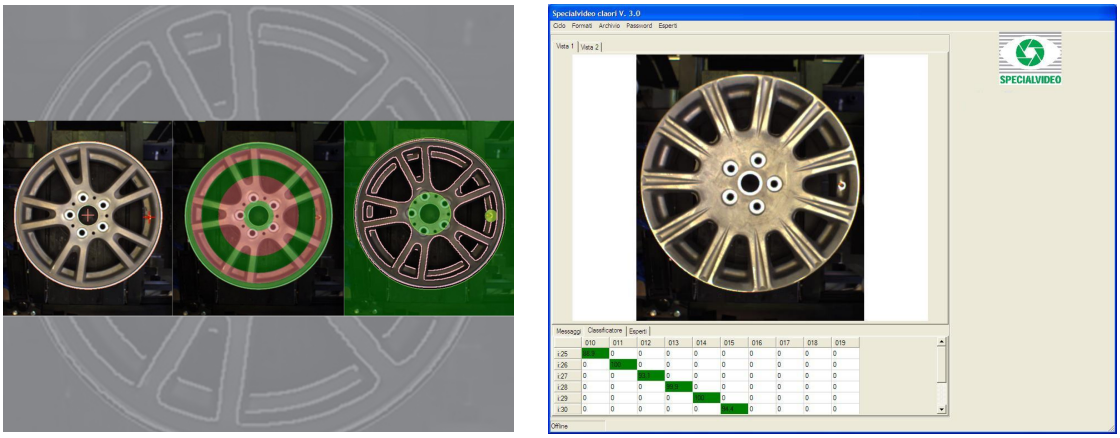
Riconoscimento colore: Classificazione ruote

Quando si richiede un controllo di qualità a colori, a differenza di altri metodi (quali spettrofotometri e colorimetri a distanza), i sistemi di visione permettono di eseguire il controllo al 100% della produzione, in maniera completamente automatica. I sistemi sviluppati da Specialvideo s.r.l. sono in grado di fornire informazioni sulla distribuzione spaziale dei colori su tutta la superficie del prodotto; ovviamente è un'operazione essenziale quando i prodotti da analizzare sono costituiti da zone di colori diversi, come nel caso di capsule bicolore, merendine con pasta e farcitura, e così via; vediamo un esempio di classificazione di ruote tramite il colore in figura 3.8 Inoltre, negli anni, nel campo del riconoscimento del colore, hanno messo a

punto speciali algoritmi che permettono di raggiungere elevate accuratezze nella determinazione dei colori, in tempi di analisi estremamente ridotti e con procedure di autoapprendimento robuste ed affidabili. Un esempio di utilizzo di queste tecniche è un sistema realizzato per la classificazione di ruote.
Questo sistema di visione, realizzato da Specialvideo s.r.l. per il settore automobilistico, è in grado di effettuare diversi controlli in un punto critico della linea di produzione delle ruote. Svariate tipologie di cerchioni vengono prodotte simultaneamente dai macchinari a monte della catena e raggiungono le macchine equilibratrici e la zona della finitura pneumatici. E' necessario distinguere tra i vari modelli di pneumatici in modo da eseguire correttamente le successive fasi della lavorazione. E' inoltre fondamentale individuare la posizione del foro valvola e verificare se esso è vuoto o se un oggetto è inserito al posto della valvola per simularne il peso. Il sistema di visione determina il tipo di ruota sulla base di altezza, diametro e offset ma, qualora si ricada in un caso dubbio, individua la tipologia specifica grazie ad un pattern matching geometrico. Il software comunica quindi il tipo di cerchione e l'orientamento alle macchine spazzolatrici ed equilibratrici. Conoscere la posizione e il tipo di riempimento (se presente) del foro valvola è necessario per poter effettuare correttamente la verifica e l'aggiustamento dell'equilibratura della ruota (simulando la valvola in caso non vi sia un riempimento fisico sostitutivo). Gli algoritmi sviluppati da Specialvideo s.r.l. per questa applicazione sono ideati per soluzioni generali e non specificatamente per alcune tipologie di cerchioni. Nella fase di memorizzazione è possibile far procedere il programma all'apprendimento di nuove tipologie di cerchioni. Se le ruote pre-memorizzate diverse dovessero essere un numero particolarmente elevato, è anche possibile restringere l'identificazione ai modelli effettivamente prodotti in un dato periodo.
I dati su altezza, diametro e offset del cerchione vengono calcolati dalle altre strumentazioni della linea e forniti al sistema di visione. Il calcolo dell'altezza è basato su una barriera di fotocellule Datalogic. Il pattern matching è invece calcolato su un'immagine catturata dalla telecamera del sistema di visione stesso. A seconda delle specifiche necessità il sistema è a colori o basato sui livelli di grigio, mentre l'illuminatore può essere a LED oppure costituito da flash.
I risultati possono venire comunicati al PLC dell'impianto oppure direttamente ad un robot. In generale questo sistema di visione può essere utilizzato sia per la verifica del prodotto finale che per gestire le fasi successive della lavorazione, come ad esempio il controllo dell'equilibratura o l'eliminazione di difetti tramite macchine spazzolatrici. L'appartenenza di una ruota ad un dato modello è basata su uno score di somiglianza, visibile nell'interfaccia e i cui parametri sono modificabili da un operatore esperto.

3.2 Introduzione al progetto

Il progetto nasce da uno studio di fattibilità richiesto da un cliente di Specialvideo s.r.l., in particolare un'azienda che si occupa di produzione industriale di prosciutti. La richiesta parte dal bisogno di accelerare e automatizzare il processo produttivo, introducendo un robot che svolga autonomamente la funzione di *"sugnatura"* dei prosciutti.
La **sugnatura** è una fase della lavorazione del prosciutto crudo, vediamo un esem-



(a) Sistema di classificazione a colori delle ruote.

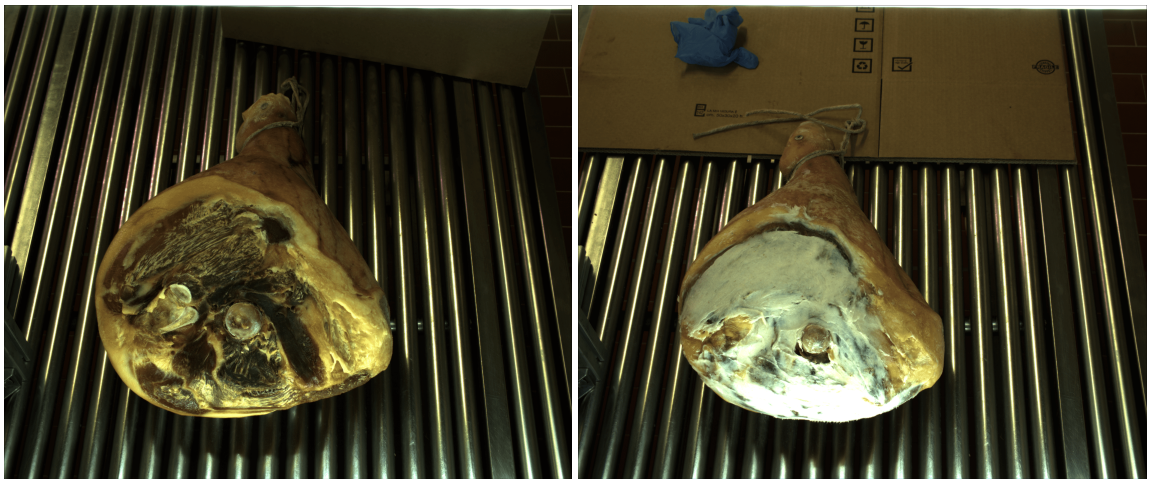
(b) Esempio di videata disponibile all'operatore.

Figura 3.8: Sistema di visione che sfrutta il riconoscimento colore per la classificazione[5].

pio in figura 3.9; dopo che le cosce sono state sottoposte a salatura e dopo essere state lasciate a riposare qualche mese, sulla parte non ricoperta dalla cotenna viene applicato a mano un sottile strato di sugna ovvero un impasto composto da grasso di maiale, sale, pepe ed eventuale farina di riso. Questo procedimento, che viene effettuato la prima volta tra il 4°/7° mese e che può essere ripetuto più volte nei mesi successivi, serve per proteggere la carne ed evitare che la parte esterna asciughi troppo rovinando così la stagionatura della parte interna. Inoltre fa da barriera contro gli agenti esterni come gli insetti. In seguito alla sugnatura i prosciutti vengono nuovamente rimessi a stagionare.
Il progetto parte da uno studio preliminare in laboratorio svolto sui campioni forniti dal cliente. Il frutto di una buona progettazione è dato dall'interazione con il cliente, in quanto il sistema viene costruito per rispondere alle sue specifiche esigenze e dovrà essere integrato nella linea produttiva. La soluzione di illuminazione viene pensata già dalle fasi iniziali, affinché le immagini così ottenute siano adatte per svolgere le funzioni richieste.
Il progetto consiste in due parti che sono state svolte distintamente per essere poi integrate alla fine:

- **Sistema stereo per guida robot:** per poter sviluppare un'applicazione di guida robot ovviamente abbiamo bisogno di ricostruire la scena 3D in cui il robot dovrà "agire", ciò è possibile grazie ai sistemi di telecamere stereo citati nel capitolo 3.1.1, in particolare troviamo un esempio in figura 3.4. Questa prima parte sarà spiegata dettagliatamente nel capitolo 4;

- **Rete neurale per segmentazione immagini:** il robot ha anche bisogno di sapere quali sono le zone dell'immagine sulle quali deve svolgere un'operazione; per fare ciò, dopo uno studio di fattibilità spiegato dettagliatamente nel capitolo 5, abbiamo deciso di utilizzare una rete neurale che svolga la funzione di segmentazione automatica delle immagini. In questo modo diventa autonoma l'operazione di selezione della zona da "sugnare".



(a) Immagine di un prosciutto non *"sugnato"*.

(b) Immagine di un prosciutto già *"sugnato"*.

Figura 3.9: Immagini originali del processo di lavorazione.

Per quanto riguarda il ciclo del progetto abbiamo seguito una procedura standard e ampiamente utilizzata in tutte le commesse di Specialvideo s.r.l., composta dai seguenti 4 passaggi.

1 - Analisi di fattibilità

La prima fase prevede un'analisi di fattibilità, il cui scopo è valutare in modo realistico la possibilità di sviluppare un sistema di visione che esegua le funzioni richieste dal cliente con prestazioni adeguate, con particolare riguardo alle percentuali di successo e di falsi positivi. Assieme al cliente si analizzano le varie soluzioni, proponendo quelle più vantaggiose dal punto di vista dell'impianto nel suo complesso. Al termine dell'analisi, e senza alcun impegno da parte del cliente, viene formulata una proposta tecnico economica.

2 - Progettazione e implementazione

La progettazione del sistema prosegue in laboratorio, sulla base di ulteriori campioni forniti dal cliente. Lo scopo principale è quello di ottenere una buona immagine, illuminando i pezzi nel modo più opportuno e seguendo le indicazioni ottenute durante l'analisi di fattibilità. Si simulano inoltre le condizioni operative della linea tenendo conto dei vincoli esistenti: spazi disponibili, movimento del nastro trasportatore, posizione della telecamera e dell'illuminatore, ecc.. Si passa poi alla scelta degli algoritmi più adatti, attingendo, a seconda dell'opportunità, da una vasta libreria proprietaria sviluppata nel corso della lunga esperienza in questo settore e da consolidate librerie fornite da terzi, oppure sviluppando algoritmi su misura. Naturalmente, durante la scelta degli algoritmi, si tiene conto di quelle che dovranno essere le caratteristiche del controllo richiesto e la velocità di elaborazione. Al termine di questa fase, il lavoro di scelta dei componenti hardware è concluso e la programmazione del software è pressoché ultimata. Rimangono solo alcuni dettagli che dovranno essere messi a punto direttamente presso il cliente.

3 - Messa in opera e collaudo

Il sistema viene assemblato sulla linea produttiva ed interfacciato con gli altri componenti come ad esempio PLC e robot. Si procede poi a calibrare il sistema mettendo a punto i parametri e le soglie di accettazione per il collaudo finale. Questo viene svolto alla presenza dei tecnici del committente, assicurandosi che il sistema svolga le funzioni richieste e con le dovute prestazioni. Superato con successo il collaudo, il sistema viene affidato al cliente, mentre i servizi offerti da Specialvideo s.r.l. proseguono attraverso la garanzia e l'assistenza post vendita.

4 - Assistenza tecnica post vendita

Concluso il collaudo, Specialvideo continua ad offrire assistenza tecnica ai propri clienti. Tra i servizi inclusi nella fornitura ci sono: la garanzia sui componenti e sul funzionamento del sistema di visione e le attività di assistenza remota fino a 12 mesi dal collaudo. Grazie alla assistenza remota è possibile diagnosticare e correggere eventuali errori in tempi rapidi e senza costi aggiuntivi per il cliente. Attraverso una connessione internet protetta infatti, i tecnici Specialvideo sono in grado di: visualizzare le immagini dei pezzi in produzione, intervenire sui parametri di controllo e le soglie di accettazione, aggiornare il software. Terminato il periodo di garanzia sono disponibili i servizi di manutenzione programmata e assistenza in loco e, naturalmente, le estensioni sul periodo di assistenza remota.

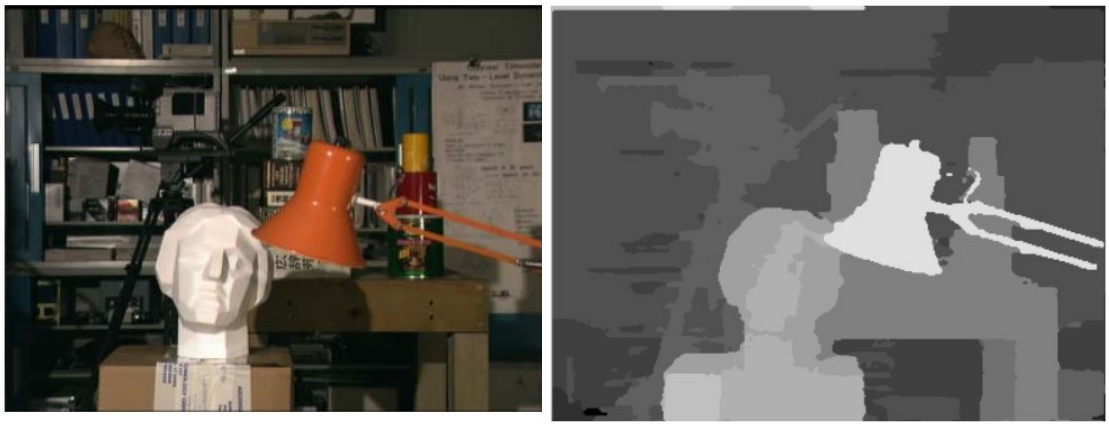
Per quanto concerne il progetto che stiamo trattando, nella prima fase ho affiancato il team di Specialvideo che mi ha guidato nell'analisi del problema e nella creazione di un piano di lavoro, inoltre ho seguito insieme a loro anche la fase di progettazione e di scelta di algoritmi e di tecnologie. Ho svolto, invece in autonomia, la parte di implementazione delle due fasi del progetto spiegate nel paragrafo precedente, confrontandomi e scambiando feedback con il team, nel momento in cui veniva raggiunto uno step posto come obiettivo. La parte di messa in opera e collaudo non è stata possibile nel momento in cui scrivo, ma probabilmente la seguirò successivamente al lavoro di tesi.

Capitolo 4

Prima parte: estrazione di informazioni 3D da un'immagine

4.1 Analisi del problema

Questa prima parte del progetto tratta la procedura di ricostruzione della scena in cui il robot dovrà svolgere operazioni mediante mappa di disparità, ovvero una immagine della scena che ci fa capire la profondità alla quale si trovano gli oggetti rispetto a un piano di origine definito durante la calibrazione del sistema stereo; ne vediamo un esempio in figura 4.1.



(a) Immagine della scena.

(b) Mappa di disparità.

Figura 4.1: Nella mappa di disparità vediamo che in base alla gradazione di grigio viene indicata la profondità alla quale si trova l'oggetto, grigio chiaro per gli oggetti più vicini e grigio più scuro per quelli più lontani.

Il processo si divide in 3 fasi principali:

- Installazione di un sistema stereo nella catena di produzione dei prosciutti e acquisizione delle immagini di calibrazione e di qualche immagine di esempio;
- Utilizzo di un pattern planare (scacchiera) per effettuare la rettificazione e la calibrazione delle immagini;

- Tramite un software proprietario di Specialvideo che sfrutta librerie di openCV ho poi eseguito il calcolo delle corrispondenze stereo e ho creato la mappa di disparità.

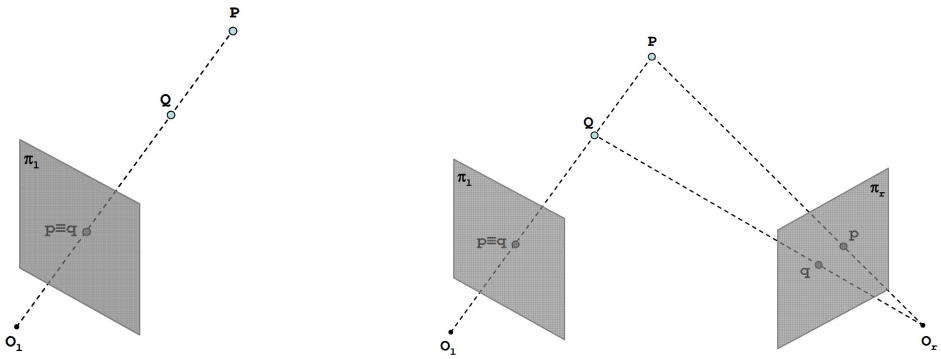
4.2 Sistema di telecamere stereo

La tecnologia che si è deciso di utilizzare per la guida robot in questo progetto è la visione stereo che abbiamo osservato in figura 3.4, la stessa funzione (ricostruzione della scena 3D) sarebbe stata possibile anche grazie a scansione laser (anche molto più precisamente), ma in questo caso la facilità di installazione e il contesto produttivo hanno fatto pendere per la prima opzione.

4.2.1 Che cosa è un sistema di visione stereo?

La trasformazione prospettica che mappa un punto dello spazio nel piano immagine di una telecamera implica la perdita dell'informazione relativa alla distanza. Questo può essere facilmente notato osservando la figura 4.2a nella quale due punti distinti (e.g. P e Q) nello spazio intersecati dalla stessa retta che parte dal centro ottico O_l della telecamera corrispondono allo stesso punto nel piano immagine.
Un metodo per poter risalire a quale punto dello spazio corrisponda la proiezione di un punto sul piano immagine di una telecamera consiste nell'utilizzo di due o più telecamere. Infatti, come mostrato nella figura 4.2b nel caso di un sistema composto da due telecamere, tra tutti punti nello spazio che giacciono sulla retta che passa per il centro ottico O_l e il punto q, proiezione di Q sul piano immagine π_l, al più un solo punto (punto omologo) viene proiettato anche sul piano immagine π_r. La determinazione dei punti omologhi consente di mettere in relazione le proiezioni dello stesso punto sui due piani immagini e di risalire, mediante una procedura denominata **triangolazione**, alle coordinate dei punti dello spazio rispetto ad un sistema di riferimento opportuno. Sia dato un punto q su un piano immagine π_l, proiezione del punto Q appartenente allo spazio 3D. Per ottenere, attraverso la triangolazione, le coordinate 3D del punto nello spazio è necessario determinare (**problema delle corrispondenze**) il punto omologo q' nel piano immagine π_r.
Tale problema, dato il punto q nel piano immagine π_l, richiederebbe una ricerca bidimensionale del punto omologo q' all'interno del piano immagine π_r.

In realtà, sfruttando una particolare caratteristica della geometria del sistema stereoscopico, è possibile effettuare la ricerca del punto omologo in uno spazio monodimensionale. Infatti, come mostrato nella figura 4.3, gli omologhi di tutti i punti dello spazio che potrebbero risultare proiezione nello stesso punto q del piano immagine π_l, (e.g. punto p proiezione di P o lo stesso punto q proiezione di Q) giacciono sulla retta generata dall'intersezione tra il piano immagine π_r e il piano (denominato piano epipolare) che passa per la retta $O_l - Q - P$ e i due centri ottici O_l e O_r. Tale vincolo, denominato vincolo epipolare, consente di limitare lo spazio di ricerca dei punti omologhi a un segmento di retta semplificando considerevolmente il problema delle corrispondenze sia da un punto di vista della complessità algoritmica sia per quanto concerne la correttezza della soluzione. Si fa notare che il problema delle corrispondenze non necessariamente ha soluzione: infatti a causa della diversa posizione delle telecamere che compongono un sistema di visione stereoscopico nello



(a) Proiezione di due distinti punti nello spazio sul piano immagine.

(b) Sistema stereoscopico.

Figura 4.2: Proiezione prospettica di due punti dello spazio nel piano immagine.

spazio è possibile che un punto non risulti proiettato su tutti i piani immagini delle telecamere. In tal caso il problema delle corrispondenze non ha soluzione e non è possibile determinare la distanza del punto esaminato dalle telecamere (occlusioni). Un sistema di visione stereoscopico è completamente caratterizzato dai **parametri intrinseci** ed **estrinseci**. I primi consentono di definire la trasformazione che mappa un punto dello spazio 3D nelle coordinate del piano immagine di ogni telecamera, ovvero le coordinate relative al piano immagine del principal point (punto di intersezione tra il piano immagine e la retta ortogonale al piano immagine stesso passante per il centro ottico), la distanza focale, ed eventualmente altri parametri che descrivono altre caratteristiche del sensore (distorsione delle lenti, forma dei pixels, etc). I parametri estrinseci rappresentano le posizioni di ogni telecamera rispetto ad una sistema di riferimento noto (trasformazioni rigide). La determinazione dei parametri intrinseci ed estrinseci, ottenuta mediante la procedura di **calibrazione**, consente quindi di descrivere completamente il sistema stereoscopico e in particolare di inferire informazioni relative alle coordinate dei punti nello spazio mediante la triangolazione di punti omologhi.

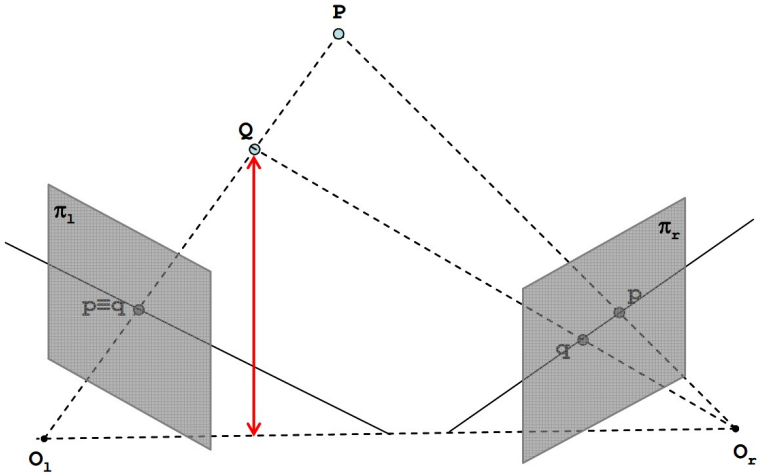


Figura 4.3: Sistema stereoscopico: vincolo epipolare.

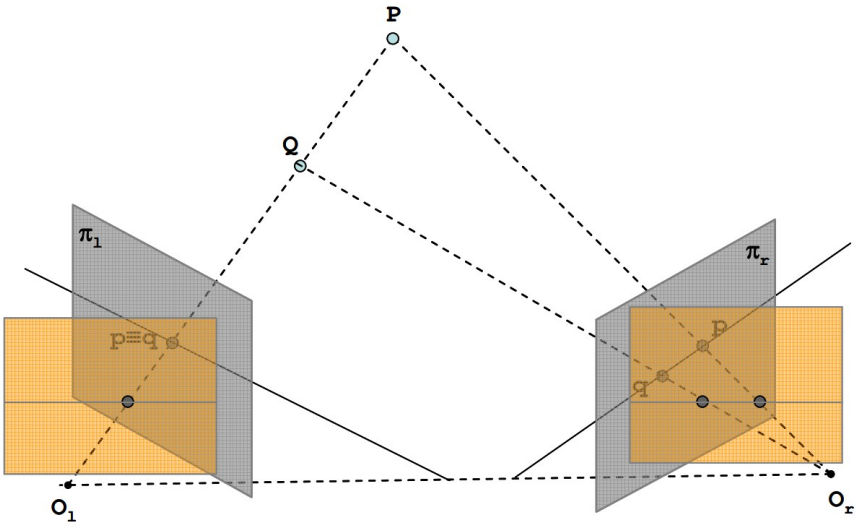


Figura 4.4: Immagini rettificate (i.e. geometria stereo standard).

La conoscenza dei parametri intrinseci ed estrinseci consente anche di trasformare le immagini acquisite dal sistema stereoscopico al fine di produrre un sistema virtuale nel quale i piani immagine delle telecamere giacciono sullo stesso piano e nel quale la ricerca dei punti omologhi avviene esaminando le medesime righe nei diversi piani immagine. Tale configurazione del sistema stereoscopico ottenuta mediante una procedura denominata **rettificazione** è mostrata in seguito. Osservando la figura 4.4 si può notare come il sistema risultante dopo la rettificazione risulti composto da due piani immagine virtuali θ_l e θ_r giacenti sullo stesso piano. Le immagini stereoscopiche ottenute da un sistema rettificato sono denominate immagini in forma standard. Si può osservare infine che nelle immagini in forma standard i piani xy dei sistemi di riferimento centrati nei centri ottici delle due telecamere sono complanari.

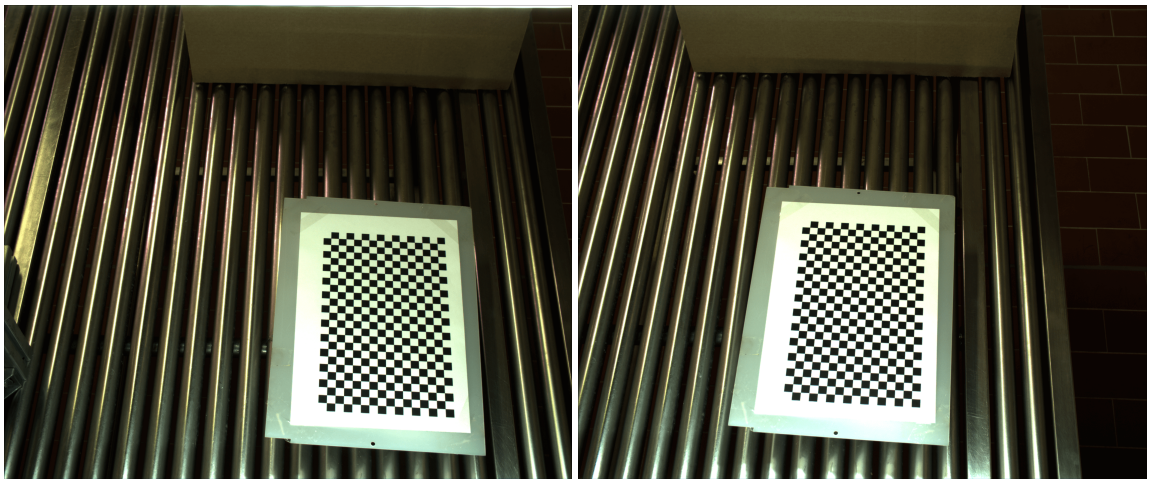
Da un punto di vista operativo i passi necessari per risalire alle coordinate dei punti dello spazio mediante un sistema di visione sono i seguenti:

- ***Calibrazione;***
- ***Rettificazione;***
- ***Ricerca delle corrispondenze (matching);***
- ***Triangolazione.***

Di seguito vedremo la teoria di questi passaggi e la loro applicazione nel progetto di cui questa tesi.

4.2.2 Calibrazione

La calibrazione è una procedura eseguita offline ed è mirata all'individuazione dei parametri che caratterizzano un sistema di visione stereoscopico. Tali parametri, denominati **parametri intrinseci** ed **estrinseci**, sono utilizzati dalla procedura di *rettificazione* per trasformare le immagini acquisite dal sistema stereoscopico, in modo da ottenere immagini stereoscopiche in una forma particolare (forma standard)



(a) Immagine dalla telecamera di sinistra. **(b)** Immagine dalla telecamera di destra.

Figura 4.5: Immagini della scacchiera utilizzata per la calibrazione del sistema stereo installato nella catena produttiva.

e per ottenere le coordinate 3D dei punti mediante la procedura di *triangolazione*. Esistono in letteratura diverse tecniche per effettuare la calibrazione di un sistema stereoscopico. Tipicamente però questa operazione viene eseguita con tecniche basate sull'utilizzo di pattern geometrici dei quali sono note con precisione le caratteristiche (dimensione e posizione delle features presenti nel pattern, etc). Mediante l'acquisizione di tali pattern (generalmente contenenti features simili a scacchiere) in diverse posizioni e utilizzando procedure ampiamente note è possibile stimare i parametri intrinseci ed estrinseci che caratterizzano il sistema stereoscopico.

Per quanto riguarda il progetto che stiamo trattando la calibrazione del sistema stereo si è basata sull'utilizzo di un pattern planare raffigurante una scacchiera, come vediamo in figura 4.5, e delle librerie openCV per il calcolo dei parametri del sistema.
I dati di input importanti necessari per la calibrazione della telecamera sono l'insieme di punti 3D del mondo reale e le corrispondenti coordinate 2D di questi punti nell'immagine. I punti immagine 2D devono essere facilmente reperibili dall'immagine, per questo di solito per la calibrazione si utilizzano dei pattern, come la scacchiera, dove abbiamo dei punti noti (come ad esempio i punti dove si incontrano i quadrati neri).
E i punti 3D dallo spazio del mondo reale? Le immagini sono prese da una telecamera statica e le scacchiere sono posizionate in posizioni e orientamenti diversi. Quindi dobbiamo conoscere i valori (x, y, z). Per semplicità, possiamo dire che la scacchiera è stata tenuta ferma sul piano XY, (quindi abbiamo sempre $z = 0$) e la fotocamera è stata spostata di conseguenza. I valori x, y, possiamo semplicemente esprimerli come (0,0), (1,0), (2,0), ... in modo da denotare la loro posizione. In questo caso, i risultati che otterremo saranno in scala rispetto alle dimensioni del quadrato della scacchiera, ma se conosciamo la dimensione del quadrato (diciamo 30 mm) possiamo esprimere i valori come (0,0), (30,0), (60,0), ... Quindi, otteniamo i risultati in mm.

L'operazione di calibrazione di un sistema stereo tramite le librerie di openCV

è un'operazione abbastanza standard, per cui non vedremo nel dettaglio il procedimento, basti sapere che si basa su ciò che abbiamo detto poco fa, ovvero riuscire a esprimere i punti immagine che conosciamo, grazie al pattern, in coordinate per poi poterli passare ad una funzione della libreria la quale ricostruirà tramite un algoritmo la matrice della telecamera, i coefficienti di distorsione, i vettori di rotazione e traslazione ecc [6].

4.2.3 Rettificazione

La rettificazione è una procedura, che sfruttando i parametri intrinseci ed estrinseci ottenuti mediante calibrazione, mira a trasformare le immagini stereoscopiche provenienti dal dispositivo di acquisizione in modo che risultino soddisfatti alcuni vincoli. Tra questi, la rettificazione consente di trasformare le immagini in forma standard: nel caso di sistemi binoculari questo assicura che dato un punto in un'immagine il suo omologo possa essere rintracciato sulla stessa riga dell'altra immagine consentendo una notevole riduzione dei calcoli ed una maggiore affidabilità nella soluzione del problema delle corrispondenze. Mediante la rettificazione è possibile trasformare le immagini in modo che i punti omologhi di una riga (scanline) dell'immagine possano essere ricercati nella corrispondente riga dell'altra immagine.
Le rettificazione è eseguita ad ogni acquisizione di una coppia di immagini stereo ed è fondamentale per poter eseguire la ricerca delle corrispondenze (matching stereo), il passo successivo per individuare la mappa di disparità.
Ovviamente la procedura descritta in figura 4.6 di standardizzazione del sistema stereo è solo teorica perchè quando si lavora con sistemi stereo, è normale fare l'assunzione di lavorare con un sistema stereo laterale, è impossibile ottenere questa configurazione tramite allineamento meccanico. Se si è calibrata la coppia stereo è però possibile rettificare via software le immagini acquisite, vediamo il risultato in figura 4.7.

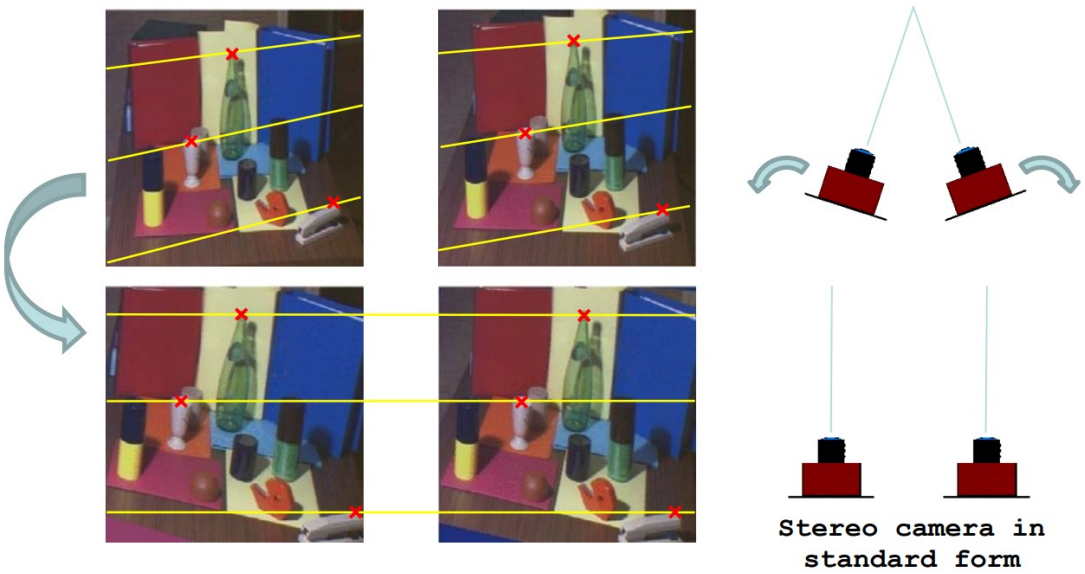


Figura 4.6: Procedura di rettificazione.

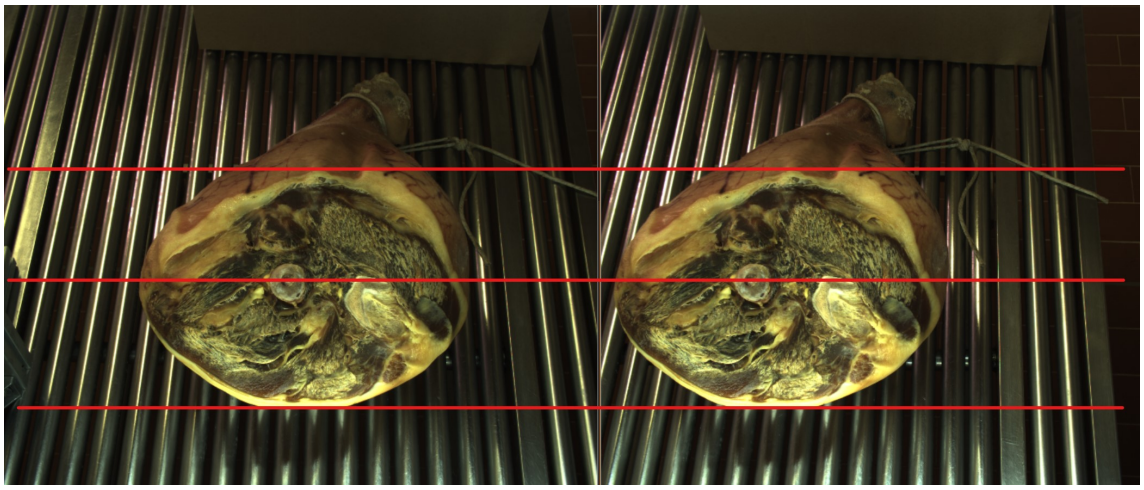


Figura 4.7: Immagini dei prosciutti ottenute dopo la procedura di rettificazione.

4.2.4 Matching stereo

Partiamo dal definire cosa si intende per *disparità* [7], prendiamo ad esempio il sistema stereo rappresentato in figura 4.8 con i piani immagine complanari, in questo caso i sistemi di riferimento solidali con le due camere differiscono solo per una traslazione orizzontale denotata come b. Definiamo x_l x_r, y_l y_r e z_l z_r come le coordinate rispettivamente del sistema di riferimento sinistro (left) e destro (right), possiamo quindi scrivere:

$$\begin{aligned} x_L - x_R &= b \\ y_L = y_R &= y \\ z_L = z_R &= z \end{aligned} \tag{4.1}$$

Segue la proiezione prospettica nel piano immagine:

$$\begin{aligned} v_L = v_R &= y \cdot f/z \\ u_L &= x_L \cdot f/z \\ u_R &= x_R \cdot f/z \end{aligned} \tag{4.2}$$

Dove v_l v_r e u_l u_r sono le coordinate dei punti p_l e p_r sui rispettivi piani immagine. Esprimiamo ora la traslazione orizzontale b tramite le coordinate immagine:

$$\begin{aligned} u_L - u_R &= b \cdot f/z \\ u_L - u_R &= d \end{aligned} \tag{4.3}$$

Possiamo quindi definire la quantità ottenuta come *disparità* $d = b \cdot f/z$, la quale è fortemente legata alla coordinata z che si ricava in questo modo:

$$z = b \cdot f/d \tag{4.4}$$

Facile intuire che la coordinata z è proprio ciò che stiamo cercando di ottenere perchè ci consentirà di individuare nello spazio 3D il punto P della scena.

E' possibile in prima istanza classificare gli algoritmi di matching stereo in due categorie principali: algoritmi di tipo **feature-based** e algoritmi di tipo denso (**dense stereo**)[8]. I primi consentono di ottenere informazioni di disparità (denominate mappe di disparità) per un numero limitato di punti delle immagini per

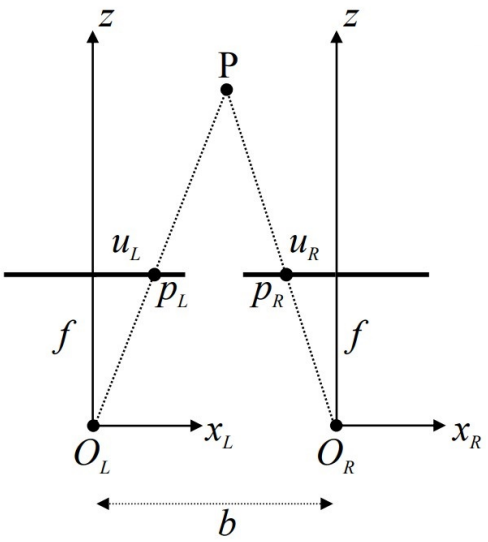


Figura 4.8: Esempio di sistema stereo visto dall'alto.

i quali sono identificate features particolari come linee, segmenti, angoli. Tali algoritmi sono efficienti dal punto di vista computazionale, per via del ridotto numero di features identificabili nelle immagini rispetto al numero totale di punti, ed anche estremamente affidabili perché le features estratte dalle immagini generalmente sono intrinsecamente distintive e non pongono problemi di ambiguità nella soluzione del problema delle corrispondenze. Tali algoritmi però sono in questo momento poco utilizzati perché le mappe di disparità sono limitate ai soli punti che presentano features distintive. Contrariamente gli algoritmi di tipo denso mirano a generare mappe di disparità per ogni punto dell'immagine ed è possibile classificare tali algoritmi in due principali categorie: algoritmi di tipo **locale** (local) e algoritmi di tipo **globale** (global)[8]. Gli algoritmi di tipo locale cercano i punti omologhi singolarmente e indipendentemente gli uni dagli altri sfruttando generalmente un supporto locale (local support) nell'intorno di ogni punto esaminato (tale area nell'intorno di ogni punto è denominata anche finestra di correlazione) allo scopo di aumentare il rapporto segnale/rumore. Utilizzando un supporto locale tali algoritmi implicitamente assumono che all'interno del supporto locale la disparità sia costante, anche se tale ipotesi non essendo sempre verificata, conduce a una discrepanza tra gli oggetti della scena e le mappe di disparità in prossimità di punti posti a differente distanza dalle telecamere. E' importante osservare che esistono anche algoritmi locali che adattano il proprio supporto in base alle caratteristiche locali delle immagini mediante le quali è possibile ottenere eccellenti risultati ad un costo computazionale tipicamente molto più alto rispetto agli algoritmi locali standard. Oltre all'assunzione sulla costanza della disparità all'interno della finestra di correlazione gli algoritmi di tipo locale hanno un limite legato al fatto che non necessariamente sono in grado di produrre mappe di disparità completamente dense (e.g. non è garantito che per ogni punto dell'immagine sia possibile stabilire un valore di disparità). Questo avviene principalmente per due motivi: a causa di *regioni uniformi* e a causa di *occlusioni*. Nel primo caso non è possibile determinare informazioni di disparità in regioni che presentano intensità costante o comunque pochi elementi distintivi a

causa dell'elevata ambiguità nella soluzione del problema delle corrispondenze. Nel secondo caso, a causa della geometria del sistema stereoscopico, in presenza di punti adiacenti posti a differente distanza dalle telecamere risulta che alcuni punti siano visibili in una immagine ma non nell'altra dello stereo-pair: in casi come questo non è possibile risolvere il problema delle corrispondenze. Diversamente dagli algoritmi di tipo locale gli algoritmi densi di tipo globale non utilizzano tipicamente alcun supporto locale ma, formulando alcune ipotesi sulla natura degli oggetti (smoothness assumption), ricercano un assegnamento di disparità tale che venga minimizzata una funzione costo globale (global cost function). La maggior differenza tra tali algoritmi è proprio nel metodo con il quale viene minimizzata la funzione costo. Tali algoritmi, in particolare quelli basati su graph-cuts consentono di ottenere mappe di disparità decisamente migliori rispetto agli algoritmi di tipo local, ma ad un costo computazione estremamente elevato che li rende attualmente non utilizzabili in applicazioni realtime. Inoltre, anche se alcuni algoritmi sfruttando il metodo della minimizzazione della funzione di costo sono in grado di risolvere il problema delle occlusioni, le occlusioni analogamente agli algoritmi di tipo local possono causare mappe di disparità non completamente dense. E' importante osservare che esistono anche algoritmi di tipo locale o semi-globale che utilizzano una funzione costo non puntuale ma basata su supporto variabile mediante i quali si ottengono eccellenti risultati.

Per quanto riguarda il progetto in questione è stato utilizzato un algoritmo standard dalla libreria openCV tramite la funzione stereoSGBM(). Il metodo **Semi-Global Matching (SGBM)** si basa sull'idea di corrispondenza pixel per pixel di mutua informazione e approssimazione di un vincolo di smoothing 2D globale combinando molti vincoli 1D. L'algoritmo è descritto in vari passaggi, alcuni di loro sono opzionali, a seconda dell'applicazione [9].

Calcolo dei costi di corrispondenza tra pixel

Nel caso dell'algoritmo utilizzato dalla libreria openCV ci sono alcune differenze con il SGM originale, tra cui il calcolo dei costi di corrispondenza tra pixel [10], infatti nel nostro caso non si basa sulla mutua informazione pixel per pixel, ma su una misura della dissimilarity dei pixel insensibile al campionamento, perché utilizza le funzioni di intensità dei pixel dell'intorno interpolate linearmente [11]. Quando un punto della scena viene ripreso da una coppia di telecamere stereo, i valori di intensità dei pixel corrispondenti sono in generale differenti. Molti fattori contribuiscono a questa differenza, come il fatto che la luce riflessa dal punto non è la stessa nelle due direzioni, le due fotocamere hanno guadagni e bias diversi, le intensità dei pixel sono quantizzate e la fotocamera è soggetta a rumore. Supponiamo di avere una coppia di telecamere stereo rettificate. Lungo due linee di scansione corrispondenti, siano i_L e i_R le funzioni di intensità continua unidimensionale che risultano dalla convoluzione della quantità di luce incidente sui due sensori con una funzione box il cui supporto è uguale alla larghezza di un pixel. Questa convoluzione è dovuta al fatto che un sensore di immagine reale può essere modellato come integrazione di intensità su ogni pixel seguito da un campionatore ideale, quindi per consentire di concentrarci sul campionamento ideale, rimuoviamo l'integrazione fin dall'inizio. Le funzioni i_L e i_R vengono campionate in punti discreti dal campionatore ideale del sensore, risultando in due matrici discrete unidimensionali di valori di intensità I_L e

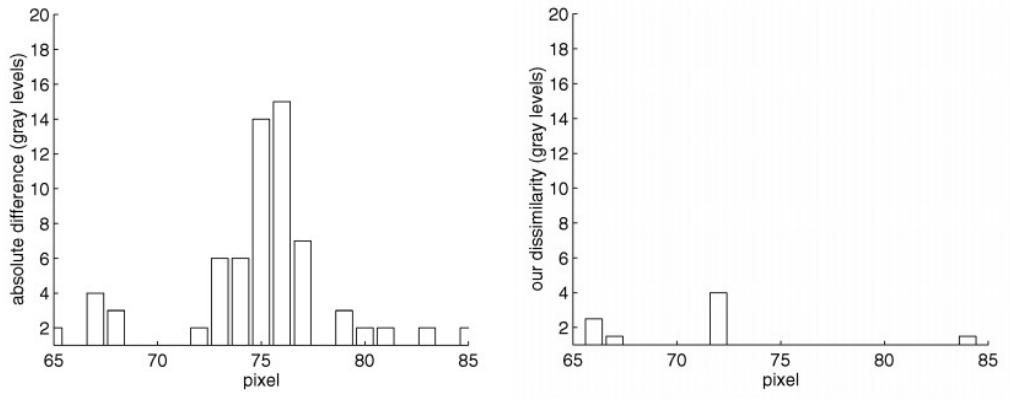


Figura 4.9: La differenza assoluta di intensità (a sinistra) rispetto alla nostra dissimilarity (a destra).

I_R, come mostrato in figura 4.9. Il nostro obiettivo è calcolare la dissimilarity tra un pixel nella posizione x_L nella retta epipolare di sinistra e un pixel nella posizione x_R nella retta epipolare di destra; gli altri pixel mostrati in figura 4.9 sono adiacenti a questi due. Per prima cosa, definiamo $\hat{I}_R$ come la funzione interpolata linearmente tra i punti campione della retta epipolare destra, quindi misuriamo quanto l'intensità in x_L si adatta alla regione interpolata linearmente che circonda x_R. Definiamo la seguente quantità:

$$\bar{d}\left(x_L, x_R, I_L, I_R\right) = \min_{x_R - \frac{1}{2} \leq x \leq x_{R+\frac{1}{2}}} \left| I_L\left(x_L\right) - \hat{I}_R(x) \right| \tag{4.5}$$

Definendo $\hat{I}_L$ allo stesso modo, otteniamo una quantità simmetrica:

$$\bar{d}\left(x_R, x_L, I_R, I_L\right) = \min_{x_L - \frac{1}{2} \leq x \leq x_{L+\frac{1}{2}}} \left| \hat{I}_L(x) - I_R\left(x_R\right) \right| \tag{4.6}$$

La dissimilarity d tra i pixel è definita simmetricamente come il minimo delle due quantità:

$$d\left(x_L, x_R\right) = \min\left\{\bar{d}\left(x_L, x_R, I_L, I_R\right), \bar{d}\left(x_R, x_L, I_R, I_L\right)\right\} \tag{4.7}$$

Poiché i punti estremi di una funzione lineare a "tratti" devono essere i suoi punti di interruzione, il calcolo di d è semplice. Per prima cosa calcoliamo l'intensità interpolata linearmente a metà tra x_R e il suo intorno sinistro:

$$I_R^- \equiv \hat{I}_R\left(x_R - \frac{1}{2}\right) = \frac{1}{2}\left(I_R\left(x_R\right) + I_R\left(x_R - 1\right)\right) \tag{4.8}$$

E la quantità analoga:

$$I_R^+ \equiv \hat{I}_R\left(x_R + \frac{1}{2}\right) = \frac{1}{2}\left(I_R\left(x_R\right) + I_R\left(x_R + 1\right)\right) \tag{4.9}$$

A questo punto definiamo $I_{\min} = \min\left\{I_R^-, I_R^+, I_R\left(x_R\right)\right\}$
e $I_{\max} = \max\left\{I_R^-, I_R^+, I_R\left(x_R\right)\right\}$, per cui:

$$\bar{d}\left(x_L, x_R, I_L, I_R\right) = \max\left\{0, I_L\left(x_L\right) - I_{\max}, I_{\min} - I_L\left(x_L\right)\right\} \tag{4.10}$$

Questo calcolo, insieme alla sua controparte simmetrica $\bar{d}\,(x_L, x_R, I_L, I_R)$, richiede solo una piccola quantità di tempo costante in più rispetto alla differenza assoluta di intensità. In pratica, abbiamo capito che il tempo di calcolo totale di questo algoritmo stereo aumenta di meno del 10% [11].

Aggregazione dei costi

Il calcolo dei costi per pixel è generalmente ambiguo e le corrispondenze sbagliate possono facilmente avere un costo inferiore a quelle corrette, dovuto al rumore, ecc. Pertanto, viene aggiunto un ulteriore vincolo per supportare l'uniformità penalizzando i cambiamenti delle disparità nell'intorno. Il costo per pixel e i vincoli di uniformità sono espressi definendo l'energia $E(D)$ che dipende dall' immagine di disparità D [9]. Il problema è che a questo punto non abbiamo ancora calcolato l'immagine di disparità; Kim et al. [12] suggeriva una soluzione iterativa, ovvero iniziare il calcolo dei costi con una immagine di disparità casuale, questo costo viene quindi utilizzato per abbinare entrambe le immagini e calcolare una nuova immagine di disparità, che funge da base per la successiva iterazione. Il numero di iterazioni è piuttosto basso (ad esempio 3), perché anche le immagini di disparità sbagliate (es. casuali) consentono una buona stima della probabilità di distribuzione P, a causa dell'elevato numero di pixel. Questa soluzione è adatta per algoritmi stereo iterativi come Graph Cuts, ma aumenterebbe il tempo di esecuzione degli algoritmi non iterativi inutilmente. Poiché una stima approssimativa della disparità iniziale è sufficiente per stimare P, potrebbe essere utilizzato un metodo di correlazione veloce nelle prime iterazioni, in modo che verrà eseguita solo l'ultima iterazione con un metodo più accurato e che richiede tempo; tuttavia, questo comporterebbe l'implementazione di due diversi metodi stereo. L'utilizzo di un unico metodo sembra più elegante. Pertanto, viene suggerito un calcolo gerarchico, che usa in modo ricorsivo l'immagine di disparità (ingrandita), che è stata calcolata a metà risoluzione, come disparità iniziale. Se la generale complessità dell'algoritmo è $O(WHD)$ (cioè, larghezza $\times$ altezza $\times$ intervallo di disparità), allora il tempo di esecuzione a metà risoluzione viene ridotto di un fattore $2^3 = 8$. A partire da un'immagine di disparità casuale a una risoluzione di 1/16 e calcolando inizialmente 3 iterazioni aumenta il runtime di un fattore:

$$1 + \frac{1}{2^3} + \frac{1}{4^3} + \frac{1}{8^3} + 3\frac{1}{16^3} \approx 1.14 \tag{4.11}$$

Quindi, il tempo di esecuzione teorico del calcolo gerarchico sarebbe solo del 14% più lento. E' degno di nota che l'immagine della disparità del livello di risoluzione inferiore viene usata solo per stimare la distribuzione di probabilità P e calcolare i costi del livello di risoluzione maggiore. Tutto il resto è calcolato da zero per evitare di passare errori dai livelli inferiori di risoluzione a quelli superiori.

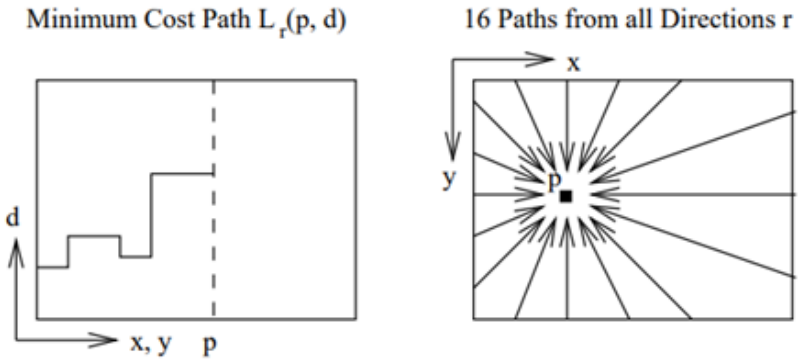


Figura 4.10: Aggregazione dei costi nello spazio di disparità.

Quindi utilizzando questo metodo abbiamo a disposizione l'immagine di disparità D e possiamo dunque definire l'energia $E(D)$ di cui parlavamo prima come:

$$E(D) = \sum_{\mathbf{P}} \left(C\left(\mathbf{p}, D_{\mathbf{p}}\right) + \sum_{\mathbf{q} \in N_{\mathbf{p}}} P_1 \mathbf{T}\left[|D_{\mathbf{p}} - D_{\mathbf{q}}| = 1\right] + \sum_{\mathbf{q} \in N_{\mathbf{p}}} P_2 \ \mathrm{T}\left[|D_{\mathbf{p}} - D_{\mathbf{q}}| > 1\right] \right) \tag{4.12}$$

Il primo termine è la somma di tutti i costi di corrispondenza dei pixel per le disparità di D. Il secondo termine aggiunge una penalità costante P_1 per tutti i pixel q nell'intorno N_p di p per i quali la disparità cambia leggermente (massimo 1 pixel). Il terzo termine aggiunge una penalità costante maggiore P_2, per tutte le variazioni di disparità maggiori. L'uso di una penalità inferiore per piccoli cambiamenti consente un adattamento su superfici inclinate o curve. La penalità costante per tutti i più grandi cambiamenti (cioè indipendentemente dalla loro dimensione) preserva le discontinuità.
Le discontinuità sono spesso visibili al variare dell'intensità, questo viene sfruttato adattando P_2 al gradiente di intensità, ovvero $P_2 = \frac{P_2'}{|I_{bp} - I_{bq}|}$ per gli intorni dei pixel p e q nell'immagine di base I_b. Tuttavia, è sempre necessario garantire che $P2 \geq P1$. ***Il problema della corrispondenza stereo può ora essere formulato come trovare l'immagine di disparità D che minimizza l'energia E(D).*** Sfortunatamente, una tale minimizzazione globale, cioè in 2D, è NP-completo per molte discontinuità che preservano le energie. Al contrario, la riduzione al minimo lungo le singole righe di immagini, ovvero in 1D, può essere eseguita in modo efficiente in tempo polinomiale utilizzando la Dynamic Programming [13]. Tuttavia, le soluzioni basate sulla programmazione dinamica soffrono facilmente di "striature" [8], a causa della difficoltà nel correlare le ottimizzazioni 1D delle singole righe di immagini tra loro in un'immagine 2D. Il problema è che vincoli molto forti in una direzione, cioè lungo le righe dell'immagine, sono combinate con nessuno o con vincoli molto più deboli nell'altra direzione, cioè lungo le colonne dell'immagine. Questo porta alla nuova idea di aggregare i costi di corrispondenza in 1D da tutte le direzioni allo stes-

so modo. Il costo aggregato (uniformato) $S(p,d)$ per un pixel p e disparità d viene calcolato sommando i costi di tutti i percorsi di costo minimo 1D che terminano nel pixel p e disparità d, come mostrato in figura 4.10. Questi percorsi attraverso lo spazio di disparità vengono proiettati come linee rette nell'immagine di base, ma come linee non rette nell'immagine corrispondente, secondo i cambiamenti di disparità lungo i percorsi. È interessante notare che è richiesto solo il costo del percorso e non il percorso stesso. Il costo $L'_{\mathbf{r}}(\mathbf{p},d)$ lungo un percorso attraversato in direzione r dal pixel p alla disparità d è definito ricorsivamente come:

$$\begin{aligned} L'_{\mathbf{r}}(\mathbf{p},d) =& C(\mathbf{p},d) + \min\big(L'_{\mathbf{r}}(\mathbf{p}-\mathbf{r},d), \\ & L'_{\mathbf{r}}(\mathbf{p}-\mathbf{r},d-1)+P_1 \\ & L'_{\mathbf{r}}(\mathbf{p}-\mathbf{r},d+1)+P_1 \\ & \min_i L'_{\mathbf{r}}(\mathbf{p}-\mathbf{r},i)+P_2\big) \end{aligned} \tag{4.13}$$

Il primo termine è il costo di corrispondenza pixel per pixel, Il resto dell'equazione aggiunge il costo più basso del precedente pixel p - r del percorso, inclusa la penalità appropriata per le discontinuità, questo implementa il comportamento di E(D) lungo un percorso 1D arbitrario. Questo costo non impone la visibilità o vincolo di ordinamento, perché entrambi non possono essere realizzati per percorsi che non sono identici alle linee epipolari.
I valori di L' aumentano in modo permanente lungo il percorso, che può portare a valori molto grandi. Tuttavia, l'equazione sopra può essere modificata sottraendo il costo minimo del percorso del pixel precedente dall'intero termine:

$$\begin{aligned} L_{\mathbf{r}}(\mathbf{p},d) =& C(\mathbf{p},d) + \min\big(L_{\mathbf{r}}(\mathbf{p}-\mathbf{r},d) \\ & L_{\mathbf{r}}(\mathbf{p}-\mathbf{r},d-1)+P_1 \\ & L_{\mathbf{r}}(\mathbf{p}-\mathbf{r},d+1)+P_1 \\ & \min_i L_{\mathbf{r}}(\mathbf{p}-\mathbf{r},i)+P_2\big) - \min_k L_{\mathbf{r}}(\mathbf{p}-\mathbf{r},k) \end{aligned} \tag{4.14}$$

Questa modifica non cambia il percorso attuale attraverso lo spazio di disparità finchè il valore sottratto è costante per tutte le disparità di un pixel p. Quindi, la posizione del minimo non cambia. Tuttavia, ora è possibile fornire il limite superiore come $L \leq C_{\max} + P_2$. I costi L_r sono sommati sui percorsi in tutte le r direzioni. Il numero di percorsi deve essere almeno 8 e dovrebbe essere 16 per fornire una buona copertura dell'immagine 2D. In quest'ultimo caso, percorsi che non sono orizzontali, verticali o diagonali si realizzano facendo un passo in orizzontale o in verticale seguito da un passo in diagonale.

$$S(\mathbf{p},d) = \sum_{\mathbf{r}} L_{\mathbf{r}}(\mathbf{p},d) \tag{4.15}$$

Il limite superiore per S è facilmente determinabile come $\mathrm{S} \leq 16\,(\mathrm{C_{max}} + \mathrm{P_2})$, per 16 percorsi. Un'implementazione efficiente precalcola il costo di corrispondenza tra pixel $C(p,d)$, ridotto a valori interi di 11 bit, cioè, $\mathrm{C_{max}} < 2^{11}$, tramite un fattore s. Il ridimensionamento a 11 bit garantisce che i costi aggregati nei calcoli successivi non superino il limite di 16 bit. Tutti i costi sono memorizzati in un vettore di 16 bit $C[]$ di dimensione W x H x D. Quindi, $C[\mathbf{p},d] = sC(\mathbf{p},d)$. Un secondo vettore di interi a 16 bit $S[]$ della stessa dimensione viene utilizzato per memorizzare i

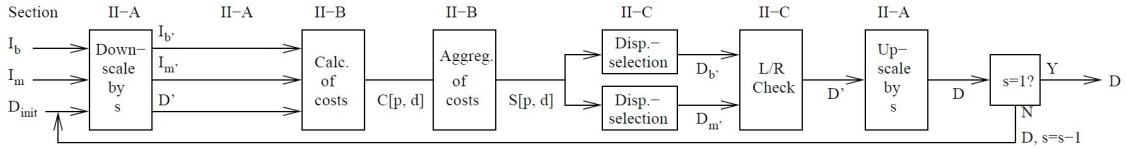


Figura 4.11: Riassunto dei passaggi del processo fin qui.

valori di costo aggregati, inizializzato con valori nulli. Il calcolo inizia per ogni direzione r da tutti i pixel b del bordo dell'immagine con $L_{\mathrm{r}}(\mathrm{b}, d) = C[\mathbf{b}, d]$, il percorso viene attraversato in avanti. Per ciascun pixel p visitato lungo il percorso, i costi $L_{\mathrm{r}}(\mathrm{p}, d)$ vengono aggiunti ai valori $S[b, d]$ per tutte le disparità d. Il calcolo richiede passi $O(D)$ per ogni pixel, fino a che il costo minimo del pixel precedente, ad esempio $\min_k L_r(p - r, k)$, è costante per tutte le disparità di un pixel e può essere precalcolato. Ogni pixel viene visitato esattamente 16 volte, il che risulta in una complessità totale di $O(WHD)$.

Calcolo della disparità

L'immagine di disparità D_b che corrisponde all'immagine di base I_b è determinata come nei metodi stereo locali selezionando per ciascun pixel p la disparità d che corrisponde al costo minimo, cioè $\min_d S[\mathbf{p}, d]$. Per la stima dei sub-pixel, viene adattata una curva quadratica attraverso i costi vicini, cioè alla disparità successiva più alta e a quella più bassa, e viene calcolata la posizione del minimo. L'utilizzo di una curva quadratica è teoricamente giustificato solo per la correlazione che utilizza la somma delle differenze al quadrato. Tuttavia, è usata come approssimazione dovuta alla semplicità di calcolo utile per velocizzare il calcolo. L'immagine della disparità D_m che corrisponde all'immagine di corrispondenza I_m può essere determinata dagli stessi costi, attraversando la linea epipolare, che corrisponde al pixel q della immagine di corrispondenza. Di nuovo, viene selezionata la disparità d, che corrisponde al costo minimo, cioè $\min_d S\left[e_{mb}(\mathbf{q}, d), d\right]$. Comunque, la fase di aggregazione dei costi non tratta le immagini di base e di corrispondenza simmetricamente. Ci si possono aspettare risultati leggermente migliori, se D_m è calcolata da zero, ovvero eseguendo la corrispondenza tra pixel e di nuovo l'aggregazione, ma con I_m come base e I_b come immagine di corrispondenza. Valori anomali sono filtrati da D_b e D_m, utilizzando un filtro mediano con un kernel piccolo, cioè 3×3. Il calcolo di D_b così come D_m consente la determinazione di occlusioni e false corrispondenze eseguendo un controllo di coerenza. Ogni disparità di D_b viene confrontata con la sua corrispondente disparità di D_m. La disparità viene impostata su "non valida" (D_inv) se differiscono.

$$D_{\mathbf{p}} = \begin{cases} D_{b\mathbf{p}} & \text{if } |D_{b\mathbf{p}} - D_{m\mathbf{q}}| \leq 1 \\ D_{\text{inv}} & \text{otherwise} \end{cases} \tag{4.16}$$

$$\mathbf{q} = e_{bm}\left(\mathbf{p}, D_{b\mathbf{p}}\right)$$

Il controllo di coerenza applica il vincolo di unicità, consentendo solo le mappature uno a uno. Il calcolo della disparità e il controllo della coerenza richiedono la visita di ogni pixel ad ogni disparità un numero costante di volte. Pertanto, la complessità

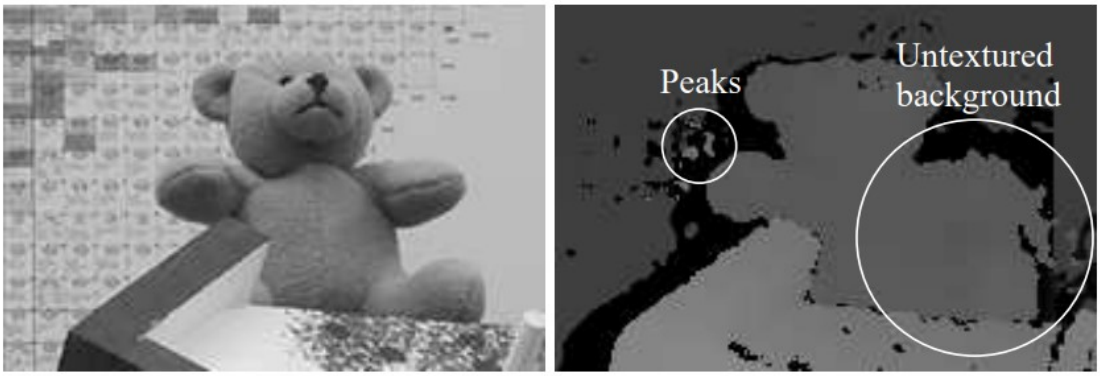


Figura 4.12: Possibili errori nelle immagini di disparità.

di questo passaggio è di nuovo $O(WHD)$. Un riepilogo di tutte le fasi di elaborazione del metodo SGM si trova in figura 4.11.

Perfezionamento della disparità

L'immagine della disparità risultante può ancora contenere alcuni tipi di errori, inoltre, ci sono generalmente aree di valori non validi che devono essere recuperati. Entrambi questi problemi possono essere gestiti in post processing, di seguito vediamo le principali metodologie applicabili.

1. ***Rimozione dei picchi:*** le immagini di disparità possono contenere valori anomali, cioè disparità completamente sbagliate, a causa di bassa consistenza, riflessi,rumore, ecc. Di solito si presentano come piccole macchie di disparità, sono molto diverse dalle disparità circostanti, vengono detti picchi, come mostrato in figura 4.12. In base alla scena, le dimensioni di piccole patch di disparità possono anche rappresentare strutture valide. Spesso una soglia può essere definita sulla dimensione di quest'ultime, in modo tale che le patch più piccole è improbabile rappresentino una struttura di scena valida. Per identificare i picchi, l'immagine di disparità viene segmentata, consentendo alle disparità vicine all'interno di un segmento di variare di un pixel, considerando una griglia dell'immagine 4-connessa. Le disparità di tutti i segmenti al di sotto di una certa dimensione sono impostati come non validi. Questo algoritmo di segmentazione e filtraggio dei picchi può essere implementato nell'ordine di $O(WH)$.

2. ***Disparità con intensità consistente:*** negli ambienti indoor capita spesso che si trovino oggetti in primo piano davanti a uno sfondo basso o privo di texture, ad esempio un muro come in figura 4.12. La funzione energetica E(D) definita in (4.12) non include una preferenza sulla posizione di uno step di disparità. Pertanto, E(D) non distingue tra il posizionamento di uno step di disparità corretto accanto a un oggetto in primo piano o un po' più in là all'interno di uno sfondo senza texture. Adattare il costo P_2 in funzione del gradiente di intensità aiuta a posizionare correttamente lo step di disparità proprio accanto a un oggetto in primo piano, perché questa posizione coincide con un gradiente di intensità in contrasto con una posizione all'interno di

un'area priva di texture. Tuttavia, SGM applica la funzione di energia non in 2D sull'immagine intera, ma lungo percorsi 1D individuali da tutte le direzioni e poi li somma. Se si incontra un'area priva di texture lungo un percorso 1D, una modifica della disparità è applicata solo se la corrispondenza delle aree texturizzate su entrambi i lati dell'area non texturizzata lo richiede. Le aree non texturizzate possono avere forme e dimensioni diverse e possono estendersi oltre i bordi dell'immagine, come abbastanza comune per i muri in scene indoor. A seconda della posizione e della direzione dei percorsi 1D, si possono incontrare texture di primo piano e oggetti sullo sfondo attorno a una parte non texturizzata, nel qual caso ci si aspetterebbe uno step di disparità corretto. Sommare tutti quei percorsi incoerenti può facilmente condurre a discontinuità sfocate intorno agli oggetti in primo piano di fronte a uno sfondo senza texture. È interessante notare che questo problema è solo un caso speciale che si applica a determinate scene in ambienti strutturati. Tuttavia, è abbastanza importante darne una soluzione; prima facciamo alcune ipotesi:

- Le discontinuità nell'immagine di disparità non devono verificarsi all'interno di aree non texturizzate;
- Sulla stessa superficie fisica dell'area non texturizzata è visibile anche qualche texture;
- La superficie dell'area non texturizzata può essere approssimata da un piano.

La prima ipotesi è per lo più corretta, discontinuità di profondità di solito causano almeno qualche cambiamento visivo nelle intensità, altrimenti la discontinuità non sarebbe rilevabile. La seconda ipotesi è necessaria in quanto la disparità di uno sfondo assolutamente privo di texture sarebbe indeterminabile. La terza ipotesi è la più debole, la sua giustificazione è che superfici non strutturate con distanza variabile di solito appaiono con intensità variabili. Quindi, l'intensità costante a tratti può essere trattata come planare. L'identificazione delle aree non texturizzate viene effettuata da una larghezza di banda fissa, Mean Shift Segmentation [14] sull'immagine di intensità I_b. La larghezza di banda radiometrica σ_r è impostata su P_1, che di solito è 4. Per tanto, vengono trattate le variazioni di intensità al di sotto della penalità di uniformità come fossero rumore. La larghezza di banda spaziale σ_s è impostata su un valore piuttosto basso per una elaborazione veloce (es. 5). Inoltre, tutti i segmenti che sono più piccoli di una certa soglia (cioè 100 pixel) vengono ignorati, perché piccole aree prive di texture dovrebbero essere gestite bene dall'SGM. Come descritto sopra, il problema atteso è che le discontinuità siano posizionate in modo sfocato all'interno di aree prive di texture, quindi, aree non texturizzate dovrebbero contenere disparità errate di oggetti in primo piano, ma anche le disparità corrette dello sfondo, purché la superficie di sfondo contenga alcune texture, cioè l'ipotesi 2. Questo significa che alcune disparità all'interno di ogni segmento S_i dovrebbero essere corrette. Quindi, diverse ipotesi per la corretta disparità di S_i possono essere identificate segmentando la disparità all'interno di ogni segmento S_i. Questo viene fatto mediante una semplice segmentazione, cioè consentendo alle disparità vicine all'interno di un

segmento di variare di un pixel. Questa segmentazione veloce risulta in diversi segmenti S_{ik} per ogni segmento S_i. Successivamente le ipotesi di superficie F_{ik} vengono create calcolando i migliori piani adattati attraverso le disparità di S_{ik}. La scelta dei piani si basa sull'ipotesi 3. Segmenti molto piccoli, cioè $\leq$ 12 pixel, vengono ignorati, poiché è improbabile che patch così piccole appartengano all'ipotesi corretta. Quindi, ogni ipotesi viene valutata all'interno di S_i sostituendo tutti i pixel di S_i con l'ipotesi della superficie e calcolando E_{ik} per tutti i pixel non occlusi di S_i. Un pixel p è occluso, se un altro pixel con maggiore disparità viene mappato allo stesso pixel q nell'immagine corrispondente. Questo rilevamento viene eseguito prima mappando p nell'immagine corrispondente tramite $\mathbf{q} = e_{bm}\left(\mathbf{p}, D'_{\mathbf{p}}\right)$. Quindi, la retta epipolare di q nella immagine di base $e_{mb}(\mathbf{q}, d)$ è seguita per $d > D'_{\mathrm{p}}$. Il pixel p è occluso se la retta epipolare attraversa un pixel con una disparità maggiore di d. Per ogni segmento di intensità costante S_i viene scelta l'ipotesi di superficie F_{ik} con il costo minimo E_{ik}. Tutte le disparità all'interno di S_i sono sostituite da valori sulla superficie scelta per rendere la selezione della disparità coerente con le intensità dell'immagine di base, cioè soddisfare l'ipotesi 1.

$$\begin{aligned} F_i &= F_{ik'} \text{ with } k' = \underset{k}{\operatorname{argmin}} E_{ik} \\ D'_{\mathbf{p}} &= \begin{cases} F_i(\mathbf{p}) & \text{if } \mathbf{p} \in S_i \\ D_{\mathbf{p}} & \text{otherwise.} \end{cases} \end{aligned} \tag{4.17}$$

L'approccio presentato è simile a metodi precedenti poiché utilizza la segmentazione dell'immagine e l'adattamento del piano per perfezionare un'immagine di disparità iniziale. A differenza di altri metodi, l'immagine di disparità iniziale calcolata da SGM è già abbastanza precisa, quindi vengono modificate solo le aree non texturizzate al di sopra di una certa dimensione. Perciò, solo le aree critiche vengono affrontate senza pericolo di corruzione di aree probabilmente ben abbinate. Un'altra differenza è che le disparità delle aree considerate viene selezionata considerando un piccolo numero di ipotesi che sono inerenti all'immagine della disparità iniziale, non ci sono iterazioni che richiedono tempo. Calcolare il miglior adattamento dei piani implica la visita di tutti i pixel segmentati, testare tutte le ipotesi richiede la visita di tutti i pixel di tutti i segmenti, per tutte le ipotesi. Inoltre, il test di occlusione richiede di superare al massimo D disparità per ogni pixel, pertanto, il limite superiore della complessità è $O(WHDN)$. Tuttavia i pixel segmentati di solito sono solo una frazione dell'intera immagine e il numero massimo di ipotesi N per un segmento è comunemente piccolo e spesso uguale a 1, in quest'ultimo caso, non è necessario calcolare il costo dell'ipotesi.

3. ***Discontinuità che preservano l'interpolazione:*** il controllo coerente nonché fusione di immagini di disparità o il filtro picchi, possono invalidare alcune disparità. Questo porta a buchi nell'immagine della disparità, che devono essere interpolati per un risultato più denso. Le disparità non valide sono classificate come occlusioni o mis-match; deve essere eseguita l'interpolazione di entrambi i casi diversamente. Le occlusioni non devono essere interpolate dall'occlusore, ma solo dall'occluso per evitare un errato uniformamento di discontinuità, ciò significa un'estrapolazione dello sfondo in regioni occluse. Al

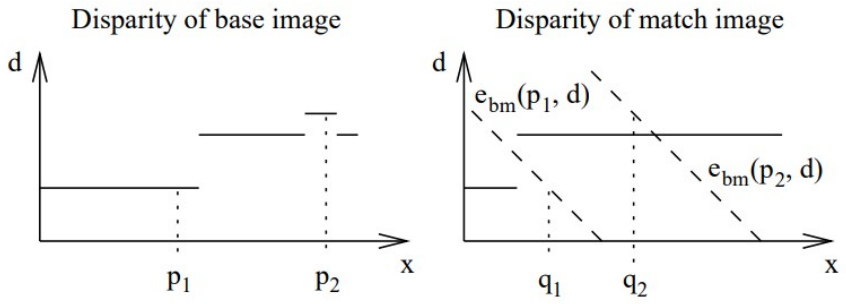


Figura 4.13: distinzione tra pixel mis-matched e occlusi.

contrario, buchi dovuti al disallineamento possono essere facilmente interpolati da tutti i pixel vicini. Le occlusioni e i mis-match possono essere distinti come parte del controllo di coerenza sinistra/destra. La figura 4.13 mostra che la retta epipolare del pixel occluso p_1 passa attraverso la discontinuità che provoca l'occlusione e non interseca la funzione di disparità D_m. Al contrario, la retta epipolare del mis-match p_2 si interseca con D_m. Pertanto, per ogni pixel invalidato, viene cercata un'intersezione della corrispondente retta epipolare con D_m, per contrassegnarla come occlusione o mis-match. Ai fini dell'interpolazione, le aree di pixel mis-matched che sono vicini diretti dei pixel occlusi vengono trattati come occlusioni, perché anche questi pixel devono essere estrapolati da pixel di background validi. L'interpolazione viene eseguita propagando disparità valide tra le aree di disparità non valide vicine, ciò è svolto in modo simile a SGM lungo percorsi da 8 direzioni. Per ciascun pixel non valido, tutti gli 8 valori v_pi vengono memorizzati. L'immagine finale della disparità D' è creata da:

$$D'_{\mathbf{p}} = \begin{cases} \text{seclow}_{\,i} v_{\mathbf{p}i} & \text{if } \mathbf{p} \text{ is occluded,} \\ \text{med}_i\, v_{\mathbf{p}i} & \text{if } \mathbf{p} \text{ is mismatched} \\ D_{\mathbf{p}} & \text{otherwise} \end{cases} \tag{4.18}$$

Il primo caso garantisce che le occlusioni siano interpolate dallo sfondo più basso selezionando il secondo valore più basso, mentre il secondo caso sottolinea l'uso di tutte le informazioni senza una preferenza per il primo piano o lo sfondo. Viene utilizzata la mediana, invece della media, per mantenere le discontinuità nei casi in cui l'area non corrispondente si trova sul bordo di un oggetto.
Il metodo di interpolazione presentato ha il vantaggio che è indipendente dal metodo di abbinamento stereo utilizzato. Gli unici requisiti sono una geometria epipolare nota e il calcolo delle immagini di disparità per l'immagine di base e per l'immagine corrispondente, per distinguere tra occlusioni e mismatch.
Infine il filtraggio mediano può essere utile per rimuovere i residui di irregolarità e inoltre uniforma l'immagine di disparità risultante. La complessità dell'interpolazione è lineare al numero di pixel, cioè $O(WH)$, poiché c'è un numero costante di operazioni per ogni pixel non valido.

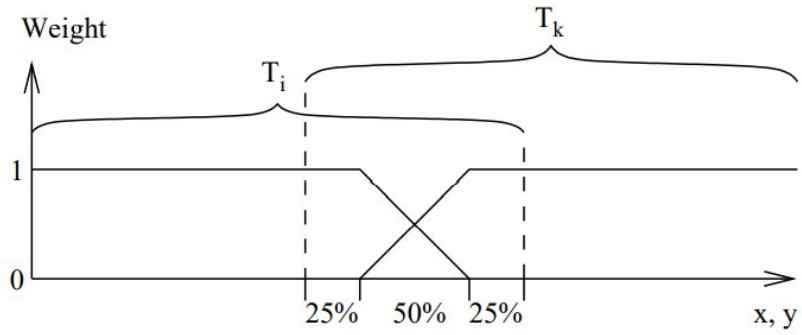


Figura 4.14: Definizione dei pesi per la fusione di crop sovrapposti.

4. ***Elaborazione di immagini enormi:*** il metodo SGM richiede memoria temporanea per l'archiviazione dei costi di corrispondenza tra pixel $C[]$, per i costi aggregati $S[]$, immagini di disparità prima della fusione, ecc. La dimensione della memoria temporanea dipende dalla dimensione dell'immagine W x H , dall'intervallo di disparità D o entrambi, come nel caso di $C[]$ e $S[]$. Quindi, anche dimensioni di immagine moderate come 1 MPixel con intervalli di disparità di svariati 100 pixel richiedono array temporanei di grandi dimensioni che possono superare la memoria disponibile.
   La soluzione proposta è dividere l'immagine di base in crop, calcolare la disparità di ciascun crop individualmente, unendo i crop insieme nella immagine di disparità totale prima della eventuale fusione multi-baseline. I crop vengono scelti sovrapponibili perché il processo di aggregazione dei costi può utilizzare solo percorsi da un lato per i pixel vicino ai bordi dei crop, che porta a una minore precisione di corrispondenza o anche a discrepanze. Questo può essere particolarmente critico in aree a bassa texture vicino ai bordi dei crop. L'unione dei crop viene eseguita calcolando la media ponderata delle disparità di tutti i crop nelle aree sovrapposte. I pesi sono scelti in modo tale che i pixel vicino al bordo del riquadro siano ignorati e quelli più lontani vengano uniti linearmente come mostrato in figura 4.14. La dimensione dei crop viene scelta il più grande possibile, in modo che tutti gli array temporanei richiesti si adattino semplicemente alla principale memoria disponibile.
   Pertanto, la memoria disponibile determina automaticamente la dimensione dei crop utilizzati internamente. Questa strategia consente la corrispondenza di immagini più grandi, tuttavia, ci sono alcune tecnologie come le telecamere aeree che possono produrre immagini singole di 1 miliardo di pixel o più. Quindi, potrebbe essere impossibile anche caricare due immagini complete nella memoria principale, per non parlare della loro corrispondenza. In questi casi, in aggiunta al cropping interno, si utilizza un cropping esterno dell'immagine di base. Ogni crop dell'immagine di base insieme al range di disparità e alla geometria nota della telecamera definisce le parti corrispondenti delle immagini di corrispondenza. Tutti i passaggi, compresa la eventuale fusione multi-baseline e, facoltativamente, la post-elaborazione vengono eseguiti e la disparità risultante viene memorizzata per ciascun crop individualmente. L'unione dei crop esterni viene eseguita allo stesso modo della fusione di crop interni. A seconda del tipo di scena, è probabile che l'intervallo di disparità

richiesto per ciascun crop è solo una frazione del range di disparità di tutte le immagini.
Pertanto, si utilizza una riduzione automatica dell'intervallo di disparità in combinazione con la corrispondenza. L'intervallo di disparità completo viene applicato per le corrispondenze alle risoluzioni più basse, successivamente un range di disparità raffinato è determinato dall'immagine della disparità risultante; il range viene esteso di un certo valore fisso per tener conto delle piccole strutture che potrebbero non essere rilevate durante la corrispondenza a bassa risoluzione. L'intervallo di disparità perfezionato e aumentato viene utilizzato per la corrispondenza alla successiva risoluzione più alta. Il meccanismo di cropping interno ed esterno consente l'abbinamento stereo di immagini arbitrariamente grandi. Un altro vantaggio del cropping esterno è che tutti i crop possono essere calcolati in parallelo su computer diversi.

Risultati

In figura 4.15 possiamo vedere il matching stereo ottenuto grazie all'algoritmo stereoSGBM(), cliccando su un punto qualsiasi dell'immagine vediamo sul terminale il valore della disparità media in quel punto e i corrispondenti punti nelle immagini rettificate. Nel paragrafo 4.2.5 spiegherò nel dettaglio i parametri utilizzati per ottenere questi risultati e l'immagine di disparità in figura 4.16.

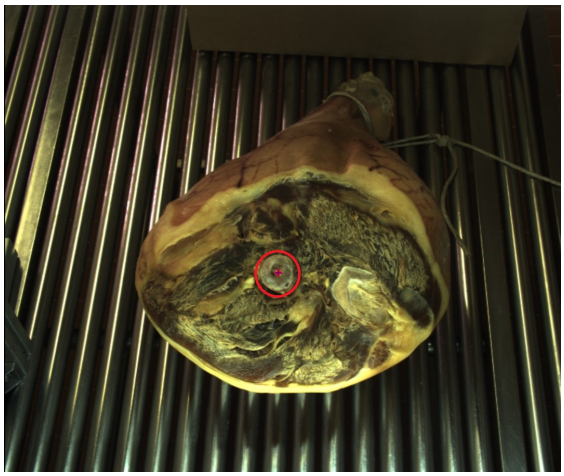
4.2.5 Creazione immagine di disparità

Come spiegato dettagliatamente nella sezione precedente nel progetto di questa tesi per le corrispondenze stereo ho utilizzato l'algoritmo stereoSGBM() con le modifiche presenti nella libreria openCV. Ci tengo a precisare che, oltre al diverso calcolo dei costi per i pixel, ci sono altre 2 differenze: l'algoritmo fa il match delle corrispondenze a blocchi e non a singoli pixel come in quello originale [9], tuttavia impostando il parametro "blockSize" a 1 si riducono i blocchi a singoli pixel; inoltre sono inclusi alcuni passaggi di pre e post elaborazione presi dall'algoritmo StereoBM di K. Konolige [15], ad esempio: prefiltraggio e post-filtraggio (controllo di unicità, interpolazione quadratica e filtro speckle).
L'algoritmo SGM di openCV dà la possibilità di migliorare la mappa di disparità in output adattando, alle immagini che si stanno utlizzando, alcuni parametri. Di seguito vediamo la lista dei parametri più rilevanti [10] e di come sono stati adattati al nostro progetto per ottenre la mappa di disparità in figura 4.15:

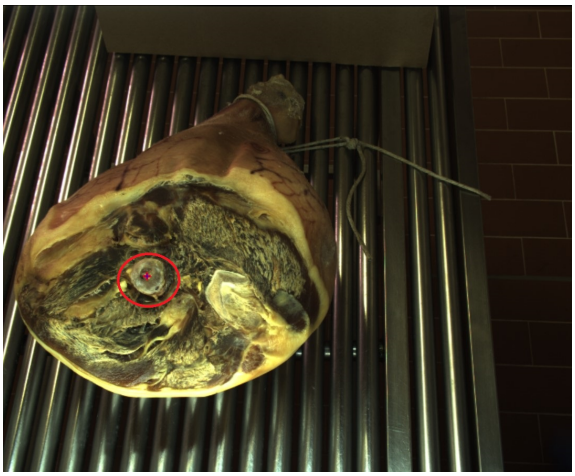
- **minDisparity:** valore di disparità minimo possibile. Normalmente è zero ma a volte gli algoritmi di rettifica possono spostare le immagini, quindi questo parametro deve essere regolato di conseguenza;

- **numDisparity:** disparità massima meno disparità minima. Il valore è sempre maggiore di zero. Nell'attuale implementazione, questo parametro deve essere divisibile per 16;

- **blockSize:** dimensione del blocco corrispondente. Deve essere un numero dispari ≥ 1. Normalmente, dovrebbe essere compreso tra 3 e 11;



(a) Immagine dalla telecamera di sinistra.



(b) Immagine dalla telecamera di destra.

```
computing disparity...
Min disparity  0  Max disparity  592  Range  592
( 1203 ,  1163 ) Median disparty:  589.0
```

(c) Selezionando un punto col mouse sulle immagini viene mostrata a terminale la disparità media in quel punto e le sue coordinate.

Figura 4.15: In questa immagine vediamo i risultati del matching stereo con stereoSGBM(), cliccando su un punto dell'immagine di disparità vengono evidenziati anche i corrispondenti punti nelle immagini rettificate come in (a) e (b).

- **P1:** il primo parametro che controlla l'uniformità della disparità (smoothing);
- **P2:** : Il secondo parametro che controlla la uniformità della disparità. Più grandi sono i valori, più uniforme è la disparità. P1 è la penalità sulla variazione della disparità di più o meno 1 tra i pixel vicini. P2 è la penalità sulla variazione della disparità di più di 1 tra i pixel vicini. L'algoritmo richiede P2 > P1. Alcuni valori P1 e P2 ragionevolmente buoni possono essere rispettivamente 8 x numero dei canali dell'immagine x blockSize x blockSize e 32 x numero dei canali dell'immagine x blockSize x blockSize;
- **disp12MaxDiff:** differenza massima consentita (in unità di pixel interi) nel controllo della disparità sinistra-destra. Impostare su un valore non positivo per disabilitare il controllo;
- **uniquenessRatio:** margine in percentuale in base al quale il valore migliore (minimo) della funzione di costo calcolato dovrebbe "vincere" il secondo miglior valore per considerare corretta la corrispondenza trovata. Normalmente, un valore compreso nell'intervallo 5-15 è abbastanza buono;
- **speckleWindowSize:** dimensione massima delle regioni di disparità uniforme per considerare i loro punti di rumore e invalidarli. Si imposta a 0 per

disabilitare il filtro speckle, altrimenti da qualche parte nell'intervallo 50-200 (una specie di filtro gaussiano);

- **speckleRange:** massima variazione di disparità all'interno di ogni componente collegato. Se speckleWindowSize è attivo, impostare il parametro su un valore positivo, verrà implicitamente moltiplicato per 16. Normalmente, 1 o 2 è sufficiente;
- **mode:** impostarlo su StereoSGBM :: MODE_HH per eseguire l'algoritmo di programmazione dinamica a due passaggi su vasta scala. Occuperà O(W x H x numDisparities) byte, che è grande per lo stereo 640x480 ed enorme per le immagini in formato HD. Comprende:
  - MODE_SGBM (default=0);
  - MODE_HH;
  - MODE_SGBM_3WAY;
  - MODE_HH4.

Risultati

Parliamo ora dell'immagine di disparità ottenuta e mostrata in figura 4.16, di seguito possiamo vedere come sono stati impostati i parametri della funzione openCV stereoSGBM() per ottenere il risultato ottimale (impostati basandosi sulle nozioni dette sopra e facendo diverse prove per trovare il risultato migliore):

1. **min_disp** = int(round(0 / 16) x 16)
2. **num_disp** = int(round(596.0 / 16) x 16)
3. **block_size** = 11
4. **uniquenessRatio** = 4
5. **speckleRange** = 2
6. **speckleWindowSize** = 200
7. **disp12MaxDiff** = 0
8. **P1** = 1400
9. **P2** = 5600
10. **mode** = 0

Nel capitolo 7 capiremo nel dettaglio come trattare l'immagine di disparità per ottenere un modello 3D del prosciutto e quindi come ricostruire la scena 3D analizzata per poter guidare il robot nello spazio con coordinate precise.

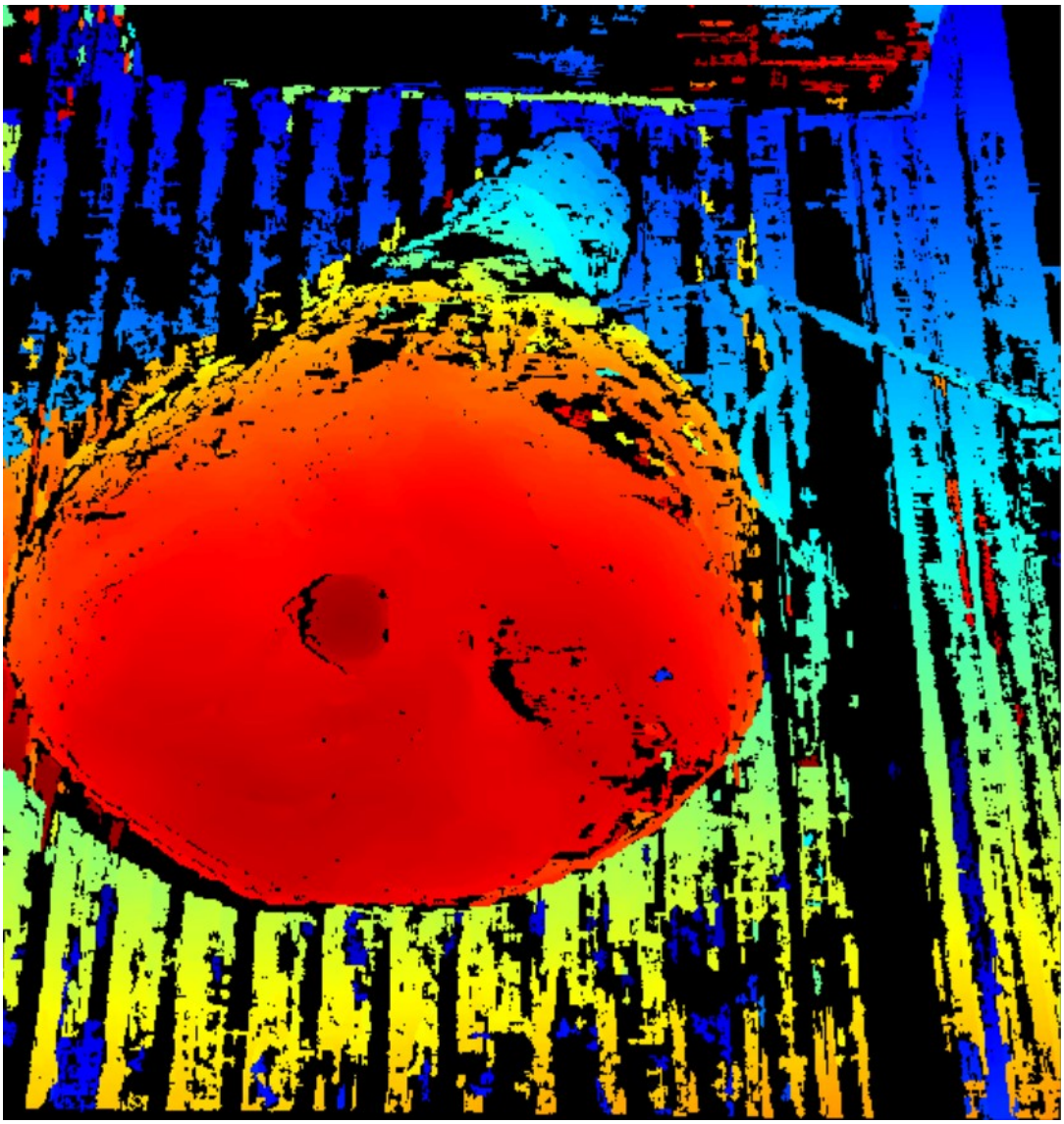


Figura 4.16: Mappa di disparità derivante dalle immagini 4.15, ottenuta con i parametri specificati, la scala colori va dal blu scuro/nero che indica la disparità minima ad un rosso molto accesso che indica la disparità massima.

Capitolo 5

Deep learning nella computer vision

5.1 Cosa si intende per deep learning?

La traduzione strettamente letterale di questo termine, di chiara matrice anglosassone, è apprendimento profondo. E' proprio questo il centro del suo significato, perché il deep learning, sottocategoria del Machine Learning, non fa altro che creare modelli di apprendimento su più livelli. Il concetto è molto semplice: immaginiamo di esporre una nozione, la apprendiamo e subito dopo ne esponiamo un'altra. Il nostro cervello raccoglie l'input della prima e la elabora insieme alla seconda, trasformandola e astraendola sempre di più. Scientificamente, è corretto definire l'azione del deep learning come l'apprendimento di dati che non sono forniti dall'uomo, ma sono appresi grazie all'utilizzo di algoritmi di calcolo statistico. Questi algoritmi hanno uno scopo: comprendere il funzionamento del cervello umano e come riesca ad interpretare le immagini e linguaggio. L'apprendimento così realizzato ha la forma di una piramide: i concetti più alti sono appresi a partire dai livelli più bassi.

Il deep learning ha compiuto passi da gigante, ottenendo risultati che, fino a qualche decennio fa, erano pura utopia. Tale successo è dovuto alle numerose conquiste in campo informatico, relative soprattutto alla sfera dell'hardware. Abbiamo visto come sia importante portare il calcolatore a fare esperienza su un quantitativo sempre maggiore di dati sensibili e come, fino a poco tempo fa, il tempo per ottenere tale addestramento fosse abbastanza elevato. Oggi, grazie all'introduzione delle GPUs, ovvero nuove unità che concorrono all'elaborazione dati, questo processo è diventato molto più snello. Un altro importante aiuto è derivato dalla facilità di trovare numerose collezioni di dati (dataset) fondamentali per allenare il sistema.
Il deep learning fa una cosa fondamentale: ci regala la rappresentazione dei dati, ma lo fa a livello gerarchico e soprattutto a livelli diversi tra loro, riuscendo a elaborarli e a trasformarli. Questa trasformazione è stupefacente perché ci consente di assistere ad una macchina che riesce a classificare i dati in entrata (input) e quelli in uscita (output), evidenziando quelli importanti ai fini della risoluzione del problema e scartando quelli che non servono. La rivoluzione apportata dal deep learning è tutta nella capacità, simile a quella umana, di elaborare i dati, le proprie conoscenze a livelli che non sono affatto lineari. Grazie a questa facoltà, la macchina apprende e perfeziona funzionalità sempre più complesse.

5.2 Reti neurali artificiali

I circuiti neurali artificiali sono la base di sofisticate forme di intelligenza artificiale, sempre più evolute, in grado di apprendere sfruttando meccanismi simili (almeno in parte) a quelli dell'intelligenza umana, il risultato sono prestazioni impossibili per altri algoritmi.
Le reti neurali artificiali riescono oggi a risolvere determinate categorie di problemi avvicinandosi sempre più all'efficienza del nostro cervello, e trovando perfino soluzioni inaccessibili alla mente umana. Dalla nascita del concetto di neurone artificiale ad oggi è stata fatta molta strada. In moltissimi ed eterogenei settori scientifici, dalla biomedicina al data mining, le reti neurali hanno ormai un impiego quotidiano. Si tratta di un trend in crescita. I continui progressi permettono di ottenere circuiti sempre più sofisticati. Tutto lascia prevedere, insomma, che le reti neurali e il machine learning saranno parte notevole delle fondamenta del mondo ipertecnologico in cui ci stiamo addentrando.

Il prototipo delle reti neurali artificiali sono quelle biologiche. Le reti neurali del cervello umano sono la sede della nostra capacità di comprendere l'ambiente e i suoi mutamenti e di fornire quindi risposte adattive calibrate sulle esigenze che si presentano; sono costituite da insiemi di cellule nervose fittamente interconnesse fra loro, al loro interno troviamo:

- **i somi neuronali**, ossia i corpi dei neuroni. Ricevono e processano le informazioni; in caso il potenziale di azione in ingresso superi un certo valore, generano a loro volta degli impulsi in grado di propagarsi nella rete;

- **i neurotrasmettitori**, composti biologici di diverse categorie (ammine, peptidi, aminoacidi), sintetizzati nei somi e responsabili della modulazione degli impulsi nervosi;

- **gli assoni o neuriti**: la via di comunicazione in uscita da un neurone. Ogni cellula nervosa ne possiede di norma soltanto uno; i dendriti: la principale via di comunicazione in ingresso; sono multipli per ogni neurone, formando il cosiddetto albero dendritico;

- **le sinapsi**, o giunzioni sinaptiche: i siti funzionali ad alta specializzazione nei quali avviene il passaggio delle informazioni fra neuroni. Ognuno di questi ne possiede migliaia. A seconda dell'azione esercitata dai neurotrasmettitori, le sinapsi hanno una funzione eccitatoria, facilitando la trasmissione dell'impulso nervoso, oppure inibitoria, tendente a smorzarlo. Le connessioni hanno luogo quando il neurotrasmettitore viene rilasciato nello spazio intersinaptico, raggiungendo così i recettori delle membrane post-sinaptiche (ossia del neurone successivo), e, alterandone la permeabilità, trasmettendo l'impulso nervoso.

Un singolo neurone può ricevere simultaneamente segnali da diverse sinapsi, una sua capacità intrinseca è quella di misurare il potenziale elettrico di tali segnali in modo globale, stabilendo quindi se è stata raggiunta la soglia di attivazione per generare a sua volta un impulso nervoso. Tale proprietà è implementata anche nelle reti artificiali.
La configurazione sinaptica all'interno di ogni rete neurale biologica è dinamica.

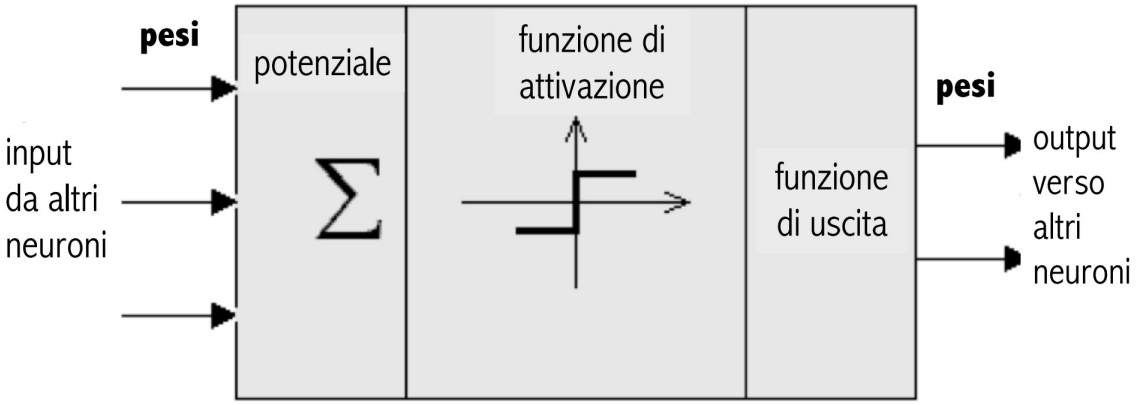


Figura 5.1: Schematizzazione di un neurone artificiale.

Si tratta di un fattore determinante per la loro efficienza. Il numero di sinapsi può incrementare o diminuire a seconda degli stimoli che riceve la rete. Più sono numerosi, maggiori connessioni sinaptiche vengono create, e viceversa. In questo modo, la risposta adattiva fornita dai circuiti neurali è più calibrata e anche questa è una peculiarità implementata nelle reti neurali artificiali.

Dal punto di vista generale ci sono molti diversi tipi di reti neurali, ma quasi tutte si basano sugli stessi concetti. Se si vuole simulare il cervello umano utilizzando una rete neurale, è ovvio che alcune drastiche semplificazioni devono essere fatte. Prima di tutto, non è possibile "replicare" il funzionamento parallelo di tutte le cellule neurali, anche se ci sono computer che hanno capacità di elaborazione parallela, il gran numero di processori che sarebbe necessario non può essere implementato dall'hardware attualmente disponibile. Un altro limite è che la struttura interna di un computer non può essere modificata durante l'esecuzione di qualsiasi compito.
E come implementare le stimolazioni elettriche in un programma per computer?
Come il cervello umano, una rete neurale artificiale consiste di neuroni e delle connessioni tra di loro, i neuroni trasportano le informazioni in entrata sulle loro connessioni in uscita verso altri neuroni. Nella terminologia delle reti neurali queste connessioni sono chiamate pesi. Le informazioni "elettriche" sono simulate con valori specifici memorizzati nei pesi. Il cambiamento della struttura di connessione può essere simulata semplicemente cambiando i valori dei pesi. La figura 5.1 mostra un esempio di neurone di una rete neurale artificiale, è simile a una cellula neurale biologica e funziona nello stesso modo. Le informazioni (valori di input) arrivano al neurone e vengono combinate (moltiplicandole) con i relativi pesi. Una funzione di propagazione somma tali valori, fornendo il "potenziale" del neurone, il valore risultante viene confrontato con un dato valore di soglia in base alla funzione di attivazione del neurone. Se la somma supera il valore di soglia, il neurone si attiva, altrimenti sarà inibito; se attivato, il neurone invia un output sulle connessioni pesate in uscita a tutti i neuroni collegati, e così via.
In una rete neurale, i neuroni sono raggruppati in strati (layer), denominati strati di neuroni (neuron layers), di solito ogni neurone di uno strato è collegato a tutti i neuroni dello strato precedente e successivo (eccetto il livello di input e lo strato di uscita della rete). Le informazioni fornite da una rete neurale sono propagate layer-by-layer dallo strato di input a quello di output attraverso nessuno, uno o più

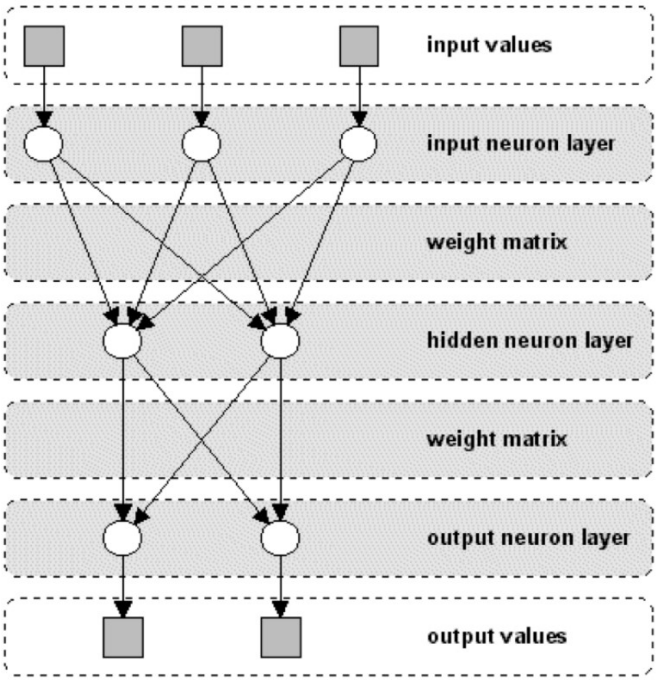


Figura 5.2: Schematizzazione di una rete neurale a tre strati. Si noti che questa non è la struttura generale di una rete neurale, ad esempio, alcuni tipi di reti non hanno livelli nascosti o i neuroni in uno strato sono disposti come una matrice. Ciò che è comune a tutti i tipi di reti neurali è la presenza di almeno una matrice dei pesi, che individua le connessioni tra due strati di neuroni.

strati nascosti (hidden). A seconda dell'algoritmo di apprendimento, è anche possibile che l'informazione si propaghi all'indietro attraverso la rete (backpropagation). In figura 5.2 vediamo un esempio di rete neurale a tre strati

5.2.1 Possibilità e limiti

L'utilizzo delle varie tipologie di reti neurali nasce dagli importanti vantaggi che presentano:

- Elevato parallelismo, grazie al quale possono processare in tempi relativamente rapidi grandi moli di dati;
- Tolleranza ai guasti, anche questo grazie all'architettura parallela;
- Tolleranza al rumore, ossia la capacità di operare, in molti casi, in modo corretto nonostante input imprecisi o incompleti;
- In alcune categorie di problemi costituiscono lo strumento migliore per gestirlo. Data mining, optimization, elaborazione di modelli predittivi e simulativi e classificazione sono i campi di impiego preferenziali per le reti neurali;
- Evoluzione adattiva: una rete neurale ben implementata è in grado di autoaggiornarsi in presenza di modifiche ambientali.

Le reti neurali artificiali hanno comunque dei limiti, ed è difficile prevedere se col tempo potranno essere eliminati o attenuati. I più importanti sono:

- Funzionamento a black box. Un handicap rimarchevole delle reti neurali artificiali è il fatto che la loro computazione non è analizzabile in modo completo. Con questo si intende dire che sono in grado di fornire output corretti, o sufficientemente corretti, ma non permettono di esaminare i singoli stadi di elaborazione che li determinano;

- Non è possibile avere la certezza a priori che un problema sarà risolto;

- Gli output forniti spesso non rappresentano la soluzione perfetta, anche se in molti casi questo non è necessario;

- Il periodo di learning è più o meno lungo. Le iterazioni necessarie dipendono da fattori quali numero e complessità delle variabili di input, algoritmo utilizzato, etc. In realtà, in tale ambito sono stati fatti importanti progressi, ed è ragionevole ipotizzare che in futuro il periodo di learning potrà ulteriormente ridursi;

- Le reti neurali non sono idonee a risolvere determinate categorie di problemi. Un esempio è un tipo di input costituito da un numero elevato di variabili categoriche.

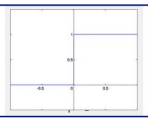
Sebbene le reti neurali siano in grado di trovare soluzioni a problemi difficili, i risultati non possono essere garantiti.
Essi sono solo approssimazioni della soluzione desiderata e un certo errore è sempre presente.
Inoltre, esistono problemi che non possono essere correttamente risolti con reti neurali.

5.2.2 Tipologie di reti neurali

Esistono diverse tipologie di reti neurali, che possono essere distinte per tipo (feedforward o feedback), struttura e algoritmo di apprendimento utilizzato. Il *tipo* di una rete neurale indica se i neuroni dei vari strati della rete possono essere collegati tra loro. Le reti neurali di tipo **Feedforward** permettono solo collegamenti neuronali tra due diversi strati, mentre le reti del tipo **Feedback** consentono anche connessioni tra i neuroni dello stesso strato. Di seguito verrà esaminata brevemente una selezione di tipologie di reti neurali.

Perceptron

Il Perceptron è stato introdotto da F. Rosenblatt nel 1958 [16] (figura 5.3). Si tratta di un tipo molto semplice di rete neurale con due strati di neuroni che accetta solo input ed output binari (0 e 1). Il processo di apprendimento è supervisionato e la rete è in grado di risolvere operazioni logiche di base come AND e OR. È utilizzato anche per la classificazione di modelli (pattern classification). Operazioni logiche più complesse (come XOR) non possono essere risolte da un Perceptron.

tipo	Feedforward
strati	1 strato di input, 1 strato di output
valori di input/output	binari
funzione di attivazione	hard limit 
metodo di apprendimento	supervisionato
algoritmo di apprendimento	regola di Hebb
utilizzo	semplici operazioni logiche, pattern classification

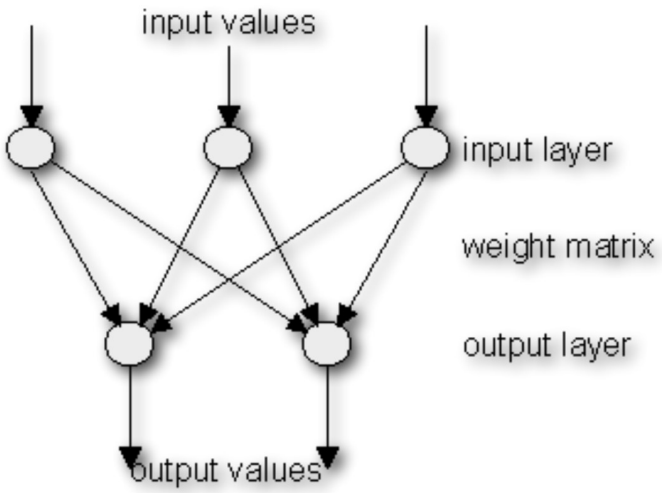
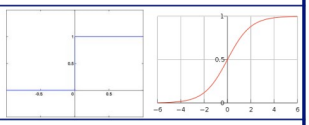


Figura 5.3: Rete neurale Perceptron.

Multi-Layer Perceptron

Il Multi-Layer Perceptron è stato introdotto da M. Minsky e S. Papert nel 1969 [16] (figura 5.4). Si tratta di un Perceptron esteso, con uno o più livelli nascosti tra lo strato di input e quello di output. Grazie alla sua struttura estesa, un Multi-Layer Perceptron è in grado di risolvere ogni operazione logica, compreso il problema XOR.

tipo	Feedforward
strati	1 strato di input, 1 o più strati nascosti, 1 strato di output
valori di input/output	binari
funzione di attivazione	hard limit, sigmoidea 
metodo di apprendimento	supervisionato
algoritmo di apprendimento	delta rule
utilizzo	operazioni logiche complesse, pattern classification

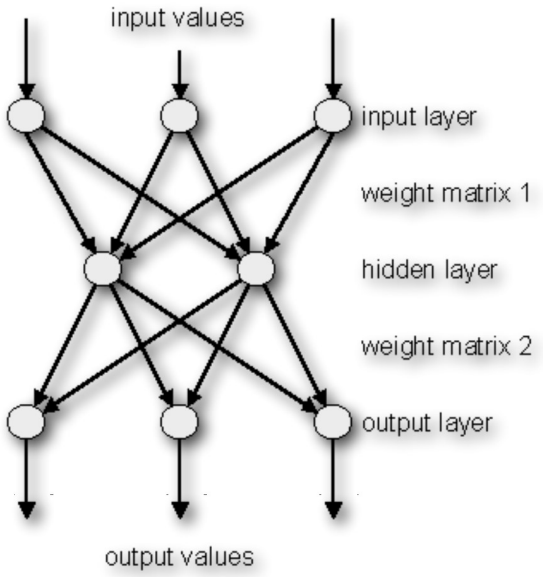
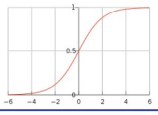


Figura 5.4: Rete neurale Multi-Layer Perceptron.

MLP Backpropagation

La rete MLP Backpropagation è stata introdotta da Hinton, Rumelhart e Williams nel 1986 [16] ed è uno dei tipi più potenti di rete neurale (figura 5.5). Ha la

tipo	Feedforward
strati	1 strato di input, 1 o più strati nascosti, 1 strato di output
valori di input/output	binari
funzione di attivazione	sigmoidea 
metodo di apprendimento	supervisionato
algoritmo di apprendimento	backpropagation
utilizzo	operazioni logiche complesse, pattern classification, analisi del parlato

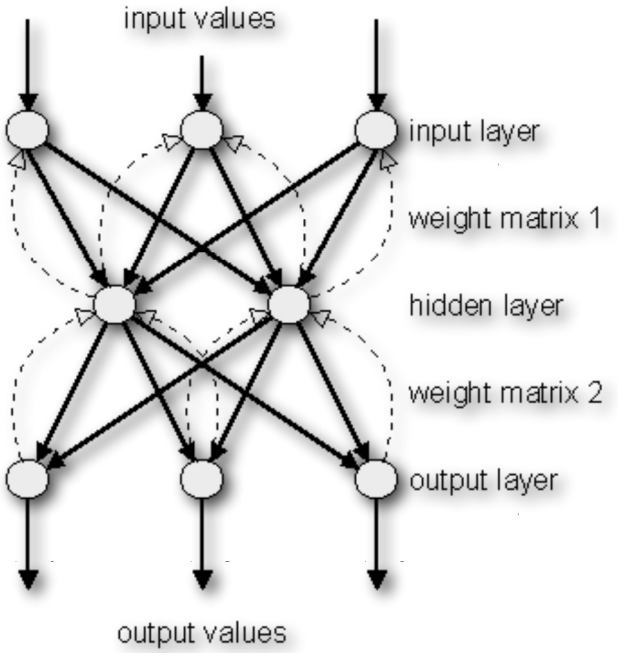
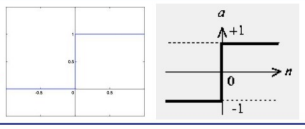


Figura 5.5: Rete neurale Multi-Layer Perceptron con backpropagation.

stessa struttura del Multi-Layer Perceptron e utilizza l'algoritmo di apprendimento "backpropagation".

Hopfield

Questa rete è stata introdotta dal fisico Hopfield nel 1982 [16] e appartiene ai tipi di reti neurali denominate "modelli termodinamici" (figura 5.6). Si compone di un insieme di neuroni, in cui ogni neurone è collegato ad ogni altro neurone. Non c'è differenziazione tra neuroni di input e di output. L'applicazione principale di una rete di Hopfield è la conservazione e il riconoscimento di modelli, per esempio immagini.

tipo	Feedback
strati	1 matrice di connessioni
valori di input/output	binari
funzione di attivazione	segno, hard limiter 
metodo di apprendimento	non supervisionato
algoritmo di apprendimento	delta rule simulated annealing (più usata)
utilizzo	pattern association, problemi di ottimizzazione

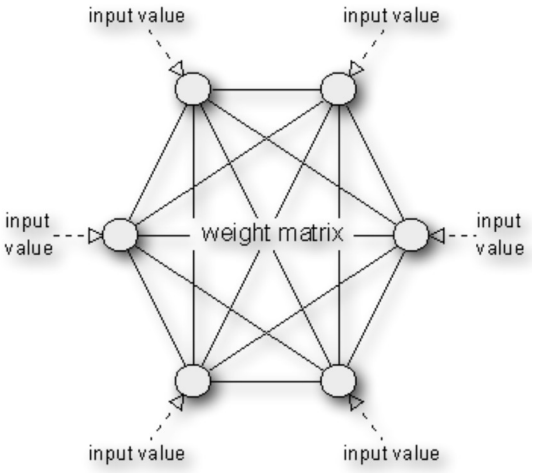
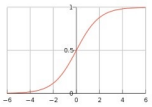


Figura 5.6: Rete neurale Hopfield.

Mappa di Kohonen

Questa rete è stata introdotta dal professore finlandese Teuvo Kohonen (Università di Helsinki) nel 1982 [16] (figura 5.7). È probabilmente il tipo di rete neurale più utile per la simulazione del processo di apprendimento del cervello umano. Il suo "cuore" è la mappa caratteristica, uno strato di neuroni in cui i neuroni si organizzano secondo certi valori di input. Questo tipo di rete neurale è sia feedforward che feedback (neuroni della mappa interconnessi).

tipo	Feedforward/Feedback
strati	1 strato di input, 1 strato "mappa"
valori di input/output	binari, reali
funzione di attivazione	sigmoidea 
metodo di apprendimento	non supervisionato
algoritmo di apprendimento	auto-organizzazione (esempio)
utilizzo	pattern classification, problemi di ottimizzazione simulazione

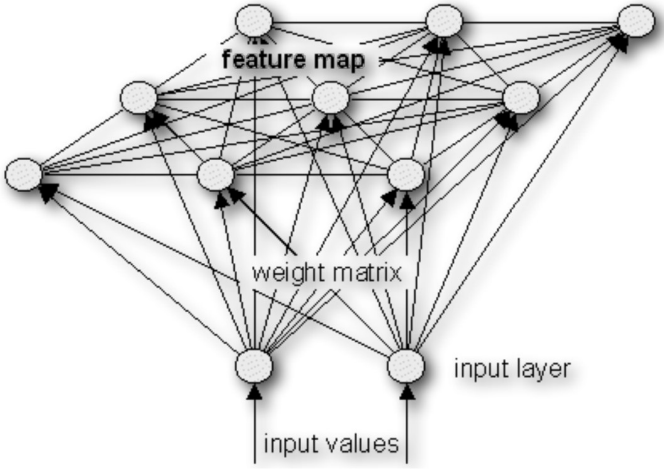


Figura 5.7: Rete neurale mappa di Kohonen. Durante il processo di apprendimento, i neuroni sulla mappa si organizzano in funzione dei valori di ingresso. Questo si traduce in una struttura neuronale clusterizzata, in cui i neuroni con proprietà (valori) simili si dispongono in settori connessi sulla mappa.

5.2.3 Processo di apprendimento

Questa sezione descrive le capacità di apprendimento delle reti neurali. In primo luogo, viene illustrato il termine apprendimento, successivamente, viene presentata una panoramica di specifici algoritmi di apprendimento per reti neurali.

Nel cervello umano, l'informazione viene passata tra i neuroni in forma di stimolazione elettrica lungo i dendriti, se una certa quantità di stimolazione viene ricevuta da un neurone, esso genera un output verso tutti gli altri neuroni connessi e così le informazioni prendono la loro strada verso la destinazione, dove si verificherà qualche reazione. Se lo stimolo in entrata è troppo basso, nessuna uscita è generata dal neurone e il successivo trasporto delle informazioni sarà bloccato.
Spiegare come il cervello umano impari certe cose è molto difficile e nessuno lo sa esattamente, si suppone che durante il processo di apprendimento la struttura di collegamento tra i neuroni venga modificata, in modo che certe stimolazioni sono accettate solo da certi neuroni. Ciò significa che esistono solidi legami tra le cellule neurali una volta che hanno imparato un fatto specifico, consentendo il richiamo rapido delle informazioni, se informazioni analoghe arrivano successivamente vengono stimolate le stesse cellule nervose, che adatteranno la loro struttura di connessione in

funzione di queste nuove informazioni. D'altra parte, se una informazione specifica non viene richiamata per molto tempo, la struttura di connessione stabilita tra le cellule neurali responsabili diverrà più "debole". Questo accade ad esempio quando si "dimentica" un fatto in precedenza acquisito o lo si ricorda solo vagamente.
Come accennato prima, le reti neurali cercano di simulare la capacità di apprendimento del cervello umano. Infatti, la rete neurale artificiale è fatta anch'essa di neuroni e dendriti. A differenza del modello biologico, una rete neurale ha una struttura immutabile, costruita su un determinato numero di neuroni e un determinato numero di connessioni tra di loro (chiamate "pesi"), che hanno certi valori, ciò che cambia nel processo di apprendimento sono i valori di tali pesi.
Rispetto all'originale biologico questo significa che:

1. Le informazioni in arrivo "stimolano" (superano un determinato valore di soglia di) certi neuroni;

2. Questi passano le informazioni ai neuroni collegati o prevengono l'ulteriore trasporto lungo le connessioni "ponderate" in uscita;

3. Il valore di un peso sarà aumentato se le informazioni devono essere trasportate e ridotto in caso contrario.

Durante l'apprendimento di diversi input, i valori dei pesi vengono cambiati dinamicamente fino a quando non sono in equilibrio, in modo tale che ogni ingresso porterà all'output desiderato. L'addestramento di una rete neurale risulta in una matrice che contiene i valori dei pesi tra i neuroni. Una volta che una rete neurale è stata correttamente addestrata, sarà probabilmente in grado di trovare il risultato desiderato per un ingresso precedentemente appreso, utilizzando i valori della matrice dei pesi.
Perché probabilmente? Purtroppo non si può garantire che una rete neurale produca i risultati corretti in ogni caso. Molto spesso vi è un certo errore residuo dopo il processo di apprendimento, per cui l'output generato è solo una buona approssimazione dell'uscita perfetta nella maggior parte dei casi. Gli algoritmi di apprendimento utilizzati per istruire le reti neurali sono divisi in 3 categorie principali. La scelta di quale usare dipende dal campo di applicazione per cui la rete è progettata e dalla sua tipologia (feedforward o feedback). Gli algoritmi sono:

- Supervisionato;

- Non supervisionato;

- Di rinforzo.

Apprendimento supervisionato

Una rete neurale è supervisionata, se l'output desiderato è già noto.
Facciamo un esempio di un problema di pattern recognition per capire meglio di cosa si tratta; supponiamo che, una rete neurale debba imparare ad associare le coppie di modelli in figura 5.8. I modelli d'ingresso (input pattern) sono numeri decimali, ognuno rappresentato da una sequenza di bit. I modelli di uscita (output pattern) sono forniti in forma di valori binari dei numeri decimali. I passaggi che compie la rete sono i seguenti:

input pattern	output pattern
0001	001
0010	010
0100	011
1000	100

Figura 5.8: Pattern di input e di output dati come training alla rete di esempio.

1. Durante l'apprendimento, uno dei pattern di input viene "presentato" al livello di input della rete;

2. Questo pattern si propaga attraverso la rete (indipendentemente dalla sua struttura) sino al livello di uscita;

3. Lo strato di uscita genera un pattern di output che viene poi confrontato con il pattern reale. A seconda della differenza tra la il risultato prodotto dalla rete e l'obiettivo, viene calcolato un valore di errore;

4. Questo errore indica lo sforzo di apprendimento della rete, che può essere controllato dal "supervisore immaginario";

5. Quanto maggiore è il valore di errore calcolato, tanto più i valori dei pesi verranno modificati.

Apprendimento non supervisionato

Le reti neurali non supervisionate (cioè che "imparano" senza supervisione) non hanno uscite predefinite, il risultato del processo di apprendimento non può essere determinato a priori. Durante il processo di apprendimento, le unità (i pesi) della rete neurale vengono "disposte" all'interno di un determinato intervallo di valori, a seconda dei valori in ingresso; l'obiettivo è quello di raggruppare insieme unità simili, "vicine" in certe aree del range di valori. Questo effetto può essere utilizzato in modo efficace ai fini della classificazione di modelli (clustering).

Forward propagation

Si tratta di un algoritmo di apprendimento supervisionato che descrive il "flusso di informazioni" attraverso una rete neurale dal suo strato di ingresso al suo livello di output. L'algoritmo funziona nel modo seguente:

1. Impostare tutti i pesi a valori casuali compresi tra -1.0 e +1.0;

2. Fornire un pattern di ingresso (valori binari) ai neuroni del livello di input;

3. Attivare ogni neurone dello strato successivo nel modo seguente:

- moltiplicare i valori di peso delle connessioni che portano al neurone con i valori di output dei neuroni precedenti;
- sommare questi valori;
- passare il risultato ad una funzione di attivazione, che calcola il valore di uscita del neurone.

4. Ripetere questa operazione fino a quando lo strato di uscita non viene raggiunto;
5. Confrontare il pattern di uscita calcolato con il pattern previsto e calcolare l'errore;
6. Modificare tutti i pesi aggiungendo il valore di errore ai (vecchi) valori di peso;
7. Tornare al punto 2;
8. L'algoritmo termina se tutti i pattern di output corrispondono ai loro pattern previsti.

Backward propagation

Il Backpropagation è un algoritmo di apprendimento supervisionato ed è principalmente usato dal Multi-Layer Perceptron per cambiare i pesi collegati agli strati di neuroni nascosti. L'algoritmo di backpropagation utilizza l'errore di output per cambiare i valori dei pesi a ritroso. Per ottenere l'errore deve essere stata precedentemente eseguita una fase "forward propagation", durante la quale i neuroni vengono attivati utilizzando la funzione di attivazione sigmoidea. La formula dell'attivazione sigmoidea è $f(x) = \frac{1}{1+e^{-x}}$, l'algoritmo funziona nel modo seguente:

1. Eseguire la fase di forward propagation per un pattern di input e calcolare l'errore di uscita;
2. Cambiare tutti i valori di ogni matrice dei pesi utilizzando la formula: (vecchio) peso + tasso di apprendimento × errore di output × neurone di output (i) × neurone di output (i+1) × [1− neurone di output (i+1)];
3. Tornare al punto 1;
4. L'algoritmo termina se tutti i pattern di output corrispondono ai target.

Self organization

"Self organization" (auto-organizzazione) è un algoritmo di apprendimento non supervisionato utilizzato dalle reti neurali con mappa caratteristica di Kohonen (Kohonen Feature Map). Come accennato nei paragrafi precedenti, una rete neurale cerca di simulare il cervello biologico umano, e l'auto-organizzazione è probabilmente il modo migliore per rendersene conto. È comunemente noto che la corteccia del cervello umano è suddivisa in diverse regioni, ognuna responsabile di determinate funzioni. Le cellule neurali si organizzano in gruppi, secondo le informazioni in arrivo. Le informazioni in input non solo sono ricevute dalle singole cellule neurali,

ma influenzano anche le altre cellule vicine. Questa organizzazione si traduce in una specie di mappa, nella quale le cellule neurali con funzioni simili vengono disposte in raggruppamenti omogenei (cluster).
Questo processo di auto-organizzazione può essere eseguito anche da una rete neurale. Questo tipo di rete neurale è per lo più utilizzato per la classificazione, perché i valori di input simili vengono agglomerati in zone della mappa. Una struttura tipica di una mappa di Kohonen che utilizza l'algoritmo self organization è mostrata in figura 5.7, come si può vedere, ogni neurone del livello di input è collegato ad ogni neurone sulla mappa. La matrice dei pesi risultante è usata per propagare i valori di input della rete ai neuroni della mappa. Inoltre, tutti i neuroni nella mappa sono collegati tra loro, questi collegamenti vengono utilizzati per influenzare i neuroni in una certa area di attivazione intorno al neurone con la massima attivazione, ricevuta dall'uscita del livello di input.
La quantità di feedback tra i neuroni della mappa viene di solito calcolata mediante la funzione di Gauss:

$$\text{feedback}_{ci} = e^{\frac{-|x_c - x_i|^2}{2r^2}} \tag{5.1}$$

Dove:

- xc è la posizione del neurone più attivato (centroide);
- xi sono le posizioni degli altri neuroni della mappa;
- r è l'area di attivazione (raggio).

Inizialmente, l'area di attivazione è di grandi dimensioni come anche il feedback tra i neuroni della mappa, ciò si traduce in una attivazione dei neuroni in una vasta area intorno al neurone più attivo. Con il progredire dell'apprendimento, l'area di attivazione viene costantemente diminuita e solo i neuroni più vicini al centro di attivazione sono influenzati dal neurone più attivo.
A differenza del modello biologico, i neuroni della mappa non cambiano la loro posizione sulla mappa. L' "organizzazione" è simulata modificando i valori nella matrice dei pesi (allo stesso modo delle altre reti neurali). Poichè questo è un algoritmo di apprendimento non supervisionato, non vi sono pattern di ingresso/target preesistenti. I valori in input alla rete sono presi da un range predefinito e rappresentano i "dati" che devono essere organizzati. L'algoritmo "self organization" funziona nel modo seguente:

1. Definire l'intervallo dei valori di input;
2. Impostare tutti i pesi a valori casuali presi nel campo dei valori di ingresso;
3. Definire l'area iniziale (raggio) di attivazione;
4. Prendere un valore a caso dal set di input e passarlo ai neuroni dello strato di input;
5. Determinare il neurone più attivo sulla mappa:
    - moltiplicare l'uscita del livello di input con i valori dei pesi;

- il neurone mappa con il maggior valore risultante è detto essere “il più attivo”;
- calcolare il valore di feedback di ogni altro neurone mappa utilizzando la
- funzione di Gauss.

6. Ottenere i nuovi valori dei pesi w_i+1 con la formula: w_i+1 = w_i + feedback × (valore di ingresso - w_i) × tasso di apprendimento;
7. Ridurre l’area di attivazione;
8. Tornare al punto 4;
9. L’algoritmo termina se l’area di attivazione è più piccola di un valore specificato.

5.3 Applicazioni del deep learning

Nonostante le problematiche che abbiamo illustrato, i sistemi di Deep Learning hanno compiuto enormi passi evolutivi e sono migliorati moltissimo negli ultimi anni, soprattutto per la grandissima quantità di dati a disposizione, ma soprattutto per la disponibilità di infrastrutture ultra performanti (CPU e GPU in particolare). Nell’ambito della ricerca sull’Artificial Intelligence, l’apprendimento automatico ha riscosso un notevole successo negli ultimi anni, consentendo ai computer di superare o avvicinarsi alle prestazioni umane corrispondenti in aree che vanno dal riconoscimento facciale al riconoscimento vocale e linguistico. L’apprendimento profondo invece consente ai computer di fare un passo in avanti, in particolare di risolvere una serie di problemi complessi.
Un campo dove tale metodologia può dare risultati importanti è senza dubbio quello della diagnostica medica. L’applicazione del concetto delle reti neurali in tale ambito è molto semplice perché i medici, molto spesso, si servono già degli algoritmi, soprattutto in ambito specialistico. Quando un dottore formula una diagnosi lo fa basandosi sulle sue conoscenze e sull’esperienza, ovvero quel bagaglio culturale accumulato negli anni. Il deep learning potrebbe intervenire con successo proprio in questo punto, ampliando e migliorando l’ambito delle conoscenze del medico. L’applicazione dell’apprendimento approfondito spazia con successo dai programmi finalizzati alla diagnostica medica fino al controllo di qualità nelle produzioni farmaceutiche. Un campo affascinante relativo alla possibile applicazione del deep learning è quello relativo alla guida automatica. Siamo ancora molto lontani dalla commercializzazione di automobili completamente automatiche, ma i prototipi elaborati fanno ben sperare. La guida automatica consente di riconoscere gli ostacoli in entrambi i lati della carreggiata, grazie all’ausilio di sensori e telecamere in grado di elaborare le immagini. La computer vision in questo caso riproduce la vista umana, riconoscendo l’ambito nel quale si sta muovendo e fornendo tutte le indicazioni utili per muoversi in sicurezza. La spinta in avanti per la realizzazione di automobili automatiche è stata data dalla possibilità, offerta dal deep learning, di elaborare ben 20 miliardi di operazioni al secondo. Immaginiamo una grande azienda che produce un quantitativo considerevole di prodotti sui quali ha bisogno di effettuare un controllo di qualità. Nella piccola distribuzione il sistema è semplice: l’operatore umano,

attraverso il tatto e la vista, controlla i vari prodotti. Nella grande distribuzione, dove il ritmo è alto e i quantitativi da controllare enormi, il deep learning può essere l'arma vincente. Le reti neurali permettono di passare al vaglio dell'Intelligenza artificiale qualità, difetti, standard sbagliati e così via in un lasso di tempo molto breve.

5.3.1 Deep learning e computer vision

Il deep learning viene in aiuto ai task di visione artificiale in diverse applicazioni pratiche. Possiamo identificare alcune macro categorie [5] di problemi che è quasi sempre più efficiente risolvere tramite l'applicazione di una rete neurale piuttosto che tramite algoritmi "tradizionali", di seguito le introduco.

Classification

Le reti neurali possono essere utilizzate per il riconoscimento di oggetti, permettendo così di risolvere problemi di classificazione. La suddivisione può avvenire tra due o più classi, senza che questo abbia effetti sull'efficienza del software. Casi pratici per questa applicazione possono essere lo smistamento di prodotti aventi caratteristiche differenti oppure il controllo qualità, dove la rete neurale impara a riconoscere i pezzi difettosi.

Object detection

Un altro task in cui può essere sfruttato l'approccio deep learning è la guida robot, infatti esistono reti neurali specializzate nella localizzazione di uno o più tipologie di oggetti. L'individuazione del prodotto avviene tramite bounding box che possono anche essere orientate, ossia indicare la direzione in cui è disposto l'oggetto.

Anomaly/defeat detection

Anche la ricerca di anomalie può essere un ottimo caso per l'uso di una rete neurale. Infatti esistono alcuni modelli capaci di imparare e generalizzare le caratteristiche dei prodotti conformi ad un certo standard individuando la presenza di eventuali anomalie. Un altro caso analogo può essere la detection di anomalie all'interno di un pattern sia regolare che irregolare di una superficie.

Segmentation

Con il termine segmentazione si intende la mappatura dell'immagine attraverso l'individuazione dei contorni dei vari oggetti in essa rappresentati e un'eventuale successiva classificazione. Queste tipologie di reti neurali possono essere utilizzate per fare la suddivisione di una superficie in base alle sue caratteristiche oppure l'elaborazione di alcune parti dell'immagine riconosciute come simili dalla rete.
Vedremo più nel dettaglio la segmentazione tramite reti neurali nel capitolo 6, si tratta proprio dell'applicazione utilizzata nel progetto di cui questa tesi.

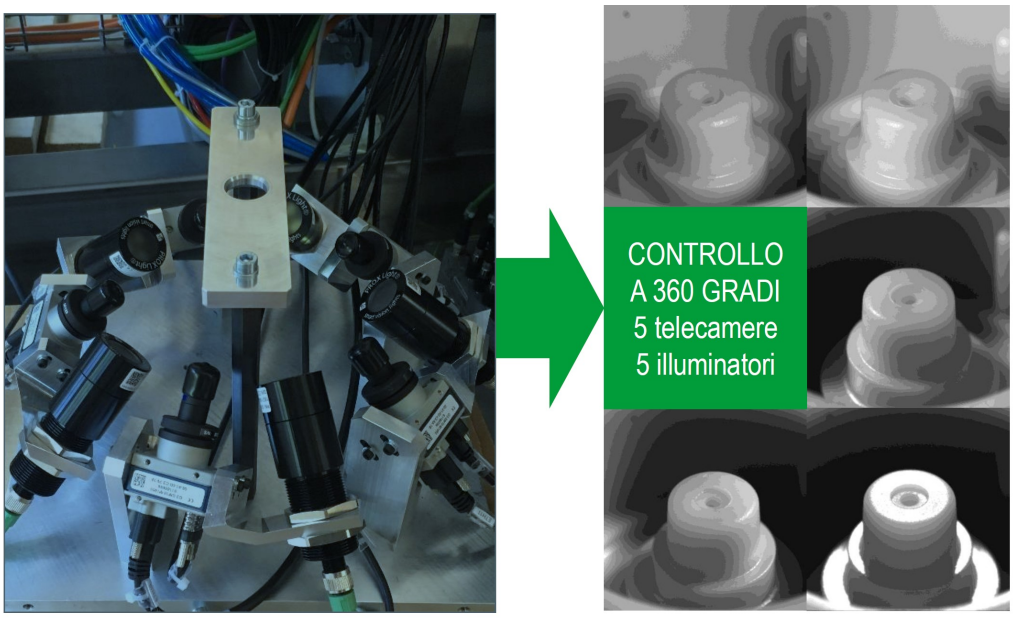


Figura 5.9: Sistema di visione a sinistra ed esempi di immagini scattate dalle telecamere a destra.

5.3.2 Stato dell'arte

Come abbiamo visto le applicazioni del deep learning alla computer vision sono molteplici, per completezza vediamo brevemente un progetto portato a termine dal team di Specialvideo per una commessa legata alla defeat detection in filtri automobilistici.

Filtri automobilistici

Questo progetto nasce da una commessa arrivata da un cliente, il problema consiste nel rilevare tutti i possibili difetti su dei filtri automobilistici durante la catena di produzione. Per fare ciò Specialvideo ha realizzato un sistema di visione artificiale che implementa 3 reti neurali volte alla identificazione dei difetti, il sistema è composto da 5 telecamere e 5 illuminatori . La disposizione in senso circolare e la loro inclinazione rispetto all'oggetto consentono il controllo sull'intera superficie, vediamo un esempio in figura 5.9. Essendo i filtri oggetti molto piccoli, per ottenere le fotografie utili all'addestramento delle reti sono state utilizzate diverse modalità di illuminazione pur mantenendo le telecamere ferme. Se consideriamo le 5 telecamere ad angolo 0° sono state acquisite 5 immagini per ogni filtro modificando l'angolazione dell'illuminazione in questo modo:

- Frontale sinistra: -36°;
- Frontale destra: 36°;
- Posteriore destra: 108°;
- Posteriore: 180°;
- Posteriore sinistra: -108°.

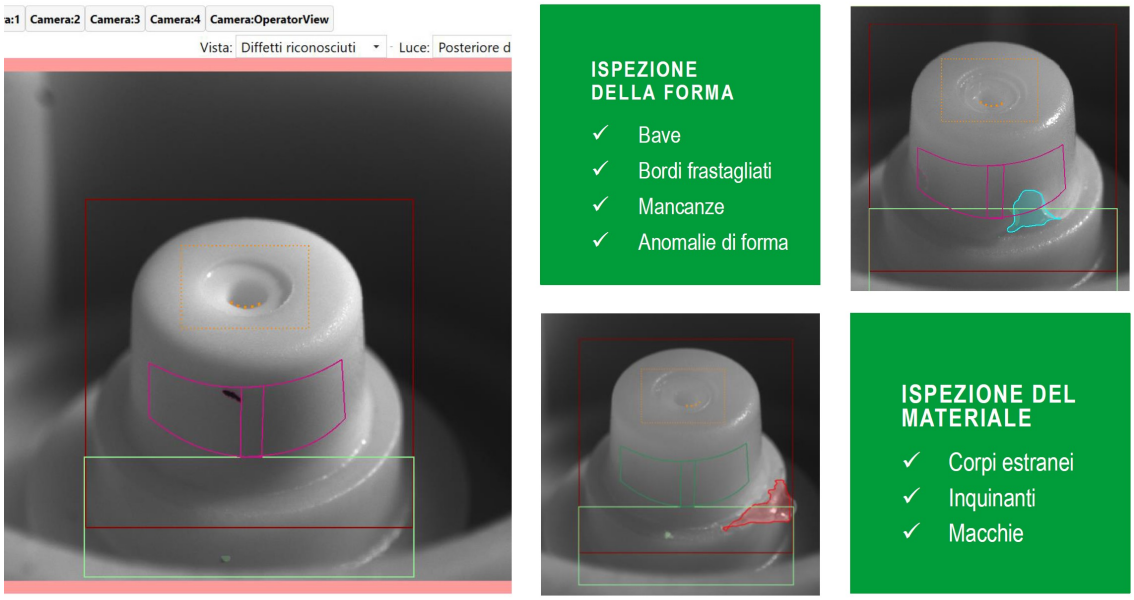


Figura 5.10: Sistema di visione a sinistra e esempi di immagini scattate dalle telecamere a destra.

Vediamo un esempio delle immagini scattate dalle telecamere in figura 5.9.
Per questo progetto sono state usate 3 reti neurali con architettura U-Net, di cui parleremo dettagliatamente nel capitolo 6, a ognuna di queste reti vengono fornite in input le 5 immagini del filtro. Le funzioni delle reti sono le seguenti:

1. La prima restituisce due classi, indicanti se manca del silicone (chiamate mancanze e buchi);

2. La seconda indica se c'è del silicone di troppo sui bordi (chiamate bave);

3. La terza indica se nella parte in basso dove c'è il supporto di plastica c'è sporcizia.

Infine in figura 5.10 vediamo uno screenshot del software realizzato per la visualizzazione dei difetti identificati dalle reti neurali.

Capitolo 6

Seconda parte: segmentazione delle immagini

6.1 Analisi del problema

Questa seconda parte del progetto tratta per prima cosa del lavoro di creazione e adattamento del dataset di immagini stereo raccolte inizialmente, al fine di renderlo utilizzabile per il training e il testing della rete neurale volta alla segmentazione delle immagini, ovvero identificare la parte dei prosciutti dove dovrà poi andare applicata la sugna. Inoltre dopo aver esaminato il processamento dei dati analizzerò le scelte fatte riguardo la rete neurale: la struttura, i vari parametri, le funzioni utilizzate e così via. Infine vedremo e analizzeremo i risultati della rete neurale sul dataset utilizzato come test.

Quando si vuole utilizzare una rete neurale per risolvere un task in modo efficace bisogna preoccuparsi prima di tutto di fornirle i dati per il suo addestramento nel modo migliore possibile, così da sfruttarne appieno il potenziale.
Per " modo migliore" si intende riuscire a processare i dati "grezzi" ovvero quelli raccolti inizialmente, ad esempio immagini, dati statistici ecc. per renderli "leggibili" dalla rete e darle la possibilità di imparare da essi.

Nel nostro caso abbiamo a che fare con delle immagini raccolte da un sistema di telecamere ad alta definizione, quindi la prima cosa che ho fatto è stata scegliere quale vettore di immagini utilizzare per il training della rete, quello derivante dalla telecamera di sinistra oppure da quella di destra. La scelta in questo caso si è basata semplicemente sulla qualità delle immagini, ho optato per il vettore di immagini derivante dalla telecamera di destra perchè il soggetto, ovvero il prosciutto, risultava più centrato rispetto a quelle di sinistra.
In secondo luogo ho analizzato l'organizzazione del dataset di immagini fornitomi, il quale era composto da: 194 cartelle identificate da un numero sequenziale con all'interno di ognuna le due immagini stereo identificate dal rispettivo sequenziale e alcuni file non interessanti per il nostro fine.
Infine, prima di organizzare le immagini per il training della rete, ho dovuto decidere quale rete neurale utilizzare e con quale strategia di apprendimento affrontare il problema, perchè su questa scelta si basa il preprocessamento dei dati.
Queste scelte sono state fatte insieme al team di Specialvideo, che grazie alla sua esperienza nel settore ha saputo guidarmi nel primo approccio al problema. La rete

Figura 6.1: Esempio di segmentazione in cui vengono classificati tutti gli elementi della scena.

neurale scelta per affrontare il task di semantic segmentation è **U-net** sfruttando un **apprendimento supervisionato**, spiegato nel paragrafo 5.2.3.

Di seguito giustificherò le scelte fatte e illustrerò ciò che è stato svolto anche con qualche cenno teorico.

6.2 Semantic segmentation

L'obiettivo della segmentazione semantica di un'immagine è etichettare ogni pixel con una classe o label corrispondente a ciò che viene rappresentato. Una cosa importante da notare è che non stiamo separando istanze della stessa classe; ci interessa solo la categoria di ogni pixel. In altre parole, se hai due oggetti della stessa categoria nell'immagine di input, la mappa di segmentazione non li distingue intrinsecamente come oggetti separati.
Vediamo un esempio di segmentazione in figura 6.1 utilizzata per la guida autonoma. Semplicemente, l'obiettivo è quello di prendere un'immagine a colori RGB (altezza $\times$ larghezza $\times$ 3) o un'immagine in scala di grigi (altezza $\times$ larghezza $\times$ 1) e produrre una mappa di segmentazione (figura 6.2) in cui ogni pixel contiene un'etichetta rappresentata come un numero intero (altezza $\times$ larghezza $\times$ 1). Viene quindi creato un canale di output per ciascuna delle classi possibili; per visualizzare una mappa di segmentazione come quella in figura 6.2 basterà sovrapporre tutti gli output e prendere per ogni pixel solamente il valore più alto. Per problemi di semantic segmentation l'approccio tipico è quello di impiegare le **reti neurali convoluzionali** per produrre la mappa di segmentazione.

6.3 Reti neurali convoluzionali: CNN

Le reti neurali convoluzionali (Convolutional Neural Networks o ConvNets) sono reti neurali profonde realizzate per operare su input a griglia caratterizzati da forti dipendenze spaziali in regioni locali. Un esempio di input a griglia è un'immagine

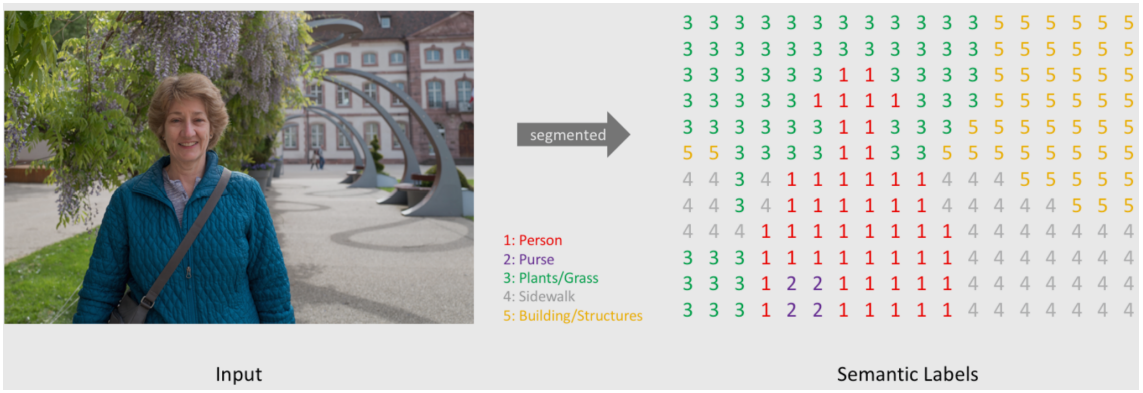


Figura 6.2: Esempio di segmentazione in cui vengono classificati tutti gli elementi della scena, ovviamente per semplificare non sono stati inseriti tutti i pixel nell'immagine di segmentazione.

bidimensionale: essa non è altro che una matrice di valori compresi fra 0 e 255. Pixel adiacenti presenti in un'immagine sono interlacciati l'uno all'altro e nel loro insieme definiscono un pattern, una texture, un contorno, etc. Le CNN associano queste caratteristiche a valori detti "pesi", che saranno simili per regioni locali con pattern simili. La qualità che le contraddistingue è anche l'operazione che da loro il nome, ossia la convoluzione. Essa non è altro che il prodotto scalare tra matrici, nello specifico tra una struttura a griglia di pesi e una simile struttura proveniente dall'input. Le reti convoluzionali sono deep network in cui lo strato convoluzionale appare almeno una volta, benchè la maggioranza ne usi ben più di uno.
L'input di una ConvNet è un'immagine, ossia una matrice di valori di ogni singolo pixel occupante una precisa posizione all'interno dell'immagine. Le immagini in formato RGB sono descritte da una terna di matrici delle intensità dei colori primari (rosso, verde, blu); le dimensioni di un'immagine non sono quindi limitate ad altezza e larghezza ma ve n'è una terza: depth, profondità. Le dimensioni dell'immagine, inclusa la profondità, saranno conferite allo strato di input, il primissimo layer di una CNN. Le mappe di attivazione successive, ossia gli input degli strati successivi, hanno anch'esse una struttura a più dimensioni e saranno in numero congruente con le proprietà indipendenti rilevanti alla classificazione.

L'insieme di matrici degli strati nascosti (hidden), risultato della convoluzione o di altre operazioni, è definita feature map o activation map, i parametri addestrabili sono tensori denominati filtri o kernel. Una CNN è composta da successioni di strati, illustrati in dettaglio nei paragrafi seguenti. Gli strati principali sono:

- strato convoluzionale;
- strato di attivazione;
- strato di pooling.

6.3.1 Strato convoluzionale

La convoluzione è l'operazione fondamentale delle CNN. Essa dispone un filtro in ogni possibile posizione dell'immagine, coprendola interamente e computa il prodotto scalare tra il filtro stesso e la matrice corrispondente del volume di input, avente

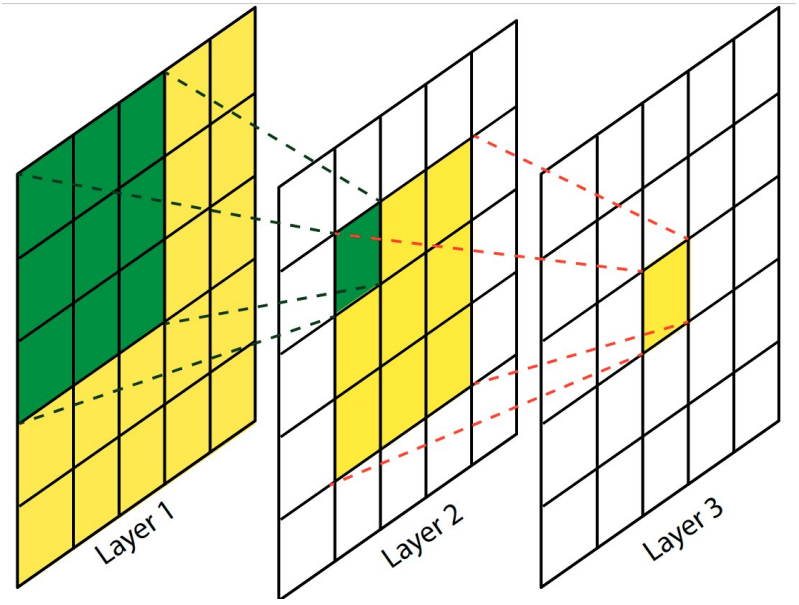


Figura 6.3: Confronto tra due convoluzioni consecutive con dimensioni kernel di 3 × 3 (dallo strato 1 allo strato 2 in verde, e poi dallo strato 2 allo strato 3 in giallo) e una singola convoluzione con dimensione kernel di 5 × 5 (direttamente dallo strato 1 allo strato 3 in giallo). Si noti che il quadrato centrale del livello 3 ha lo stesso campo visivo nel livello 1 (tutti i 25 quadrati) indipendentemente da quale delle convoluzioni consideriamo.

eguale dimensioni. È possibile visualizzare la convoluzione come una sovrapposizione del kernel sull'immagine in input (o strato nascosto). Un filtro è caratterizzato dai seguenti iperparametri (hyperparameters):

- Altezza;
- Larghezza;
- Profondità;
- Numero.

Dimensione del kernel di convoluzione

La dimensione del kernel è uno dei principali iperparametri, con la sua dimensione stiamo specificando anche la sua forma. Aumentare la dimensione del kernel influenzerà la complessità totale del modello, poiché aumenteremo il numero di pesi che dovremo allenare. Nella pratica comune il kernel ha solitamente la forma di un quadrato/cubo e il lato è un numero intero positivo dispari. La ragione di ciò è per evitare ambiguità in cui l'output pixel (quello al centro del quadrato/cubo, per i kernel di dimensioni dispari) corrisponderà al pixel di input. Inoltre, nell'ultimo decennio abbiamo osservato un trend decrescente nella dimensione del kernel. Questo fatto può essere spiegato confrontando due convoluzioni 3 × 3 consecutive con una singola convoluzione 5 × 5, come in figura 6.3 [17]. Chiamiamo campo ricettivo il numero di input da cui dipende il nostro valore di output centrale. Ovviamente è

possibile definire il campo ricettivo dopo convoluzioni composte, con la conseguenza che dovremo tenere conto delle sovrapposizioni. Il motivo per cui confrontiamo questi due casi è perché sono due modi diversi per ottenere lo stesso campo recettivo. Fondamentalmente, stiamo confrontando una struttura più profonda (più strati) con una struttura più "larga" (più parametri). Il primo caso dovrà addestrare $2 \times (3 \times 3) = 18$ parametri, mentre il secondo dovrà addestrare $5 \times 5 = 25$ parametri. Pertanto, anche avendo lo stesso campo ricettivo, è preferibile l'utilizzo di due kernel più piccoli in termini di complessità computazionale. Inoltre, la presenza di due strati andrà a "stratificare" in un certo senso l'effetto degli input, dando maggiore importanza a quelli più vicini al centro (rilevati e utilizzati dalla prima 3×3 convoluzione). Oltre a questi due buoni effetti nell'utilizzo di strutture più profonde ma più strette, c'è anche uno svantaggio negativo. L'iterazione di due circonvoluzioni 3×3 consecutive non può generare alcun kernel 5×5 possibile, e questo è logicamente riflesso dal numero di parametri precedentemente confrontati. Anche date queste considerazioni precedenti, di solito dovrebbe essere preferibile usare 3×3 kernel, poiché sono i kernel di dimensioni dispari più piccoli che aumentano il campo recettivo. Ad ogni modo, è comune vedere anche kernel 1×1 (il motivo sarà discusso nella sezione dei filtri) e 5×5 kernel, mentre è sempre più raro vedere kernel più grandi di 7×7 anche tra le reti più performanti.

Numero dei filtri

Il termine filtro è talvolta usato come sinonimo di kernel nella convoluzione delle immagini. Questo iperparametro viene utilizzato per eseguire più convoluzioni nello stesso livello. Il numero di filtri può anche essere pensato come il numero di canali nelle operazioni relative alle immagini, ad esempio, se vogliamo avere un'immagine colorata come output finale di uno strato convoluzionale, dobbiamo fissare il suo numero di filtri a 3, che può essere interpretato come i canali RGB nell'ordine che preferiamo. Più in generale, possiamo pensare al numero di filtri come al numero di neuroni che avrà lo strato convoluzionale, ogni neurone eseguirà una convoluzione possibilmente diversa cambiando i pesi del suo kernel e apprendendo quindi una caratteristica diversa che verrà utilizzata dagli strati successivi. Per questo motivo, ogni volta che si ha a che fare con un kernel n dimensionale, l'output di uno strato convoluzionale sarà (n + 1) dimensionale per la presenza dell'iperparametro del numero dei filtri.

Strides e padding

Consideriamo ora strides e padding insieme, perché sono entrambi legati al "movimento" del kernel sull'input. Strides di un kernel n-dimensionale sono una sequenza n-dimensionale di numeri interi, che specifica quante posizioni sono necessarie per spostarsi lungo una data direzione prima di eseguire nuovamente l'operazione di convoluzione. Ricordando la definizione che abbiamo dato di convoluzione, fondamentalmente **strides = 1** significa *"eseguire l'operazione di convoluzione per ogni possibile (x, y) nell'input"*. **Strides = 2** significherà *"eseguire la convoluzione solo se (x, y) una volta ogni due posizioni consecutive"*, quindi stiamo sostanzialmente saltando metà delle possibili combinazioni ed eseguendo la convoluzione solo quando (x, y) sono entrambi dispari ((1,1) , (1,3), (3,1), (3,3), ...). Abbiamo detto che strides

hanno la stessa dimensionalità del suo kernel, perché è possibile specificare differenti lunghezze di strides per ogni dimensione del kernel, quindi per una convoluzione 2D può essere possibile specificare strides = (2,3), che significa: *"Eseguire nuovamente la convoluzione solo dopo essersi spostati di 2 posizioni sulla prima dimensione o di 3 posizioni sulla seconda dimensione"*. L'iperparametro strides influenzerà la dimensione e la forma dell'output, maggiore viene impostato e minore è l'output. L'iperparametro padding è quello usato per affrontare i problemi dei contorni delle matrici, a cui ci si riferisce quando il kernel sborda dai contorni dell'input. Le opzioni più popolari sono: **valid** and **same**. Il padding valid significa che ogni volta che il kernel sborda dall'input, quella singola operazione verrà saltata, risultando in un output di dimensione inferiore rispetto all'input (se il kernel ha una dimensione maggiore di 1). Padding same significa idealmente "mantenere la stessa dimensione". Anche se questa conclusione non è garantita (poiché altri parametri possono comunque ridurre la dimensione), lo scopo è quello di evitare di saltare la posizione a causa di ambiguità su come trattare i bordi. Pertanto l'opzione padding = same andrà ad aggiungere, in qualsiasi posizione richiesta ma non esistente nell'input, una posizione immaginaria con la minore interferenza possibile in termini di valori. Il valore all'interno della posizione immaginaria dipenderà dall'operazione, per le convoluzioni vengono inseriti zeri, per il min o max pooling $\pm\inf$. Fondamentalmente si cercherà di inserire il valore più neutro.

6.3.2 Funzione di attivazione

Questa sezione non riguarda un vero e proprio iperparametro, ma esattamente come per un iperparametro, la scelta della funzione di attivazione avviene dopo un primo approccio al problema in base allo scopo e alla situazione generale. Per funzione di attivazione definiamo una funzione $f : \mathbb{R} \to \mathbb{R}$ che prende come input il risultato di una convoluzione per un singolo neurone e fornisce come output l'output del neurone. Questo comportamento di regolazione dell'attivazione di un neurone è ciò che dà il nome alla funzione. Tra le funzioni di attivazione possibili la più usata è sicuramente ReLU.

ReLU

Rectifier Linear Unit (ReLU), ne vediamo il grafico in figura 6.4. Viene definita come:

$$f(x) = \mathrm{ReLU}(x) = \max(0, x) = \begin{cases} 0 & \text{if } x < 0 \\ x & \text{if } x \geq 0 \end{cases}$$
$$f'(x) = \mathrm{ReLU}'(x) = \begin{cases} 0 & \text{if } x < 0 \\ 1 & \text{if } x \geq 0 \end{cases} \tag{6.1}$$

Una delle funzioni di attivazione maggiormente utilizzate nel contesto deep learning [18].
Uno dei suoi maggiori vantaggi è essere molto semplice e quindi più veloce da calcolare, ciò porta a una formazione più rapida e quindi a un apprendimento più rapido e prestazioni migliori per la CNN [19]. Inoltre, la derivata di ReLU è costantemente uguale a 1 se l'input è positivo, ciò evita il fatto che il gradiente diventi sempre

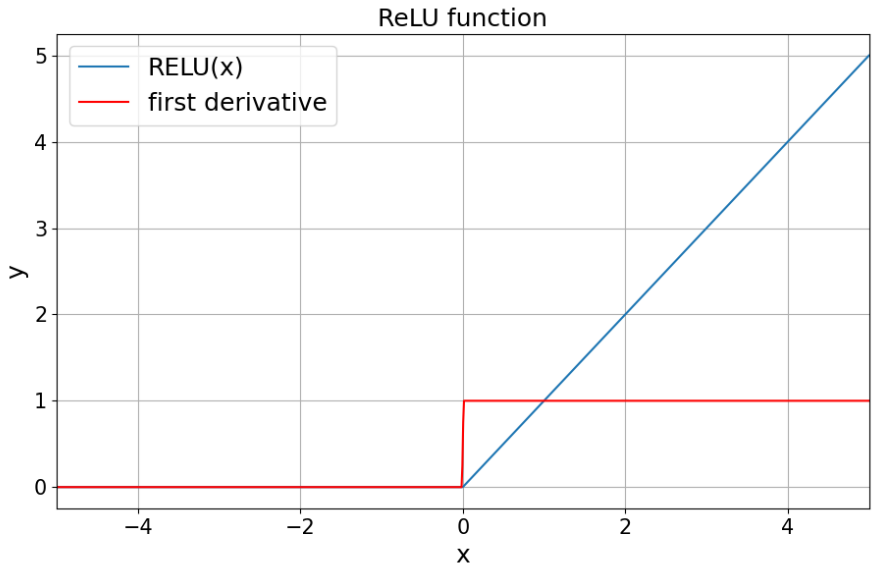


Figura 6.4: Funzione di attivazione ReLU e sua derivata prima, tracciati per un intervallo di input da -5 a +5.

più piccolo tra gli strati durante la backpropagation, e quindi risolve il problema dell'annullamento del gradiente. Alcuni inconvenienti sono legati al fatto che ogni input non positivo è mappato a 0, evitando la possibilità di trattare in modo diverso i valori negativi, evidente nel grafico in figura 6.4. Se da un lato la possibilità di produrre uno zero reale porta più facilmente ad una rappresentazione sparsa dei dati strato per strato (grazie alla presenza di molti zeri), lo stesso effetto può portare alla "morte" di alcuni neuroni, ciò significa che alcuni neuroni non verranno mai aggiornati e utilizzati se finiscono in una configurazione di peso che fornisce loro input negativi, poiché la derivata di ReLU(x) è 0 for $x \leq 0$ e quindi l'algoritmo di backpropagation avrà effetto nullo su quel dato neurone. Quindi, una volta che il neurone diventa negativo, è molto improbabile che si riprenda da questa situazione [20][21].
Un altro grande svantaggio è il fatto che l'output medio è sempre positivo, portando a uno spostamento del bias per il livello successivo, processo che rallenta la formazione.

6.3.3 Strato di pooling

Il max-pooling estrae il massimo valore contenuto in matrici P $\times$ P di ogni mappa di attivazione e produce un altro hidden layer di eguale profondità. Anche in questo caso, come per la convoluzione, si fa uso di stride: se lo stride è 1 lo strato 2 così generato avrà dimensioni:

$$L_2 = L_1 - P + 1$$

$$B_2 = B_1 - P + 1 \tag{6.2}$$

$$d_2 = d_1$$

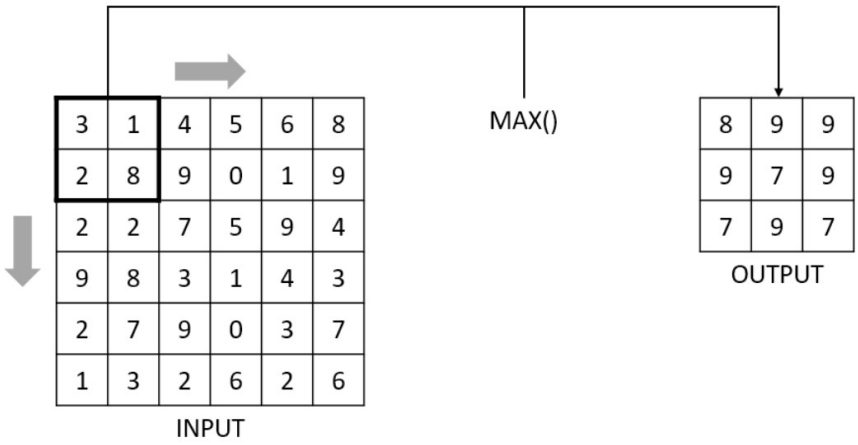


Figura 6.5: Esempio di max-pooling: dimensione 2, stride 2.

Dove L e B sono rispettivamente altezza e larghezza e d rappresenta la profondità. Per stride maggiori di 1, come solitamente si usa, altezza e larghezza saranno, rispettivamente:

$$L_2 = \frac{L_1 - P}{Stride} + 1$$
$$B_2 = \frac{B_1 - P}{Stride} + 1 \tag{6.3}$$

Rispetto alla convoluzione, il pooling è effettuato al livello di ciascuna mappa di attivazione, quindi il loro numero rimane inalterato e l'output è uno strato della stessa profondità (ma di altezza e larghezza differenti). Una configurazione tipica è dimensione 2×2 e stride 2: così facendo non c'è sovrapposizione tra regioni (figura 6.5).
Esistono altre varianti del pooling, meno diffuse, come ad esempio l'average-pooling, denominato subsampling, termine con il quale ora ci si riferisce ad una generica operazione di riduzione dell'impronta spaziale. L'uso dello stride nel pooling è importante per tre motivi. Il primo è la riduzione dell'impronta spaziale delle mappe di attivazione, il secondo è un certo grado di invarianza alla traslazione e il terzo è un incremento del campo ricettivo. Si osservi che per ridurre l'impronta spaziale possono essere utilizzati unicamente strati convoluzionali con stride maggiori di 1. Nonostante ciò, si preferisce tuttora utilizzare max-pooling o qualche altra variante dato il grado di non-linearità e invarianza alla traslazione che introducono.

6.3.4 Numero dei layer

Questo è uno degli iperparametri più importanti. Il suo utilizzo è abbastanza intuitivo, maggiore è il numero di livelli, maggiore è il numero di parametri nella rete e quindi maggiore è la complessità della CNN. Il solito compromesso è tra **underfitting** (a causa di una struttura a bassa complessità) e **overfitting** (causato da una CNN ad alta complessità che si avvicina a un problema più semplice).
Anche fissando il parametro principale relativo alla complessità della struttura, che è il numero totale di neuroni, c'è un altro compromesso che dobbiamo trovare, ovvero quello tra "larghezza" e "profondità". Consideriamo, per semplicità, il numero totale di neuroni in una CNN come sinonimo della complessità della CNN. Fissare

il numero totale di neuroni a N implica che abbiamo ancora 2^{N-1} possibili architetture tra cui scegliere. Solitamente, fisso N, è preferibile andare più in profondità che più in larghezza [22], il motivo risiede nella definizione di Deep Learning, poiché le caratteristiche o i concetti che la CNN imparerà sono costruiti in modo profondo. Pertanto, l'utilizzo di un singolo strato con N neuroni creerà N funzionalità semplici, mentre l'utilizzo di 2 livelli con $N/2$ neuroni creerà $N/2$ funzionalità più complesse combinando $N/2$ funzionalità più semplici. Per questi motivi, l'approccio è solitamente quello di regolare la maggior parte degli iperparametri su un paio di livelli, quindi aumentare il numero di livelli prestando attenzione all'overfitting e riducendo il numero di neuroni per strato se necessario. Ovviamente il numero di neuroni per strato non è costante ed è possibile cambiare la larghezza lungo la profondità degli strati. Le strutture più comuni sono di numero di neuroni costante o decrescente all'aumentare della profondità dello strato, ma è possibile avere anche l'opposto.

6.3.5 Optimizer

L'ottimizzatore è letteralmente l'algoritmo che useremo per aggiornare i nostri parametri CNN durante l'apprendimento. Inoltre, questa variabile non riguarda la nostra CNN, ma riguarda il processo di training, non è un iperparametro, ma si comporta come tale. Tutti gli ottimizzatori hanno istruzioni e calcoli molto diversi, portando a un diverso insieme di parametri regolabili. L'unico parametro che ogni ottimizzatore avrà di sicuro è il **learning rate**. Questo particolare parametro influisce sulla grandezza degli aggiornamenti ai parametri addestrabili della CNN. Fondamentalmente, dopo aver "allenato" la CNN sui dati di training, avremo una sorta di "errore" (che verrà discusso più avanti nella sezione riguardante la funzione di loss) maggiore è il tasso di apprendimento maggiore è l'entità degli aggiornamenti. Poiché l'apprendimento senza un ottimizzatore fornisce la direzione (nello spazio dei parametri allenabili) per il miglior aggiornamento, allora l'ottimizzatore ci dice di quanto spostarci lungo quella determinata direzione. La durata di questo passaggio dipenderà in qualche modo dal tasso di apprendimento scelto. Questo non sempre accade, alcuni ottimizzatori usano uno strumento matematico chiamato *momentum* che influenza anche la direzione menzionata, ma ci dà l'idea di quale sia il tasso di apprendimento. Considerando $\mathbf{p}$ come il vettore dei parametri addestrabili della CNN, X come dati, y come obiettivi e $f(\mathbf{p}; X, y)$ come loss scelta, analizzeremo di seguito alcuni degli ottimizzatori più popolari e spiegheremo brevemente i loro parametri interni.

Adam

Uno degli ottimizzatori più recenti e ancora ampiamente utilizzati è l'Adaptive Moment Estimation (ADAM). Formulato per la prima volta da Diederik P. Kingma e Jimmy Lei Ba nel 2015 [23], si dice che il metodo abbia pochi requisiti di memoria, sia computazionalmente efficiente e adatto a problemi che sono grandi in termini di

dati e/o parametri [23].

$$\mathbf{m_0} = \mathbf{v_0} = \mathbf{0} \tag{6.4}$$
$$\mathbf{m_{n+1}} = \beta_1 \mathbf{m_n} + (1 - \beta_1)\nabla_{\mathbf{p}}(f(\mathbf{p_n}; X, y)) \tag{6.5}$$
$$\mathbf{v_{n+1}} = \beta_2 \mathbf{v_n} + (1 - \beta_2)(\nabla_{\mathbf{p}}(f(\mathbf{p_n}; X, y)))^2 \tag{6.6}$$
$$\hat{\mathbf{m}}_{\mathbf{n+1}} = \frac{\mathbf{m_{n+1}}}{(1 - \beta_1^{n+1})} \tag{6.7}$$
$$\hat{\mathbf{v}}_{\mathbf{n+1}} = \frac{\mathbf{v_{n+1}}}{(1 - \beta_2^{n+1})} \tag{6.8}$$
$$\mathbf{p_{n+1}} = \mathbf{p_n} - l\frac{\hat{\mathbf{m}}_{\mathbf{n+1}}}{\sqrt{\hat{\mathbf{v}}_{\mathbf{n+1}}} + \epsilon} \tag{6.9}$$

Dove l rappresenta il learning rate, $(\mathbf{m}, \mathbf{v})$ sono i vettori del primo momento e del secondo momento grezzo, rispettivamente con i loro tassi di decadimento (β_1, β_2). Ciò che questi vettori cercano di stimare sono il primo momento (la media) e il secondo momento grezzo (la varianza non centrata) della funzione gradiente, attraverso una media mobile esponenziale su iterazioni passate. Inoltre, poiché vogliamo che (β_1, β_2) agiscano come tassi di decadimento, abbiamo $\beta_1, \beta_2 \in [0, 1[$. La versione senza cappello dei vettori momento corrisponde alla stima distorta, poiché i vettori sono inizializzati come pieni di zeri avremo una polarizzazione verso 0. La versione cappello indica la stima non distorta. Infine, ϵ viene utilizzato per evitare divisioni per 0 (che altrimenti si verificherebbero sempre nella prima iterazione) e poiché non ha significato matematico è consigliabile utilizzarne un valore molto piccolo. In letteratura ci sono alcuni valori suggeriti che sono altamente raccomandati [23]:

$$l = 0.001 \tag{6.10}$$
$$\beta_1 = 0.9 \tag{6.11}$$
$$\beta_2 = 0.999 \tag{6.12}$$
$$\epsilon = 10^{-8} \tag{6.13}$$

E' possibile ricercarne una migliore sintonia, con particolare riferimento al tasso di apprendimento. ADAM ha mostrato molti vantaggi in termini di proprietà di generalizzazione ed è risultato essere robusto e adatto a un'ampia gamma di problemi di ottimizzazione non convessi nel campo dell'apprendimento automatico [23]. D'altra parte, alcuni altri risultati hanno mostrato che ADAM non riesce a convergere su alcuni particolari problemi convessi unidimensionali semplici [24], dimostrando ancora una volta che non esiste un ottimizzatore universale per le attività di machine learning e deep learning.

Nadam

La versione Nesterov Accelerated di ADAM (NADAM) cerca di migliorare ulteriormente i risultati di ADAM. L'operazione di "indovinare" il prossimo aggiornamento e calcolare il gradiente su quella ipotesi si ottiene introducendo un programma di decadimento per un nuovo parametro μ che dipende da β_1. Riportiamo gli aggiornamenti implementati dall'algoritmo NADAM (per spiegazioni precise, vedere la

sezione Metodi di [Dozat, 2015] [25]).

$$\mathbf{m_0} = \mathbf{v_0} = \mathbf{0} \tag{6.14}$$

$$\mu_n = \beta_1(1 - 0.5 \cdot 0.96^{(n)/250}) \tag{6.15}$$

$$\mathbf{m_n} = \beta_1 \mathbf{m_{n-1}} + (1 - \beta_1)\nabla_{\mathbf{p}}(f(\mathbf{p_{n-1}}; X, y)) \tag{6.16}$$

$$\mathbf{v_n} = \beta_2 \mathbf{v_{n-1}} + (1 - \beta_2)(\nabla_{\mathbf{p}}(f(\mathbf{p_{n-1}}; X, y)))^2 \tag{6.17}$$

$$\hat{\mathbf{m}}_\mathbf{n} = \frac{\mathbf{m_n}}{(1 - \prod_{i=1}^{1=n} \mu_i)} \tag{6.18}$$

$$\hat{\mathbf{v}}_\mathbf{n} = \frac{\mathbf{v_n}}{(1 - \beta_2^n)} \tag{6.19}$$

$$\bar{\mathbf{m}}_\mathbf{n} = \frac{(1 - \mu_n)\nabla_{\mathbf{p}}(f(\mathbf{p_{n-1}}; X, y))}{(1 - \prod_{i=1}^{1=n} \mu_i)} + \mu_{n+1}\hat{\mathbf{m}}_\mathbf{n} \tag{6.20}$$

$$\mathbf{p_n} = \mathbf{p_{n-1}} - l\frac{\bar{\mathbf{m}}_\mathbf{n}}{\sqrt{\hat{\mathbf{v}}_\mathbf{n}} + \epsilon} \tag{6.21}$$

L'implementazione della versione Nesterov del momentum può essere riconosciuta nel passaggio 6.20 per il calcolo di $\bar{\mathbf{m}}_n$, dove ora la "supposizione" non è più eseguita nel calcolo del gradiente, ma aggiungendo la quantità di moto "successiva" $\mu_{n+1}\mathbf{m_n}$. L'autore di NADAM suggerisce un valore leggermente diverso per un parametro, che è $\beta_1 = 0,99$. Dal confronto tra le prestazioni di NADAM e ADAM, è ormai comune credere che l'applicazione Nesterov momentum porti in generale a risultati migliori rispetto al semplice momentum classico.

6.3.6 Funzione di loss

Quando abbiamo a che fare con la formazione della CNN abbiamo bisogno di un modo per esprimere quanto sono buoni i nostri risultati dopo ogni iterazione. Nell'apprendimento supervisionato abbiamo i dati target "veri" e vogliamo arrivare a predirli attraverso la nostra CNN. Pertanto avremo alcuni campioni di dati X, con un vettore di dati target $\mathbf{y_t}$ "veri" e, eseguendo la nostra CNN sui campioni di dati, genereremo un vettore target previsto $\mathbf{y_p}$ da confrontare con quelli target "veri". Tale confronto è valutato in termini di una funzione $G(\mathbf{y_p}, \mathbf{y_t})$ che è chiamata funzione di loss o funzione di costo. Di solito, il requisito principale per $G(\mathbf{y_p}, \mathbf{y_t})$ è di avere il suo minimo globale quando $\mathbf{y_p} = \mathbf{y_t}$. Possiamo distinguere due diverse sottoclassi di funzione di loss, a seconda della natura dei vettori target. Le **funzioni di loss probabilistiche** sono usate per confrontare due vettori $\mathbf{y_p}$ e $\mathbf{y_t}$ che possono essere interpretati come distribuzione di probabilità. Il caso d'uso principale di queste funzioni sono i task di *classification*, in questi casi di solito il vettore $\mathbf{y_t}$ è un vettore di vettori *one hot encoded*, il che significa che:

$$\mathbf{y_{i,t}} = (0, 0, \ldots, 0, 1, 0, \ldots, 0, 0) \tag{6.22}$$

Dove $\mathbf{y_{i,t}}$ è il vettore one hot encoded per l'i-esimo campione, e il suo k-esimo elemento è 1 se il campione appartiene alla k-esima classe, 0 altrimenti. Quindi il vettore $\mathbf{y_t}$ può essere pensato come una matrice, con una riga per ogni campione e una colonna per ogni classe, e l'elemento $\mathbf{y_{ik}}$ è 1 se l'i-esimo sample appartiene alla classe k-esima, 0 altrimenti. Pertanto ogni riga di $\mathbf{y_t}$ può essere pensata come

una distribuzione di probabilità normalizzata discreta. D'altra parte, le righe di $\mathbf{y_t}$ verranno confrontate con le righe di $\mathbf{y_p}$ prodotte dalla nostra CNN. Queste righe non saranno one hot encoded (anche se miriamo a questo), poiché i valori nel livello di output della nostra CNN saranno raramente interi esatti (a meno che non li forziamo ad essere con una funzione di attivazione del passo). Possiamo superare il problema di non avere un output one hot encoded in termini di classificazione, semplicemente assegnando il campione alla classe j se il neurone j-esimo nello strato di output è quello che contiene il valore massimo su tutto quello strato . Oltre a indovinare la classe corretta, dovremmo fare di più in termini di confronto, e quindi possiamo interpretare anche lo strato di output come una distribuzione di probabilità discreta (che sarà la nostra $\mathbf{y_{i,p}}$) e confrontalo con la distribuzione di probabilità del vero obiettivo ($\mathbf{y_{i,t}}$). In generale, le funzioni di loss probabilistiche tendono a trattare in modo diverso alcune piccole variazioni che possono verificarsi nella distribuzione di probabilità prevista, il loro scopo principale è allineare i due picchi di distribuzione. Invece, le **funzioni di loss di regressione** si adattano a più attività di regressione, dove anche la più piccola differenza tra $\mathbf{y_p}$ e $\mathbf{y_t}$ potrebbe essere utile data la più continua natura del target. A proposito, la differenza principale tra queste due sottoclassi è che le funzioni di loss probabilistiche dovrebbero funzionare con distribuzioni di probabilità, quindi parte della loro efficienza è legata al fatto che i target "veri" e le predizioni della CNN sono normalizzate. Di seguito analizziamo le funzioni loss più utilizzate per risolvere problemi come il nostro, la semantic segmentation.

Cross-entropy

Ogni probabilità di classe prevista viene confrontata con l'output 0 o 1 desiderato della classe effettiva e viene calcolato un punteggio che penalizza la probabilità in base a quanto è distante dal valore atteso effettivo. La penalità è di natura logaritmica e produce un punteggio elevato per grandi differenze vicine a 1 e un punteggio basso per piccole differenze tendenti a 0. La Cross-entropy loss viene utilizzata quando si regolano i pesi del modello durante l'addestramento. L'obiettivo è ridurre al minimo la loss, ovvero, minore è la loss, migliore è il modello. Un modello perfetto ha una Cross-entropy loss pari a 0. La Cross-entropy è definita come:

$$L_{\text{CE}} = -\sum_{i=1}^{n} t_i \log\left(p_i\right), \text{ per n classi,} \tag{6.23}$$

Dove t_i è la label "vera" e p_i è la probabilità softmax per la classe i-esima.

Binary Cross-entropy

Per la classificazione binaria, abbiamo la Binary Cross-entropy definita come:

$$\begin{aligned} L &= -\sum_{i=1}^{2} t_i \log\left(p_i\right) \\ &= -[t\log(p) + (1-t)\log(1-p)] \end{aligned} \tag{6.24}$$

Dove t_i è la label "vera" che assume un valore 0 o 1 e p_i è la probabilità softmax per la classe i-esima.

Categorical Cross-entropy

Ha la stessa funzione di loss definita nell'equazione 6.23. L'unica differenza è su come vengono definite le label "vere", la Categorical Cross-entropy viene utilizzata quando le label "vere" sono one hot encoded, ad esempio, se abbiamo i seguenti valori "veri" per il problema di classificazione a 3 classi [1,0,0], [0,1,0] e [0,0,1].

Weighted Cross-entropy

La weighted Cross-entropy è una variante della Cross-entropy in cui tutti gli esempi positivi vengono pesati in base a un coefficiente. Viene utilizzato in caso di classi non equilibrate, nella segmentazione, spesso non è necessario. Tuttavia, può essere utile quando l'addestramento della rete neurale è instabile.
Si può definire come:

$$\text{WCE} = -(\beta t_i \log(p_i) + (1 - t_i) \log(1 - p_i)) \tag{6.25}$$

Dove t_i è la label "vera" e p_i è la probabilità softmax per la classe i-esima. Per diminuire il numero di falsi negativi, impostare $\beta > 1$, per diminuire il numero di falsi positivi, impostare $\beta < 1$.

6.4 U-Net

U-Net è una rete neurale convoluzionale sviluppata per la segmentazione di immagini biomediche presso il Dipartimento di Informatica dell'Università di Friburgo [26]. Si basa su un architettura cosiddetta "Fully convolutional network" [27]. Questa architettura è stata estesa e modificata in modo tale che la rete risulta molto efficace anche con poche immagini di training e produce segmentazioni più precise, possiamo vedere l'architettura di U-Net in figura 6.6. L'idea principale di questa architettura è quella di integrare una normale rete "contratta" con strati successivi, in cui gli operatori di pooling vengono sostituiti da operatori di sovracampionamento.
Quindi, i livelli aggiunti aumentano la risoluzione dell'output, per localizzare le feature ad alta risoluzione del percorso di contrazione vengono combinate con l'output di sovracampionato. Un successivo livello di convoluzione può quindi imparare ad assemblare un output più preciso in base a queste informazioni. Una modifica importante all'architettura è che nella parte di sovracampionamento ci sono un gran numero di feature channel, che consentono alla rete di propagare le informazioni di contesto a livelli di risoluzione più elevata, di conseguenza, il percorso espansivo è più o meno simmetrico al percorso di contrazione e produce un'architettura a forma di U. La rete non ha livelli completamente connessi e utilizza solo la parte valida di ciascuna convoluzione, ovvero la mappa di segmentazione contiene solo i pixel, per i quali è disponibile il contesto completo nell'immagine di input.

6.4.1 Architettura della rete

L'architettura di rete è illustrata nella figura 6.6, consiste in un **percorso di contrazione** (lato sinistro) e un **percorso espansivo** (lato destro). Il percorso contrattuale segue l'architettura tipica di una rete convoluzionale, consiste nell'applicazione

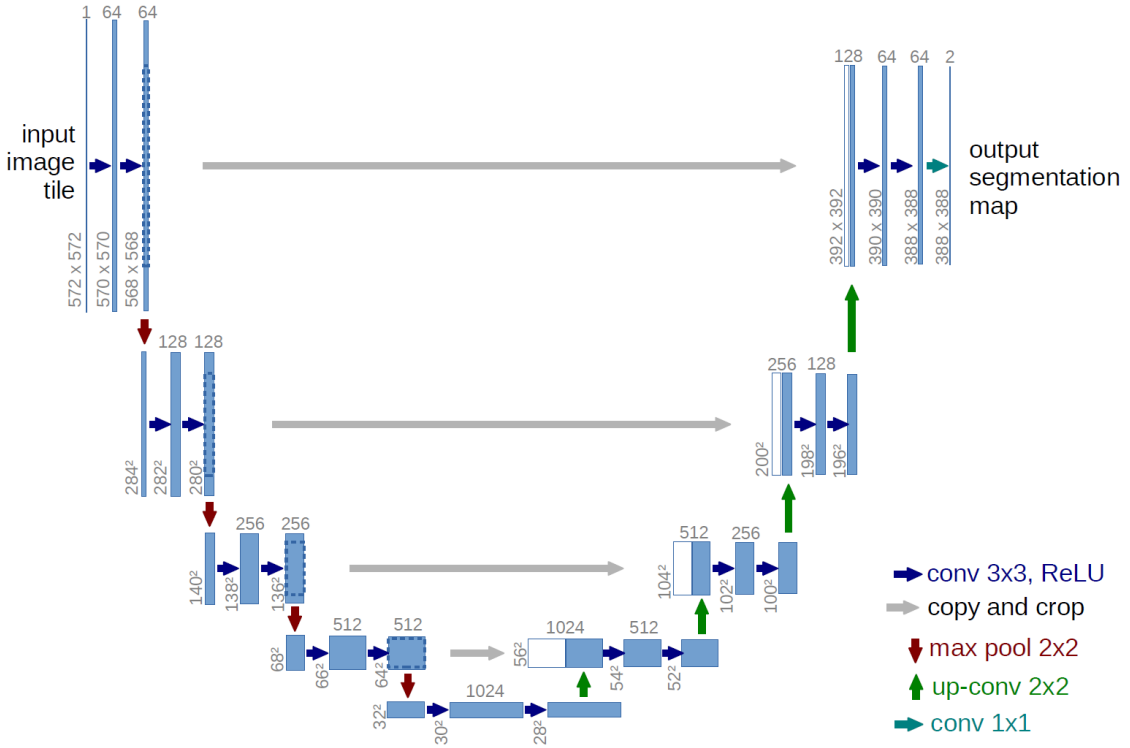


Figura 6.6: Architettura di U-net (esempio per 32x32 pixel nella risoluzione più bassa). Ciascun riquadro blu corrisponde a una mappa delle funzionalità multicanale. Il numero di filtri è indicato nella parte superiore della casella. La dimensione x-y è fornita sul bordo inferiore sinistro del riquadro. I riquadri bianchi rappresentano le mappe delle caratteristiche copiate. Le frecce indicano le diverse operazioni.

ripetuta di due convoluzioni 3x3, ciascuna seguita da Rectified Linear Unit (ReLU) e un'operazione di max pooling 2x2 con stride = 2 per il downsampling. Ad ogni passaggio di downsampling raddoppiamo il numero di feature channel. Ogni fase del percorso espansivo consiste in: un sovracampionamento della mappa delle feature seguito da una convoluzione 2x2 ("convoluzione verso l'alto") che dimezza il numero di feature channel, una concatenazione con la mappa delle feature ridotta dal percorso di contrazione e due convoluzioni 3x3, ciascuna seguita da un ReLU. La riduzione è necessaria a causa della perdita di pixel del bordo in ogni convoluzione. Nel livello finale viene utilizzata una convoluzione 1x1 per mappare ciascun vettore di feature a 64 componenti per il numero di classi desiderato.
In totale la rete ha 23 strati convoluzionali. Per consentire un affiancamento senza interruzioni della mappa di segmentazione dell'output (figura 6.7), è importante selezionare la dimensione del riquadro di input in modo tale che tutte le operazioni di max pooling 2x2 siano applicate a un layer con dimensioni x e y pari.

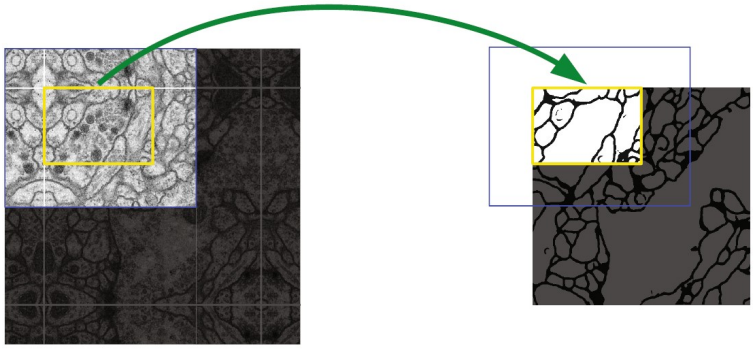


Figura 6.7: Strategia di sovrapposizione dei crop per la segmentazione senza interruzioni di immagini arbitrarie di grandi dimensioni. La previsione della segmentazione nell'area gialla richiede come input i dati dell'immagine all'interno dell'area blu. I dati di input mancanti vengono estrapolati dal mirroring.

6.5 Preparazione del dataset

Come detto precedentemente il dataset a mia disposizione per questo progetto era organizzato in 194 cartelle in ordine sequenziale con all'interno entrambe le immagini stereo corrispondenti a quel sequenziale. La preparazione del dataset per il training e per il testing della rete neurale si è svolto solamente sulle immagini derivanti dalla telecamera di destra, in virtù del fatto che erano maggiormente centrate nella scena. Sono stati svolti 2 passaggi principali:

1. Labeling a mano delle immagini originali tramite un programma di labeling modificato ad hoc per questo progetto;

2. Caricamento delle immagini sotto forma di file e loro trasformazione in matrici Numpy molto più veloci da leggere dalla macchina.

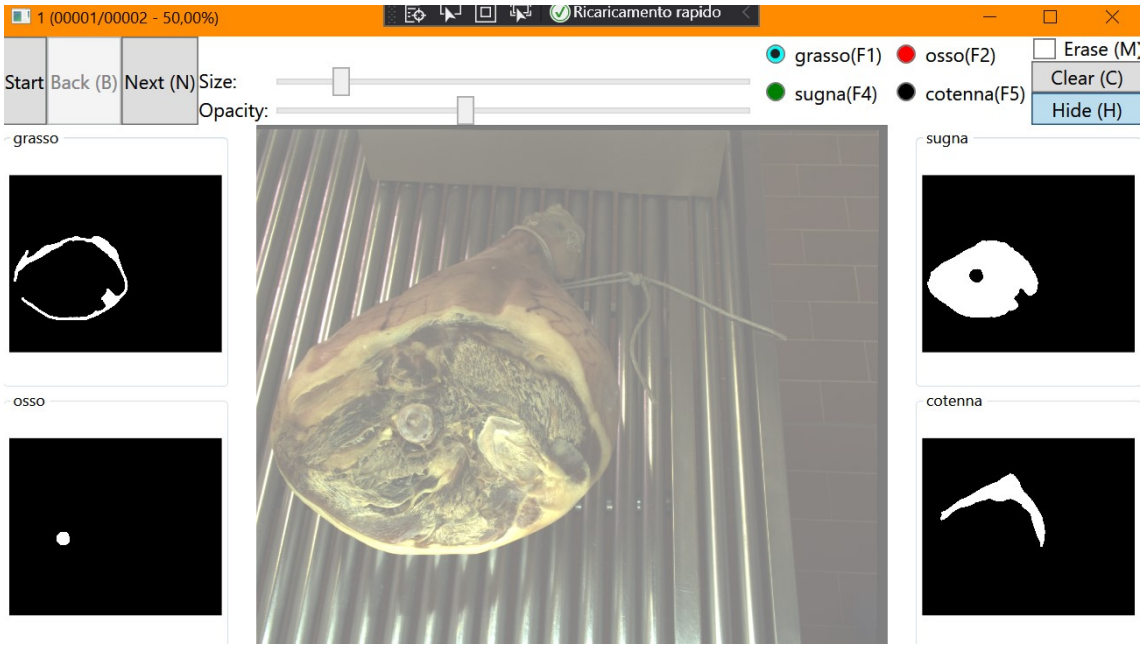


Figura 6.8: Screenshot del programma Segmentator, in questo caso sono state nascoste le label sovrapposte all'immagine originale con il tasto "hide". Le quattro maschere ai lati rappresentano le label di output.

6.5.1 Labeling

Avendo scelto di intraprendere un approccio di apprendimento supervisionato ovviamente la rete ha bisogno di esempi "veri" o "giusti" da cui poter imparare e con cui poter valutare le proprie prestazioni. Nel nostro caso le classi che abbiamo bisogno di segmentare all'interno delle immagini sono 4:

- **Sugna**: la parte più interessante, dove il robot dovrà applicare la sugna;
- **Grasso**: il grasso del prosciutto, non deve essere toccato;
- **Osso**: l'osso al centro del prosciutto, non deve essere toccato;
- **Cotenna**: per completezza è stata aggiunta anche la classe relativa alla parte restante del prosciutto.

In questo progetto si è deciso di non introdurre una classe per identificare lo sfondo dell'immagine perchè non utile ai nostri fini. Per svolgere il procedimento di labeling ho sfruttato un programma proprietario di Specialvideo (scritto in C#) messomi a disposizione, modificandolo e adattandolo alle immagini che avevo a disposizione, vediamo la schermata principale di questo Segmentator in figura 6.8. Il Segmentator da la possibilità di segmentare le immagini nelle 4 classi elencate prima tramite l'utilizzo di un "pennello", comandato tramite il mouse, per segnalare le porzioni di immagini appartenenti ad una certa classe. Inoltre è anche possibile definire la grandezza del pennello in base alla grandezza del dettaglio da segmentare e definire la trasparenza del colore delle label sull'immagine originale, vediamo un esempio di immagine labelizzata in figura 6.9. Dopo ogni labeling il Segmentator salvava all'interno della cartella dell'immagine originale anche 4 maschere a 1 canale della

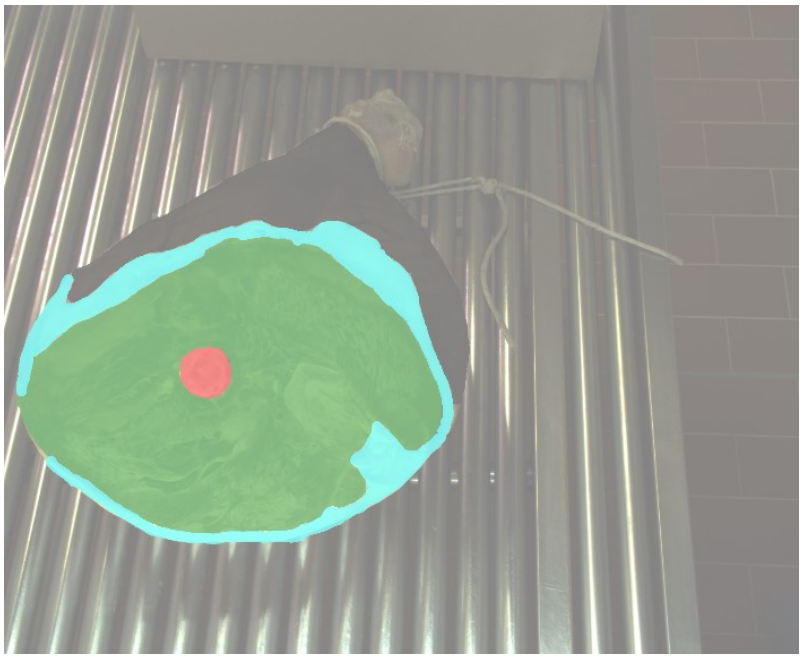


Figura 6.9: Immagine di output dopo il processo di labeling tramite Segmentator, col colore rosso vediamo l'osso, con il nero la cotenna, con il verde la sugna e con l'azzurro il grasso.

stessa dimensione, una per ogni classe, dove ogni pixel era bianco se apparteneva alla label e nero altrimenti (come quelle rappresentate in figura 6.8).

6.5.2 Caricamento delle immagini in memoria

Per il training ovviamente le immagini vanno caricate nella memoria della macchina e poi passate alla rete neurale per l'addestramento, per fare ciò non è conveniente mantenere le immagini sotto forma di file perchè troppo dispendiose da manipolare e da mantenere in memoria, per questo dopo aver "letto" le immagini dalle loro cartelle ho deciso di trasformarle, tramite la libreria **Numpy**, in matrici molto meno impattanti in termini di memoria.
La libreria Numpy consente di lavorare con vettori e matrici in maniera più efficiente e veloce di quanto non si possa fare con le liste e le liste di liste (matrici). Il costrutto di base è l'ndarray, che può avere dimensioni qualunque e tipi corrispondenti a quelli classici del C o del Fortran, che restano i linguaggi tuttora più utilizzati per il calcolo numerico. Uno dei punti di forza di Numpy è di poter lavorare sui vettori sfruttando le ottimizzazioni di calcolo vettoriale del processore della macchina. Ciò rende particolarmente efficiente i calcoli, rispetto alle liste [28]. Quindi grazie a Numpy ho trasformato ogni immagine originale e ogni mappa di segmentazione in un ndarray sotto forma di matrice, a 3 canali per le immagini originali RGB e a 4 canali per la mappa di segmentazione corrispondente (4 canali perchè in ogni canale ho inserito una label a 1 canale: grasso, sugna, osso e cotenna).
La rete neurale durante il training ha bisogno sì delle immagini, ma anche di poter capire gli abbinamenti tra immagine originale e mappa di segmentazione corrispondente, per fare ciò ho utilizzato una particolare struttura dati in python detta Tensor

Slice Dataset che praticamente crea al suo interno un'associazione, basata sulle posizioni degli elementi, tra i 2 vettori di matrici dategli in input; nel nostro caso il vettore contenente tutte le immagini originali, e il vettore contenente tutte le mappe di segmentazione. Quindi ricapitolando, alla fine abbiamo caricato in memoria 3 Tensor Slice Dataset, uno per il training, uno per la validation e uno per il test contenenti tutte le 194 immagini originali abbinate a tutte le loro mappe di segmentazione. La suddivisione nei 3 dataset segue queste percentuali: 70% per il training, 20% per la validation e 10% per il test.

6.6 Creazione del modello della rete

Dopo aver svolto la prima parte di preparazione della rete offline è giunto il momento di creare il modello di rete neurale pronta per l'apprendimento, come detto precedentemente l'architettura della rete scelta per questo progetto è U-Net, spiegata dettagliatamente nel paragrafo 6.4, perchè molto precisa e veloce ad eseguire task di segmentazione anche con immagini molto definite (come le nostre: 2464 x 2056) e con poche immagini a disposizione per il training, infatti nel nostro caso abbiamo utilizzato soltanto 135 immagini per il training. A causa di evidenti limiti fisici e computazionali delle macchine a mia disposizione ho optato per la creazione del modello e per il training della rete online su una piattaforma di cloud computing chiamata Google Colab.

6.6.1 Google Colab

Google Colab è una piattaforma che ci permette di eseguire codice direttamente sul Cloud. Per sfruttare le funzionalità di tale piattaforma, tutto ciò di cui abbiamo bisogno è un account Google, mediante il quale potremo effettuare il login ed avere accesso alla piattaforma stessa.
Prima di capire meglio come funziona, è bene fare una premessa. Per eseguire codice, Google Colab sfrutta i cosiddetti Jupyter Notebook. Questi non sono altro che documenti interattivi nei quali possiamo scrivere (e quindi eseguire) il nostro codice. Più precisamente, tali documenti permettono di suddividere il nostro codice in celle, ognuna delle quali può contenere anche del testo informativo, eventualmente formattato in Markdown, ne vediamo un esempio in figura 6.10. L'uso di tali notebook è abbastanza comodo (e, non a caso, piuttosto popolare) per chi si occupa di data science e machine learning. Tramite un unico documento, è infatti possibile sia eseguire tutti gli step di un processo di analisi o processing, sia descriverne il comportamento in linguaggio naturale. Di fatto, i notebook rappresentano un ottimo modo per spiegare o semplicemente mostrare come agisce un algoritmo, e si prestano molto bene alla realizzazione di un portfolio da data scientist.

Quindi su Colab ho creato un nuovo notebook e ho installato, ovvero importato la mia cartella Git contenente tutti gli script Python necessari per il caricamento delle immagini spiegato prima e per la definizione del modello di U-Net. Fatto ciò ho dovuto fare i conti con la memoria Ram disponibile in Google Colab, ovvero 12 Gb, che non mi rendeva possibile caricare le immagini originali a risoluzione massima (2464 x 2056) perchè superavano quella soglia. Per cui, il passaggio successivo

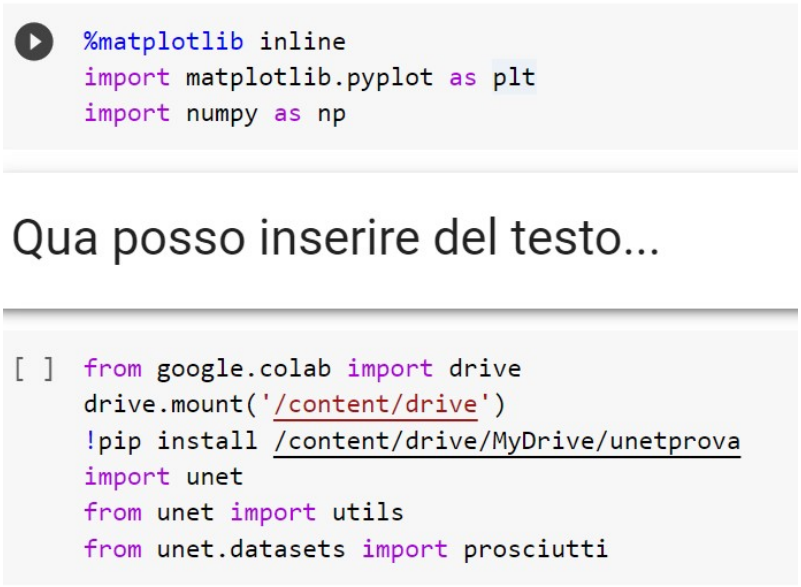


Figura 6.10: Esempio di Jupyter notebook su Colab, si possono eseguire singolarmente le celle di codice o creare celle solo destinate a commenti.

è stato modificare il codice precedentemente scritto per il caricamento delle immagini applicando un ridimensionamento alle stesse di 1/4 della dimensione originale, ottenendo immagini e label di dimensione 616 x 514.

6.6.2 Creazione del modello

L'architettura di U-Net la conosciamo bene, per adattare però la rete al mio progetto ho attuato qualche modifica in termini di hyperparameter, in tabella 6.1 vediamo un riassunto. Gli unici parametri differenti rispetto al modello originale di U-Net

Hyperparameter	Value
layer depth	5
filter roots	64
kernel size	3×3
Pool size	2×2
Dropout rate	0.5
Stride	1
Padding	same
Activation function	Relu
Loss function	Weighted Cross-entropy
Optimizer	Nadam

Tabella 6.1: Gli hyperparameter utilizzati per il progetto.

sono l'optimizer e il padding; come optimizer ho deciso di utilizzare Nadam perchè come spiegato in 6.3.5 da normalmente risultati migliori rispetto ad Adam. La scelta di padding = same, invece, è semplicemente legata al fatto che in questo progetto avevamo bisogno di mantenere la stessa dimensione anche nell'immagine di output.

Layer (type)	Output Shape	Param #	Connected to
inputs (InputLayer)	[(None, 512, 608, 3)	0	
conv_block (ConvBlock)	(None, 512, 608, 64)	38720	inputs[0][0]
max_pooling2d (MaxPooling2D)	(None, 256, 304, 64)	0	conv_block[0][0]
conv_block_1 (ConvBlock)	(None, 256, 304, 128	221440	max_pooling2d[0][0]
max_pooling2d_1 (MaxPooling2D)	(None, 128, 152, 128	0	conv_block_1[0][0]
conv_block_2 (ConvBlock)	(None, 128, 152, 256	885248	max_pooling2d_1[0][0]
max_pooling2d_2 (MaxPooling2D)	(None, 64, 76, 256)	0	conv_block_2[0][0]
conv_block_3 (ConvBlock)	(None, 64, 76, 512)	3539968	max_pooling2d_2[0][0]
max_pooling2d_3 (MaxPooling2D)	(None, 32, 38, 512)	0	conv_block_3[0][0]
conv_block_4 (ConvBlock)	(None, 32, 38, 1024)	14157824	max_pooling2d_3[0][0]
upconv_block (UpconvBlock)	(None, 64, 76, 512)	2097664	conv_block_4[0][0]
crop_concat_block (CropConcatBl	(None, 64, 76, 1024)	0	upconv_block[0][0] conv_block_3[0][0]
conv_block_5 (ConvBlock)	(None, 64, 76, 512)	7078912	crop_concat_block[0][0]
upconv_block_1 (UpconvBlock)	(None, 128, 152, 256	524544	conv_block_5[0][0]
crop_concat_block_1 (CropConcat	(None, 128, 152, 512	0	upconv_block_1[0][0] conv_block_2[0][0]
conv_block_6 (ConvBlock)	(None, 128, 152, 256	1769984	crop_concat_block_1[0][0]
upconv_block_2 (UpconvBlock)	(None, 256, 304, 128	131200	conv_block_6[0][0]
crop_concat_block_2 (CropConcat	(None, 256, 304, 256	0	upconv_block_2[0][0] conv_block_1[0][0]
conv_block_7 (ConvBlock)	(None, 256, 304, 128	442624	crop_concat_block_2[0][0]
upconv_block_3 (UpconvBlock)	(None, 512, 608, 64)	32832	conv_block_7[0][0]
crop_concat_block_3 (CropConcat	(None, 512, 608, 128	0	upconv_block_3[0][0] conv_block[0][0]
conv_block_8 (ConvBlock)	(None, 512, 608, 64)	110720	crop_concat_block_3[0][0]
conv2d_18 (Conv2D)	(None, 512, 608, 4)	260	conv_block_8[0][0]
activation_22 (Activation)	(None, 512, 608, 4)	0	conv2d_18[0][0]
outputs (Activation)	(None, 512, 608, 4)	0	activation_22[0][0]

Figura 6.11: Modello di U-Net utilizzato, i parametri totali della rete sono 31,031,940.

Dopo aver settato tutti i parametri come in tabella 6.1 ho stampato a video il modello risultante accorgendomi però di un errore: essendo la rete composta da 5 layer le immagini di dimensioni 616 x 514 venivano, per ogni layer, divise per 2 a lato a causa del max pooling 2 x 2. Il problema nasce dal fatto che 616 e 514 non sono divisibili per 16 (2^4, vedi figura 6.6), quindi nell'ultimo layer si otterrebbe un valore non intero che ovviamente crea un errore. Per questo motivo ho deciso di tagliare tutte le immagini di pochi pixel fino ad arrivare al primo multiplo di 16 disponibile, quindi 608 x 512. Il "resize" delle immagini non è stato possibile svolgerlo in modo classico, ovvero eliminando pixel da tutti e 4 i lati, ma soltanto dai lati destro e inferiore dell'immagine, questo perchè nel momento in cui dovremo calcolare la corrispondenza stereo dell'immagine originale il punto di origine (0,0) sarà quello in alto a sinistra, quindi quel punto deve combaciare con le label predette dalla rete. In figura 6.11 troviamo il modello definitivo della rete.

6.7 Training

Dopo aver caricato le immagini in memoria e aver creato il modello è il momento di allenare la rete e vedere i risultati. Per l'allenamento della rete ho scelto alcune metriche valutative in modo da poter monitorare l'andamento del processo, utili anche alla rete stessa per poter tarare i pesi di conseguenza ai valori ottenuti epoca per epoca.

6.7.1 Metriche

Durante il training la rete neurale sfrutta alcune metriche di valutazione per controllare che le modifiche effettuate ai pesi ogni epoca stiano migliorando i risultati finali o meno. Nel nostro caso ovviamente le metriche devono essere improntate alla valutazione di quanto l'immagine predetta dalla rete sia simile a quella "giusta", per questo motivo le metriche utilizzate per il mio modello sono: Mean intersection over union, Dice coefficent, Intersection over union coefficent e Area under the curve. Ovviamente oltre queste anche la funzione di loss Weighted Cross-entropy viene utilizzata per la validazione.

Intersection Over Union

L'Intersection-Over-Union (IoU), noto anche come indice Jaccard, è una delle metriche più comunemente utilizzate nella segmentazione semantica. L'IoU è una metrica molto semplice ed estremamente efficace.
In poche parole, l'IoU è l'area di sovrapposizione tra la segmentazione predetta e il labeling "giusto" divisa per l'area di unione tra le stesse due, come mostrato nella figura 6.12. Questa metrica varia da 0-1 (0-100%) con 0 che indica l'assenza di sovrapposizione e 1 che indica la segmentazione perfettamente sovrapposta. Per la segmentazione binaria (due classi) o multi-classe, la **mean IoU** dell'immagine viene calcolata prendendo l'IoU di ciascuna classe e calcolandone la media (è implementato in modo leggermente diverso nel codice).
La metrica più semplice da utilizzare potrebbe sembrare quella della **Pixel accuracy**, ovvero la corrispondenza tra i pixel della ground truth e quelli della segmen-

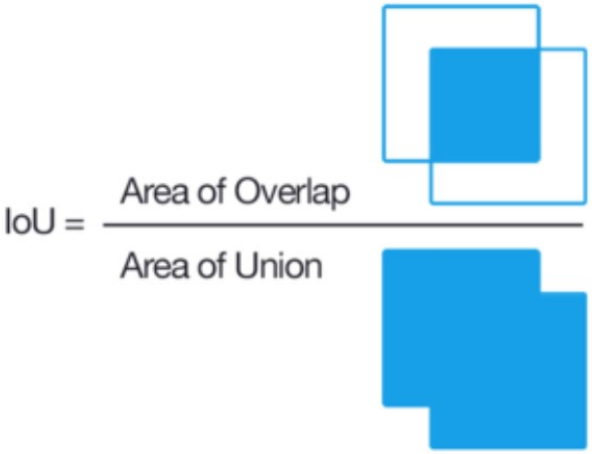


Figura 6.12: Formula per calcolare IoU.

tazione predetta, ma nel nostro caso il non bilanciamento delle classi rende inutile utilizzare questa metrica.

Dice coefficent

Il coefficiente di Dice è pari a 2 x l'area di sovrapposizione tra ground truth e previsione della rete divisa per il numero totale di pixel in entrambe le immagini, vediamo un esempio in figura 6.13.
Il coefficiente di Dice è molto simile all'IoU. Sono positivamente correlati, il che significa che se uno dice che il modello A è migliore del modello B nel segmentare un'immagine, l'altro dirà lo stesso. Come l'IoU, entrambi vanno da 0 a 1, con 1 che indica la massima somiglianza tra predizione e ground truth.

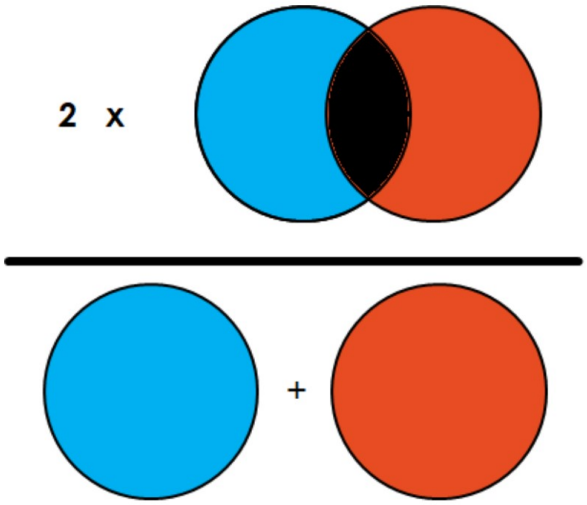


Figura 6.13: Formula per il calcolo del coefficiente di Dice.

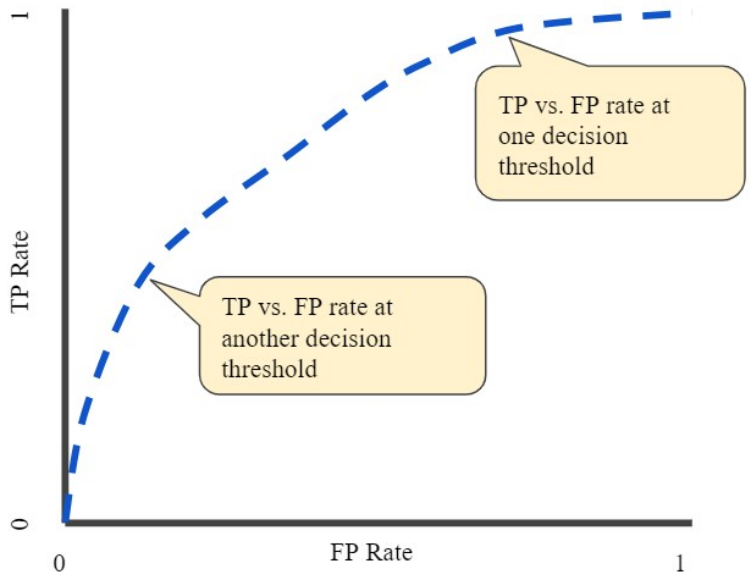


Figura 6.14: Tasso di TP vs FP a diverse soglie di classificazione.

Area Under the Curve

La **ROC curve** (Receiver Operating Characteristic curve) è un grafico che mostra le prestazioni di un modello di classificazione in tutte le soglie di classificazione. Questa curva traccia due parametri:

- Il tasso di "veri" positivi;
- Il tasso di falsi positivi.

True Positive Rate (TPR) è sinonimo di recall ed è quindi definito come segue:

$$TPR = \frac{TP}{TP + FN} \tag{6.26}$$

Il **False Positive Rate** (FPR) è definito come segue:

$$FPR = \frac{FP}{FP + TN} \tag{6.27}$$

Una ROC curve traccia TPR vs. FPR a diverse soglie di classificazione. L'abbassamento della soglia di classificazione classifica più elementi come positivi, aumentando così sia i falsi positivi che i "veri" positivi. La figura 6.14 mostra una tipica curva ROC. Per calcolare i punti in una ROC curve, potremmo valutare molte volte un modello di regressione logistica con soglie di classificazione diverse, ma questo sarebbe inefficiente. Fortunatamente, esiste un efficiente algoritmo basato sull'ordinamento che può fornirci queste informazioni, chiamato AUC. **AUC** sta per "Area under the ROC Curve". Cioè, l'AUC misura l'intera area bidimensionale sotto la curva ROC da (0,0) a (1,1), evidente l'area in figura 6.15.
L'AUC fornisce una misura aggregata delle prestazioni attraverso tutte le possibili soglie di classificazione. Un modo di interpretare l'AUC è come la probabilità che

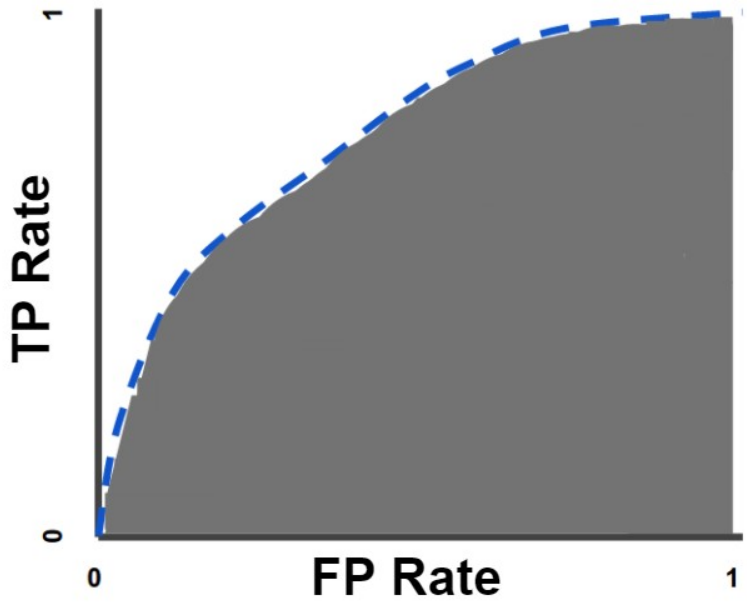


Figura 6.15: Esempio di area calcolata da AUC.

il modello classifichi un esempio positivo in modo più elevato rispetto a un esempio negativo.
Il valore dell'AUC varia da 0 a 1. Un modello le cui previsioni sono errate al 100% ha un'AUC di 0,0; uno le cui previsioni sono corrette al 100% ha un'AUC di 1,0.
L'AUC è utile per i seguenti due motivi:

- L'AUC ha l'invarianza di scala. Misura quanto bene vengono classificate le previsioni, piuttosto che i loro valori assoluti;
- L'AUC è invariante alla soglia di classificazione. Misura la qualità delle previsioni del modello indipendentemente dalla soglia di classificazione scelta.

Tuttavia, entrambi questi motivi sono accompagnati da avvertenze, che possono limitare l'utilità dell'AUC in alcuni casi d'uso:

- L'invarianza di scala non è sempre desiderabile. Ad esempio, a volte abbiamo davvero bisogno di output di probabilità ben calibrati e l'AUC non ce lo dirà;
- L'invarianza alla soglia di classificazione non è sempre desiderabile. Nei casi in cui vi sono ampie disparità nel costo dei falsi negativi rispetto ai falsi positivi, può essere fondamentale ridurre al minimo un tipo di errore di classificazione. Ad esempio, quando si esegue il rilevamento dello spam tramite posta elettronica, è probabile che si desideri dare la priorità alla riduzione al minimo dei falsi positivi (anche se ciò comporta un aumento significativo dei falsi negativi). L'AUC non è una metrica utile per questo tipo di ottimizzazione.

6.7.2 Addestramento del modello

Il training della rete si è svolto su 135 immagini delle 194 totali, utilizzandone altre 38 per la validation durante l'addestramento, le restanti immagini sono invece state

utilizzate per la fase di testing, quindi non sono state prese in considerazione in questa fase.
Il validation dataset è il campione di dati utilizzato per fornire una valutazione imparziale di un modello adattato precedentemente al dataset di training durante l'ottimizzazione degli iperparametri del modello. La valutazione diventa più parziale man mano che la competenza sul validation dataset viene incorporata nella configurazione del modello. Il validation dataset viene utilizzato per valutare un determinato modello, utilizziamo questi dati per mettere a punto gli iperparametri del modello. Quindi il modello occasionalmente vede questi dati, ma non "impara" mai da questi. Usiamo i risultati sul validation dataset e aggiorniamo gli iperparametri di livello superiore. Quindi il validation dataset influisce sul modello, ma solo indirettamente. Il validation dataset è noto anche come set di sviluppo, ciò ha senso poiché questo set di dati aiuta durante la fase di "sviluppo" del modello.

Per il training ho seguito questa procedura: ho svolto 100 epoche di addestramento in modo che la rete si stabilizzasse un minimo nelle ultime epoche e poi ho ripetuto questa procedura fino a raggiungere circa i 30000 step di apprendimento (epoche x numero di immagini), dopo i quali ho ritenuto la rete "stabile".
I parametri che ho utilizzato per il training sono:

1. **Learning rate:** è un parametro riferito all'optimizer che determina la dimensione del "passo" a ogni iterazione mentre ci si sposta verso un minimo di una funzione di loss. Influenza in che misura le informazioni acquisite di recente hanno la precedenza su quelle vecchie, rappresenta metaforicamente la velocità con cui un modello di apprendimento automatico "apprende". Nel nostro progetto è stato settato inizialmente a 0.001;

2. **Batch size:** La dimensione del batch è un iperparametro che definisce il numero di campioni su cui lavorare prima di aggiornare i parametri del modello interno, nel nostro caso è stata settata a 2 perchè già con 2 immagini per volta veniva riempita quasi tutta la memoria RAM disponibile su Colab.

Per adattare il learning rate al procedere dell'apprendimento, in modo da non "cadere" in minimi locali ho utilizzato una callback di Keras chiamata **ReduceLROnPlateau**, la sua funzione è quella di ridurre il learning rate quando una metrica smette di migliorare. I modelli spesso traggono vantaggio dalla riduzione del tasso di apprendimento una volta che l'apprendimento ristagna. Questa callback monitora una quantità e se non si vede alcun miglioramento per un numero di epoche impostato", il tasso di apprendimento viene ridotto.
Nel nostro caso ho settato questa callback per il monitor della validation loss in modo che diminuisse il learning rate di un fattore 0.1 ogni qualvolta questa loss "ristagnasse" per almeno 5 epoche, fino a raggiungere un minimo corrispondente a 0.00000001.

6.8 Testing e risultati

La fase di testing del modello addestrato ha portato buonissimi risultati sia visivi che a livello di metriche, in tabella 6.2 vediamo un riassunto e in figura 6.17 vediamo il confronto tra label originale e predizione della rete.

Le metriche utilizzate per valutare oltre che visivamente i risultati sono le stesse utilizzate nella fase di validation, cioè: MeanIou e coefficiente di Dice.

Metriche	Risultati
Mean IoU	tra 0.90 e 0.93
Area Under the ROC curve	tra 0.92 e 0.93
Runtime	∼84 s/epoca e ∼538 ms/step

Tabella 6.2: Risultati medi della fase di testing.

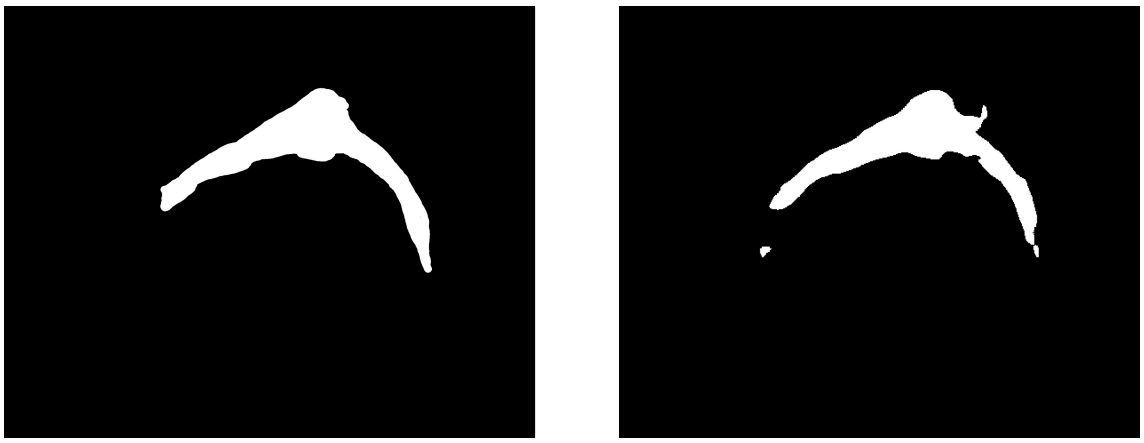
La prima cosa che salta all'occhio in figura 6.16 è che la label dell'osso non compare nell'immagine predetta dalla rete, questo perchè si è deciso di escluderla dal modello in un secondo momento, visto che nei risultati iniziali l'osso veniva predetto veramente con scarse performance e solo in alcuni casi. Il motivo di questa mancata segmentazione lo ritengo dovuto alla dimensione molto ridotta dell'osso e alla somiglianza di colore con il resto del prosciutto, probabilmente se avessi avuto a disposizioni immagini di qualità superiore con un contrasto più marcato la rete sarebbe riuscita a riconoscerlo. Quindi per il riconoscimento dell'osso, che ovviamente non va sugnato, ci baseremo sulla disparità molto più alta rispetto alla parte da sugnare del prosciutto, questo perchè l'osso è l'unica parte del prosciutto che "sporge" in maniera riconoscibile rispetto al piano rappresentato dal grasso e dalla sugna. Vedremo in maniera approfondita tutto ciò nel capitolo 7.

Nonostante ciò, per i risultati ottenuti a livello visivo (figura 6.16 e 6.17) e a livello di metriche (tabella 6.2) mi ritengo più che soddisfatto e inoltre credo che con un dataset di immagini maggiore e con più qualità si sarebbe riusciti a raggiungere performance ancora migliori.

(a) Predizione della rete. **(b)** Ground truth.

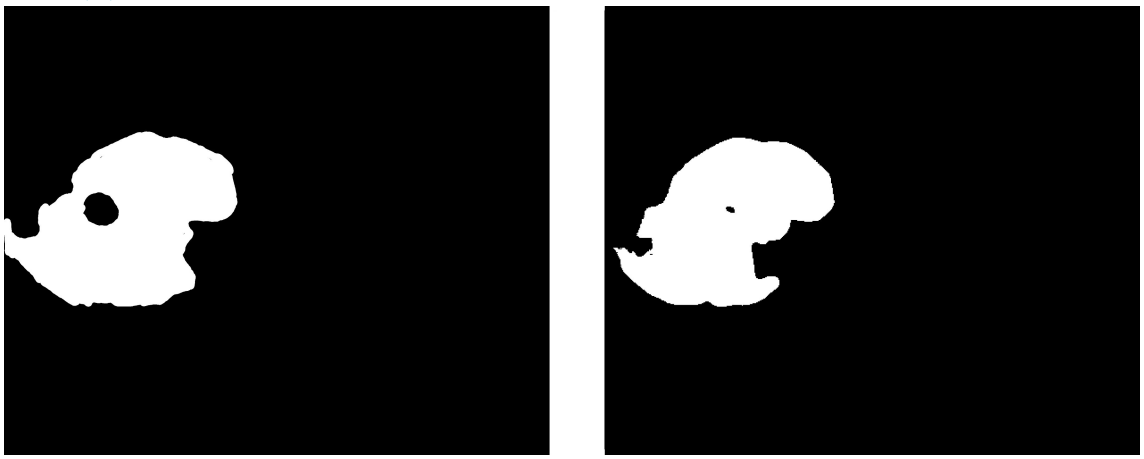
Figura 6.16: In questo confronto notiamo una buona qualità della predizione della rete.

Infine, al termine delle operazioni di testing, ho creato una copia del dataset di test eliminando le ground truth e sostituendole con le predizioni della rete, sotto forma di file immagine.



(a) Confronto tra una cotenna originale a sinistra e la cotenna predetta a destra.

(b) Confronto tra un grasso originale a sinistra e il grasso predetto a destra.



(c) Confronto tra una sugna originale a sinistra e la sugna predetta a destra.

Figura 6.17: Confronto tra Ground truth a sinistra e previsioni della rete a destra.

6.8.1 Visualizzazione dei risultati

Per visualizzare i risultati delle predizioni della rete in modo facile e comodo, insieme al team di Specialvideo, ho deciso di modificare il programma usato per il labeling nella prima fase, il Segmentator, in modo che fornisse l'immagine originale con sovrapposte le label predette. In realtà la struttura dell'output grafico è la stessa, semplicemente è stato eliminato il pennello e la possibilità di modificare le label. In figura 6.18 vediamo uno screenshot del visualizator.

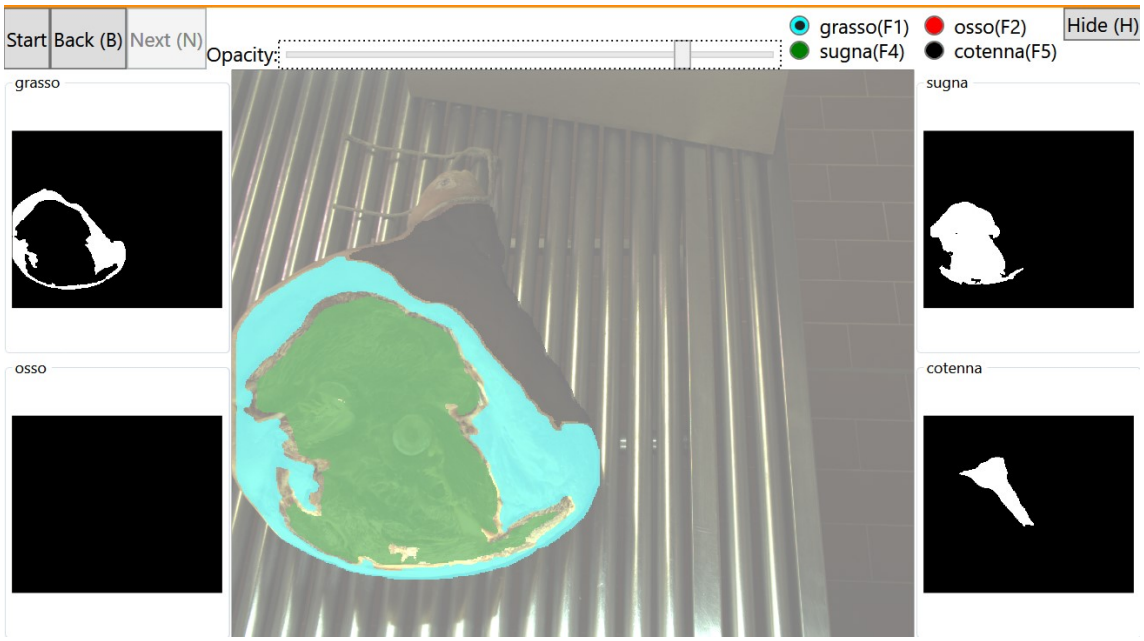


Figura 6.18: Screenshot del visualizator, è stato tolto il pennello e la possibilità di modifica delle label predette.

Capitolo 7

Parte finale: ricostruzione della scena 3D

7.1 Analisi del problema

In questo capitolo tratteremo l'ultima parte del progetto, ovvero la ricostruzione finale della scena 3D dalle immagini 2D acquisite inizialmente.
Ci eravamo lasciati nel capitolo 4 alla creazione della mappa di disparità tramite il matching stereo, il passaggio successivo è la triangolazione. In particolare per il progetto di questa tesi ho creato, a partire dalla mappa di disparità, la **point cloud** (depth map) del prosciutto e infine la mesh che rappresenta l'oggetto in 3D con il calcolo delle normali per ogni punto della superficie.

7.2 Triangolazione

Una volta risolto il problema delle corrispondenze stereo e noti i parametri del sistema stereoscopico ottenuti mediante calibrazione, per i punti per cui esiste e per quelli per cui è stato possibile determinare l'omologo, si può inferire la distanza del punto dalle telecamere. Si consideri la figura 7.1, nella quale si considera un sistema stereoscopico ideale o nel quale le immagini siano state poste in forma standard mediante la procedura di rettificazione. Si indichi inoltre con (x_l, y_l) e (x_r, y_r) le coordinate delle proiezioni del punto P sui due piani immagini θ_l e θ_r rispetto ai sistemi di riferimento con origine nei centri ottici delle due telecamere. Tali punti hanno coordinate (u_l, v_l) e (u_r, v_r) rispetto ai sistemi di riferimento con origine nel punto in alto a sinistra di ogni immagine. Si fa notare che, per ottenere le coordinate dei punti (u_l, v_l) e (u_r, v_r) appartenenti ai due piani immagine rispetto al sistema di riferimento centrato nell'asse ottico di ciascuna telecamera, si trasla l'origine delle coordinate nel punto principale (principal point) di ogni telecamera e si moltiplica tale valore per la dimensione orizzontale del pixel nel caso della coordinata x e per la dimensione verticale del pixel nel caso della coordinata y. Le dimensioni del pixel possono coincidere nel caso di sensori immagine con pixel quadrati o essere diversi. In questo ultimo caso è necessario conoscere sia le dimensioni orizzontali (Dim_x) sia le dimensioni verticali (Dim_y) del pixel per ottenere le coordinate. Indicando con (C_{ul}, C_{vl}) e (C_{ur}, C_{vr}) le coordinate del principal point rispetto al sistema di

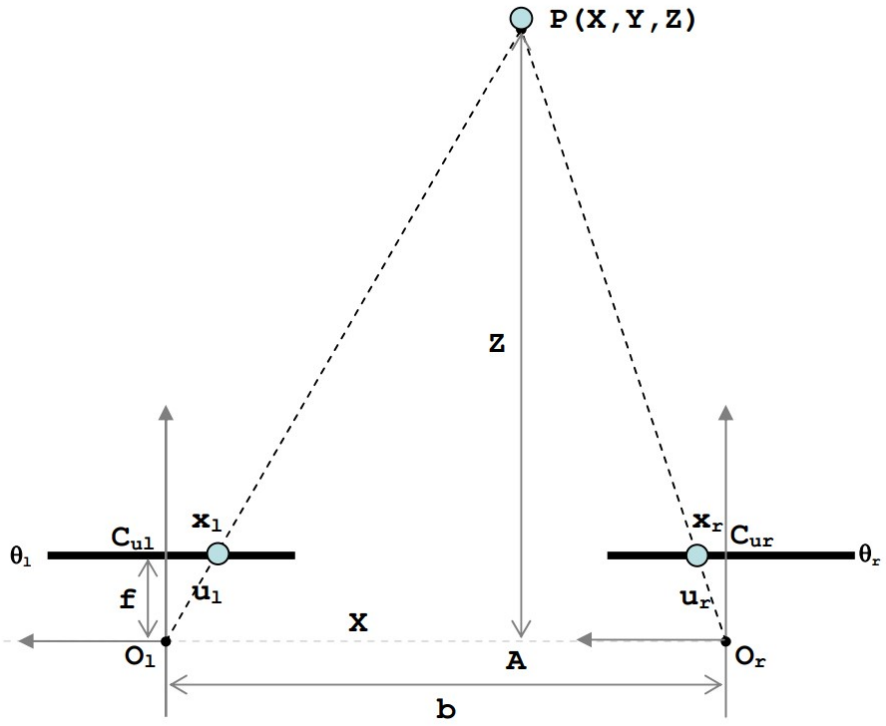


Figura 7.1: Triangolazione in un sistema stereoscopico in forma standard.

riferimento immagine centrato nel punti in alto a sinistra di ogni immagine risulta rispettivamente per i punti sulle immagini θ_l e θ_r:

$$\begin{aligned} x_l &= (u_l - C_{ul}) \cdot \mathrm{Dim}_x \\ y_l &= (v_l - C_{vl}) \cdot \mathrm{Dim}_y \\ &\text{e} \\ x_r &= (u_r - C_{ur}) \cdot \mathrm{Dim}_x \\ y_r &= (v_r - C_{vr}) \cdot \mathrm{Dim}_y \end{aligned} \tag{7.1}$$

Si assuma come sistema di riferimento quello con origine nel centro ottico O_l avente assi x e y paralleli ai piani immagine e asse z passante per il centro ottico O_l e il centro dell'immagine, è possibile scrivere le relazioni che individuano le coordinate X e Y rispetto a tale sistema di riferimento a partire dalle coordinate immagine x_l e y_l. Indicando con f la distanza focale, con b la baseline e utilizzando la similitudine tra i triangoli O_lPO_r e x_lPx_r è possibile scrivere la seguente relazione che individua la distanza Z del punto P dalla retta che passa per i centri ottici O_l e O_r delle due telecamere:

$$\frac{b + x_l - x_r}{Z - f} = \frac{b}{Z} \tag{7.2}$$

Indicando con $d = \frac{x_l - x_r}{\mathrm{Dim}_x}$ la disparità, risulta:

$$Z = f \frac{b}{d \cdot \mathrm{Dim}_x} \tag{7.3}$$

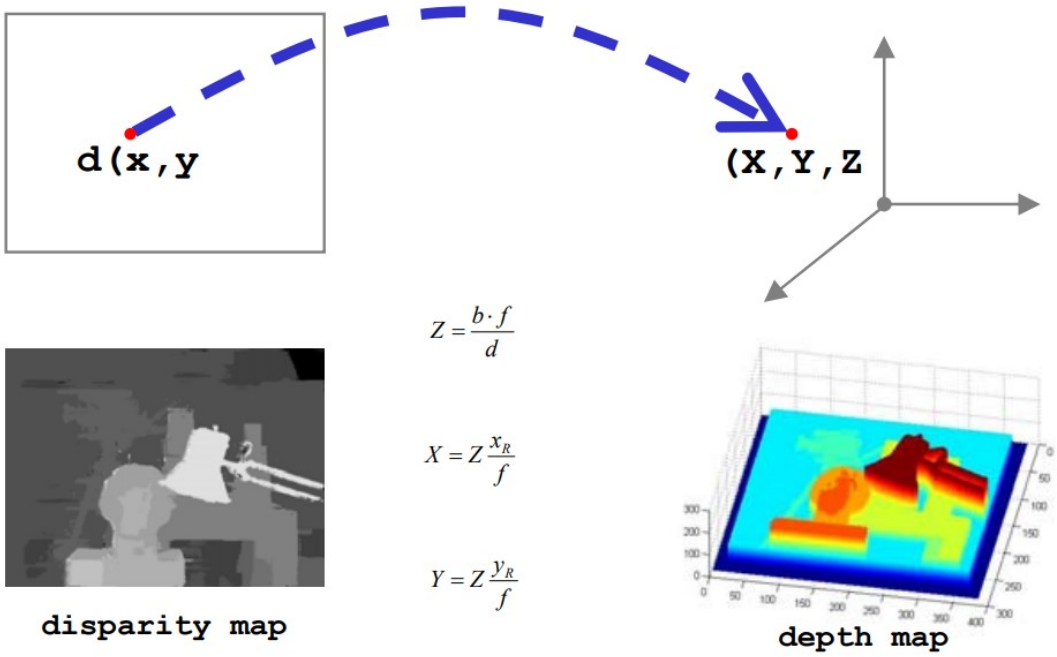


Figura 7.2: Generazione della point cloud a partire dalla mappa di disparità e parametri del sistema stereo.

Per la coordinata X, dalla similitudine tra i triangoli $O_l PA$ e $O_l C_l x_l$, risulta:

$$\frac{Z}{X} = \frac{f}{x_l} \tag{7.4}$$

Da cui:

$$X = Z\frac{x_l}{f} \tag{7.5}$$

Analogamente per Y, risulta:

$$Y = Z\frac{y_l}{f} \tag{7.6}$$

Mediante le relazioni precedenti si ottengono le coordinate X,Y,Z del punto P rispetto al sistema di riferimento con origine nel centro ottico O_l. La figura 7.2 schematizza il processo di calcolo della nuvola di punti (X,Y,Z) a partire dalla mappa di disparità e i parametri del sistema stereo calcolati durante la calibrazione.

7.3 Dalla mappa di disparità ai punti 3D

Per il calcolo della point cloud, detta anche depth map, ho utilizzato una funzione apposita della libreria OpenCV chiamata reprojectImageTo3D(), la quale ha come parametri di ingresso [29]:

- **Disparità**: immagine di disparità in ingresso a canale singolo a 8 bit senza segno, oppure con segno a 16 bit, con segno a 32 bit o a virgola mobile a 32 bit. Si presume che i valori dei formati con segno a 8 bit/16 bit non abbiano

bit frazionari. Se la disparità è in formato con segno a 16 bit, come calcolato da StereoBM() o StereoSGBM() e forse altri algoritmi, dovrebbe essere diviso per 16 (e ridimensionato in virgola mobile) prima di essere utilizzato qui;

- **Q**: matrice di trasformazione prospettica 4 × 4 ottenuta durante la rettificazione delle immagini e output della funzione stereoRectify();

- **handleMissingValues**: indica se la funzione deve gestire i valori mancanti (ovvero i punti in cui la disparità non è stata calcolata). Se handleMissingValues = true, i pixel con la disparità minima che corrisponde ai valori anomali (vedere StereoMatcher.compute()) vengono trasformati in punti 3D con un valore Z molto grande (attualmente impostato su 10000);

- **ddepth**: profondità dell'array di output. Se è -1, l'immagine di output avrà una profondità CV_32F. ddepth può anche essere impostato su CV_16S, CV_32S o CV_32F.

Come output si ha l'immagine 3dImage a virgola mobile a 3 canali della stessa dimensione della disparità. Ogni elemento di 3dImage (x, y) contiene le coordinate 3D del punto (x, y) calcolato dalla mappa di disparità. Se si utilizza Q ottenuto da stereoRectify(), i punti restituiti sono rappresentati nel sistema di coordinate rettificate della prima telecamera.
La funzione trasforma una mappa di disparità a canale singolo in un'immagine a 3 canali che rappresenta una superficie 3D. Cioè, per ogni pixel (x, y) e la corrispondente disparità d = disparità (x, y), calcola:

$$\begin{bmatrix} X \\ Y \\ Z \\ W \end{bmatrix} = Q \begin{bmatrix} x \\ y \\ \text{disparity } (x,y) \\ z \end{bmatrix} \tag{7.7}$$

Questa funzione mi ha permesso di calcolare la point cloud e quindi di ottenere i punti 3D della mappa di disparità. Per quanto riguarda la visualizzazione a video, argomento molto importante soprattutto a supporto del cliente, ho sfruttato una libreria chiamata open3D, la quale permette la visualizzazione di geometrie 3D e viene spiegata dettagliatamente nella sezione successiva.

7.4 Open3D

Il mondo è tridimensionale.
I sistemi che operano nel mondo fisico o che si occupano della sua simulazione devono spesso elaborare dati tridimensionali. Tali dati assumono la forma di nuvole di punti (point cloud), mesh e altre rappresentazioni. I dati in questa forma sono prodotti da sensori come LiDAR e telecamere di profondità e da sistemi software che supportano la ricostruzione e la modellazione 3D [30]. Nonostante il ruolo centrale dei dati 3D in campi come la robotica e la computer grafica, scrivere un software che elabori tali dati è piuttosto laborioso rispetto ad altri tipi di dati. Ad esempio, un'immagine può essere caricata e visualizzata in modo efficiente con poche righe

di codice OpenCV. Non è emerso un framework software altrettanto facile da usare per i dati 3D paragonabile a open3D. Un notevole sforzo precedente è la Point Cloud Library (PCL) [30]. Sfortunatamente, dopo un afflusso iniziale di contributi opensource, il PCL è diventato ingombrante ed è ora in gran parte dormiente.
Altri sforzi open source includono MeshLab [30], che fornisce un'interfaccia utente grafica per l'elaborazione delle mesh; libigl [30], che supporta la geometria differenziale discreta e la ricerca correlata e una varietà di soluzioni per la ricostruzione basata su immagini [30]. Tuttavia, attualmente non esiste una libreria open source veloce, facile da usare, che supporti i flussi di lavoro di elaborazione dati 3D comuni e sia sviluppata in conformità con le moderne pratiche ingegneristiche, paragonabile a Open3D.
Open3D è stato creato per rispondere a tutte queste esigenze. È una libreria opensource che supporta lo sviluppo rapido di software che si occupa di dati 3D. Il frontend Open3D espone un insieme di strutture di dati e algoritmi accuratamente selezionati sia in C ++ che in Python. Il backend Open3D è implementato in C ++, è altamente ottimizzato ed è configurato per la parallelizzazione OpenMP. Open3D è stato sviluppato con un insieme di dipendenze piccolo e attentamente considerato. Può essere impostato su diverse piattaforme e compilato dai sorgenti con il minimo sforzo. Open3D è in sviluppo dal 2015 ed è stato utilizzato in numerosi progetti di ricerca pubblicati [30].

7.4.1 Design

Due principi di progettazione principali di Open3D sono l'utilità e la facilità d'uso. L'utilità ha motivato il supporto per rappresentazioni, algoritmi e piattaforme popolari. La facilità d'uso funge da forza di contrasto che protegge dalle dipendenze pesanti e dallo scivolamento delle funzionalità. Open3D fornisce strutture di dati per tre tipi di rappresentazioni: nuvole di punti, mesh e immagini RGB-D. Per ogni rappresentazione, esiste un set completo di algoritmi di elaborazione di base come I/O, campionamento, visualizzazione e conversione dei dati.
Inoltre, esiste una raccolta di algoritmi ampiamente utilizzati, come la stima delle normali, la registrazione ICP e l'integrazione volumetrica [30].L'insieme di dipendenze "leggere" comprende Eigen per l'algebra lineare, GLFW per il supporto di finestre OpenGL e FLANN per la ricerca veloce del vicino più vicino [30]. Per una compilazione facile, alcune librerie come Boost e Ceres sono escluse [30]. Invece Open3D utilizza alternative "leggere" o implementazioni interne. Il codice sorgente di tutte le dipendenze è distribuito come parte di Open3D. Le dipendenze possono così essere compilate dai sorgenti se non rilevate automaticamente dallo script di configurazione. Ciò è particolarmente utile per la compilazione su sistemi operativi privi di software di gestione dei pacchetti, come Microsoft Windows.
Lo sviluppo di Open3D è iniziato da zero e la libreria è mantenuta il più semplice possibile. Vengono aggiunti solo algoritmi che risolvono un problema di ampio interesse.
Se un problema ha più soluzioni, viene scelta quella che la comunità considera standard. Un nuovo algoritmo viene aggiunto solo se la sua implementazione dimostra risultati significativamente migliori su un benchmark ben noto. Open3D è scritto

nello standard C ++ e utilizza CMake per supportare le comuni toolchain C ++ tra cui:

- GCC 4.8 e versioni successive su Linux;
- XCode 8.0 e versioni successive su OS X;
- Visual Studio 2015 e versioni successive su Windows.

Una caratteristica fondamentale di Open3D è l'onnipresente associazione Python. Open3D è avvezza a problemi di visione artificiale e deep learning: il backend è implementato in C ++ ed è esposto attraverso interfacce frontend in Python. Gli sviluppatori usano Python come linguaggio collante per assemblare i componenti implementati nel backend. L'implementazione che utilizza l'interfaccia Open3D Python è circa la metà della lunghezza dell'implementazione che utilizza l'interfaccia Open3D C ++ e circa cinque volte più breve dell'implementazione basata su PCL. Come ulteriore vantaggio, il codice Python può essere modificato, ed eseguito il debug in modo interattivo in un notebook Jupyter.

7.4.2 Dati

Il modulo Geometry implementa tre rappresentazioni geometriche: PointCloud, TriangleMesh e Image. La struttura dati PointCloud ha tre campi dati: PointCloud.points, PointCloud.normals, PointCloud.colors. Sono usati per memorizzare coordinate, normali e colori. Il campo di dati principale è PointCloud.points, gli altri due campi sono considerati validi solo quando hanno lo stesso numero di record di PointCloud.points. Open3D fornisce accesso alla memoria diretto a questi campi dati tramite un array numpy. Allo stesso modo, TriangleMesh ha due campi di dati principali TriangleMesh.vertices e TriangleMesh.triangles oltre a tre campi di dati ausiliari: TriangleMesh.vertex_normals, TriangleMesh.vertex_colors e TriangleMesh.triangle_normals. La struttura dei dati dell'immagine è implementata come un array 2D o 3D e può essere convertita direttamente in un array numpy. Una coppia di immagini di profondità e colore della stessa risoluzione può essere combinata in una struttura dati denominata RGBDImage.

Nel nostro caso per la Point Cloud ho utilizzato la struttura dati specifica assegnando a PointCloud.points i punti ottenuti tramite la funzione reprojectImageTo3D() citata precedentemente per poi stimare le normali di ognuno dei punti, tramite la funzione estimate_normals(), e assegnarle a PointCloud.normals.

7.4.3 Visualizzazione

Open3D fornisce una funzione draw_geometries() per la visualizzazione. Prende un elenco di geometrie come input, crea una finestra e le renderizza simultaneamente utilizzando OpenGL. Sono implementate molte funzioni nel visualizzatore, come rotazione, traslazione e ridimensionamento tramite operazioni del mouse, modifica dello stile di rendering e cattura dello schermo.
Oltre a draw_geometries(), Open3D ha una serie di funzioni simili con funzionalità più avanzate: draw_geometry_with_custom_animation(), consente al programmatore

di definire una traiettoria di visualizzazione personalizzata e riprodurre un'animazione nella GUI;
draw_geometry_with_animation_callbacks() e draw_geometry_with_key_callbacks(), accettano le funzioni di callback Python come input. La funzione di callback viene chiamata in un ciclo di animazione automatico o in seguito a un evento di pressione di un tasto.

7.5 Ricostruzione della scena 3D: point cloud

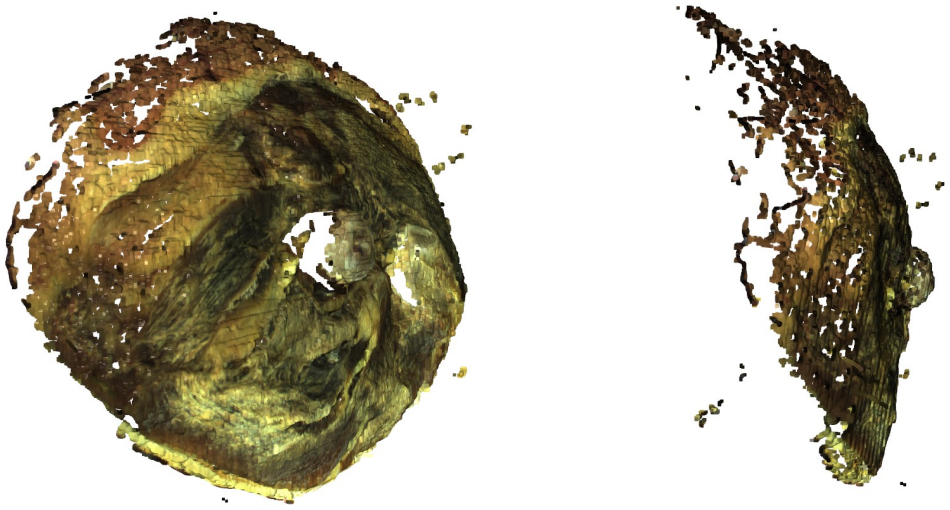
7.5.1 Point cloud e eliminazione dello sfondo

Abbiamo visto nella sezione 7.3 come ho calcolato i punti 3D tramite i parametri di rettificazione e la mappa di disparità, fatto ciò sono passato alla creazione della point cloud, ma che cos'è esattamente la point cloud?
Una **nuvola di punti** (point cloud) è un insieme di punti caratterizzati dalla loro posizione in un sistema di coordinate e da eventuali valori di intensità (colore, profondità, ecc.) ad essi associati, nel nostro caso vediamo le coordinate omogenee dei punti che la compongono nell'equazione 7.7.
Le nuvole di punti servono, di solito, come rappresentazione di strutture tridimensionali come oggetti o superfici in rilievo.
In Open3D è molto semplice passare dai punti 3D ottenuti dalla mappa di disparità all'"oggetto" point cloud; fatto ciò il passaggio successivo è stato quello di eliminare quei punti che non facevano parte del prosciutto, ovvero tutti quei punti che rappresentavano lo sfondo, oppure degli outliers.
Per eliminare lo sfondo c'erano 2 metodi possibili:

1. Eliminare dalla mappa di disparità tutti quei punti che stavano al di sotto di un certo valore di disparità (una sorta di soglia), scelto con la logica di eliminare tutti quei punti al di sotto del valore più basso che appartiene al prosciutto;

2. Utilizzare la segmentazione ottenuta tramite la rete neurale in modo da eliminare tutti i punti che non fanno parte delle label predette e che quindi corrispondono allo sfondo.

Il primo metodo ha un problema: selezionare un valore di soglia che elimini completamente lo sfondo senza però toccare il prosciutto è molto difficile perchè le disparità non sono omogenee in tutte le zone e inoltre anche la mappa di disparità è affetta da outliers, come vediamo in figura 4.16. Il secondo metodo risolve il problema relativo al primo perchè vengono mantenute soltanto le zone che effettivamente appartengono al prosciutto, ma ha un altro problema, i punti non riconosciuti dalla rete, i quali però appartengono fisicamente al prosciutto, vengono scartati.
Per questi motivi ho deciso di intraprendere una soluzione "mista", ho eliminato tutti i punti 3D non appartenenti alle label ottenute dalla segmentazione della rete (secondo metodo), i punti non riconosciuti dalla rete, ma che sono un intorno o comunque sono contornati da altri punti che invece sono stati riconosciuti, vengono comunque inseriti nella point cloud. In questo modo sono riuscito a ricostruire in 3D l'intero prosciutto.



(a) Vista a tre quarti.

(b) Vista di profilo.

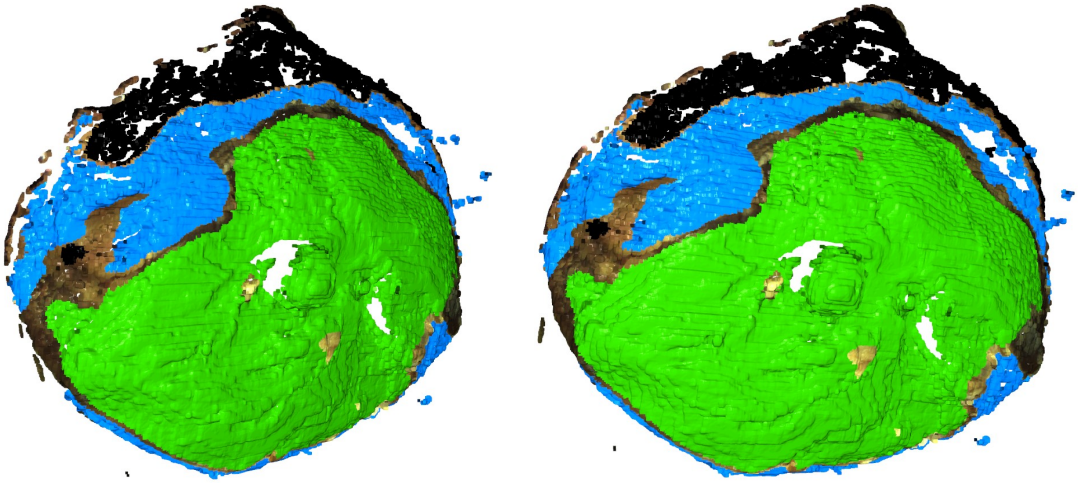
Figura 7.3: Ricostruzione del prosciutto in 3D con eliminazione dello sfondo, visualizzazione tramite Open3D.

In secondo luogo ho aggiunto un controllo di soglia (primo metodo), nonostante lo sfondo fosse stato eliminato rimaneva sempre qualche outliers con disparità molto più bassa rispetto a quella del prosciutto, per questo ho impostato una soglia molto grossolana per eliminare gli outliers molto evidenti.
In figura 7.3 vediamo il risultato finale di questo procedimento.

7.5.2 Segmentazione in 3D

Ottenuto il modello 3D del prosciutto il passaggio successivo è quello di visualizzare in 3D anche la segmentazione ottenuta tramite la rete. Per fare ciò basterà modificare il colore dei punti che appartengono alle varie label predette dalla rete, i punti che rimarranno esclusi da tutte le label rimarranno del loro colore originale.
Come spiegato dettagliatamente nella sezione 7.4.2 in Open 3D l'oggetto PointCloud è composto da 3 campi: PointCloud.points, PointCloud.normals, PointCloud.colors. Quindi per inserire la segmentazione nella visualizzazione 3D mi è bastato andare a modificare il campo PointCloud.colors; ho modificato il colore dei punti della point cloud (ovviamente con un colore diverso in base alla label) che corrispondono in coordinate x,y a quelli delle label predette dalla rete, lasciando inalterati quelli che invece non corrispondevano. Il risultato è mostrato in figura 7.4.

Come detto nel capitolo 6 la segmentazione dell'osso non è stata possibile tramite la rete neurale con il database fornitomi. Questo non è un grosso problema, i valori di disparità dell'osso sono molto riconoscibili perchè decisamente più alti rispetto al piano della sugna del prosciutto, come evidenziato in figura 7.3b. Per questi motivi la segmentazione dell'osso del prosciutto risulta molto più facile ed è possibile anche eseguirla nel momento in cui vengono forniti i punti 3D da sugnare al robot (in quel momento oltre ai punti appartenenti all'osso verranno esclusi anche



(a) Vista a tre quarti.

(b) Vista di fronte.

Figura 7.4: Ricostruzione del prosciutto in 3D con segmentazione predetta dalla rete, visualizzazione tramite open3D.

quelli appartenenti al grasso e alla cotenna). Infatti basterà trovare un valore di soglia di disparità abbastanza valido per escludere dal labeling della sugna i punti appartenenti all'osso. In figura 7.5 vediamo il labeling ottimale posto come obiettivo da raggiungere.

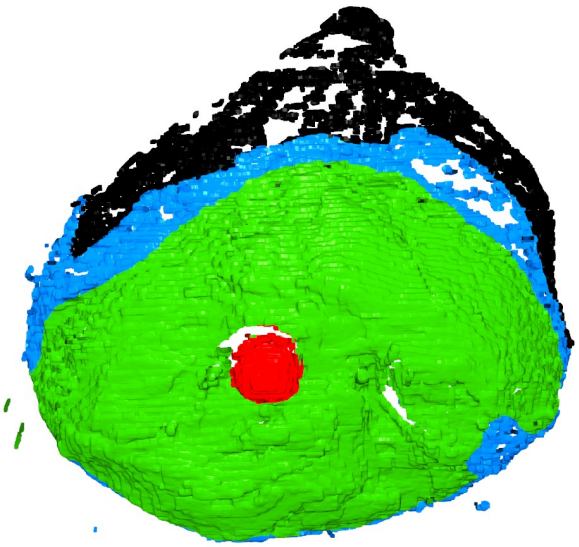


Figura 7.5: Segmentazione 3D ottimale con anche la label dell'osso del prosciutto.

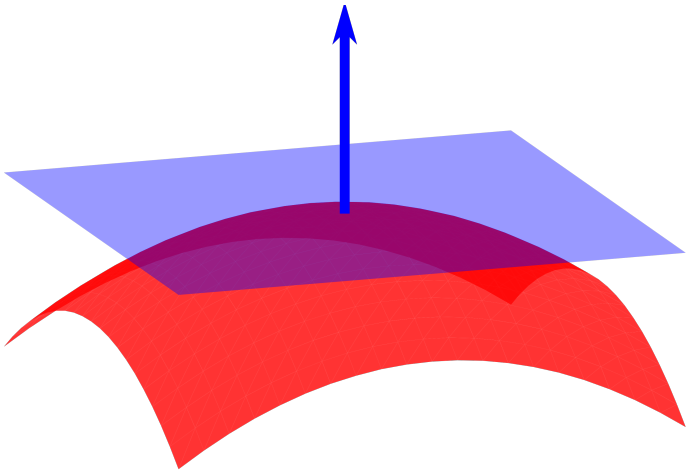


Figura 7.6: Esempio di retta normale a una superficie non piana in un punto.

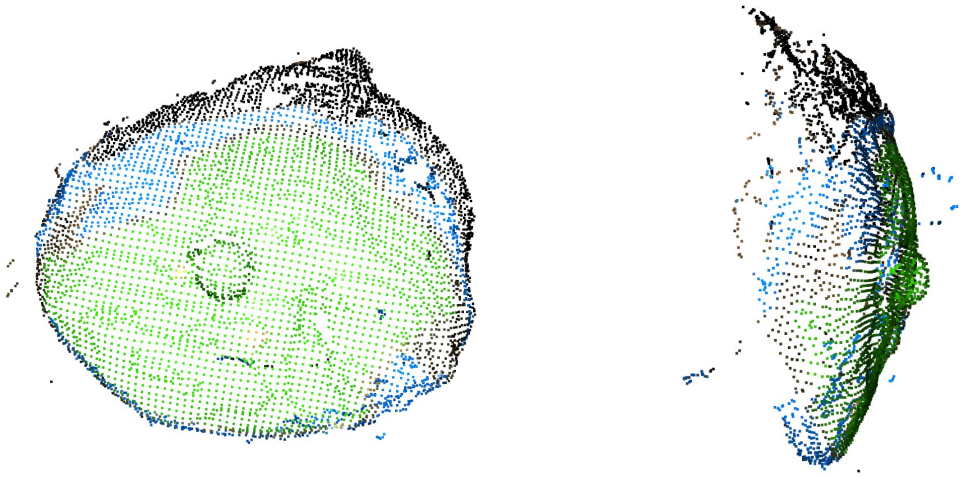
7.5.3 Stima delle norme dei punti 3D

Il robot che dovrà poi "spennellare" la parte del prosciutto indicata come "da sugnare" ha bisogno di sapere in che modo impugnare il pennello e con che angolazione approcciarsi alla superficie per non lasciare buchi vuoti o per evitare di perdere tempo inutilmente. Per questo motivo è indispensabile la stima delle normali alla superficie nei punti 3D calcolati.
In matematica, una normale a una superficie piana è un vettore tridimensionale perpendicolare a quella superficie. Una normale ad una superficie non piana nel punto p su quella superficie è un vettore perpendicolare al piano tangente a quella superficie in p, vediamo un esempio in figura 7.6.

Con la triangolazione abbiamo già calcolato le coordinate z dei punti della mappa di disparità in modo da poterli collocare nello spazio 3D, quindi il calcolo delle normali alla superficie totale in quei punti è un calcolo standard e abbastanza semplice. Con Open3D ciò avviene tramite la funzione estimate_normals(), la quale viene "chiamata" dall'oggetto point cloud. Lo scopo di questa funzione è quello di calcolare le normali alla superficie in tutti i punti appartenenti alla point cloud e inserire questi vettori nella struttura dato PointCloud.normals.
Matematicamente parlando il procedimento svolto per il calcolo delle normali alla superficie nei punti è standard: viene identificato il piano tangente alla superficie nel punto p (di cui conosco le coordinate) tramite la derivata della funzione, quindi il vettore normale è il vettore prodotto vettoriale dei due lati non paralleli del piano tangente nel punto p.
La normale ad una superficie non ha un unico verso; il vettore che punta nel verso opposto della normale alla superficie è anch'esso una normale a quella superficie. Per una superficie orientata, la normale alla superficie è solitamente determinata dalla "regola della mano destra".

Per la visualizzazione delle normali dei punti in Open3D è consigliato apportare un sotto campionamento alla point cloud in modo da avere meno punti e quindi una visualizzazione più semplice. Per fare ciò ho utilizzato una funzione di Open3D chiamata voxel_down_sample().

In generale un voxel (volumetric picture element) è un'unità di misura del volu-



(a) Vista di fronte.

(b) Vista di profilo.

Figura 7.7: Point cloud sotto campionata di un 20% rispetto l'originale tramite downsample voxel.

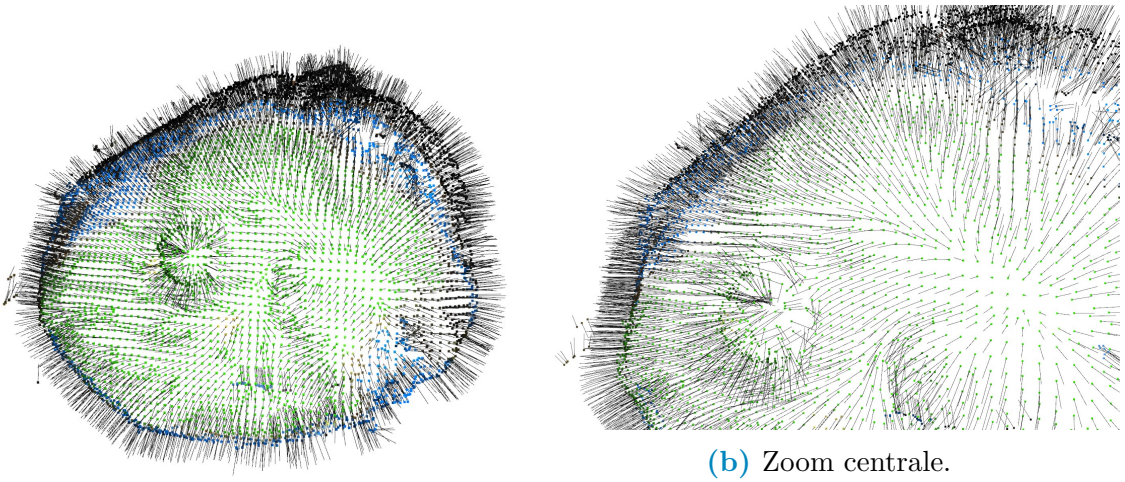
me. Il voxel è la controparte tridimensionale del pixel bidimensionale (che rappresenta l'unità dell'area) e perciò il volume buffer (un ampio array 3D di voxel) dei voxel può essere considerato come la controparte tridimensionale del frame buffer bidimensionale dei pixel.
Nel nostro caso la definizione giusta di voxel è: *"un punto (campione) in una griglia 3D ad intervalli regolari"* Il downsampling voxel utilizza una griglia voxel regolare per creare una nuvola di punti uniformemente sotto campionata da una nuvola di punti di input. Viene spesso utilizzato come fase di pre-elaborazione per molte attività di elaborazione delle nuvole di punti.
L'algoritmo opera in due fasi:

1. I punti vengono raggruppati in voxel;

2. Ogni voxel occupato genera esattamente un punto calcolando la media di tutti i punti all'interno.

Per quanto riguarda il mio progetto ho deciso di sotto campionare la point cloud di un 20% rispetto all'originale, in questo modo ho ottenuto il risultato in figura 7.7.

A questo punto l'ultimo passaggio di questa prima fase è quello di visualizzare le normali calcolate precedentemente con la funzione estimate_normals(). Nell'interfaccia grafica di Open3D per mostrare le normali nei punti basta premere il tasto n durante la visualizzazione. Nel nostro caso il risultato è quello di figura 7.8.



(a) Vista di fronte.

(b) Zoom centrale.

Figura 7.8: Point cloud sotto campionata di un 20% rispetto l'originale tramite downsample voxel, con visualizzazione delle normali.

7.6 Ricostruzione della scena 3D: mesh

Una mesh poligonale è un reticolo che definisce un oggetto nello spazio, composto da vertici, spigoli e facce. Il termine mesh in inglese significa letteralmente "maglia", "rete".
Negli anni recenti, le mesh poligonali sono divenute via via sempre più popolari e al giorno d'oggi sono usate intensivamente in molte differenti aree della computer grafica e della geometry processing (o elaborazione della geometria, che è un campo relativamente nuovo dell'informatica che concerne algoritmi e modelli matematici per l'analisi e la manipolazione di dati geometrici) [31].

Intuitivamente, una mesh poligonale è la partizione di una superficie continua in celle poligonali, come triangoli, quadrilateri, ecc. Più formalmente, una mesh $\mathcal{M}$ può essere definita come una tupla $(\mathcal{V}, \mathcal{K})$, dove $\mathcal{V} = \{v_i \in \mathbb{R}^3 | i = 1...N_v\}$ è l'insieme dei vertici del modello (punti in $\mathbb{R}^3$) e $\mathcal{K}$ contiene l'adjacency information o, in altre parole, come i vertici sono connessi per formare gli spigoli e le facce della mesh. Per esempio una mesh composta da una singolo triangolo sarebbe $(\{v_0, v_1, v_2\}, \{\{v_0, v_1\}\{v_1, v_2\}\{v_2, v_0\}\{v_0, v_1, v_2\}\})$, ossia i tre vertici, i tre spigoli e il triangolo.
Una mesh, differentemente da un oggetto solido reale, non presenta una massa; è quindi una sorta di volume vuoto, privo di spessore, le cui facce sono appunto dei "veli" superficiali. I componenti visibili di una mesh sono:

- **Vertice**: punto dello spazio, dotato quindi di coordinate x, y, z che ne determinano la posizione. In inglese: vertex;
- **Spigolo**: segmento che congiunge due vertici nello spazio. In inglese: edge;
- **Faccia**: definita attraverso la connessione e chiusura di almeno tre spigoli. In inglese: face.

Le mesh più usate in computer grafica sono le triangle meshes e le quadrilateral meshes (abbreviate in quad meshes). In questo progetto, per "mesh" intendiamo triangle mesh. Altre mesh, nelle quali gli elementi di base sono quadrilateri o altri poligoni, sono a volte utilizzate, ma possono verificarsi inconvenienti. Per esempio, è facile creare un quadrilatero i cui vertici non giacciono tutti sullo stesso piano, mentre c'è sempre un piano passante per tre vertici. Inoltre, lavorare esclusivamente con triangle meshes semplifica la memorizzazione e riduce il numero degli algoritmi.

Una **mesh triangolare** è un tipo di mesh poligonale in computer grafica. Comprende un insieme di triangoli (tipicamente in tre dimensioni) collegati dai loro bordi o angoli comuni. Molti pacchetti software grafici e dispositivi hardware possono funzionare in modo più efficiente su triangoli raggruppati in mesh rispetto a un numero simile di triangoli presentati individualmente. Ciò è in genere dovuto al fatto che la grafica computerizzata esegue operazioni sui vertici agli angoli dei triangoli. Con i singoli triangoli, il sistema deve operare su tre vertici per ogni triangolo. In una mesh di grandi dimensioni, potrebbero esserci otto o più triangoli che si incontrano in un singolo vertice: elaborando quei vertici una sola volta, è possibile fare una frazione del lavoro e ottenere un effetto identico. In molte applicazioni di computer grafica è necessario gestire una mesh di triangoli. I componenti della mesh sono vertici, spigoli e triangoli. Un'applicazione potrebbe richiedere la conoscenza delle varie connessioni tra i componenti della mesh. Queste connessioni possono essere gestite indipendentemente dalle posizioni effettive dei vertici.
Un modo per condividere i dati dei vertici tra i triangoli è la **triangle strips**. In questo modo ogni triangolo condivide un bordo completo con un vicino e un altro con il successivo. Un altro modo è il ventaglio triangolare che è un insieme di triangoli collegati che condividono un vertice centrale. Con questi metodi i vertici vengono gestiti in modo efficiente, determinando la necessità di elaborare solo N + 2 vertici per disegnare N triangoli. Le triangle strips sono efficienti, tuttavia lo svantaggio è che potrebbe non essere ovvio o conveniente tradurre una mesh triangolare arbitraria in strips.

Open3D mette a disposizione un oggetto TriangleMesh che racchiude tutte le caratteristiche delle mesh citate sopra. Inoltre è possibile arrivare alla mesh partendo dalla point cloud, ed è proprio quello che ho fatto.
Infatti per arrivare alla mesh del prosciutto ho sfruttato la funzione di Open3D chiamata TriangleMesh.create_from_point_cloud_ball_pivoting(), la quale calcola una mesh triangolare da una point cloud orientata. Implementa l'algoritmo Ball Pivoting proposto [32]. L'implementazione si basa anche sugli algoritmi delineati in [33]. La ricostruzione della superficie viene eseguita facendo rotolare una pallina con un dato raggio sopra la nuvola di punti, ogni volta che la pallina tocca tre punti viene creato un triangolo. Vedremo nel dettaglio questo algoritmo nel paragrafo successivo, per ora ci basti sapere che la funzione TriangleMesh.create_from_point_cloud_ball_pivoting() ha come parametri di input:

- **pcd**: point cloud da cui viene ricostruita la superficie TriangleMesh. Deve contenere le normali;
- **radii**: i raggi della palla che vengono utilizzati per la ricostruzione della superficie.

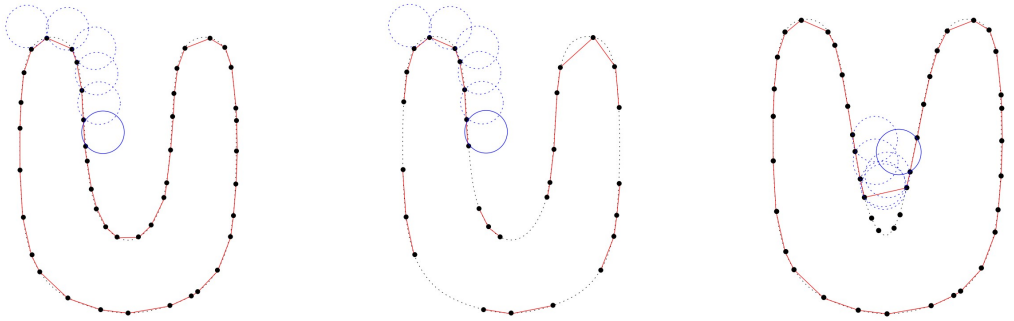
E come output ovviamente l'oggetto TriangleMesh.

7.6.1 Ball-Pivoting per la ricostruzione della superficie

Negli ultimi anni si è assistito a una proliferazione di strumenti di scansione e algoritmi per la sintesi di modelli dai dati scansionati [32]. Nella sezione seguente ci concentriamo sul ruolo che gli schemi di mesh interpolanti possono svolgere nella scansione di oggetti e sul motivo per cui non sono stati utilizzati nei sistemi di scansione pratici.

Interpolating meshes nei sistemi di scansione

Definiamo il problema della scansione: dato un oggetto, si trovi una rappresentazione continua della superficie dell'oggetto che cattura le feature in una scala di lunghezza 2d o più grande. Il valore di d è dettato dall'applicazione. L'acquisizione di elementi in scala 2d richiede il campionamento della superficie con una risoluzione spaziale uguale a d o inferiore. La superficie può essere costituita da ampie aree che possono essere ben approssimate da maglie molto più rade; tuttavia, in assenza di informazioni a priori, è necessario iniziare con una risoluzione di campionamento pari o inferiore a d per garantire che nessuna feature venga persa. Consideriamo sistemi di acquisizione che producono insiemi di immagini di distanza, cioè matrici di profondità, ciascuna delle quali copre un sottoinsieme dell'intera superficie. Poiché sono campi di altezza con campionamento regolare, le singole immagini dell'intervallo sono facilmente "meshabili". Le singole mesh possono essere utilizzate per calcolare una normale alla superficie stimata per ogni punto campione. Un sistema di acquisizione ideale restituirebbe campioni che giacciono esattamente sulla superficie dell'oggetto. Qualsiasi sistema di misurazione reale introduce qualche errore. Tuttavia, se un sistema restituisce campioni con un errore dell'ordine di grandezza inferiore alla dimensione minima dell'elemento, il campionamento può essere considerato perfetto. Una superficie può quindi essere ricostruita trovando una mesh interpolante senza operazioni aggiuntive sui dati misurati. La maggior parte dei sistemi di scansione deve ancora tenere conto dell'errore di acquisizione. Esistono due fonti di errore: **errore nella registrazione** e **errore lungo la linea di visione del sensore** [32]. Le stime dei punti di superficie effettivi sono generalmente derivate dalla media dei campioni da scansioni ridondanti. Queste stime vengono quindi collegate in una triangle mesh. La maggior parte dei metodi per stimare i punti della superficie dipendono da strutture di dati che facilitano la costruzione della mesh. Due classi di metodi sono state utilizzate con successo per set di dati di grandi dimensioni; entrambi presumono un errore di registrazione trascurabile e calcolano stime per correggere l'errore di linea di vista. La prima di queste classi è costituita dai **metodi volumetrici**, come quello introdotto da Curless e Levoy [32]. In questi metodi, le singole mesh allineate vengono utilizzate per calcolare una funzione di distanza, con segno, su una griglia di volume che racchiude l'oggetto. I punti superficie stimati vengono calcolati come i punti sulla griglia in cui la funzione di distanza è zero. La seconda classe di metodi sono i metodi di **mesh stiching**, come la tecnica di Soucy e Laurendeau [32]. Le mesh disgiunte del campo altezza vengono cucite in una singola superficie. Le regioni disgiunte vengono definite individuando aree di sovrapposizione di diversi sottoinsiemi dell'insieme di scansione. I punti superficie stimati per ciascuna regione vengono calcolati come medie ponderate di punti dalle scansioni sovrapposte. I punti stimati in ciascuna regione vengono quindi



(a) (b) (c)

Figura 7.9: L'algoritmo di Ball-Pivoting in 2D. (a) Un cerchio di raggio ρ ruota da punto campione a punto campione, collegandoli con i bordi. (b) Quando la densità di campionamento è troppo bassa, alcuni dei bordi non verranno creati, lasciando dei buchi. (c) Quando la curvatura del collettore è maggiore di $1/\rho$, alcuni dei punti di campionamento non verranno raggiunti dalla pivoting ball e le feature verranno perse.

re-triangolati e le mesh risultanti vengono cucite in una singola mesh. Turk e Levoy hanno sviluppato un metodo simile [32], che prima "ricama" le mesh disgiunte e poi calcola i punti di superficie stimati. Osserviamo che in entrambe le classi di metodi, il metodo di stima dei punti della superficie non deve essere strettamente collegato al metodo per costruire la mesh finale. Nell'approccio volumetrico, una tecnica diversa marching cube potrebbe essere utilizzata per trovare una maglia triangolare che passa attraverso i punti di superficie stimati. Negli approcci di unione delle maglie, una tecnica per trovare una maglia che collega tutti i punti superficie stimati potrebbe essere utilizzata al posto di unire insieme le maglie esistenti. Ancora più importante, con un algoritmo efficiente per il calcolo di una mesh che unisce i punti, si potrebbe utilizzare qualsiasi metodo per calcolare i punti superficie stimati, compresi quelli che non impongono una struttura aggiuntiva ai dati e non trattano separatamente la registrazione e l'errore di visuale. Ad esempio, è stato dimostrato che la riduzione dell'errore nelle singole maglie prima dell'allineamento può ridurre l'errore di registrazione [32]. Infine, può essere desiderabile trovare una mesh interpolante dai dati misurati anche se contiene un errore non compensato.

Le tecniche di interpolazione esistenti si dividono in due categorie: **sculpting-based** and **region-growing** [32], come il **BPA**. Nei metodi sculpting-based, una tetraedrizzazione del volume viene calcolata dai punti dati, tipicamente la triangolazione 3D Delaunay. I tetraedri vengono quindi rimossi dalla superficie convessa per estrarre la forma originale. I metodi di region-growing iniziano con un triangolo seme, prendono in considerazione un nuovo punto e lo uniscono al confine della regione esistente e continuano fino a quando tutti i punti sono stati considerati. Il punto di forza degli approcci sculpting-based è che spesso forniscono garanzie teoriche per la qualità della superficie risultante, ad esempio che la topologia è corretta e che la superficie converge alla superficie reale all'aumentare della densità di campionamento. Tuttavia, il calcolo della triangolazione 3D Delaunay richiesta può essere proibitivo in termini di tempo e memoria richiesti e può portare a instabilità numerica quan-

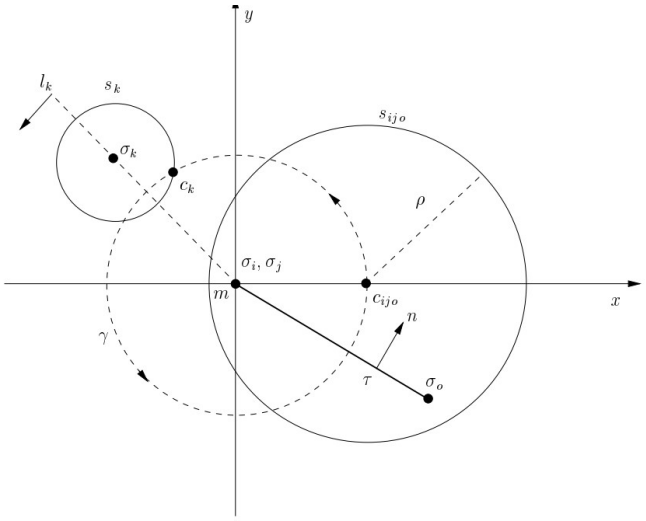


Figura 7.10: Operazione di ball pivoting. La sfera girevole è in contatto con i tre vertici del triangolo $\tau = (\sigma_i; \sigma_j; \sigma_o)$, la cui normale è n. Il bordo girevole $e_{(i;j)}$ giace sull'asse z (perpendicolare alla pagina e rivolto verso lo spettatore), con il suo punto medio m all'origine. Il cerchio s_{ijo} è l'intersezione della sfera rotante con z = 0. Il frame delle coordinate è tale che il centro c_{ijo} della sfera giace sull'asse x positivo. Durante la rotazione, la palla rimane in contatto con i due estremi del bordo σ_i, σ_j, e il suo centro descrive una traiettoria circolare γ con centro in m e raggio $||c_{ijo} - m||$. Nel suo movimento di rotazione, la palla colpisce un nuovo punto dati σ_k. Sia s_k l'intersezione di una ρ-sfera centrata in σ_k con z = 0. Il centro c_k della pivoting ball quando tocca k è l'intersezione di γ con s_k giacente sul semipiano negativo della retta orientata l_k.

do si tratta di set di dati di milioni di punti. L'obiettivo del BPA è mantenere i punti di forza delle precedenti tecniche di interpolazione in un metodo che mostra complessità temporale lineare e robustezza sui dati scansionati reali.

Ricostruzione della superficie e Ball-Pivoting

Il concetto principale alla base dell'algoritmo ball-pivoting è abbastanza semplice. Sia la varietà M la superficie di un oggetto tridimensionale e S un campionamento puntuale di M. Supponiamo per ora che S sia abbastanza denso in modo che una $\rho - ball$ (una palla di raggio ρ) non possa passare attraverso la superficie senza toccare punti campione (vedere la figura 7.9 per un esempio 2D). Iniziamo mettendo una $\rho - ball$ a contatto con tre punti campione. Mantenendo la palla a contatto con due di questi punti iniziali, facciamo "perno" fino a che arrivi a toccare un altro punto, come illustrato in figura 7.10. Ruotiamo attorno a ciascun bordo del confine della mesh corrente.
Triplette di punti con cui la palla entra in contatto formano nuovi triangoli. L'insieme dei triangoli formati mentre la pallina "cammina" sulla superficie costituisce la mesh interpolante. Il BPA è strettamente correlato alle alpha-shapes [32]. Infatti ogni triangolo τ calcolato dalla $\rho - ball$ ha ovviamente una pallina aperta più piccola vuota b_τ il cui raggio è minore di ρ.
Pertanto, il BPA calcola un sottoinsieme delle 2-facce della $\rho - shapes$ di S. Queste

facce sono anche un sottoinsieme del 2-skeleton della triangolazione tridimensionale di Delaunay dell'insieme di punti. Le alpha-shapes sono uno strumento efficace per calcolare la "forma" di un insieme di punti. La superficie ricostruita dal BPA conserva alcune delle qualità delle alpha-shapes: ha garanzie di ricostruzione dimostrabili sotto determinate ipotesi di campionamento e un significato geometrico intuitivamente semplice. Tuttavia, il 2-skeleton di una alpha-shapes calcolato dal campionamento rumoroso di un collettore liscio può contenere più connessioni non multiple. Non è banale filtrare i componenti indesiderati.
Inoltre, nella loro formulazione originale, le alpha-shapes vengono calcolate estraendo un sottoinsieme della triangolazione Delaunay 3D del set di punti, una struttura di dati che non è facilmente calcolabile per set di dati di milioni di punti. Il BPA calcola in modo efficiente e robusto un molteplice sottoinsieme di una alpha-shapes che è adatto per questa applicazione. In [34], sono state derivate condizioni sufficienti sulla densità di campionamento di una curva nel piano che garantiscono che la ricostruzione in forma alfa sia omeomorfa alla varietà originale e che si trovi entro una distanza limitata. Il teorema può essere facilmente esteso alle superfici: supponiamo che per il collettore liscio M il campionamento S soddisfi le seguenti proprietà:

1. L'intersezione di qualsiasi sfera di raggio ρ con il collettore è un disco topologico;

2. Qualsiasi sfera di raggio ρ centrata sul collettore contiene almeno un punto di campionamento al suo interno.

La prima condizione garantisce che il raggio di curvatura del collettore sia maggiore di ρ e che la ρ-ball possa passare anche attraverso cavità e altre caratteristiche concave senza contatti multipli con la superficie.
La seconda condizione ci dice che il campionamento è abbastanza denso da consentire alla pallina di camminare sui punti di campionamento senza lasciare buchi (vedi figura 7.9 per esempi 2D). Il BPA produce quindi un'approssimazione omeomorfica T della varietà liscia M. Possiamo anche definire un omeomorfismo $h : T \mapsto M$ tale che la distanza $\|p - h(p)\| < \rho$. In pratica, dobbiamo spesso occuparci di campionamenti tutt'altro che ideali. Qual è il comportamento dell'algoritmo in questi casi? Consideriamo il caso di dati reali scansionati. I problemi tipici sono i punti mancanti, la densità non uniforme, le scansioni della distanza sovrapposta non perfettamente allineate e l'errore di visuale dello scanner. Il BPA è progettato per elaborare l'output di un algoritmo di registrazione/conformità accurato, non cerca di calcolare la media del rumore o degli errori di registrazione residui. Tuttavia, il BPA è robusto in presenza di dati imperfetti.
Aumentiamo i punti dati con normali approssimative della superficie calcolate dalle mappe di intervallo per chiarire i casi che si verificano quando si tratta di dati mancanti o rumorosi. Ad esempio, se parti della superficie non sono state scansionate, ci saranno fori più grandi rispetto al campionamento. È quindi impossibile distinguere una regione interna ed una esterna. Usiamo le normali alla superficie (per le quali assumiamo l'orientamento verso l'esterno) per decidere l'orientamento della superficie. Ad esempio, quando si sceglie un triangolo seme, controlliamo che le normali alla superficie ai tre vertici siano orientate in modo coerente. Aree di densità superiori a quelle presenti, nessun problema. La pivoting ball continuerà a "camminare" sui punti, formando piccoli triangoli. Se i dati sono privi di rumore e sono più piccoli

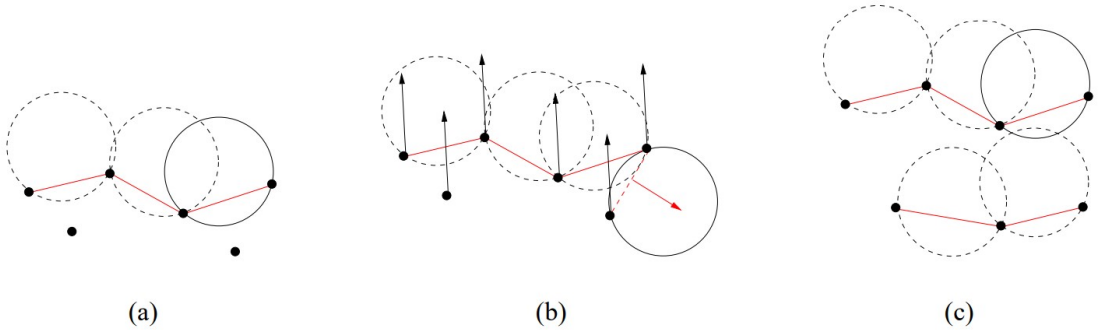


Figura 7.11: Rotazione della palla in presenza di dati rumorosi. (a) I campioni di superficie che si trovano "sotto" il livello della superficie non vengono toccati dalla sfera girevole e rimangono isolati (e vengono scartati dall'algoritmo). (b) A causa della mancanza di dati, la pallina ruota attorno a un bordo fino a toccare un campione che appartiene a una parte diversa della superficie. Controllando che le normali del triangolo e del punto dati siano orientate in modo coerente, in questo caso evitiamo di generare un triangolo. (c) I campioni rumorosi formano due strati, sufficientemente distanti da consentire alla palla di "camminare" su entrambi gli strati. Viene creato un piccolo componente spurio. La strategia di selezione dei semi evita la creazione di un gran numero di questi piccoli componenti. Quelli rimanenti possono essere rimossi con un semplice passaggio di post-elaborazione. In tutti i casi, il BPA emette un collettore triangolare orientabile.

della curvatura locale, tutti i punti verranno interpolati. Più probabilmente, i punti sono influenzati dal rumore e alcuni di quelli che giacciono sotto la superficie non saranno toccati dalla palla e non faranno parte della mesh ricostruita (vedi figura 7.11a).
I punti mancanti creano buchi che non possono essere riempiti dalla ball pivoting. Può essere impiegato qualsiasi algoritmo di post-elaborazione del riempimento dei buchi; in particolare, il BPA può essere applicato più volte, con raggi di sfera crescenti. Tuttavia, dobbiamo gestire le possibili ambiguità che i dati mancanti possono introdurre. Quando si ruota attorno a un bordo perimetrale, la palla può toccare un punto inutilizzato che si trova vicino alla superficie. Ancora una volta usiamo le normali alla superficie per decidere se il punto toccato è valido o meno (vedi figura 7.11b). Un triangolo viene rifiutato se il prodotto scalare della sua normale con la normale alla superficie è negativo. La presenza di scansioni di distanza sovrapposte disallineate può portare a risultati scadenti se l'errore di registrazione è simile alla dimensione della sfera rotante. Si formeranno piccoli componenti collegati indesiderati che giacciono vicino alla superficie principale e la superficie principale sarà interessata dal rumore ad alta frequenza (vedere figura 7.11c). La strategia di selezione dei semi evita di creare un gran numero di componenti così piccoli. Un semplice postprocessing che rimuova i piccoli componenti e la levigatura superficiale può migliorare notevolmente il risultato in questi casi, almeno esteticamente.
Indipendentemente dai difetti nei dati, il BPA è garantito per costruire un collettore orientabile. Si noti che il BPA cercherà sempre di costruire il collettore connesso più grande possibile da un dato triangolo seme. La scelta di un valore adatto per il raggio della ball pivoting è in genere facile. Gli attuali scanner per triangolazione a

luce strutturata o a laser producono campionamenti molto densi, superando il nostro requisito che la distanza tra i campioni sia inferiore alla metà delle dimensioni degli elementi di interesse. La conoscenza delle caratteristiche di densità di campionamento dello scanner e della dimensione del tratto che si vuole catturare sono sufficienti per scegliere un raggio appropriato. In alternativa, si potrebbe analizzare un piccolo sottoinsieme di dati per calcolare la densità dei punti. Potrebbe verificarsi un campionamento irregolare durante la scansione di una superficie complessa, con regioni che proiettano in piccole aree nella direzione dello scanner. L'approccio migliore è eseguire scansioni aggiuntive con lo scanner perpendicolare a tali regioni, per acquisire dati aggiuntivi. Si noti tuttavia che il BPA può essere applicato più volte, con raggi di sfera crescenti, per gestire densità di campionamento non uniformi.

7.6.2 Algoritmo Ball-Pivoting

Il BPA segue il paradigma advancing-front per costruire in modo incrementale una triangolazione interpolante. BPA prende come input un elenco di punti dati del campione di superficie σ_i, ciascuno associato a un n_i normale (e altri attributi opzionali, come le coordinate della trama) e un raggio della sfera ρ. L'algoritmo di base funziona trovando un triangolo seme (cioè tre punti dati $(\sigma_i; \sigma_j; \sigma_k)$ tali che una sfera di raggio ρ che li tocca non contenga altri punti dati) e aggiungendo un triangolo alla volta eseguendo l'operazione di ball pivoting spiegata precedentemente. La *front* F è rappresentata come una raccolta di elenchi concatenati di bordi ed è inizialmente composta da un unico anello contenente i tre bordi definiti dal primo triangolo seme. Ogni bordo $e_{(i;j)}$ del fronte, è rappresentato dai suoi due estremi $(\sigma_i; \sigma_j)$, il vertice opposto σ_o, il centro c_{ijo} della palla che tocca tutti e tre i punti e si collega al bordo precedente e a quello successivo nello stesso loop della parte anteriore. Un bordo può essere **active**, **boundary** o **frozen**. Un bordo attivo è quello che verrà utilizzato per il pivoting. Se non è possibile ruotare da un bordo, viene contrassegnato come boundary. Lo stato frozen è spiegato di seguito. Mantenere tutte queste informazioni su ciascun bordo rende più semplice ruotare la palla attorno ad esso. Il motivo per cui la parte anteriore è una raccolta di elenchi concatenati, anziché uno singolo, è che quando la pallina ruota lungo un bordo, a seconda che tocchi un punto dati appena incontrato o uno utilizzato in precedenza, la parte anteriore cambia topologia. BPA gestisce tutti i casi con due semplici operatori topologici, **join** e **glue**, che assicurano che in ogni momento il fronte sia una raccolta di liste concatenate.

L'algoritmo BPA di base è mostrato nella figura 7.12. Di seguito vengono descritte in dettaglio le funzioni e le strutture dati utilizzate. In particolare, descriveremo in seguito una semplice modifica necessaria all'algoritmo di base per supportare un'esecuzione out-of-core efficiente. Ciò consente a BPA di triangolare set di dati di grandi dimensioni con un utilizzo minimo della memoria.

Spatial queries

Sia *ball_pivot* che *find_seed_triangle* (linee 3 e 10 nella figura 7.12) richiedono una ricerca efficiente del sottoinsieme di punti contenuti in un piccolo intorno spaziale. L'implementazione di questa spatial query utilizza una griglia regolare di celle

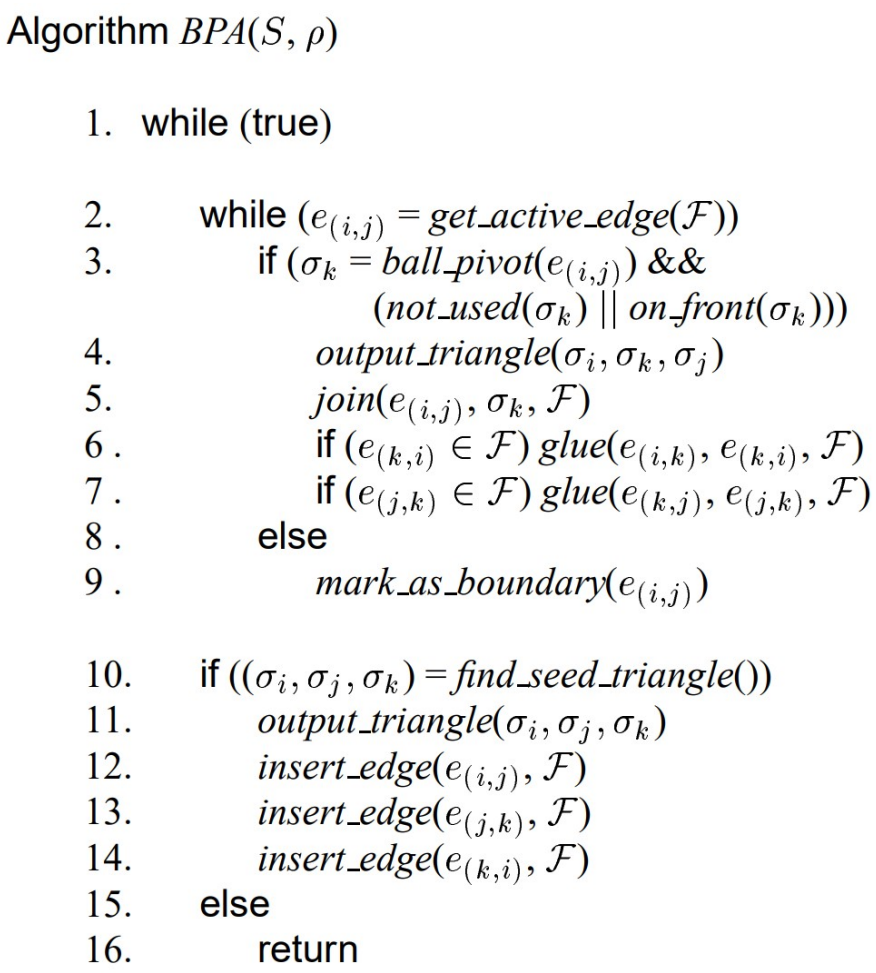


Figura 7.12: Scheletro dell'algoritmo BPA. Diversi test di errore necessari sono stati tralasciati per la leggibilità, come i controlli dell'orientamento dei bordi. I bordi nella parte anteriore F sono generalmente accessibili mantenendo una coda di bordi active. L'operazione *join* aggiunge due bordi active in primo piano. L'operazione *glue* elimina due bordi dalla parte anteriore e modifica la topologia della parte anteriore suddividendo un singolo loop in due o combinando due loop in uno. Vedere il testo per i dettagli. La funzione *find_seed_triangle* restituisce un triangolo ρ-esposto, che viene utilizzato per inizializzare la parte anteriore.

cubiche o voxel. Ogni voxel ha lati di dimensione $\delta = 2\rho$. I punti dati vengono memorizzati in un elenco e l'elenco è organizzato utilizzando l'ordinamento per bucket in modo che i punti che si trovano nello stesso voxel formino una sottolista contigua. Ogni voxel memorizza un puntatore all'inizio del suo sottoelenco di punti (al successivo elenco secondario se il voxel è vuoto). Un voxel aggiuntivo alla fine della griglia memorizza un puntatore NULL. Per visitare tutti i punti in un voxel è sufficiente percorrere la lista dal nodo puntato dal voxel a quello puntato dal voxel successivo. Dato un punto p possiamo facilmente trovare il voxel V in cui si trova dividendo le sue coordinate per ρ. Di solito abbiamo bisogno di cercare tutti i punti entro 2ρ di distanza da p, che sono un sottoinsieme di tutti i punti contenuti nei 27 voxel adiacenti a V (incluso V stesso). La griglia consente un accesso costante ai punti. Le sue dimensioni sarebbero proibitive se elaborassimo un insieme di dati di grandi dimensioni in un unico passaggio; ma un'implementazione out-of-core, descritta in una sezione successiva, può elaborare i dati in blocchi gestibili. L'utilizzo della memoria può essere ulteriormente ridotto, a scapito di un accesso più lento, utilizzando rappresentazioni più compatte, come una struttura dati a matrice sparsa.

Selezione del seme

Visto che i dati soddisfano le condizioni del teorema di ricostruzione del paragrafo precedente, un seme per componente connesso è sufficiente per ricostruire l'intera varietà (la funzione *find_seed_triangle* alla riga 10 in figura 7.12). Un modo semplice per trovare un seme valido è:

- Scegliere un punto σ non ancora utilizzato dalla triangolazione ricostruita;
- Considerare tutte le coppie di punti σ_a, σ_b nelle sue vicinanze in ordine di distanza da σ. Costruire potenziali triangoli seme $\sigma, \sigma_a, \sigma_b$;
- Verificare che la normale del triangolo sia coerente con le normali del vertice, ovvero punti verso l'esterno. Verificare che una $\rho - ball$ con il centro nel semispazio esterno tocchi tutti e tre i vertici e non contenga altri punti dati;
- Fermarsi quando è stato trovato un triangolo seme valido.

In presenza di dati rumorosi e incompleti, è importante selezionare una strategia di ricerca dei semi efficiente. Dato un seme valido, l'algoritmo costruisce il componente connesso più grande possibile contenente il seme. I punti rumorosi che si trovano a una distanza leggermente maggiore di 2 dalla triangolazione ricostruita potrebbero formare altri potenziali triangoli seme, portando alla costruzione di piccoli gruppi di triangoli che giacciono vicino alla superficie principale (vedi figura 7.11c). Questi piccoli componenti sono un artefatto del rumore presente nei dati e di solito sono indesiderati. Sebbene siano facili da eliminare filtrando i dati, una quantità significativa di risorse di calcolo viene sprecata per la loro costruzione. Possiamo tuttavia osservare quanto segue: se ci limitiamo a considerare un solo punto dati per voxel come vertice candidato per un triangolo seme, non possiamo perdere componenti che coprono un volume maggiore di pochi voxel. Inoltre, per un dato voxel, considera la media normale n di punti al suo interno. Questa normale approssima la normale alla superficie in quella regione. Dato che vogliamo che la nostra palla cammini

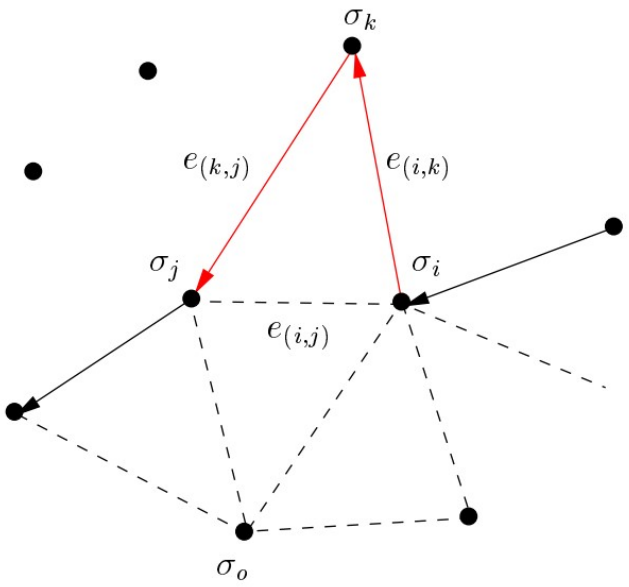


Figura 7.13: Un'operazione di *join* aggiunge semplicemente un nuovo triangolo, rimuovendo il bordo $e_{(i;j)}$ dal front e aggiungendo i due nuovi bordi $e_{(i;k)}$ ed $e_{(k;j)}$.

"sulla" superficie, è conveniente considerare prima i punti la cui proiezione su n è ampia e positiva. Pertanto, teniamo semplicemente un elenco di voxel non vuoti. Cerchiamo questi voxel per triangoli seme validi e, quando ne viene trovato uno, iniziamo a costruire una triangolazione utilizzando operazioni di pivot. Quando non è più possibile il pivot, continuiamo la ricerca di un triangolo seme dal punto in cui ci eravamo fermati, saltando tutti i voxel contenenti un punto che ora fa parte della triangolazione. Quando non si trovano più semi, l'algoritmo si ferma.

Ball pivoting

Un'operazione di pivot (linea 3 nella figura 7.12) inizia con un triangolo $\tau = (\sigma_i; \sigma_j; \sigma_o)$ e una sfera di raggio ρ che tocca i suoi tre vertici. Senza perdita di generalità, si supponga che il bordo $e_{(i;j)}$ sia il bordo di rotazione. La pallina nella sua posizione iniziale (lasciamo che c_{ijo} sia il suo centro) non contiene alcun punto dato, sia perché τ è un triangolo seme, sia perché τ è stato calcolato da una precedente operazione di pivot. La rotazione è in linea di principio un movimento continuo della palla, durante il quale la palla rimane in contatto con i due punti finali di $e_{(i;j)}$, come illustrato nella figura 7.10. A causa di questo contatto, il movimento è vincolato come segue: il centro c_{ijo} della palla descrive un cerchio che giace sul piano perpendicolare a $e_{(i;j)}$ e attraversa il suo punto medio $m = \frac{1}{2}(\sigma_j + \sigma_i)$. Il centro di questa traiettoria circolare è m, il suo raggio è $||c_{ijo} - m||$. Durante questo movimento, la palla può colpire un altro punto σ_k. Se nessun punto viene colpito, il bordo è un bordo di confine. Altrimenti, il triangolo $(\sigma_i; \sigma_j; \sigma_o)$ è un nuovo triangolo valido, e la palla nella sua posizione finale non contiene nessun altro punto, essendo quindi una palla di partenza valida per la successiva operazione di pivot. In pratica troviamo σ_k come segue. Consideriamo tutti i punti in un $2\rho - intorno$ di m. Per ciascuno di questi punti σ_x, calcoliamo il centro c_x della palla che tocca σ_i, σ_j e σ_x, se esiste. Ogni c_x

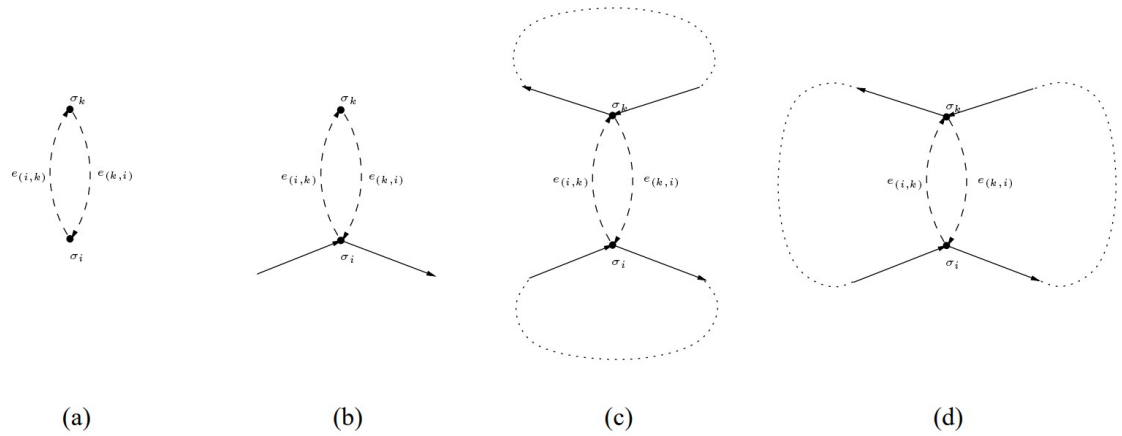


Figura 7.14: Un'operazione di glue viene applicata quando la join crea un bordo identico a un bordo esistente, ma con orientamento opposto. I due bordi coincidenti vengono rimossi e la parte anteriore regolata di conseguenza. Ci sono quattro casi possibili: (a) I due bordi formano un anello. Il loop viene cancellato dal front. (b) Due bordi appartengono allo stesso anello e sono adiacenti. I bordi vengono rimossi e il loop accorciato. (c) I bordi non sono adiacenti e appartengono allo stesso anello. Il loop è diviso in due. (d) I bordi non sono adiacenti e appartengono a due anelli diversi. I loop vengono uniti in un unico loop.

giace sulla traiettoria circolare γ attorno a m, e può essere calcolato intersecando una $\rho - sfera$ centrata in σ_x con il cerchio γ. Di questi punti c_x selezioniamo quello che si trova per primo lungo la traiettoria γ. Riportiamo il primo punto colpito e il corrispondente centro della palla. È possibile aggiungere banali test di rigetto per velocizzare la ricerca del primo hit point.

Operazioni *join* e *glue*

Queste due operazioni generano triangoli durante l'aggiunta e la rimozione di bordi dai loop anteriori (linee 5-7 nella figura 7.12). L'operazione più semplice è la *join*, che viene utilizzata quando la palla ruota attorno al bordo $e_{(i;j)}$, toccando un vertice non utilizzato σ_k (cioè σ_k è un vertice che non fa ancora parte della mesh). In questo caso, produciamo il triangolo $(\sigma_i; \sigma_k; \sigma_j)$, e modifichiamo localmente il fronte rimuovendo $e_{(i;j)}$ e aggiungendo i due bordi $e_{(i;k)}$ ed $e_{(k;j)}$ (vedi figura 7.13). Quando σ_k fa già parte della mesh, può verificarsi uno di questi due casi: σ_k è un vertice mesh interno, (cioè, nessun bordo anteriore usa σ_k). Il triangolo corrispondente non può essere generato, poiché creerebbe un vertice non-varietà. In questo caso, $e_{(i;j)}$ è semplicemente contrassegnato come un bordo boundary; σ_k appartiene al front. Dopo aver verificato l'orientamento del bordo per evitare di creare una varietà non orientabile, applichiamo un'operazione di join e generiamo il nuovo triangolo mesh $(\sigma_i; \sigma_k; \sigma_j)$. Il join potrebbe potenzialmente creare (una o due) coppie di bordi coincidenti (con orientamento opposto), che vengono rimossi dall'operazione glue.
L'operazione *glue* rimuove dal front coppie di bordi coincidenti, con orientamento opposto (i bordi coincidenti con lo stesso orientamento non vengono mai creati dall'algoritmo). Ad esempio, quando il bordo $e_{(i;k)}$ viene aggiunto al front da un'operazione di join (lo stesso vale per $e_{(k;j)}$), se il bordo $e_{(k;i)}$ è sul front, la glue rimuoverà

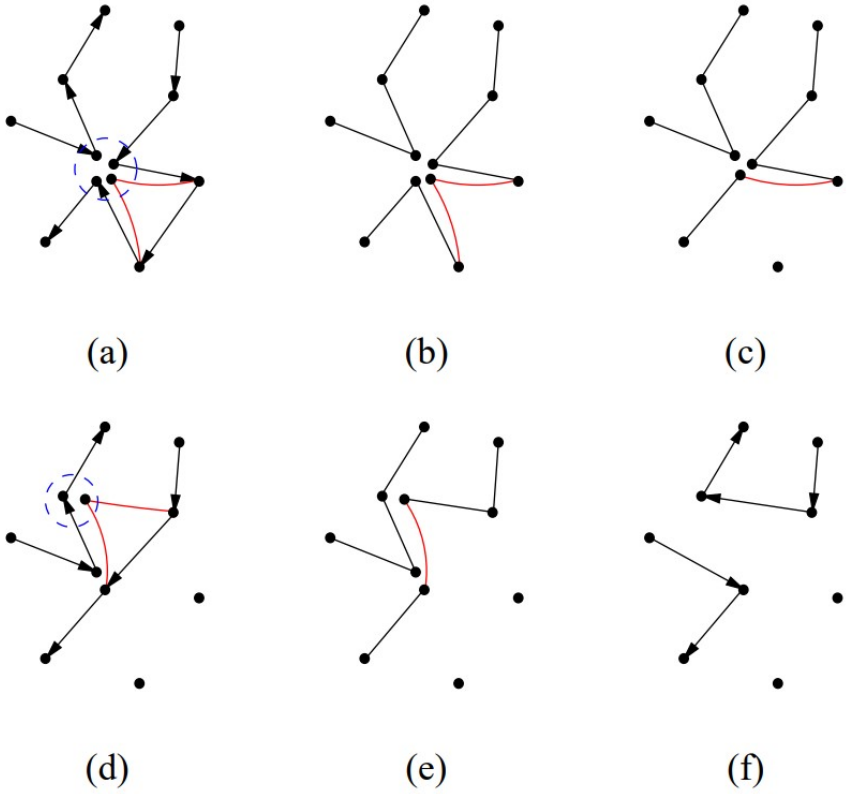


Figura 7.15: Esempio di una sequenza di operazioni di join e glue. (a) Un nuovo triangolo deve essere aggiunto al fronte esistente. I quattro vertici anteriori all'interno del cerchio tratteggiato rappresentano tutti un singolo punto dato. (b) Una join rimuove un bordo e crea due nuovi bordi anteriori, coincidenti con quelli esistenti. (c), (d) Due operazioni di glue rimuovono le coppie di bordi coincidenti. (d) mostra anche il triangolo successivo aggiunto. (e) Solo uno dei bordi creati da questa join è coincidente con un bordo esistente. (f) Una glue rimuove il paio duplicato.

la coppia di bordi $e_{(i;k)}$, $e_{(k;i)}$ e regolerà il front di conseguenza. Sono possibili quattro casi, come illustrato in figura 7.14. Una sequenza di operazioni di join e glue è illustrata in figura 7.15.

Estensione out-of-core

La possibilità di utilizzare un PC per triangolare le scansioni ad alta risoluzione consente un'elaborazione dei dati in loco a basso costo. A causa della loro località di riferimento, gli algoritmi avanzati sono adatti a estensioni out-of-core molto semplici. E' stato utilizzato un semplice schema di suddivisione dei dati per estendere l'algoritmo mostrato nella figura 7.12. L'idea di base è quella di memorizzare nella cache la parte del set di dati attualmente utilizzata per il pivot, di eseguire il dump dei dati non più utilizzati e di caricare i dati quando necessario. Vengono utilizzati due piani π_0 e π_1 allineati all'asse per definire la regione di lavoro attiva per la rotazione. Inizialmente posizioniamo π_0 in modo tale che nessun punto dati si trovi "sotto" e π_1 sopra π_0 a una certa distanza specificata dall'utente. Quando ogni bordo viene creato, verifichiamo se i suoi punti finali sono "sopra" π_1; in questo caso,

contrassegniamo il bordo come frozen. Quando tutti i bordi rimanenti nella coda sono frozen, spostiamo semplicemente π_0 e π_1 "verso l'alto" e aggiorniamo tutti i bordi frozen in bordi attivi, e così via. Un sottoinsieme di punti dati viene caricato e scartato dalla memoria quando il riquadro di delimitazione corrispondente entra ed esce dalla sezione attiva. Le scansioni possono essere facilmente pre elaborate per suddividerle in mesh più piccole, in modo che si estendano solo su poche sezioni e il carico di memoria rimanga basso. L'unico cambiamento richiesto nell'algoritmo per implementare questo perfezionamento è un loop esterno per spostare la sezione attiva e l'aggiunta delle istruzioni per sbloccare i bordi tra le righe 1–2 della figura 7.12.

Passaggi multipli

Per gestire superfici campionate in modo non uniforme, possiamo facilmente estendere l'algoritmo per eseguire più passaggi con raggi di palla crescenti. L'utente specifica un elenco di raggi $\{\rho_0, \ldots, \rho_n\}$ come parametri di input. In ogni fetta, per aumentare $\rho_i, i = 0, \ldots, n$, iniziamo inserendo i punti in una griglia di dimensione voxel $\delta = 2\rho_i$. Lasciamo funzionare il BPA fino a quando non ci sono più bordi active nella coda. A questo punto incrementiamo i, passiamo attraverso tutti i bordi anteriori e controlliamo se ogni bordo con il suo vertice opposto σ_o forma un triangolo seme valido per una sfera di raggio ρ_i. Se lo è, viene aggiunto alla coda dei bordi active. Infine, la rotazione viene riavviata.

Osservazioni

L'algoritmo BPA è stato implementato in C++ utilizzando la libreria di modelli standard. L'intero codice è inferiore a 4000 righe, comprese le estensioni out-of-core. L'algoritmo è lineare nel numero di punti dati e utilizza l'archiviazione lineare, assumendo che la densità dei dati sia limitata. Questa ipotesi è appropriata per i dati scansionati, che vengono raccolti da apparecchiature con una distanza del campione nota. Anche se più scansioni si sovrappongono, il numero totale di punti in qualsiasi regione sarà delimitato da una costante nota. La maggior parte dei passaggi sono semplici controlli $O(1)$ dello stato o aggiornamenti a code, elenchi collegati e simili. Con la densità limitata, un punto deve essere correlato solo a un numero costante di vicini. Quindi, ad esempio, un punto può essere contenuto solo in un numero costante di loop nel fronte di avanzamento. Le due operazioni ball_pivot e find_seed_triangle sono più complesse. Ogni perno della sfera opera su un diverso bordo della maglia, quindi il numero di perni è $O(n)$. Un singolo perno richiede l'identificazione di tutti i punti in un $2\rho - intorno$. Un elenco di questi punti può essere raccolto da 27 voxel che circondano il punto candidato nella nostra griglia. Con densità limitata, questa lista ha dimensione costante B. Eseguiamo alcuni calcoli algebrici su ogni punto della lista e selezioniamo il risultato minimo, tutte operazioni $O(1)$ su una lista di dimensione $O(1)$. Ogni find_seed_triangle seleziona i punti inutilizzati uno alla volta e verifica se un triangolo incidente è un seme valido. Nessun punto viene considerato più di una volta, quindi questo test viene eseguito solo $O(n)$ volte. Per testare un punto candidato, raccogliamo lo stesso elenco di punti discusso sopra e consideriamo coppie di punti fino a quando non troviamo un triangolo seme o rifiutiamo il candidato. Testare uno di questi triangoli può richie-

dere la classificazione di ogni punto vicino rispetto a una sfera che tocca i tre vertici, nel caso peggiore, $O(B^3) = O(1)$ passi. In pratica, limitiamo il numero di punti e triangoli candidati testati dall'euristica discussa nella sezione precedente. Un'implementazione in-core del BPA utilizza $O(n + L)$ di memoria, dove L è il numero di celle nella griglia voxel. Il termine $O(n)$ include i dati, il fronte di avanzamento (che può includere una sola volta ciascun bordo della mesh) e la coda del bordo candidato. La implementazione out-of-core utilizza $O(m + L')$ di memoria, dove m è il numero di punti dati nella sezione più grande e L' è la dimensione della griglia più piccola che copre una singola sezione.
Poiché l'utente può controllare la dimensione delle slice, i requisiti di memoria possono essere adattati all'hardware disponibile. La griglia voxel può essere rappresentata in modo più compatto come una matrice sparsa, con un piccolo (costante) aumento del tempo di accesso.

7.6.3 Procedimento per la visualizzazione della mesh

Nella sezione 7.5.1 abbiamo visto come calcolare la point cloud e visualizzarla con anche le norme alla superficie nei punti 3D. Il passaggio finale quindi è quello di costruire la triangle mesh partendo dalla point cloud ottenuta e sfruttando, come detto precedentemente, l'algoritmo BPA spiegato dettagliatamente nella sezione precedente.
Open3D supporta già internamente l'algoritmo BPA e ci da a disposizione la funzione TriangleMesh.create_from_point_cloud_ball_pivoting() spiegata nella sezione 7.6. L'algoritmo però, come ampiamente spiegato nella sezione 7.6.2, ha bisogno di conoscere (oltre alla point cloud) il raggio delle sfere che utilizzerà per ricostruire la superficie, quindi come calcolarlo? Il raggio, è ottenuto empiricamente in base alle dimensioni e alla scala della point cloud di input. In teoria, il diametro della palla dovrebbe essere leggermente più grande della distanza media tra i punti.
Open3D da a disposizione una funzione chiamata
PointCloud.compute_nearest_neighbor_distance() la quale calcola la distanza da un punto al vicino più vicino nella point cloud. Fatto ciò basterà fare la media delle distanze ottenute e calcolare il raggio come r = media delle distanze×fattore moltiplicativo. Il fattore moltiplicativo è utile per far sì che il diametro rispetti la "regola" citata sopra, quindi sarà un intero maggiore di 1, nel nostro caso 3.

Dopo aver creato la mesh l'ultimo passaggio prima della visualizzazione è quello di eliminare eventuali outliers e fare una sorta di "pulizia", ciò avviene in 4 passaggi:

- **simplify_quadric_decimation(*target_number_of_triangles*):** funzione per semplificare la mesh utilizzando Quadric Error Metric Decimation di Garland e Heckbert [35], nel nostro caso il *target_number_of_triangles* (il numero di triangoli che dovrebbe avere la mesh semplificata. Non è garantito che questo numero venga raggiunto) è stato posto a 100000;

- **remove_duplicated_triangles():** funzione che rimuove i triangoli duplicati, ovvero rimuove i triangoli che fanno riferimento agli stessi tre vertici, indipendentemente dal loro ordine;

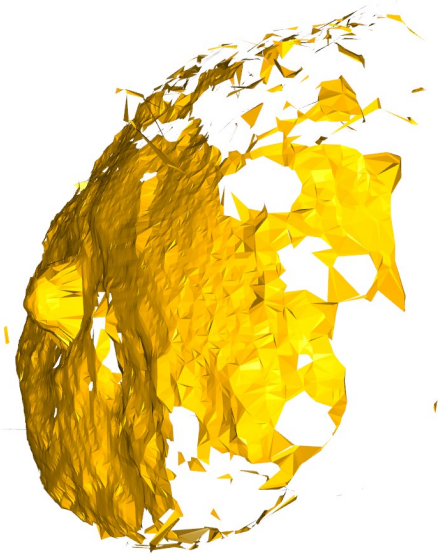
- **remove_degenerate_triangles():** funzione che rimuove triangoli degeneri, ovvero triangoli che fanno riferimento a un singolo vertice più volte in

un singolo triangolo. Di solito sono il prodotto della rimozione dei vertici duplicati;

- **remove_duplicated_vertices():** funzione che rimuove vertici duplicati, ovvero vertici che hanno coordinate identiche;
- **remove_non_manifold_edges():** funzione che rimuove tutti i bordi non manifold, eliminando successivamente i triangoli con la superficie più piccola adiacente al bordo non manifold fino a quando il numero di triangoli adiacenti al bordo è ≤ 2.

Finalmente in figura 7.16 vediamo il risultato finale del calcolo della mesh del prosciutto.

(a) Vista da sotto.



(b) vista di profilo.

(c) Vista di fronte.

Figura 7.16: Mesh 3D del prosciutto creata tramite l'algoritmo BPA.

Capitolo 8

Risultati e performance sperimentali

Durante la progettazione e l'implementazione di questo progetto siamo passati per algoritmi decisamente complicati e dispendiosi in termini di risorse, in questo capitolo vedremo come l'hardware è stato sfruttato e come i miglioramenti applicati ad alcuni algoritmi o operazioni hanno giovato sulle performance.

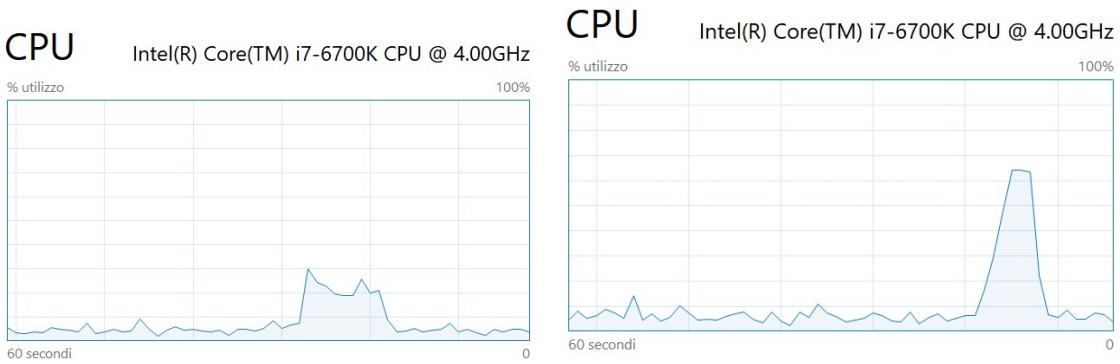
8.1 Test effettuati

Ho deciso di effettuare dei test riguardo le performance delle risorse durante le 3 fasi del progetto:

1. Costruzione della mappa di disparità tramite software proprietario Specialvideo, con alcune modifiche;
2. Segmentazione tramite rete neurale U-net (in questo caso ho testato le performance sulla macchina virtuale offerta da Google Colab);
3. Ricostruzione della scena 3D (point cloud e mesh) tramite script python interamente implementato da me.

8.1.1 Costruzione mappa di disparità

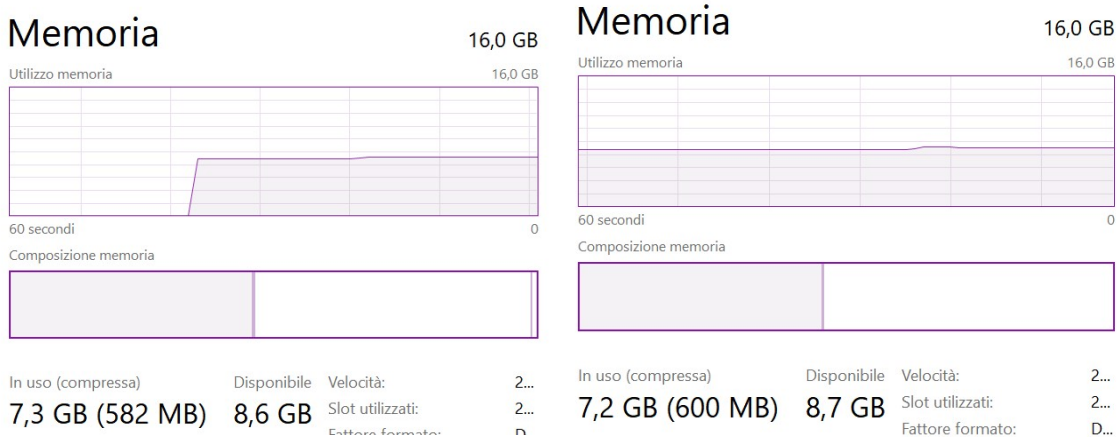
La costruzione della mappa di disparità è spiegata dettagliatamente nel capitolo 4, ma non è stato spiegato lato software quali sono le performance.
Ho utilizzato un software proprietario di Specialvideo, scritto in python, che sfrutta la libreria OpenCV (algoritmo SGM) per il calcolo della mappa di disparità.
Nelle figure 8.1, 8.2 e 8.3 analizziamo quali sono le differenze a livello di performance prima e dopo le modifiche che ho applicato per ottenere la mappa di disparità finale utilizzata per il progetto. Le modifiche di cui parlo trattano semplicemente il tuning dei parametri utilizzati dall'algoritmo SGM e qualche miglioria a livello di codice per eliminare più outliers possibili dal risultato finale. Ci tengo a precisare le specifiche hardware del computer utilizzato per questi test: **CPU** Intel i7-6700K a 4.00 GHz, **RAM** 16gb a 2600 MHz e **GPU** Nvidia GTX 1080 a 1.6 GHz e 8Gb di memoria dedicata.



(a) Originale.

(b) Dopo le modifiche.

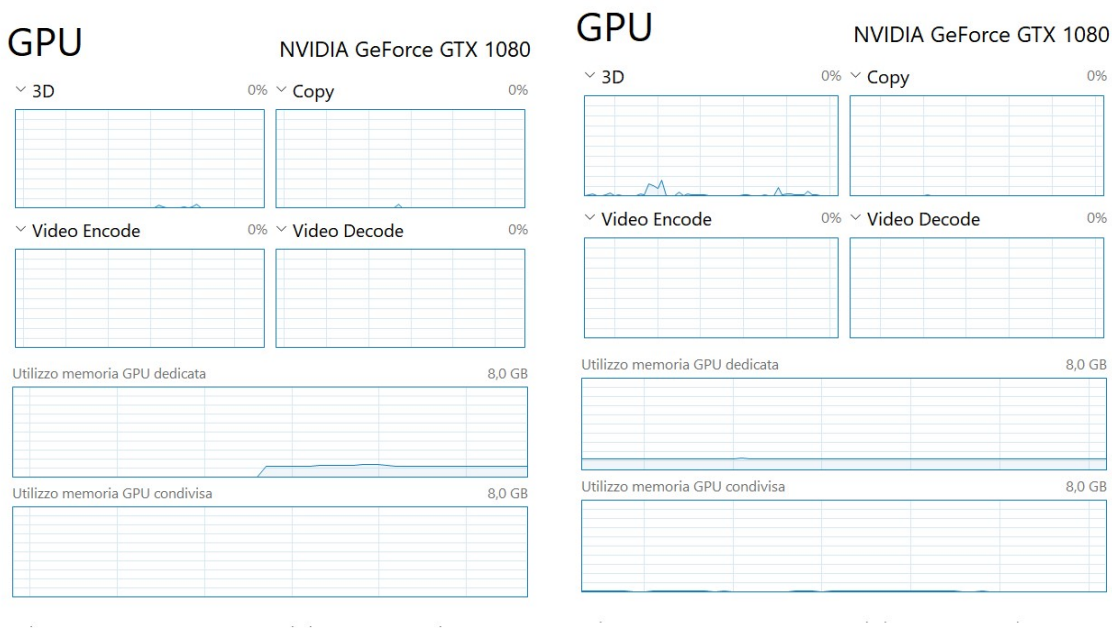
Figura 8.1: Comparazione prestazioni CPU prima e dopo le modifiche, notiamo un netto aumento nell'uso della CPU al momento della creazione dell'immagine di disparità, e una durata di runtime abbastanza simile. Questo è dovuto ovviamente alle modifiche apportate ai parametri dell'algoritmo SGM e ad alcuni miglioramenti a livello di visualizzazione del risultato.



(a) Originale.

(b) Dopo le modifiche.

Figura 8.2: Comparazione prestazioni memoria RAM prima e dopo le modifiche, in questo caso non ci sono grandi differenze, nell'immagine (b) notiamo solo un "gradino" più marcato nel momento dell'avvio del programma, ma l'occupazione di memoria totale è pressoché identica.

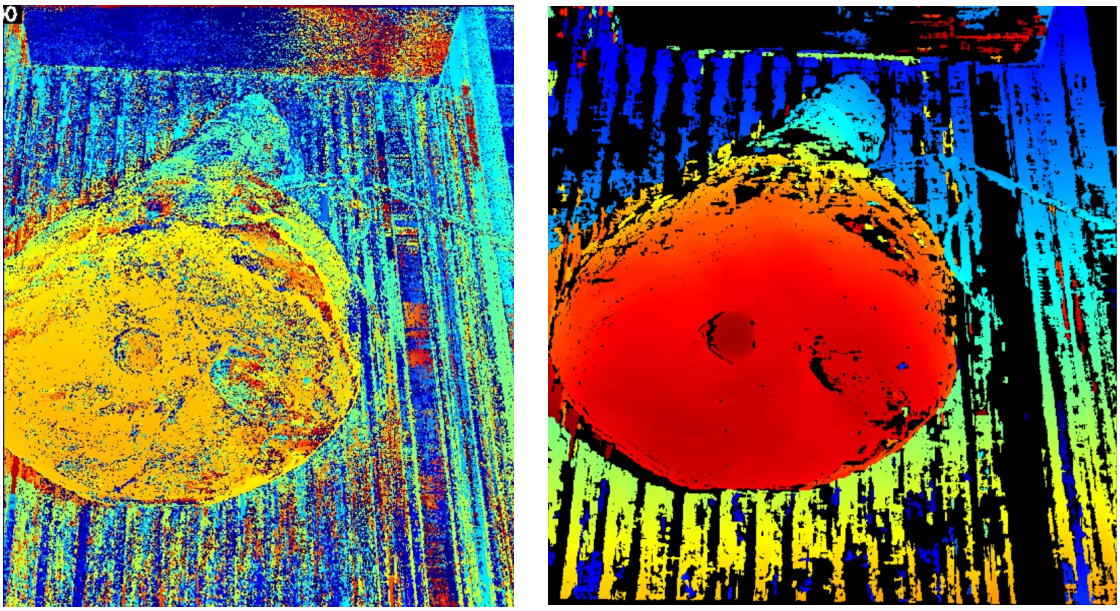


(a) Originale.

(b) Dopo le modifiche.

Figura 8.3: Comparazione prestazioni GPU prima e dopo le modifiche, la scheda video non viene praticamente utilizzata per il calcolo stereo. Possiamo notare un insignificante aumento dell'utilizzo delle funzionalità 3D nell'immagine (b) probabilmente dovuto alla visualizzazione del risultato finale.

In figura 8.4 vediamo come è cambiata l'immagine di disparità prima e dopo le modifiche di cui stiamo parlando. Il notevole miglioramento del risultato finale giustifica al 100% il maggior (seppur leggero) carico sulle risorse hardware.



(a) Originale. **(b)** Dopo le modifiche.

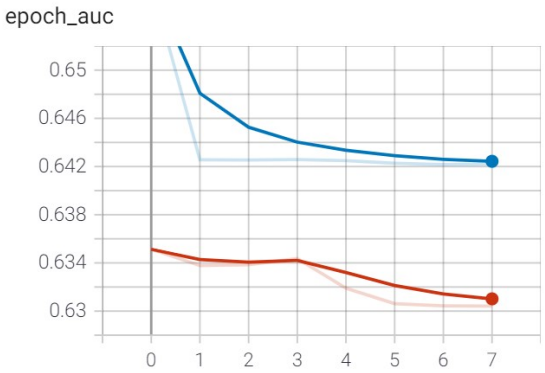
Figura 8.4: Comparazione tra mappa di disparità calcolata con parametri standard e senza miglioramenti riguardo gli outliers (a) e mappa di disparità finale utilizzata per il progetto (b).

8.1.2 Segmentazione tramite rete neurale U-Net

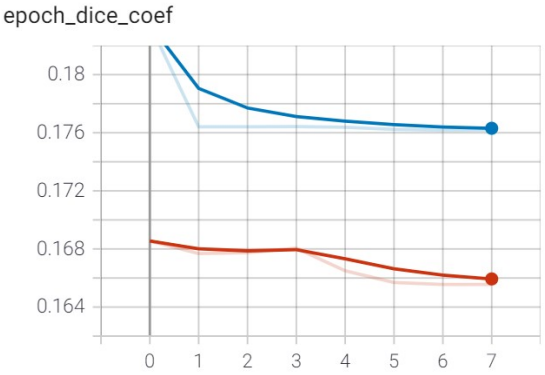
Per questa seconda parte del progetto non mi è stato possibile utilizzare il computer personale in quanto il training di una rete neurale richiede un quantitativo di risorse (soprattutto a livello di GPU) molto alto, per questo motivo, come detto nel capitolo 6, sono ricorso all'utilizzo della piattaforma Google Colab, tramite la quale ho potuto eseguire il mio codice su una macchina con specifiche di alto livello: **CPU** non nota, **GPU** che varia in base alla disponibilità della piattaforma al momento del collegamento tra Nvidia K80, T4, P4 e P100, con specifiche che vanno dai 24 Gb ai 16 Gb di memoria dedicata; e **RAM** di 13 Gb circa [36].
In figura 8.5 vediamo, durante un training, quanto la RAM e la GPU messa a disposizione dalla piattaforma viene sfruttata. La base di partenza per quanto riguarda la RAM, con la macchina in idle, è un occupazione di 0.82 Gb.
L'altro aspetto molto significante nell'uso di Colab è il tempo che la macchina ci mette per eseguire un singola epoca, si capisce bene che dovendo svolgere un numero molto alto di epoche è un parametro che influisce parecchio sulle performance. Nel nostro caso per ogni epoca vengono impiegati in media **84 s/epoca**, un tempo molto buono nonostante la risoluzione molto alta delle immagini del dataset. In figura 8.6 invece vediamo uno screenshot della tensorboard dove si possono notare gli andamenti delle 4 metriche utilizzate durante il training. In figura 6.16 possiamo vedere un esempio di segmentazione risultante dalla rete neurale.

Ultima esecuzione	RAM utilizzata	GPU utilizzata
0 minuti fa	3.85 GB	14.28 GB

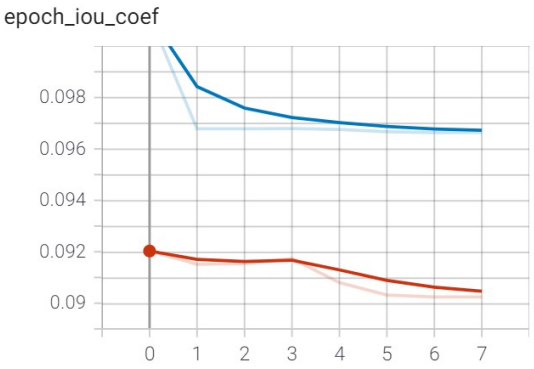
Figura 8.5: GPU e RAM utilizzate durante un sample training di 10 epoche.



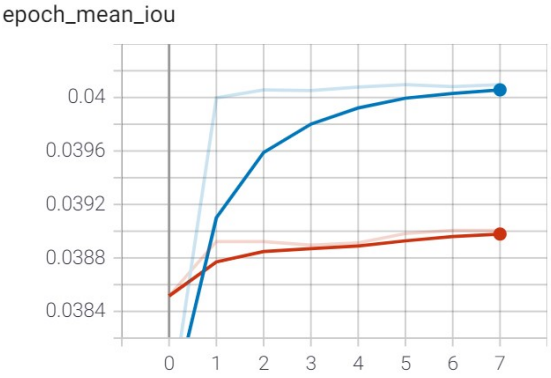
(a) Andamento metrica AUC.



(b) Andamento metrica coefficiente di Dice.



(c) Andamento metrica coefficiente di IOU.



(d) Andamento metrica meanIOU.

Figura 8.6: Andamenti delle metriche utilizzate durante un sample training di 10 epoche, sulle ordinate troviamo il valore delle metriche e sulle ascisse le epoche.

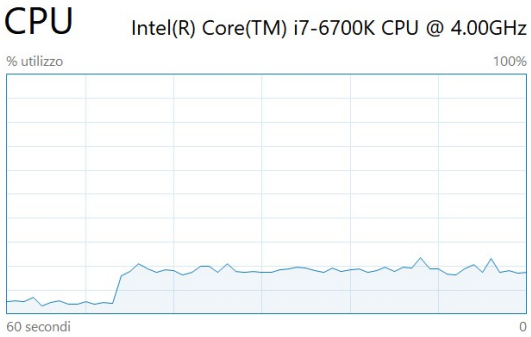
8.1.3 Ricostruzione della scena 3D

Quest'ultima parte, spiegata dettagliatamente nel capitolo 7, consiste nella ricostruzione della scena 3D partendo dalle immagini originali, tramite la creazione di point cloud e mesh 3D.
In questo caso ho utilizzato uno script python interamente implementato da me e integrato in un secondo momento con lo script riguardante la creazione dell'immagine di disparità. Analizziamo le performance delle risorse hardware durante il runtime di questo script, ricapitoliamo le specifiche: **CPU** Intel i7-6700K a 4.00 GHz, **RAM** 16 gb a 2600 MHz e **GPU** Nvidia GTX 1080 a 1.6 GHz e 8 Gb di memoria dedicata. Per il calcolo e la visualizzazione 3D della point cloud finale vista in figura 7.4 lo script impiega circa **100 secondi**, l'utilizzo delle risorse in questo periodo di tempo lo vediamo in figura 8.7.

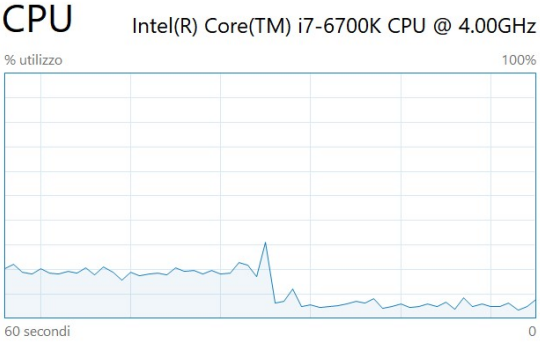
L'ultimo test sperimentale effettuato è quello riguardo il calcolo e la visualizzazione della mesh 3D. Il calcolo avviene dopo aver già calcolato la point cloud, questo giustifica lo scarso utilizzo delle risorse hardware e del ridotto tempo di runtime. Infatti il tempo totale per il calcolo e la visualizzazione della mesh si aggira intorno ai **15 secondi**.
Vediamo un riassunto delle performance per questo passaggio in figura 8.8.

8.1.4 Osservazioni

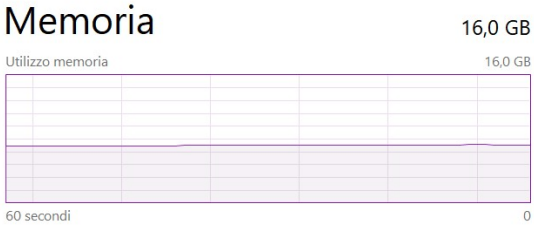
Per quanto riguarda tutti i test effettuati sull'utilizzo delle risorse hardware e sulle performance mi ritengo più che soddisfatto perchè, come abbiamo visto, sia le tempistiche di runtime sia l'utilizzo e l'usura delle risorse sono trascurabili.
Aggiungo che ovviamente i risultati dei test sono stati molto buoni anche per "merito" della macchina su cui sono stati eseguiti, infatti le specifiche hardware del computer che avevo a disposizione sono sopra la media.



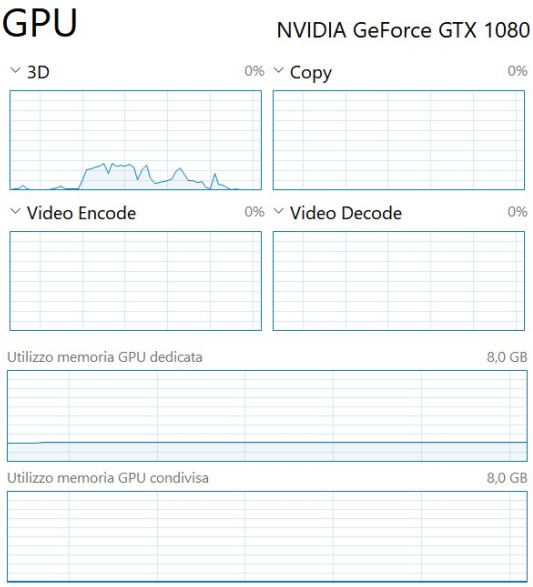
(a) Inizio del calcolo della point cloud.



(b) Visualizzazione e fine del processo.

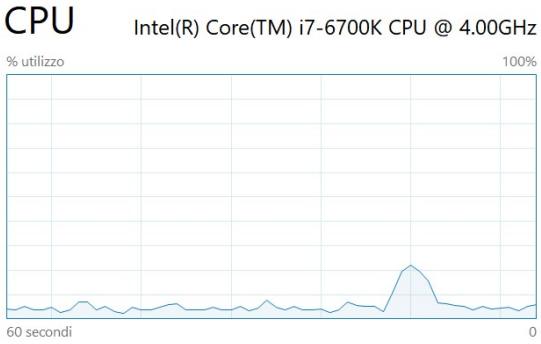


(c) Utilizzo RAM durante il processo.

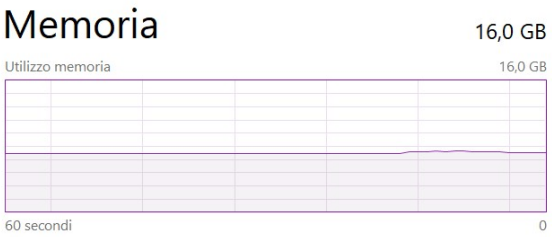


(d) Utilizzo GPU durante il processo e durante la visualizzazione 3D.

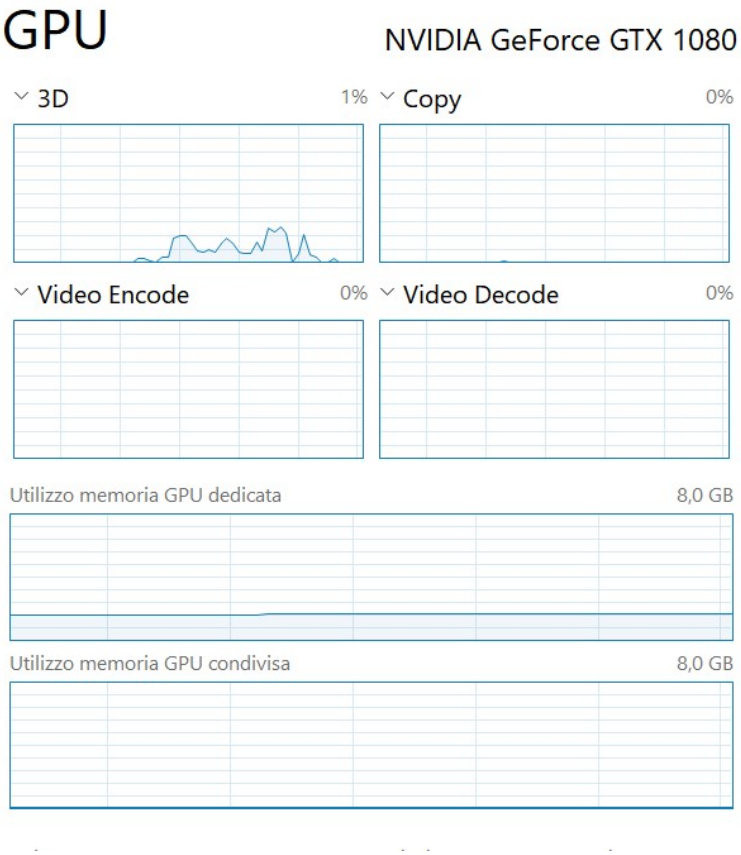
Figura 8.7: Analisi delle performance durante il calcolo e visualizzazione della point cloud, notiamo in (a) e (b) che l'utilizzo della CPU ha un incremento di circa un 15% all'inizio del processo, per poi rimanere costante durante tutto il processo. Per quanto riguarda l'utilizzo della RAM (c) non abbiamo grosse variazioni, se non 2 piccoli e quasi insignificanti aumenti uno durante il processo e uno al momento della visualizzazione della point cloud. Per la GPU le cose cambiano, l'utilizzo rimane rasente lo 0% per tutta la durata del processo, invece si nota un sostanziale aumento (d) quando durante la fase di visualizzazione si vanno ad applicare delle modifiche nell'editor 3D, come ad esempio la rotazione e lo scaling della point cloud.



(a) Utilizzo CPU durante calcolo e visualizzazione mesh.



(b) Utilizzo RAM durante calcolo e visualizzazione mesh.



(c) Utilizzo GPU durante modifiche alla visualizzazione nell'editor 3D.

Figura 8.8: Analisi delle performance durante il calcolo e visualizzazione della mesh, notiamo in (a) che l'utilizzo della CPU ha un incremento di circa un 20% durante tutto il processo. Riguardo l'utilizzo della RAM durante il processo (b) non abbiamo grosse variazioni, se non un piccolo e quasi insignificante aumento. Per la GPU le cose cambiano, l'utilizzo rimane rasente lo 0% per tutta la durata del processo, invece si nota un sostanziale aumento (d) quando durante la fase di visualizzazione si vanno ad applicare delle modifiche nell'editor 3D, come ad esempio la rotazione e lo scaling della point cloud.

Capitolo 9

Conclusioni e sviluppi futuri

Il progetto nasce da un bisogno legato ad un azienda produttrice di prosciutti, la quale aveva necessità di automatizzare la procedura di "sugnatura" tramite l'utilizzo di un robot. Come abbiamo visto il progetto si è diviso in 3 fasi: calibrazione del sistema stereo e creazione della mappa di disparità, segmentazione delle immagini acquisite tramite rete neurale e ricostruzione finale della scena 3D, eseguite tutte e 3 con ottimi risultati.
In futuro, tutto il lavoro svolto verrà utilizzato per effettivamente addestrare il robot automatico all'operazione di "sugnatura", processo che al momento non è stato possibile per ragionevoli problemi di tempo. L'aspetto di questo lavoro che mi è piaciuto di più e che mi permette di essere felice nel presentarlo è proprio il fatto che tutto ciò che svolgevo aveva uno scopo e un'applicazione reale, non solo a fini di ricerca o sperimentali.
Questo progetto per tesi si inserisce nell'ambito delle applicazioni industriali della visione artificiale e della stretta collaborazione tra computer vision e deep learning, la quale negli ultimi anni ha avuto un netto aumento. Questo progetto è proprio la dimostrazione che questo connubio di tecnologie porta a importanti passi in avanti e che probabilmente da questo momento subirà un processo di crescita importante e inarrestabile.
Specialvideo s.r.l. negli anni ha saputo adattarsi ai cambiamenti e all'evolversi di questa tecnologia, sempre più rapidamente. Per questo motivo sono entusiasta di aver avuto un'esperienza così importante in una realtà molto interessante del territorio italiano. Ringrazio il team interno che mi ha supportato durante tutto il progetto, in particolare nella persona del mio "tutor" didattico il dottor Casadio Giuseppe, per avermi lasciato libertà sia decisionale che implementativa.

Concludendo, i risultati ottenuti e trattati nei capitoli 4, 6 e 7 rendono giustizia alla qualità del progetto e all'impegno profuso, per questo mi ritengo estremamente soddisfatto, in particolare anche per quanto riguarda lo studio e i test effettuati sugli script implementati da me e sulle varie fasi sperimentali del progetto (capitolo 8). Gli obiettivi che ci eravamo prefissati inizialmente sono stati tutti rispettati e in qualche occasione anche in maniera inaspettatamente superati.
Infine, nonostante sia alla mia prima esperienza in questo campo, ci tengo a dire che le tecnologie sfruttate in questo lavoro di tesi, di cui sono grande sostenitore, possono sembrare inutili per certi aspetti o addirittura pericolose per chi non ne conosce i limiti e le potenzialità. Io credo che in questo momento storico molto complicato

il supporto a ricercatori e addetti ai lavori sia fondamentale perchè l'evoluzione tecnologica non subisca rallentamenti, in modo che sempre più persone e aziende possano giovare di questi sviluppi incredibili della tecnologia.

Bibliografia

[1] Joel Janai, Fatma Güney, Aseem Behl, Andreas Geiger, et al. Computer vision for autonomous vehicles: Problems, datasets and state of the art. *Foundations and Trends® in Computer Graphics and Vision*, 12(1–3):1–308, 2020.

[2] USM Journalists. Computer vision applications in different industries, 2020. Available on line.

[3] A. P. Jana, A. Biswas, and Mohana. Yolo based detection and classification of objects in video records. In *2018 3rd IEEE International Conference on Recent Trends in Electronics, Information Communication Technology (RTEICT)*, pages 2448–2452, 2018.

[4] Robert J Schalkoff. *Digital image processing and computer vision*, volume 286. Wiley New York, 1989.

[5] Specialvideo s.r.l. Specialvideo: sistemi di visione industriali, 2019. Available on line.

[6] OpenCV documentation. Camera calibration, 2021. Available on line.

[7] Luigi Di Stefano. Principi ed applicazioni della visione stereo. *Computer Vision Laboratory (CVLAB)-University of Bologna.*

[8] Daniel Scharstein and Richard Szeliski. A taxonomy and evaluation of dense two-frame stereo correspondence algorithms. *International journal of computer vision*, 47(1):7–42, 2002.

[9] Heiko Hirschmuller. Stereo processing by semiglobal matching and mutual information. *IEEE Transactions on pattern analysis and machine intelligence*, 30(2):328–341, 2007.

[10] OpenCV documentation. Stereosgbm, 2021. Available on line.

[11] Stan Birchfield and Carlo Tomasi. A pixel dissimilarity measure that is insensitive to image sampling. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 20(4):401–406, 1998.

[12] Junhwan Kim et al. Visual correspondence using energy minimization and mutual information. In *Proceedings Ninth IEEE International Conference on Computer Vision*, pages 1033–1040. IEEE, 2003.

[13] Stan Birchfield and Carlo Tomasi. Depth discontinuities by pixel-to-pixel stereo. *International Journal of Computer Vision*, 35(3):269–293, 1999.

[14] Dorin Comaniciu and Peter Meer. Mean shift: A robust approach toward feature space analysis. *IEEE Transactions on pattern analysis and machine intelligence*, 24(5):603–619, 2002.

[15] Kurt Konolige. Small vision systems: Hardware and implementation. In *Robotics research*, pages 203–212. Springer, 1998.

[16] Crescenzio Gallo and Centro di Ricerca Interdipartimentale Bioagromed. Reti neurali artificiali: Teoria ed applicazioni finanziarie. *Dipartimento di Scienze Economiche, Matematiche e Statistiche, Universita'di Foggia, Quaderni DSEMS (gen. 2007)*, 2007.

[17] Haoning Lin, Zhenwei Shi, and Zhengxia Zou. Maritime Semantic Labeling of Optical Remote Sensing Images with Multi-Scale Fully Convolutional Network. *Remote Sensing*, 9(5):480, May 2017.

[18] Andrinandrasana David Rasamoelina, Fouzia Adjailia, and Peter Sincak. A Review of Activation Function for Artificial Neural Network. In *2020 IEEE 18th World Symposium on Applied Machine Intelligence and Informatics (SAMI)*, pages 281–286, Herlany, Slovakia, January 2020. IEEE.

[19] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification. *arXiv:1502.01852 [cs]*, February 2015. arXiv: 1502.01852.

[20] Ludovic Trottier, Philippe Giguere, and Brahim Chaib-draa. Parametric Exponential Linear Unit for Deep Convolutional Neural Networks. In *2017 16th IEEE International Conference on Machine Learning and Applications (ICMLA)*, pages 207–214, Cancun, December 2017. IEEE.

[21] Abien Fred M Agarap. Deep Learning using Rectified Linear Units (ReLU). page 7, 2019.

[22] Charu C Aggarwal et al. Neural networks and deep learning. *Springer*, 10:978–3, 2018.

[23] D. P. Kingma and L. J. Ba. Adam: A Method for Stochastic Optimization. 2015. Publisher: Ithaca, NYarXiv.org.

[24] Sashank J Reddi, Satyen Kale, and Sanjiv Kumar. ON THE CONVERGENCE OF ADAM AND BEYOND. page 23, 2018.

[25] Timothy Dozat. INCORPORATING NESTEROV MOMENTUM INTO ADAM. page 4, 2016.

[26] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. In Nassir Navab, Joachim Hornegger, William M. Wells, and Alejandro F. Frangi, editors, *Medical Image Computing and Computer-Assisted Intervention – MICCAI 2015*, pages 234–241, Cham, 2015. Springer International Publishing.

[27] Jonathan Long, Evan Shelhamer, and Trevor Darrell. Fully convolutional networks for semantic segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 3431–3440, 2015.

[28] Numpy documentation. Numpy documentation, 2021. Available on line.

[29] OpenCV documentation. Opencv reprojectimageto3d, 2021. Available on line.

[30] Qian-Yi Zhou, Jaesik Park, and Vladlen Koltun. Open3D: A modern library for 3D data processing. *arXiv:1801.09847*, 2018.

[31] Mario Botsch, Leif Kobbelt, Mark Pauly, Pierre Alliez, and Bruno Lévy. *Polygon mesh processing*. CRC press, 2010.

[32] Fausto Bernardini, Joshua Mittleman, Holly Rushmeier, Claudio Silva, and Gabriel Taubin. The ball-pivoting algorithm for surface reconstruction. *IEEE transactions on visualization and computer graphics*, 5(4):349–359, 1999.

[33] Julie Digne. An analysis and implementation of a parallel ball pivoting algorithm. *Image Processing On Line*, 4:149–168, 2014.

[34] Fausto Bernardini and Chandrajit L Bajaj. Sampling and reconstructing manifolds using alpha-shapes. 1997.

[35] Open3D documentation. Open3d trianglemesh, 2021. Available on line.

[36] Google Colab documentation. Google colab faq, 2021. Available on line.