ALMA MATER STUDIORUM – UNIVERSITÀ DI BOLOGNA

SCUOLA DI INGEGNERIA E ARCHITETTURA

Dipartimento di Informatica – Scienza e Ingegneria (DISI)

Corso di laurea in Ingegneria Elettronica

TESI DI LAUREA

in

Logiche Riconfigurabili M

SEGMENTAZIONE E CLASSIFICAZIONE DI OGGETTI MEDIANTE IMMAGINI DEPTH E DEEP LEARNING

CANDIDATO

Filippo Cantucci

RELATORE:

Prof. Stefano Mattoccia

CORRELATORE

Dott. Matteo Poggi

Anno Accademico 2013/2014

Sessione III

Indice

Capitolo 1

Introduzione

In questo capitolo è descritto il contesto applicativo nel quale si inserisce questo lavoro di tesi, gli obiettivi e le motivazioni che hanno portato alle scelte descritte in questo documento.

1.1 Il contesto

L'*Object Categorization* è quel ramo della Visione Artificiale che ha l'obiettivo principale di identificare un oggetto specifico all'interno di una collezione di immagini o video.
A differenza dell'occhio umano, che riconosce e interpreta un oggetto appartenente a un'immagine in maniera immediata e indipendente dalle condizioni di osservazione (e.g. differente angolazione, parziale occlusione dell'oggetto, formato dell'oggetto), il calcolatore non riesce a determinare in maniera così semplice tali informazioni, ma necessita di processi più complessi per raggiungere un fine che per noi umani è naturale.
La descrizione mediante algoritmi dei compiti necessari ad un calcolatore per raggiungere questo obiettivo è complessa. Un problema di Object Categorization può essere pertanto definito come un problema di *classificazione* basata su modelli di oggetti già conosciuti.
Data una immagine che contiene uno o più oggetti e dato un set di *labels*, etichette, indotte da modelli già conosciuti, il sistema mira ad assegnare tali etichette alle regioni di interesse presenti nell'immagine, identificando così l'oggetto o gli oggetti da riconoscere. Per fare questo occorre mettere il calcolatore nelle condizioni di poter essere istruito al riconoscimento delle immagini.

1.2 Obiettivi

Il lavoro svolto in questa tesi di laurea ha come obiettivo quello di classificare immagini, identificando a quale *categoria*, o *classe*, l'immagine appartiene, in base al riconoscimento degli oggetti presenti in essa. Tale operazione, prende il nome di *Object Categorization* o *Image Classification*. La tesi si inserisce in un lavoro più ampio, che ha la finalità di creare un sistema di visione 3D per l'ausilio alla mobilità autonoma di individui non vedenti e ipovedenti.

1.3 Descrizione dell'approccio

Questo lavoro è iniziato con uno studio teorico delle principali tecniche di classificazione di immagini note in letteratura, con particolare attenzione ai più diffusi modelli di rappresentazione dell'immagine, quali il modello *Bag of Visual Words*, e ai principali strumenti di *Apprendimento Automatico (Machine Learning)*. In seguito si è focalizzata l'attenzione sulla analisi di ciò che costituisce lo stato dell'arte per la classificazione delle immagini, ovvero il *Deep Learning*.

Per sperimentare i vantaggi dell'insieme di metodologie di Image Classification, si è fatto uso di *Torch7*, un *framework* di calcolo numerico, utilizzabile mediante il *linguaggio di scripting Lua*, open source, con ampio supporto alle metodologie allo stato dell'arte di Deep Learning. Tramite Torch7 è stata implementata la vera e propria classificazione di immagini poiché questo framework, grazie anche al lavoro di analisi portato avanti da alcuni miei colleghi [1] in precedenza, è risultato essere molto efficace nel categorizzare oggetti in immagini.

Le immagini su cui si sono basati i test sperimentali, appartengono a un dataset creato *ad hoc* per il sistema di visione 3D con la finalità di sperimentare il sistema per individui ipovedenti e non vedenti; in esso sono presenti alcuni tra i principali ostacoli che un ipovedente può incontrare nella propria quotidianità. In particolare il dataset si compone di potenziali ostacoli relativi a una ipotetica situazione di utilizzo all'aperto.

Dopo avere stabilito dunque che Torch7 fosse il supporto da usare per la classificazione, l'attenzione si è concentrata sulla possibilità di sfruttare la *Visione Stereo* per aumentare l'accuratezza della classificazione stessa. Infatti, le immagini appartenenti al dataset sopra citato sono state acquisite mediante una *Stereo Camera* con elaborazione su FPGA sviluppata

dal gruppo di ricerca presso il quale è stato svolto questo lavoro. Ciò ha permesso di utilizzare informazioni di tipo 3D, quali il livello di *depth (profondità)* di ogni oggetto appartenente all'immagine, per segmentare, attraverso un algoritmo realizzato in C++, gli oggetti di interesse, escludendo il resto della scena. L'ultima fase del lavoro è stata quella di testare Torch7 sul dataset di immagini, preventivamente segmentate attraverso l'algoritmo di segmentazione appena delineato, al fine di eseguire il riconoscimento della tipologia di ostacolo individuato dal sistema.

1.4 Descrizione del contesto applicativo: il sistema di ausilio 3D

Il lavoro sviluppato in questo documento si inserisce in un progetto di sviluppo di un sistema che possa favorire la mobilità autonoma di un utente non vedente o ipovedente, mantenendo libero l'uso delle mani e cercando tramite stimoli tattili e sonori di renderlo conscio il più possibile dell'ambiente che lo circonda [26] [27]. Questo sistema è costituito da una stereo camera per l'acquisizione delle immagini e una unità di elaborazione e controllo per analizzare i dati e inviare dei segnali di comando a dei sensori vibrotattili che invieranno i feedback all'utilizzatore in base alla presenza o meno di ostacoli. Tramite un paio di cuffie audio a conduzione ossea, vengono integrati gli stimoli tattili con indicazioni sonore, per aumentare la consapevolezza dell'utente in merito a eventuali ostacoli presenti nell'ambiente nel quale si trova. Come sensore di acquisizione dati è stata utilizzata una stereo camera progettata dal DISI e per unità di elaborazione è stato utilizzato un sistema *embedded* e molto compatto denominato Odroid-U3. L'intero sistema ha un peso di alcuni centinaia di grammi ed è alimentato da una normale batteria (*power bank*) di dimensioni molto compatte. Il sistema, determina il piano su quale si muove la persona e ogni volta che evidenzia un ostacolo aziona un motore a vibrazione in corrispondenza della zona in cui si trova l'ostacolo stesso. Questa tesi ha la finalità di completare la detection dell'ostacolo con il riconoscimento della categoria a cui appartiene.

Capitolo 2

La Visione Artificiale

Il questo capitolo si introduce il concetto di visione artificiale, fornendo una panoramica globale su ciò che è e sulle principali parti che costituiscono un sistema di visione. Particolare enfasi è data alla visione stereo, una tecnica di elaborazione delle immagini molto diffusa e vantaggiosa in grado di inferire informazioni 3D a partire da coppie di immagini tradizionali. Dopo aver definito l'output di una telecamera stereo, la mappa di disparità, si descrive l'insieme di concetti, metodi e algoritmi che stanno alla base della Object Categorization. Il capitolo termina con la spiegazione teorica di uno dei punti salienti della tesi: la segmentazione di immagini da dati depth/disparità.

2.1 Cos'è la Visione Artificiale

Partiamo da due definizioni, che introducono il significato di visione artificiale:
"Estrarre descrizioni del mondo da immagini e sequenze di immagini" (Frosyth e Ponce).
"*Come capire, a partire da immagini, cosa c'è nel mondo, dove sono le cose, che eventi stanno avvenendo*" (Marr, 1982).
Per *Visione Artificiale (Computer Vision)* [2] si intende quell'insieme di concetti, tecniche, algoritmi che, nel caso della visione 3D, mirano a creare un modello approssimato del mondo, partendo da immagini bidimensionali. La visione artificiale è una disciplina che muove i primi passi a partire agli inizi degli anni '60, per conoscere uno sviluppo più significativo durante gli anni '80, con l'introduzione di sistemi per applicazioni industriali.
Oggi i sistemi di visione artificiale sono diventati componenti essenziali in molti ambiti; basti pensare al Madical Imaging, alla guida automatica dei robot, *al people tracking* [32] nei luoghi pubblici, alla ispezione industriale, eccetera. Come anticipato nell'introduzione, questa tesi rientra in un progetto più ampio di creazione di un sistema di visione che mira a

consentire alle persone ipovedenti e non vedenti di orientarsi in modo autonomo, aiutandole a capire se e quali sono gli ostacoli che si incontrano nel cammino quotidiano.

Figura 1: alcuni scenari applicativi della Computer Vision

2.2 Sistema di Visione Artificiale

Lo studio della struttura fisica di un sistema di visione artificiale non è tra gli obiettivi di questa tesi; tuttavia è utile introdurre a grandi linee il complesso di elementi, con particolare riferimento alla *Visione Stereo,* che invece ha molta importanza per il lavoro svolto.

2.2.1 Descrizione

Un sistema di visione artificiale è un insieme di apparati elettronici che permettono di acquisire informazioni relativamente alla scena che si vuole analizzare. Le parti del "mondo reale" da controllare, vengono osservate attraverso l'uso di uno o più sensori tridimensionali, attivi o passivi e basati su telecamere. Tra le ragioni per cui si opta spesso per i sensori passivi, in sostituzione di quelli basati su tecnologie di tipo attivo quali *time of flight* o basati

su tecnologia laser, possiamo evidenziare la non interferenza tra sensori dello stesso tipo nella medesima area di *sensing*, la loro capacità di poter essere utilizzati in scenari outdoor e, talvolta, il loro minore costo.
A partire dalla osservazione, la cui qualità può essere resa migliore utilizzando sistemi di illuminazione appropriata, si crea un segnale in uscita dalle telecamere, che viene digitalizzato e memorizzato.

2.2.2 La Visione Stereo

Tra le diverse tecniche di computer vision note in letteratura e mirate alla ricostruzione della struttura tridimensionale di una scena osservata da una o più telecamere, la *visione stereoscopica* [3] ha riscosso notevole attenzione, soprattutto perché non impone particolari vincoli sulle caratteristiche degli oggetti presenti nella scena (e.g presenza o meno di oggetti in movimento, presenza o meno di particolari condizioni di illuminazione).
Una *stereo camera* è un particolare tipo di telecamera dotata di due sensori di immagine, che in fig.2 sono chiamati Left Camera e Right Camera; ciò permette alla telecamera di avere una *visione binoculare* e di consentire, mediante l'utilizzo di opportuni algoritmi di ottenere immagini di profondità mediante il principio della *triangolazione* [3].

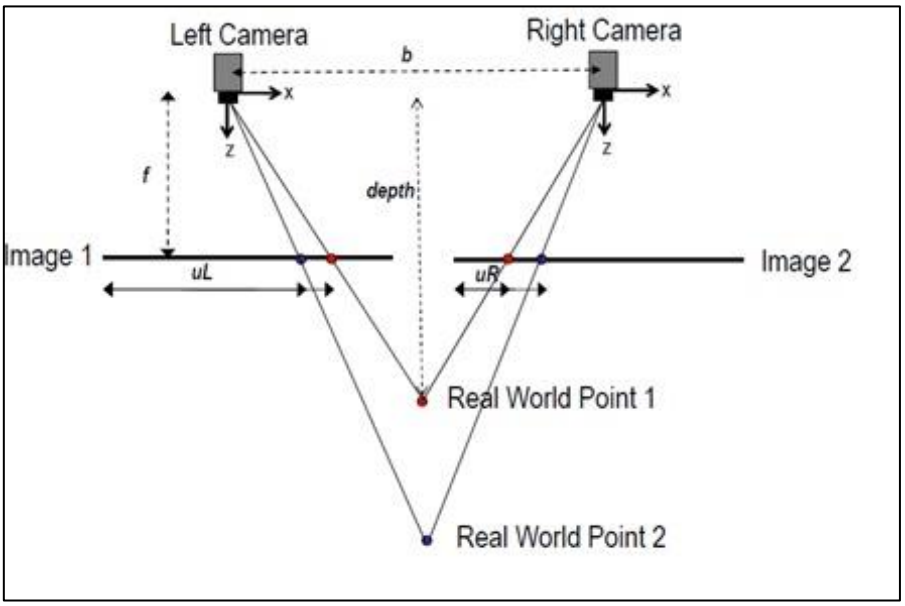


Figura 2: schema che riproduce la visione binoculare mediante Stereo Camera

La triangolazione mette in relazione la proiezione di un punto della scena sui due piani dell'immagine degli obiettivi che compongono il sistema stereoscopico. Tali punti sono definiti *omologhi* e la loro individuazione, nota come *matching stereo*, consente di ottenere una grandezza denominata *disparità*. Le tecniche di matching stereo sono le più disparate [3], ma forniscono, come output della stereo camera, la cosiddetta *mappa di disparità*, nella quale ogni pixel consente di ottenere informazioni di profondità per ogni punto corrispondente della scena reale osservata dalla telecamera, come mostrato in figura 3. Sfruttando la mappa di disparità è possibile risalire alla posizione 3D del punto considerato. La mappa di disparità e il suo utilizzo, sarà descritta in modo ampio nel capitolo 5, dove si analizzerà l'algoritmo implementato per la segmentazione di immagini.

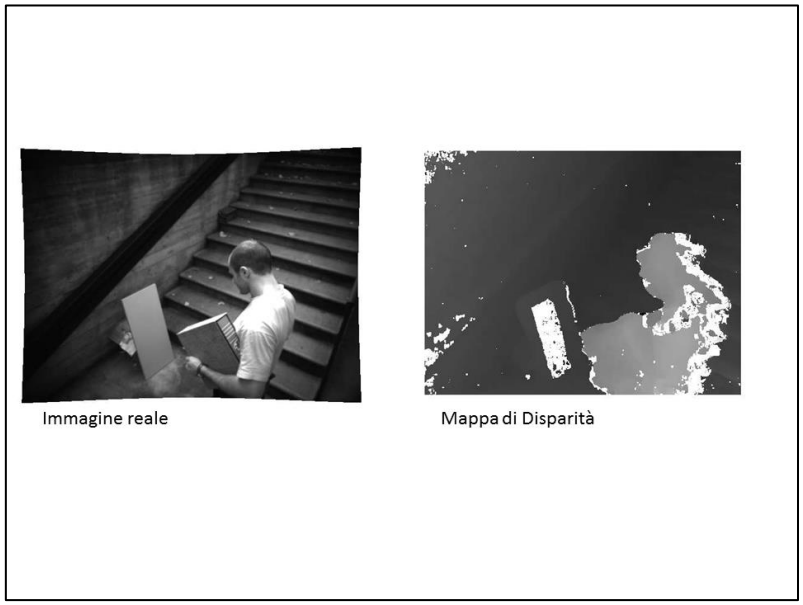


Figura 3: Immagine reale a sinistra e mappa di disparità a destra

Nella tesi è stata usata, come strumento di acquisizione delle immagini da elaborare, una stereo camera progettata al DISI, in grado di generare mappe di disparità molto accurate elaborando su FPGA due immagini sincronizzate, acquisite da sensori di immagini digitali.
Tale telecamera, descritta in [4] [5], è dotata di una interfaccia USB mediante la quale è anche alimentata, ed è in grado di generare mappe di disparità con algoritmi allo stato dell'arte implementati su una FPGA a basso costo.

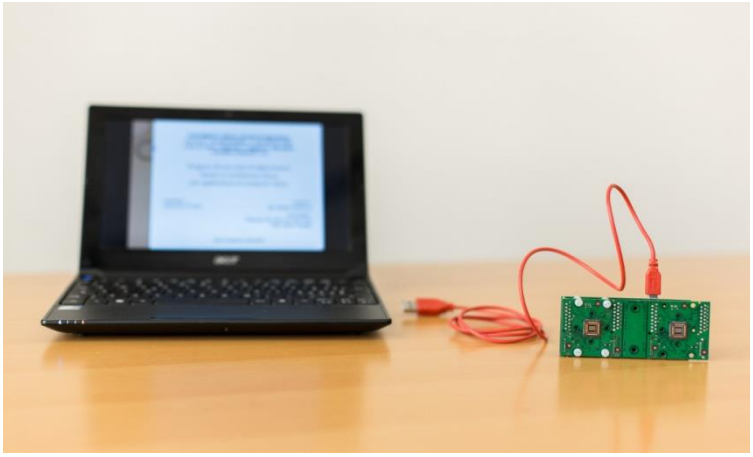


Figura 4: stereo camera progettata al DISI

2.3 Alcune tematiche di Visione Artificiale

Una volta acquisita l'immagine mediante sistema di visione, si passa alla sua elaborazione, tipicamente attraverso opportuni algoritmi implementati su architetture basate su CPU. Tale elaborazione ha molteplici finalità e i temi affrontati in computer vision sono molti. Per quella che è la mia esperienza e grazie al confronto con altri laureandi, ho potuto avvicinarmi ad alcuni problemi molto attivi in ambito di ricerca:

- Tracking 3D;
- SLAM;
- Object Categorization.

Come anticipato in precedenza, questo lavoro si inquadra prevalentemente nel contesto della Object Categorization.

2.3.1 Object Categorization

La tesi si è focalizzata in particolare sulla capacità di un calcolatore di riconoscere e classificare oggetti appartenenti a immagini. Essa si inquadra in un settore, chiamato *Pattern Recognition*, che si serve di metodi per il riconoscimento di pattern, *schemi*, specifici nei dati, acquisendo la capacità di classificarli.

2.3.1.1 Introduzione del problema

Nel caso di immagini, riuscire a identificare la classe a cui appartiene una determinata immagine, in base agli oggetti presenti nella scena, necessita di molteplici operazioni., riassunte concettualmente dallo schema a blocchi della pagina seguente:

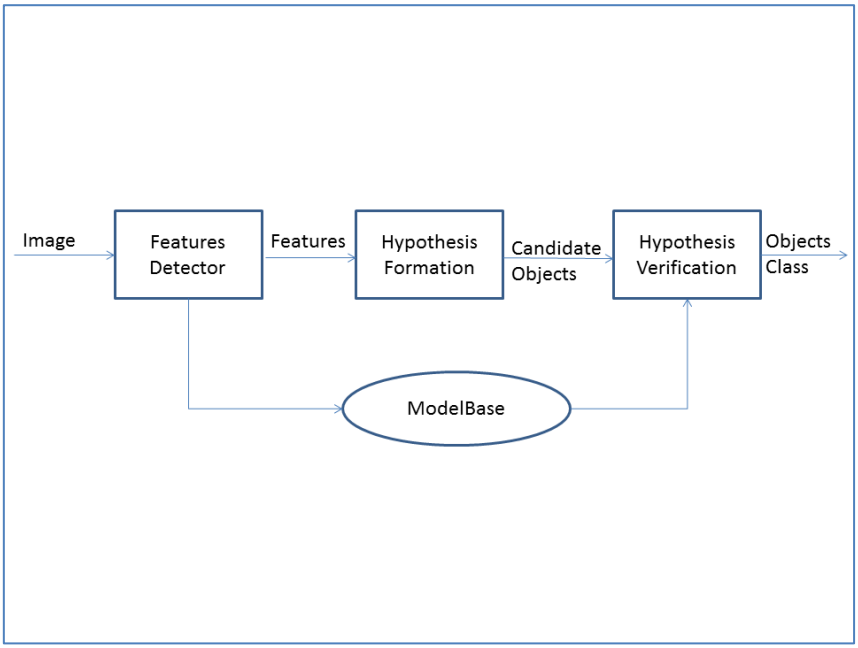


Figura 5: schema a blocchi del processo di Object Categorization

Il termine *feature* indica una informazione rilevante dal punto di vista computazionale, per riuscire a raggiungere un certo obiettivo. Nel caso di immagini, tali informazioni corrispondono alle caratteristiche che rendono gli oggetti di una scena descrivibili in maniera più possibile univoca, per poter essere individuati e confrontati con altri oggetti, aventi *features* differenti.

Le features possono essere classificate in due categorie:

- *Feature Globali*: permettono di rappresentare un oggetto nella sua totalità; non dipendono dal dettaglio delle forme, ma cercano di individuare l'andamento generale della scena. Esempi sono l'istogramma e la texture dell'immagine.
- *Feature Locali*: si tratta di parti dell'immagine facilmente rilevabili e che rendono l'identificazione di oggetti molto più robusta. Tipicamente dipendono dalla forma e dalla texture degli oggetti presenti nella scena.

Esempi di features locali possono essere i fanali per le automobili, la corteccia per gli alberi, parti delle ali per gli aerei, gli occhi e il naso per le persone, ma anche semplicemente la forma di un oggetto, il suo perimetro, l'area, il gradiente legato all'intensità dei pixel.

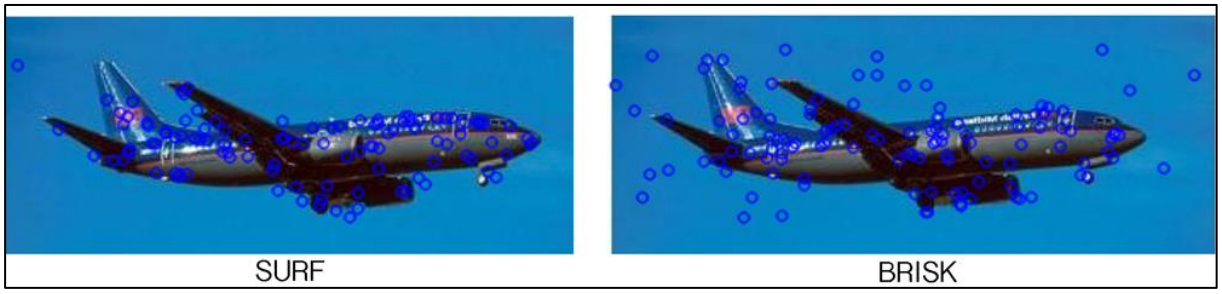


Figura 6: risultati della applicazione di due importanti algoritmi di Feautures Extraction: SURF [6] e BRISK[7]

L'individuazione e l'estrazione di features da una immagine è un tema molto importante, poiché la descrizione di un oggetto deve essere il più possibile affidabile e indipendente dalle condizioni in cui i dati vengono acquisiti (e.g illuminazione, disturbi, angolazione della telecamera).

Esistono diversi algoritmi di *Feature Extraction*: tra i più utilizzati citiamo SIFT [8], SURF [6] e BRISK [7]. Per una trattazione completa dei principali algoritmi di *Feature Extraction* si rimanda a [8].

Continuiamo a riferirci allo schema a blocchi precedente e osserviamo come i blocchi relativi alla ipotesi e verifica delle informazioni acquisite, nonché il *modelbase*, comprendono tutte le metodologie di rappresentazione dell'immagine e di apprendimento automatico (Machine Learning) che permettono di costruire un robusto sistema di classificazione.

L'approccio utilizzato in questo lavoro prevede che tutto il processo di classificazione si traduca in una operazione di training su un insieme molto vasto di immagini. Ognuno degli esempi appartenenti al *training set* è "categorizzato" tramite un'etichetta, in modo da permettere al sistema a riconoscere i pattern specifici per quella determinata categoria. Di Machine Learning e modelli di rappresentazione si parlerà in maniera esaustiva nel capitoli 3 e 4.

2.3.1.2 Elementi di Image Processing: segmentazione di immagini

Nella maggior parte di algoritmi di Feauture Detection/Feautures Extraction, la vera e propria operazione di estrazione di features è preceduta da operazioni di *elaborazione dell'immagine (Image Processing)* che hanno l'obiettivo di fornire in input agli algoritmi dati normalizzati, il più possibile semplici e significativi da analizzare. Qui di seguito sono citate solo alcune delle operazioni di Image Processing, prestando particolare attenzione alla segmentazione, poiché parte di questo lavoro è stato quello di implementare un algoritmo che realizza questa operazione:

- *Preprocessing;*
- *Edge-Detection;*
- *Segmentazione.*

Il preprocessing [9], va a modificare quello che è l'istogramma dell'immagine, ossia il grafico nel quale ad ogni tono di colore, in ascissa, si associa la frequenza di ripetizione nei pixel dell'immagine, in ordinata. L'esempio sotto mostra l'istogramma di una immagine a livelli di grigio codificata con 8 bit.

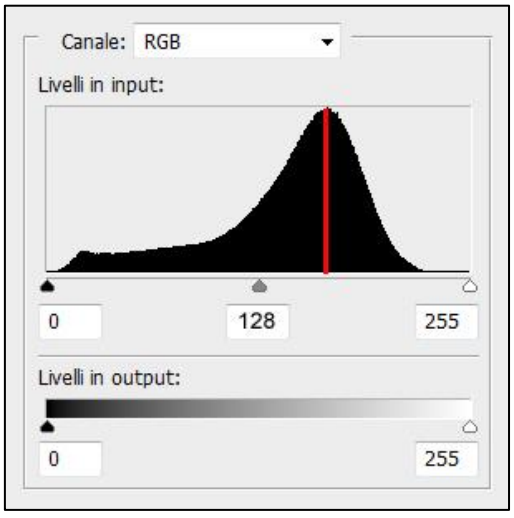


Figura 7: Istogramma di una immagine

Il preprocessing consiste tipicamente in una operazione di *normalizzazione* dell'istogramma, seguita da applicazione di filtri di *smoothing* e di *sharpening* all'immagine di partenza.

Con la normalizzazione si riesce ad evitare la presenza di porzioni poco riconoscibili nell'immagine, redistribuendo le intensità, mentre con le operazioni di smoothing e sharpening si può ridurre il rumore, le variazioni di luminosità e mettere in evidenza i dettagli dell'immagine e i suoi contorni.

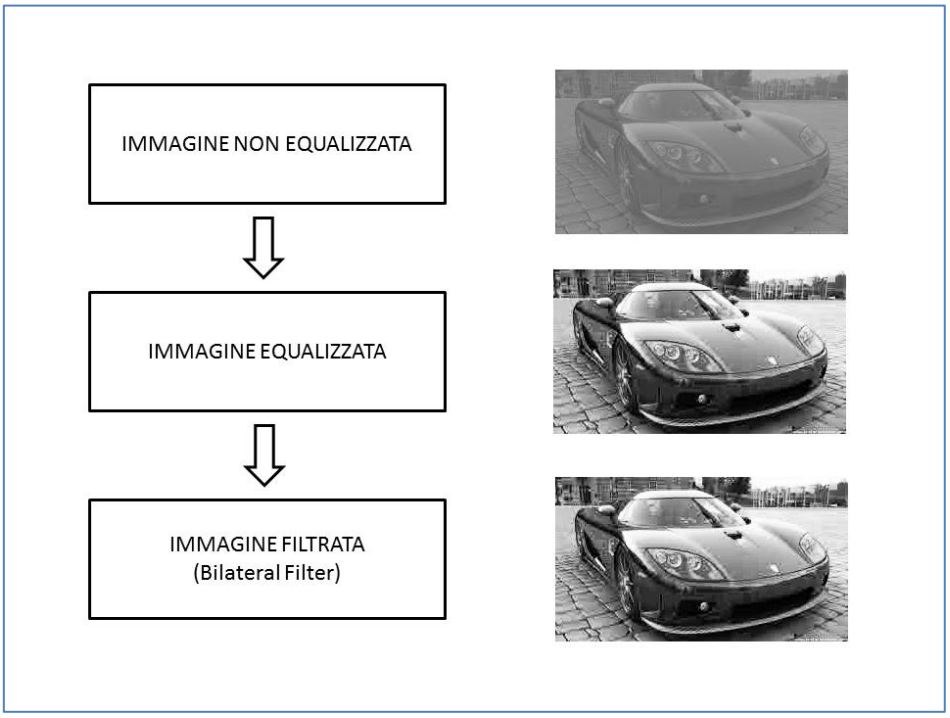


Figura 8: esempio di equalizzazione e filtraggio di immagini rumorose

Per *edge (contorno)* si intende un insieme di pixel dell'immagine, nel quale vi è un significativo punto di discontinuità, che corrisponde in molti casi ad un cambiamento del soggetto nella realtà. Matematicamente questa proprietà si traduce in brusche variazioni del gradiente della luminosità nei pixel dell'immagine.

L'applicazione di *edge-detectors* [9], può introdurre una notevole semplificazione della fisionomia della immagine; infatti l'output si presenta come un insieme di curve, che corrispondono appunto ai contorni.

Sono eliminate informazioni che sono meno significative, mantenendo quella che è la struttura geometrica degli oggetti nell'immagine di partenza.

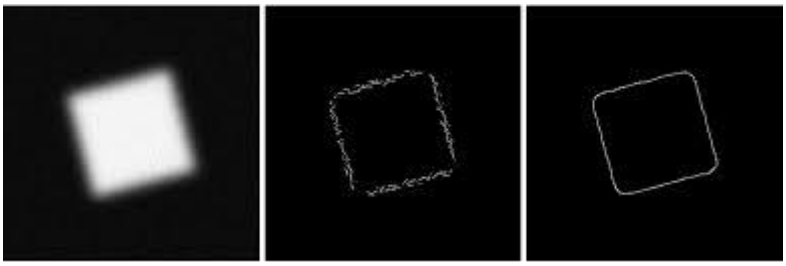


Figura 9: effetti della edge detection

Introduciamo ora l'operazione di image processing che maggiormente è stata inerente questo lavoro: la *segmentazione di immagini*.

La figura seguente, mostra in modo molto intuitivo il significato di segmentazione. Segmentare significa *ripartire* un insieme di elementi in sottoinsiemi, secondo un certo *criterio* (e.g, colore, forma, carattere).

Figura 10: concetto di segmentazione

Generalmente una immagine contiene diversi oggetti nella scena. L'obiettivo della operazione di segmentazione [9] è quello di dividere una immagine in *regioni omogene*, dove tutti i pixel corrispondenti ad un oggetto vengono identificati allo stesso modo, ad esempio con lo stesso colore. Nel momento in cui ogni oggetto presente nell'immagine è identificatocon una *label,* si può procedere in base agli scopi da raggiungere (e.g. etichettare l'oggetto, isolarlo dal contesto). In questo lavoro la segmentazione è stata utilizzata per isolare un oggetto di

interesse specifico dal resto della scena, in modo che l'operazione di image classification potesse agire esclusivamente riferendosi a quell'oggetto [10].

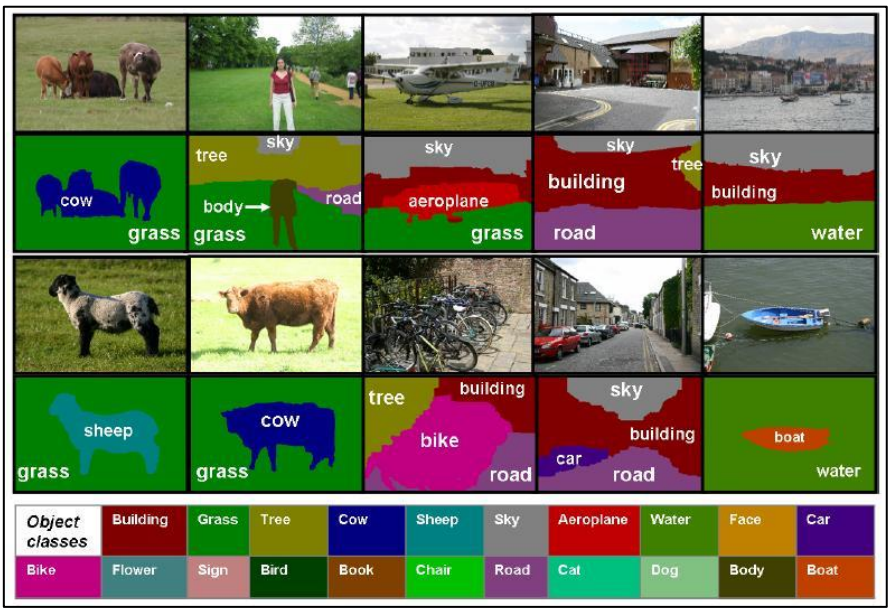


Figura 11: segmentazione per labeling

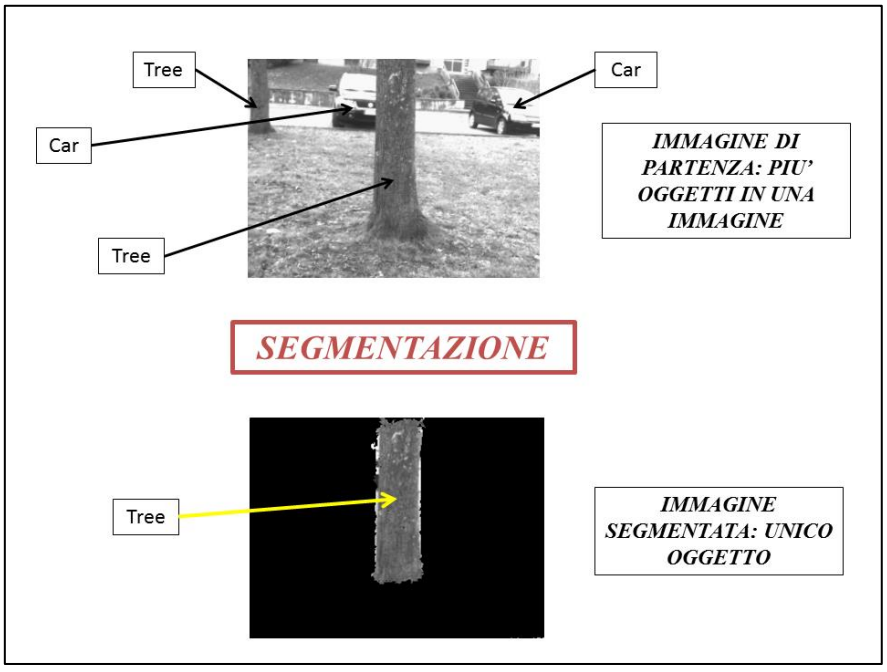


Figura 12: segmentazione per isolare un oggetto dallo sfondo/terreno

Il raggruppamento dei pixel in regioni si basa, idealmente, su alcuni criteri:

- Nessun pixel può essere condiviso da due regioni;
- Tutti i pixel dell'immagine sono assegnati ad almeno una regione della partizione;
- Tutti i pixel appartenenti ad una regione sono connessi, ad esempio in base a criteri di similarità degli attributi (e.g. colore, texture) o a valori di prossimità spaziale (e.g. distanza Euclidea).
- Tutte le regioni sono omogenee rispetto ad un criterio fissato.

I principi di similarità e di prossimità sono motivati dalla considerazione che regioni omogenee derivano dalla proiezione di punti di un oggetto nell'immagine rappresentati da pixel che sono vicini, con simili valori di tonalità.

Tuttavia queste assunzioni non sono sempre valide, come sarà mostrato nella seconda parte della tesi, in cui parlerò dei risultati ottenuti.

Esistono diversi sono gli approcci per la segmentazione, alcuni dei quali creati ad hoc per la specifica applicazione.

In particolare questi algoritmi si possono distinguere in:

1. *Algoritmi Knowledge-Based*: basati sulla conoscenza globale o parziale dell'immagine, rappresentata dagli istogrammi di alcune caratteristiche dell'immagine;
2. *Algoritmi Edge-Based*: basati sui contorni degli oggetti nelle immagini;
3. *Algoritmi Region-Based*: basati su regioni di immagini che hanno le stesse caratteristiche e che tipicamente appartengono agli stessi oggetti.

A loro volta gli algoritmi 2) e 3) si caratterizzano in base alle proprietà utilizzate per l'estrazione di bordi o regioni (e.g. luminanza, colore, texture).

In questa tesi, è stato implementato un algoritmo di segmentazione di tipo Region-Based che ha lo scopo di creare, data una immagine con più oggetti nella scena, un output nel quale esista un solo oggetto, coincidente con quello più vicino alla stereo camera con cui è stata ripresa la scena. Questo algoritmo verrà ampiamente descritto nel capitolo 5, e saranno analizzati i risultati ottenuti e i vantaggi auspicati nel contesto della classificazione di immagini.

Capitolo 3

Il Machine Learning

In questo capitolo è presentato il concetto di apprendimento automatico, attraverso cui un elaboratore è in grado di "imparare" schemi e classificare dati in base a questa fase di apprendimento. Dopo aver introdotto il Machine Learning, si passa alla descrizione di alcune tipologie di classificazione, ponendo particolare enfasi sulle reti neurali. La classificazione basata su reti neurali artificiali (ANN) è particolarmente adatta all'elaborazione delle immagini, come si potrà evincere dalla descrizione dell'architettura e del funzionamento delle ANN.

3.1 Cos'è il Machine Learning

"Un programma apprende da una certa esperienza E se: nel rispetto di una classe di compiti T, con una misura della prestazione P, la prestazione P misurata nello svolgere il compito T è migliorata dall'esperienza E" (T.M. Mitchell)

Ricercare schemi, patterns nei dati, in modo automatico, è una sfida affascinante. In Computer Science, questo tema, che va sotto il nome di pattern recognition, è fortemente legato alla capacità di un calcolatore di implementare algoritmi che utilizzino ricorrenze nei dati per azioni quali la classificazione di nuovi dati in diverse categorie.
L'apprendimento automatico, noto in letteratura come Machine Learning [8], è una parte di computer science che si occupa di individuare metodi e realizzare sistemi e algoritmi tramite i quali un calcolatore può imparare, basandosi su degli esempi dati in input.
La sfida del Machine Lerning è dunque permettere ad un Computer di imparare a riconoscere automaticamente modelli complessi e prendere decisioni il più possibile intelligenti.

Partiamo da un esempio: si vuole creare un algoritmo che, dati in input i primi 10 numeri decimali scritti a mano, rappresentati in immagini 28x28 pixel, associ in output ad ogni cifra, il corrispondente simbolo {0,1,….,9}, che ne definisce la categoria di appartenenza.

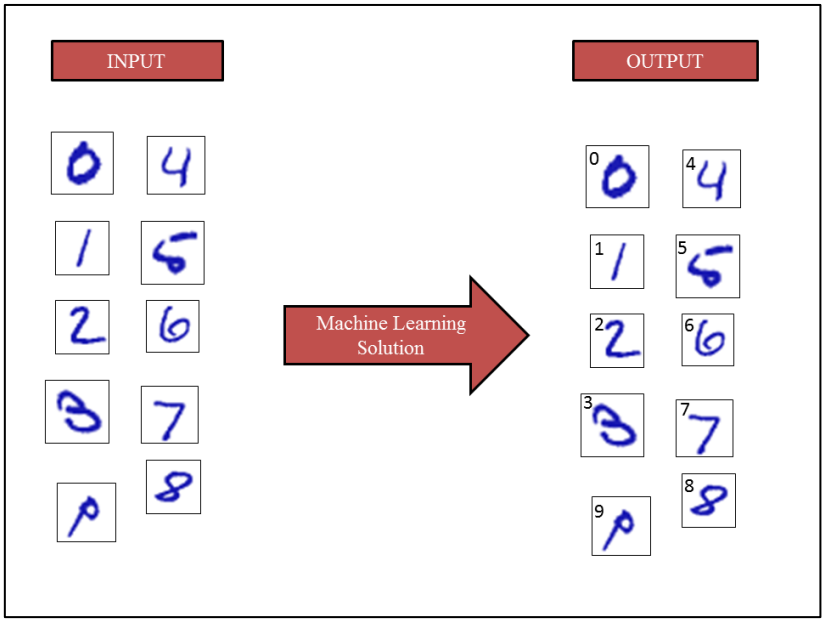


Figura 12: esemplificazione del problema inerente il riconoscimento di numeri

Questo problema può essere risolto seguendo due possibili approcci di apprendimento automatico, che corrispondono a due tra i più importanti paradigmi di Learning:

1. *Supervised Learning*
2. *Unsipervised Learning.*

Il Supervised Learning [11] parte da un insieme *N* che contiene un numero elevato di vettori che descrivono cifre scritte a mano: tale insieme è detto *training set* ed è usato per settare i parametri modello adattativo , che si basa sull'utilizzo di un *classificatore*. In seguito saranno introdotti i principali tipi di classificatore.

Le categorie di appartenenza dei numeri nel training set sono conosciute in anticipo, tipicamente attraverso un procedimento di *labeling*, ossia associando ad ogni vettore una etichetta/categoria.

Possiamo esprimere la categoria del numero usando un *target vector t*, che rappresenta l'identità del corrispondente numero scritto a mano. E' importante notare come a questo punto possa esserci per ogni immagine x un vettore t.

L’insieme N di training si presenta come una successione di coppie di vettori del tipo

$$\{(x_1, t_1), \ldots\ldots, (x_n, t_n)\}$$

Il risultato di un algoritmo di Machine Learning, può essere espresso come una funzione *g* che prende in ingresso una cifra *x* in input e genera in output un vettore *y*, codificato nello stesso modo di t. La forma della funzione y è determinata nella fase di allenamento, nota anche come fase di learning, in base ai dati di training. Lo scopo della fase di training è quello di approssimare il più possibile una funzione f ideale, che svolge la stessa funzione di g, ma con una accuratezza pari al 100%. Esistono diversi algoritmi di apprendimento supervisionato ma tutti condividono una caratteristica: l'allenamento viene eseguito mediante la minimizzazione di una *funzione di costo,* la cosiddetta *loss function,* che rappresenta l'errore dell'output fornito dalla rete rispetto all'output desiderato.

Una funzione di costo spesso utilizzata è lo scarto quadratico:

$$\frac{1}{2} * \sum_i (y_i - y)^2 \qquad (1)$$

Dove y rappresenta l’output desiderato e y_i è l’output calcolato all’i-esima iterazione, con *i* che varia in base alla cardinalità del training set.

La scelta della loss function da minimizzare è importante poiché essa sottintende il principio che sta alla base dell'apprendimento. Nella pratica è buona norma non utilizzare tutto l’insieme di training per allenare la rete, perché, minimizzando quella tipologia di funzione, la rete tende a legarsi in maniera troppo specifica agli esempi del training set e quando l'input varia, si corre il rischio di avere una predizione errata, un *overfitting* [12]. Una possibile soluzione è dividere il training set in due parti uno per l'allenamento e uno per una verifica. Si usa quello più piccolo per controllare l'efficienza dell'addestramento, valutando la loss function su di esso.

Una volta che il modello è allenato si può determinare la categoria di una nuova immagine che appartiene a un *test set*. Una possibile misura della qualità di predizione è la *verosimiglianza a posteriori*, che è la misura sul test set della probabilità che il modello assegni la giusta categoria ad una nuova immagine in input.

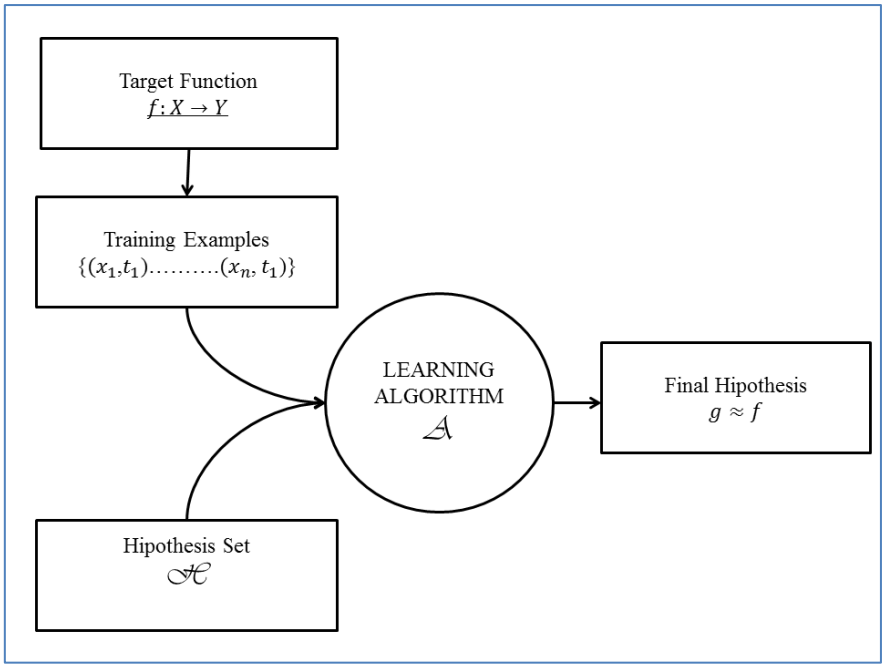


Figura 13: schema generale del Supervised Learning [13]

Per la maggior parte delle applicazioni, gli input vengono trasformati in un nuovo spazio di variabili, dove il problema di pattern recognition è più semplice. Per esempio, in un problema di image classification, ogni immagine è sottoposta alle operazioni di image processing e features extraction introdotte nel capitolo 2.

Attraverso tali tecniche l'immagine è trasformata in vettori di features: si parla di *Feature Vector* o *Feature Map*. Tali vettori contengono le informazioni essenziali per classificare una immagine: si eliminano di fatto informazioni ridondanti che diminuirebbero l'accuratezza della classificazione e aumenterebbero il carico computazionale degli algoritmi di learning.

Il diagramma di flusso precedente mostra l'approccio supervised a un problema di classificazione mediante Machine Learning.

Il secondo paradigma utilizzato per l'apprendimento automatico è quello unsupervised [13]. In questo caso l'apprendimento fornisce al modello di apprendimento una serie di input "non etichettati" e fa si che il modello stesso riclassifichi e organizzi gli input sulla base di caratteristiche comuni, effettuando ragionamenti e previsioni su input successivi. Un esempio tipico di tale approccio lo ritroviamo nei motori di ricerca, in cui dato un insieme di parole chiave, il modello crea una lista di link che sono ritenuti pertinenti alle parole inserite in input. La tecnica unsupervised procede confrontando i dati e cercando analogie e differenze.

In questo lavoro è stata considerata la metodologia del supervised learning.

3.2 Principali metodi di Classificazione

Il classificatore è uno degli strumenti fondamentali di Machine Learning. E' un insieme di algoritmi che, nel caso di nostro interesse, acquisisce una immagine dalla quale sono state estratte features e la classifica in base agli oggetti presenti nella scena.
La modalità con cui avviene questa classificazione dipende dal tipo di classificatore.
In generale possiamo distinguere [8]:

- *Classificatori Statistici*: sono algoritmi che agiscono analizzando le distribuzioni di probabilità relative alle classi e alle features nelle immagini. Per classificare un generico oggetto si stimano le probabilità di appartenenza a una classe. Un tipico esempio è il classificatore *Naive Bayes* [8]
- *Classificatori Basati sugli esempi*: questi metodi memorizzano tutti gli esempi di training e classificano un oggetto valutando la somiglianza con gli esempi memorizzati, per i quali la classe di appartenenza è nota. Un esempio è il *Nearest Neighbor.*[8]
- *Classificatori Matematici*: attraverso esempi di training questi classificatori creano una funzione matematica che modella i dati e che viene utilizzata per inidividuare l'identità corretta degli oggetti da classificare. Appartengono a questa categoria di classificatori l'algoritmo *SVM (Support Vector Machines)* [8] e le *Reti Neurali (Neural Network).*
- *Classificatori Logici*: la funzione di classificazione è espressa con condizioni logiche sui valori degli attributi estratti dai dati. Sono esempi gli *Alberi di Decisione (Decision Tree)* [8].

3.2.1 Le Reti Neurali (ANN's)

In questo lavoro di tesi, la classificazione delle immagini è stata realizzata attraverso l'uso di Torch7. Torch è un framework di Deep Learning, e come tale sfrutta le proprietà delle reti neurali per riconoscere attributi nelle immagini e associarle alla rispettiva categoria. Concentriamoci quindi sull'analisi di questo tipo di classificatore, rimandando a [8] per una descrizione esaustiva di tutti gli altri metodi introdotti nel precedente paragrafo.

Una *Rete Neurale Artificiale (Artificial Neural Network, ANN)* è un sistema di elaborazione dell'informazione i cui meccanismi di funzionamento si ispirano ai circuiti nervosi biologici. Le reti neurali, grazie alle loro caratteristiche sono protagonisti di una vera e propria rivoluzione nei sistemi di apprendimento automatico e più in generale nell'ambito di Intelligenza Artificiale.
Una rete neurale artificiale possiede molte semplici unità di elaborazione variamente connesse tra loro, secondo varie architetture. Un esempio:

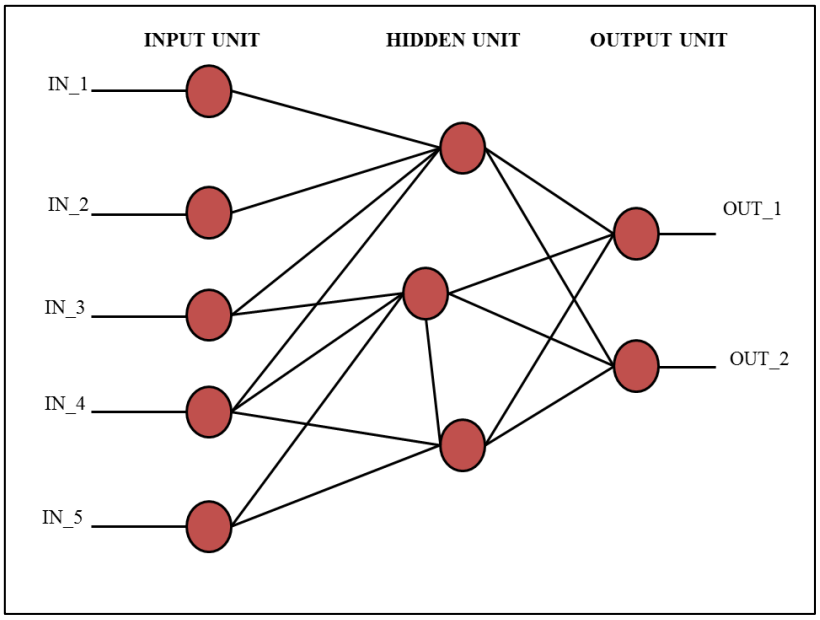


Figura 14: una possibile architettura per ANN's

Dalla figura si può evincere come esistano unità che comunicano con l'ambiente esterno, sia in input che in output, altre che comunicano solo con le unità interne alla rete : parleremo di *unità di input, unità di output* e *unità nascoste (hidden).*
Ciascuna unità o nodo simula il ruolo del neurone nelle reti neurali biologiche. Ogni nodo, detto *neurone artificiale*, svolge un'operazione molto semplice: diventa attivo se la quantità totale di segnale, che riceve supera la propria soglia di attivazione, definita da una funzione *f* detta di appunto di *funzione di attivazione*. Se un nodo diventa attivo emette un segnale *y* che viene trasmesso lungo i canali di trasmissione fino alle altre unità a cui esso è connesso. Ciascun punto di connessione agisce come un filtro che trasforma il messaggio in un segnale *inibitorio o eccitatorio* aumentandone o diminuendone l'intensità, secondo le proprie caratteristiche individuali. I punti di connessione simulano le sinapsi biologiche e hanno la

funzione fondamentale di pesare l'intensità dei segnali trasmessi, moltiplicandoli per dei *pesi w* il cui valore dipende dalla connessione stessa [14].

3.2.1.1 Tassonomia delle Reti Neurali

Il modo di connettere i nodi, il numero di *strati (layers)* presenti, ossia i livelli di nodi presenti tra ingresso e uscita, il numero di neuroni per strato, definiscono *l'architettura di una rete neurale*. Esistono vari tipi di architetture.
Una possibile tassonomia delle reti neurali segue due criteri:

- Flusso di segnali
- Organizzazione delle connessioni.

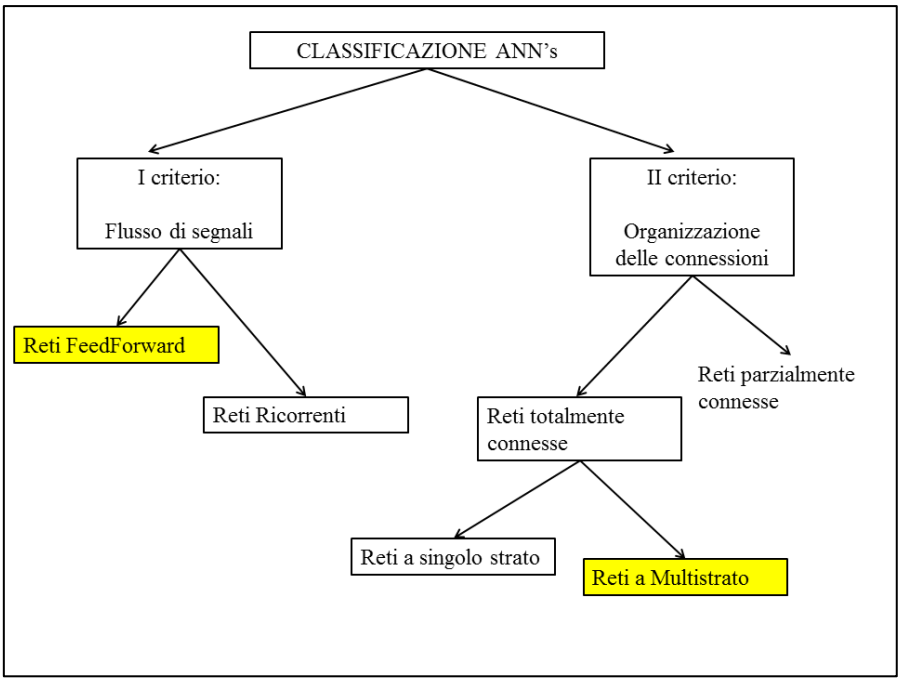


Figura 15: possibile classificazione di ANN's

Le tipologie di ANN's evidenziate in giallo sono quelle più adatte agli obiettivi della Image Classification.

Nelle reti multistrato, si individuano degli strati di neuroni artificiali tali che:

- ogni neurone è connesso con tutti quelli dello strato successivo
- non esistono connessioni tra neuroni appartenenti allo stesso strato, ne connessioni tra neuroni appartenenti a strati non adiacenti
- Il numero di strati e di neuroni per strato dipendono dal problema che si vuole risolvere.

Gli strati in ingresso e in uscita si definiscono di Input e di Output; esistono strati nascosti (*hidden layers*) in base alla cui complessità, si realizzano differenti comportamenti.

Infine, le connessioni tra i neuroni sono rappresentate mediante tante matrici quante sono le coppie di layers adiacenti. Ogni matrice contiene i pesi delle connessioni tra le coppie di nodi di due strati adiacenti.

Le *reti feedforward* sono reti prive di loop all'interno degli strati.

Parleremo dopo di reti multilayer perceptron come esempio di reti multistrato di tipo feedforward.

3.2.1.2 Il neurone artificiale

Facciamo uno zoom della figura precedente, mostrando la struttura di un nodo della rete, che è un neurone artificiale:

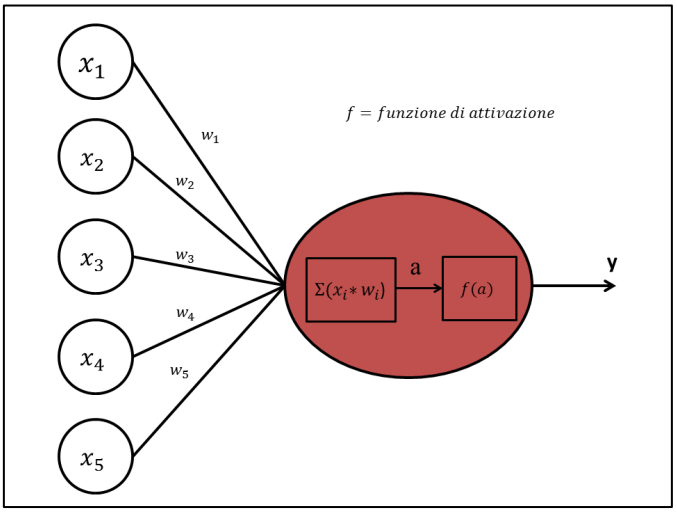


Figura 16: schema di un neurone artificiale

Abbiamo n canali di ingresso (in questo caso n=5) a ciascuno dei quali è associato un peso. I pesi sono numeri reali che riproducono le sinapsi. Se il peso è positivo, il canale è eccitatorio, se è negativo è inibitorio. Il valore assoluto di un peso rappresenta la forza della connessione. L'uscita, cioè il segnale con cui il neurone trasmette la sua attività all'esterno, è calcolata applicando la funzione di attivazione, detta anche di *trasferimento*, alla somma pesata degli ingressi.

$$y = f[a] = f[\sum_{i=1}^{n}(x_i * w_i)] \qquad (2)$$

La funzione di attivazione può essere lineare o non lineare. Andamenti tipici della f sono:

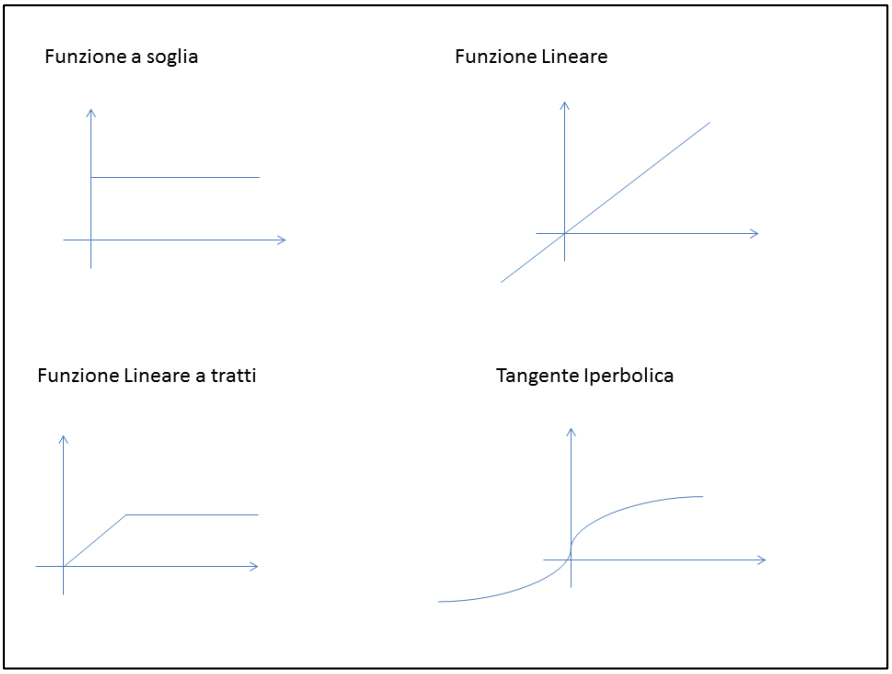


Figura 17: alcune funzioni di attivazione

Si tratta di funzioni che hanno una dinamica compresa tra -1 e 1 o tra 0 e 1. Tipicamente la funzione di attivazione utilizzata è la *tangente iperbolica* o comunque una funzione cosiddetta *sigmoidea:*

$$f(a) = \frac{1}{1 + e^{-a}} \qquad (3)$$

Una rete neurale artificiale è un modello di calcolo adattivo, capace di cambiare la sua struttura in base ai dati in input e alle informazioni che scorrono all'interno di essa. Modificando i pesi dei punti di connessione e la funzione di attivazione, si induce un particolare modello di comportamento della rete.

Il corretto funzionamento di una rete neurale dipende dall'architettura della rete, dalla funzione di attivazione dei neuroni e dai pesi. I primi due parametri sono fissati prima della fase di addestramento. Il compito dell'addestramento è quello di aggiustare i pesi in modo che la rete riproduca le risposte desiderate. Ricordiamo che i principali paradigmi di apprendimento sono il supervised e l'unsupervised.

Prima di passare alla loro descrizione, cerchiamo di capire come agisce un neurone artificiale nel momento in cui in ingresso ha un vettore di feautures estratte preventivamente da una immagine.

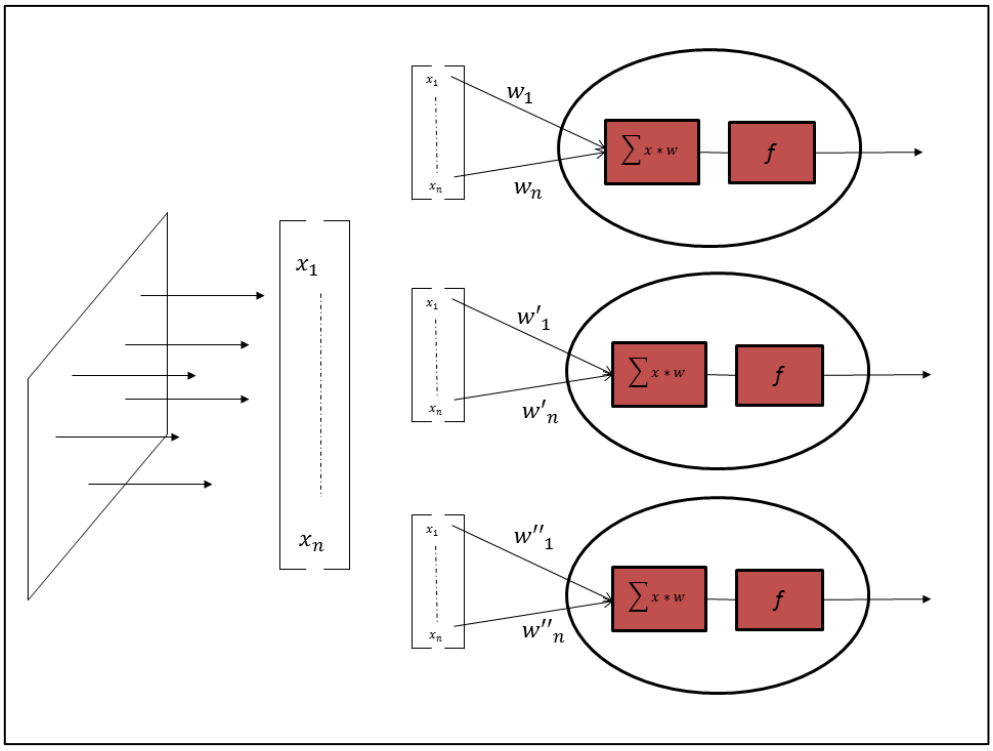


Figura 18: esempio di funzionamento di un layer nella Image Classification

Consideriamo per semplicità una rete neurale ad uno solo layer, composto da tre neuroni. Ogni neurone è caratterizzato da una certa funzione di attivazione non lineare. Il vettore di features estratto dalla immagine di input viene inviato ad ogni neurone. Ogni neurone ha un suo set di pesi, attraverso i quali peserà i valori presenti nel vettore in modo diverso. In

particolare, se la somma pesata dei valori del vettore sono compatibili con la soglia che la funzione di attivazione definisce, allora si genererà il segnale y di uscita al neurone, altrimenti non si genererà. Ciò ci induce a pensare che, ridefinendo i pesi delle connessioni e stabilendo con precisione la funzione di attivazione, la rete riesce a fare una scelta. L'entità della scelta, sarà determinata dal modo in cui avviene la fase di training della rete stessa, che come è noto consiste nella ridefinizione dei pesi associati alle connessioni e che cambia a seconda della modalità di apprendimento da parte della rete neurale.

3.2.1.3 Apprendimento di una NN

L'apprendimento supervised da parte di una rete neurale si configura come un processo iterativo di ottimizzazione dei pesi. I pesi infatti vengono modificati sulla base delle prestazioni che la rete ha su un insieme di esempi, appartenenti al training set, di cui si conosce la categoria di appartenenza.
Lo scopo è quello di minimizzare una funzione di perdita (*loss function*) che indica il grado con cui il comportamento della rete si discosta da quello desiderato. Le prestazioni della rete sono poi verificate su un insieme di test, costituito da immagini diverse da quelle di train. I passi fondamentali di addestramento sono i seguenti:

- Di solito i pesi si inizializzano con valori casuali all'inizio dell'addestramento.
- Si presenta alla rete un elemento del training set. Per ogni elemento si calcola l'errore commesso dalla reta, ossia la differenza tra l'uscita desiderata e l'uscita effettiva. Tale errore serve per aggiustare i pesi.
- Il processo viene ripetuto ripresentando alla rete, in ordine casuale, tutti gli esempi del training set fino a che l'errore commesso su tutto il training set non risulta inferiore ad una certa soglia.

Un algoritmo di apprendimento supervised particolarmente utilizzato ed efficace è l'algoritmo *BackPropagation* [1].
Le prestazioni di una rete neurale dipendono molto dall'insieme di esempi scelti per la fase di allenamento. Tali esempi devono essere rappresentativi della realtà che la rete deve apprendere e in cui verrà utilizzata. L'addestramento è in effetti un processo ad hoc dipendente dallo specifico problema trattato.

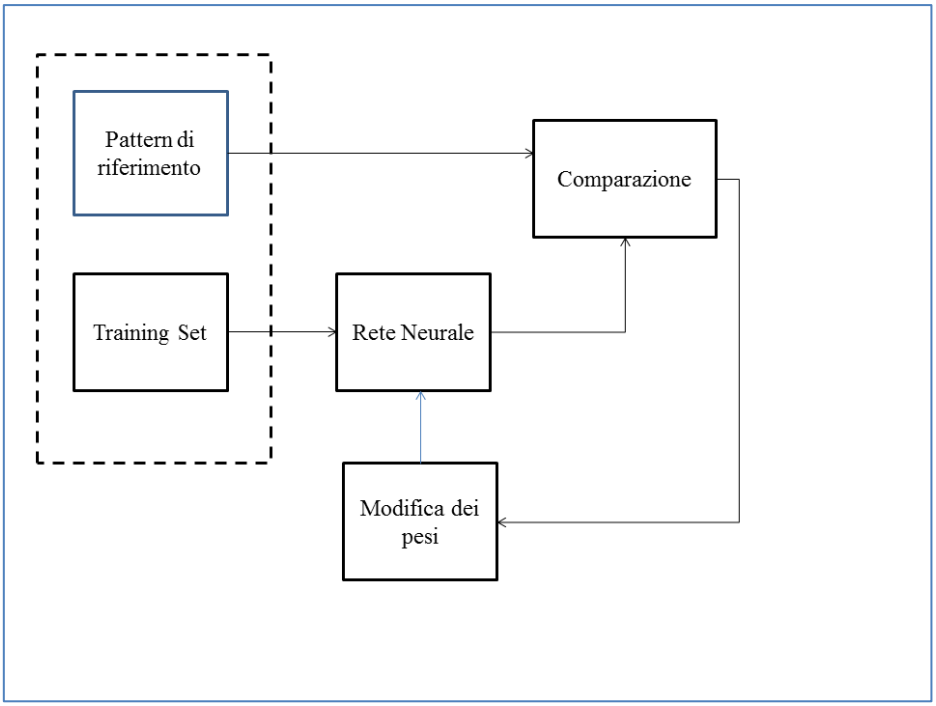


Figura 19: Supervised Learning

La rete prende in ingresso, nella fase di training, dati di cui si conosce la natura, e "impara" quali sono le caratteristiche salienti di questi dati, definite dal pattern che funge da supervisore, in modo che poi, nella fase di test, sarà in grado di classificare sfruttando ciò che ha imparato.

3.2.1.5 Multi Layer Perceptron

Come conclusione della introduzione alle reti neurali, si analizza una particolare struttura di NN che prende il nome di MultiLayer Perceptron (MLP). Si tratta di una ANN di tipo FeedForward a MultiStrato. Cercheremo di capire come agisce questa rete neurale nel caso di Image Classification.

Il *perceptron* è il modello di rete neurale più semplice [8]. Si tratta di una rete costituita da un singolo neurone artificiale, la cui funzione di attivazione è la funzione *sgn().*

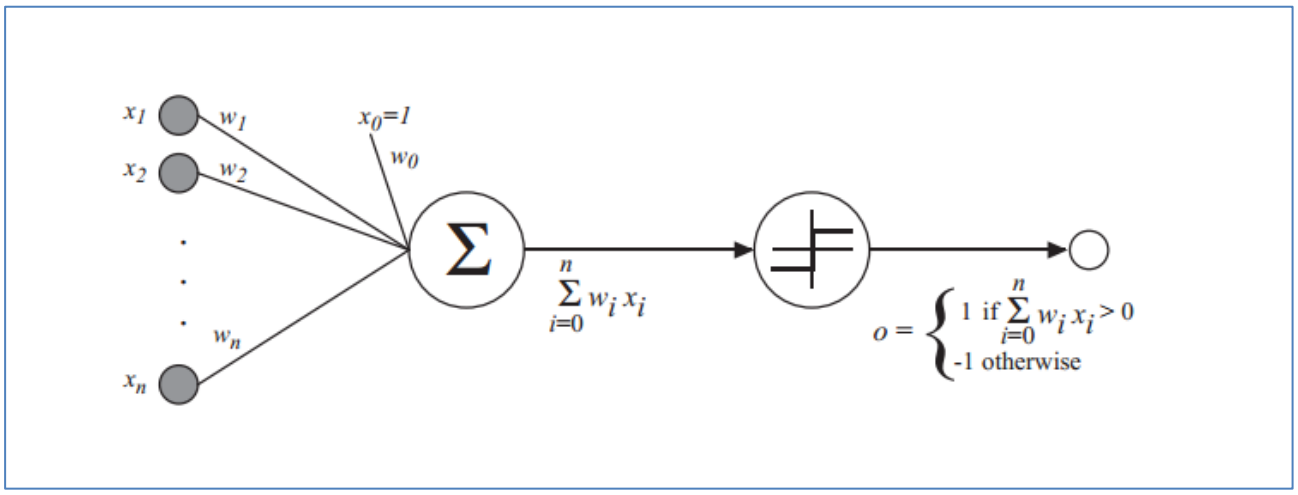


Figura20: schema di un Perceptron

Questo tipo di rete neurale elementare, permette di implementare solo poche funzioni, quali ad esempio l'AND logico o lo XOR [14].

Tuttavia, complicando la struttura della rete con più layers, e modificando le funzioni di attivazione dei neuroni artificiali, è possibile addestrare la rete ad approssimare qualsiasi funzione e risolvere una vasta gamma di problemi, come ci dice il *teorema di approssimazione universale* [15].

La rete MLP è la naturale estensione del perceptron introdotto in figura. In essa si individuano tre blocchi:

1. Unità di input: il numero di neuroni è pari alla dimensione dei vettori di ingresso.
2. Hidden layers: sono strati di neuroni nascosti nei quali avviene l'elaborazione degli input; come si vedrà ampiamente nella parte dedicata al Deep Learning, sono gli strati in cui la rete memorizza la propria rappresentazione dei dati che riceve in ingresso. Gli strati nascosti possono essere uno o più. Il loro numero definisce il livello di profondità e di complessità di di una rete neurale.
3. Unità di output: il numero di neuroni, come nella sezione d'ingresso, è pari alla dimensione dello spazio degli output.

E' importante sottolineare come la MLP sia una rete *fully connected*: ogni neurone di uno strato è necessariamente connesso ad almeno un neurone dello strato successivo.

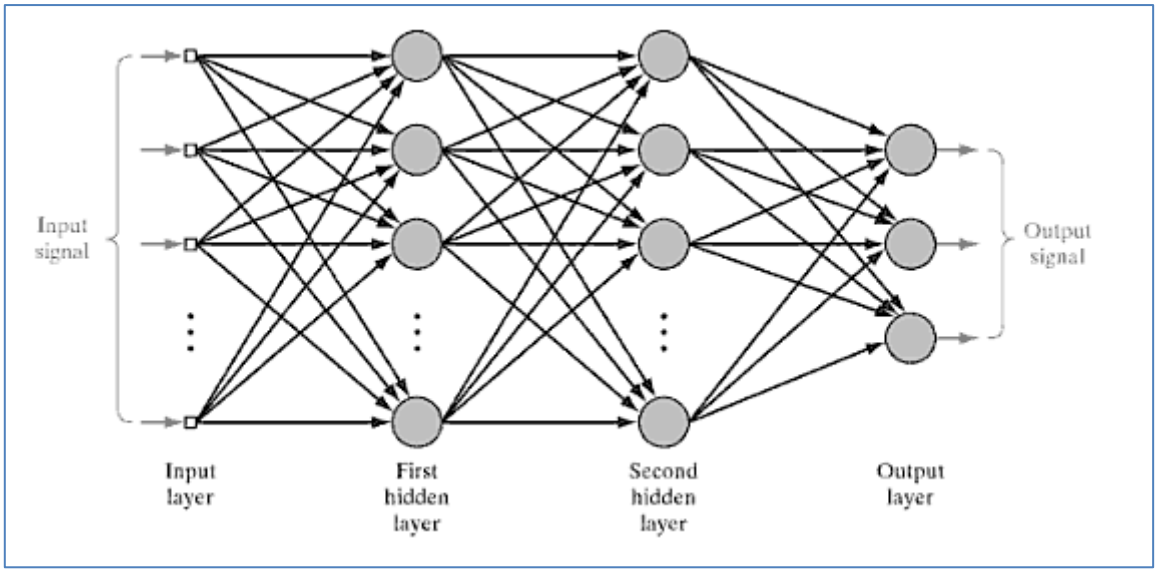


Figura 21 architettura di una MLP

Capitolo 4

Il Deep Learning

In questo capitolo sono affrontati due modelli di rappresentazione strutturata dell'immagine molto diffusi: il Bag of Visual Words e il Deep Learning. Dopo una descrizione breve del primo modello, il discorso si concentra sulla architettura di CNN's. Le Convolutional Neural Networks rappresentano uno strumento di Deep Learning molto orientato alla visione artificiale.

4.1 Modelli di Rappresentazione

Nel capitolo 3 abbiamo analizzato le principali tipologie di classificatori, con particolare riferimento alla reti neurali artificiali. Il processo di classificazione di immagini, o più in generale di dati, è costituito da più operazioni, di cui il classificatore ne implementa solo una parte.

Occorre definire modelli di rappresentazione e elaborazione dell'analisi in modo organizzato, affinché una immagine sia classificata nella rispettiva categoria. Questo problema risulta molto interessante e complesso da affrontare ed ha prodotto la definizione di molti modelli strutturati.

In questo capitolo si parlerà di due modelli in particolare: il Bag of Visual Words e il Deep Learning. Si tratta di due metodologie di Machine Learning molto diffuse: la prima è una versione modificata del modello Bag of Words [8], nato per la rappresentazione automatizzata di libri testuali, la seconda è stata ispirata dal funzionamento del cervello dei mammiferi e quindi ha caratteristiche simili a quelli descritti quando abbiamo parlato delle ANN's.

Il Deep Learning si è imposto come lo stato dell'arte in computer vision, per i risultati sorprendenti che riesce a dare nei problemi di categorizzazione degli oggetti [16].

4.1.1 Il Bag of Visual Words

Il *Bag of Visual Words* è un modello di rappresentazione dell'immagine che costituisce la naturale estensione di un altro: il *Bag of Words*. In sostanza nel BoW si considerano tutte le parole principali contenute in un libro, o più in generale in un documento scritto e le si memorizzano in un dizionario, costruito attraverso una fase di training. Un libro è rappresentato come un vettore, le cui componenti indicano la frequenza di ogni parola del dizionario, all'interno del testo di partenza.

Senza considerare la posizione e l'ordine delle singole parole nel documento, ma basandosi solo sulla frequenza con cui le parole si ripetono in esso, il Bag of Words ha la capacità di classificarne la tipologia.

Il Bag of Visual Words, rientra nella categoria di modelli di rappresentazione *Part Based* [8].

L'idea che sta alla base di tali modelli è quella di identificare frammenti di immagini o parti di esso che hanno delle analogie, ad esempio le ruote di una macchina, la posizione degli occhi di una persona. Un oggetto di una immagine è visto come un numero di parti, ciascuna delle quali possiede delle proprietà, quali ad esempio aspetto o grandezza relativa.

Una volta che regioni precise dell'oggetto sono state selezionate, occorre scegliere un metodo per rappresentare la relazione spaziale che hanno le parti dell'oggetto. Il Bag of Visual Words ad esempio, non memorizza nessuna relazione geometrica tra queste parti.

4.1.1.1 Descrizione del Modello

Anziché utilizzare parole, questo modello utilizza *Visual Words* ossia punti di interesse, *key points*, estratti da una immagine. Ad ogni immagine è associato un vettore, i cui campi indicano la frequenza delle visual words al suo interno. Questa rappresentazione non tiene conto dei concetti di posizione e ordinamento.

Attraverso una fase di apprendimento su un insieme di immagini di cui è nota la categoria, si costruisce un vocabolario di visual words.

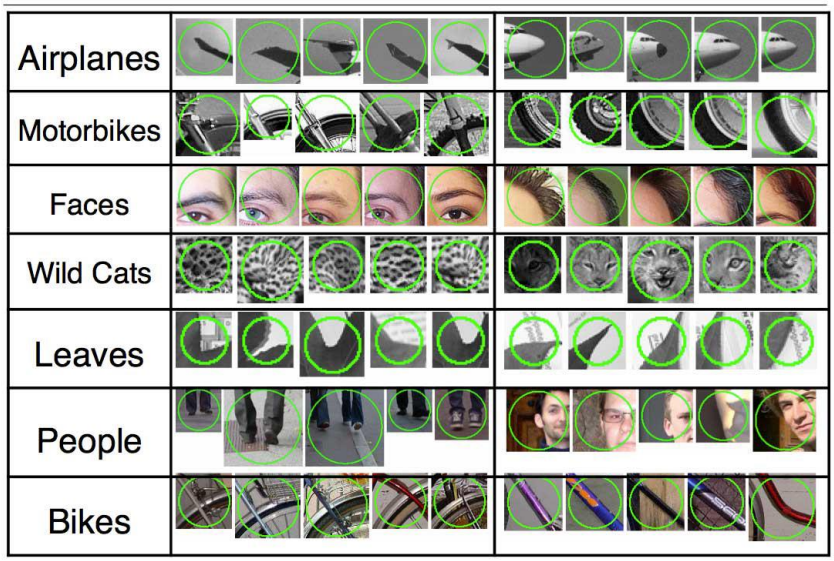


Figura 22 esempio di dizionario visuale

Per classificare una nuova immagine basta generare il vettore di frequenza delle visual words rispetto allo stesso vocabolario e osservare la sua collocazione nello spazio delle categorie. Di seguito possiamo vedere come si articolano le due fasi di Train e di Test:

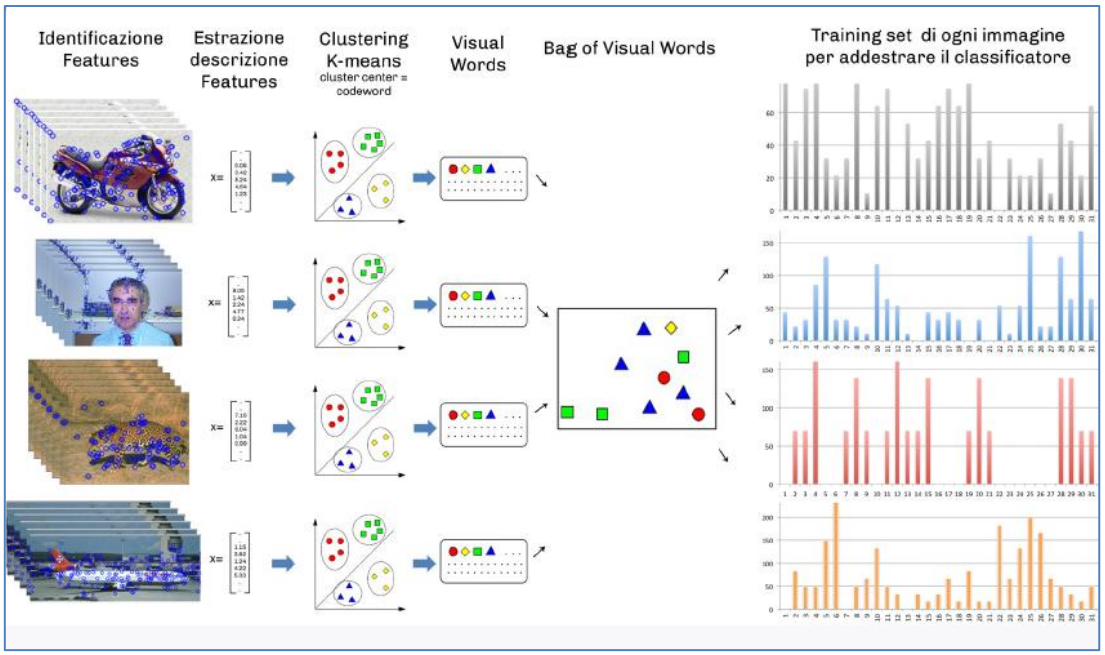


Figura 23: Fase di training in Bag of Visual Words [8]

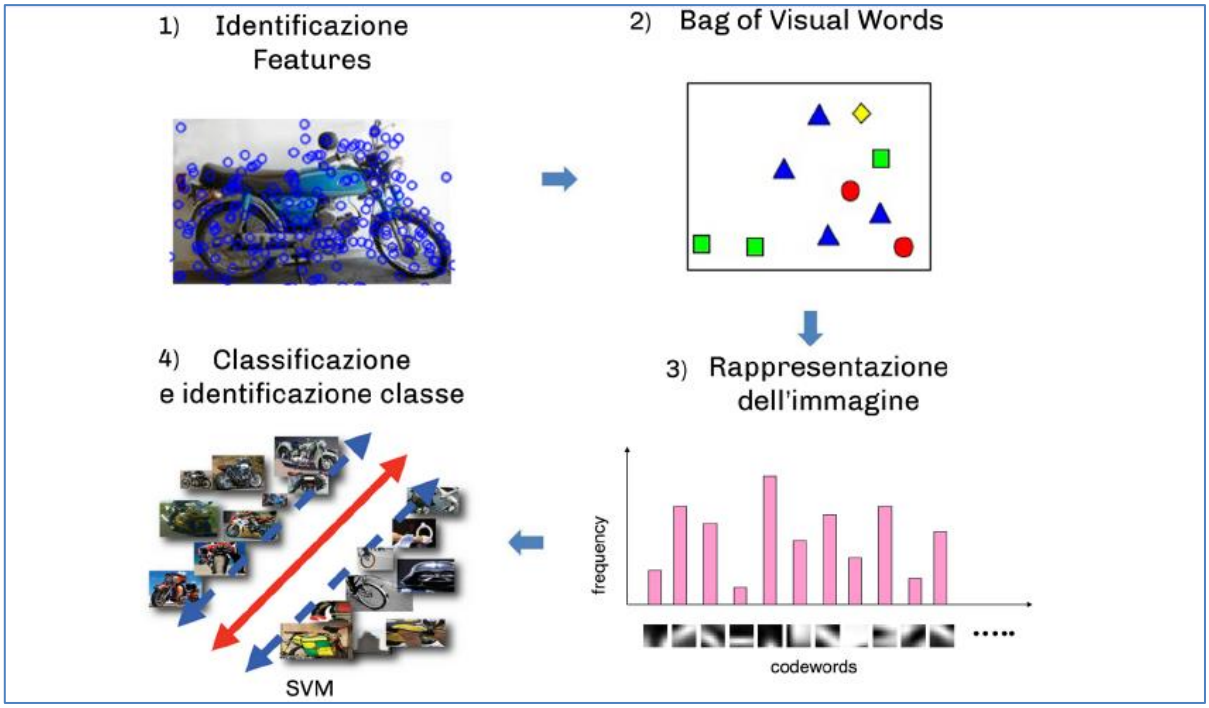


Figura 24: Fase di testing in Bag of Visual Words [8]

Per costruire il dizionario visuale si adottano *keypoints detectors* in grado di selezionare aree specifiche dell'immagine.

Terminata questa fase, occorre descrivere i keypoints in maniere robusta, attraverso dei *feature descriptors*. Le immagini vengono rappresentate come una collezione non ordinata di features descriptors, in cui non c'è relazione tra esse. Una volta estratti i keypoints dalle immagini di training, le informazioni ottenute sono raggruppate attraverso *algoritmi di clustering* [8] che compattano lo spazio delle feautures.

Per classificare una nuova si genera un istogramma, che rappresenta il numero di occorrenze di un particolare pattern estratto da una immagine e basandosi su quello si classifica l'oggetto, attraverso uno dei classificatori introdotti nel capitolo 3. Uno dei classificatori più utilizzati e accurati è il *classificatore SVM* [8].

4.1.2 Il Deep Learning

Il Deep Learning è un'area del Machine Learning, che ha lo scopo di avvicinare l'apprendimento automatico al suo modello originale: l'intelligenza artificiale. Si basa sul modo in cui il cervello dei mammiferi processa le informazioni e impara, rispondendo agli stimoli esterni.

Nel caso del modello Deep Learning, il grande passo avanti è stato quello di creare un modello di apprendimento automatico a più livelli di rappresentazione dell'immagine, per il quale i livelli più profondi prendono in input le uscite dei livelli precedenti, trasformandoli e astraendoli sempre di più [17].

Ogni livello corrisponde in questo modello ipotetico a una diversa area della corteccia cerebrale. Ad esempio, nel momento in cui il cervello riceve in ingresso delle immagini le elabora tramite diverse fasi come il rilevamento dei bordi, la percezione delle forme, da quelle primitive a quelle gradualmente sempre più complesse.

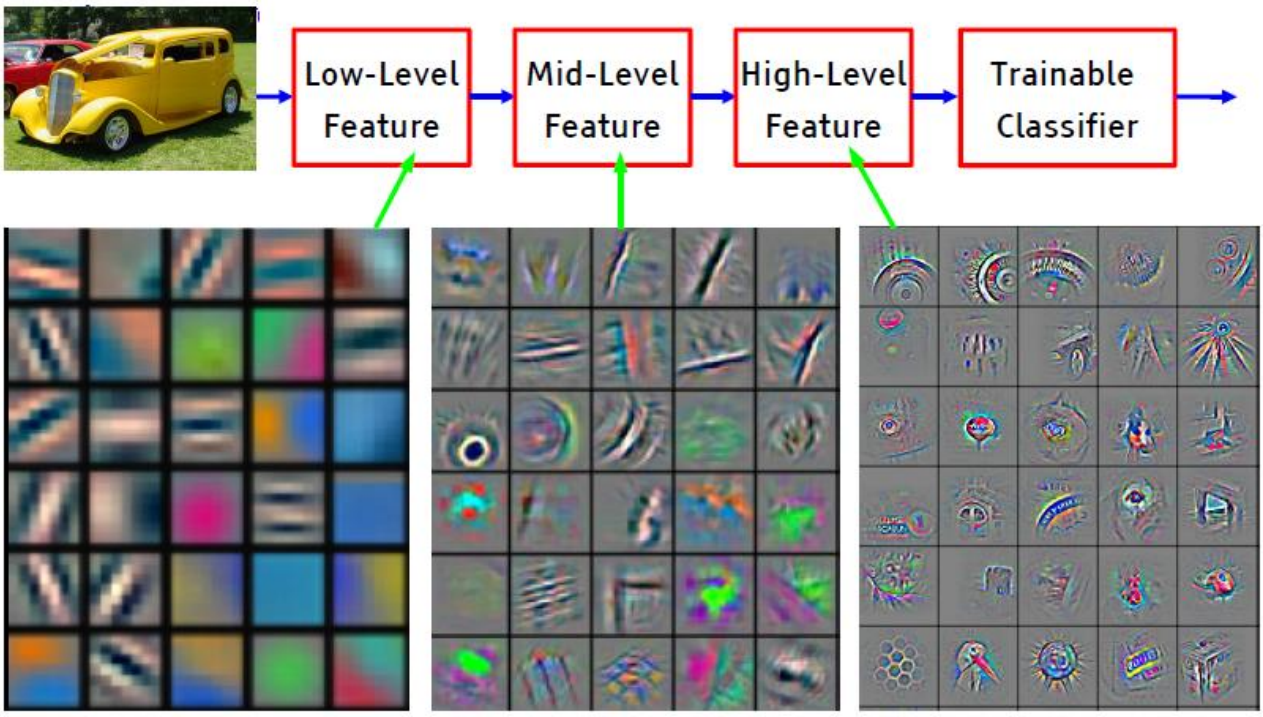


Figura 25: schema a blocchi Deep Learning [17]

Lo schema mostra ciò che è stato detto nel caso di image classification: ogni blocco estrae progressivamente le features, a vari livelli di astrazione, prendendo in ingresso dei dati già

elaborati, mediante operazioni di filtraggio, e li elabora ulteriormente, estraendo caratteristiche dell'immagine sempre più astratte: si parla appunto di rappresentazione gerarchica dell'immagine a livello di astrazione crescente. Una possibile progressione di features extraction nel caso di object categorization è la seguente:

Pixel → Edge → Texton → Motif →Part → Object

Le prestazioni e la precisione del riconoscimento possono variare molto secondo il numero delle connessioni presenti tra i vari strati di questa architettura.
Ogni stadio del diagramma precedente è un blocco di estrazione di features allenabile e progressivamente modificabile. Il cervello apprende per tentativi e genera nuovi neuroni apprendendo dall'esperienza. Anche nel caso di Deep Learning, gli stadi di estrazione si modificheranno, in base alle immagini in ingresso.
Nel caso di un modello classico, quale ad esempio il Bag of Visual Words, l'approccio è diverso e non si tiene conto del concetto di profondità in modo così vincolante:

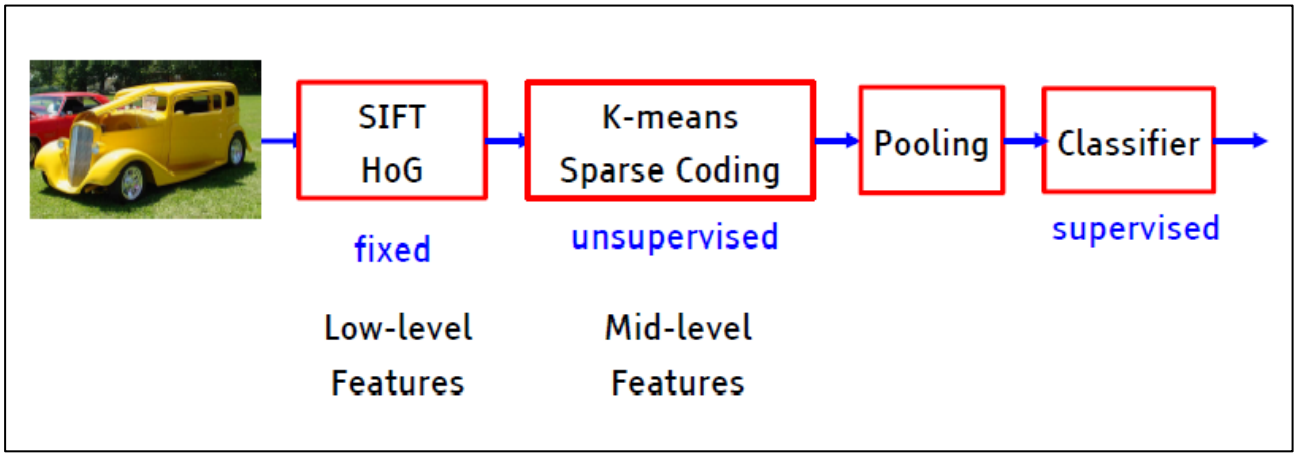


Figura 26: schema a blocchi Modello standard [17]

4.2 Le Deep Neural Networks

Le *Deep Neural Networks (DNN's)* [18] sono reti neurali artificiali fortemente orientate al Deep Learning.
Dove normali procedimenti di analisi sono inapplicabili a causa delle complessità dei dati da elaborare, tali reti sono un ottimo strumento. La loro architettura profonda, *deep architecture*,

è particolarmente adatta alla risoluzione di problemi di visione artificiale. In questo capitolo ci siamo concentrati su una particolare tipologia di deep neural networks: le *CNN, Convolutional Neural Networks*. Queste tipologie di reti rappresentano una estensione concettuale delle MLP, introdotte nel capitolo precedente.

E' importante sottolineare come i classificatori che abbiamo introdotto nel precedente capitolo rappresentano una parte del processo di image classification, che è organizzato in base al più ampio modello di rappresentazione (i.g. Bag of Visual Words, Deep Learning).

Con ciò si vuole dire che le deep neural networks sono reti neurali, quindi presentano molte analogie con quelle di cui abbiamo discusso, ma servono per implementare un modello più complesso, fatto di features extraction, classificazione e non solo di classificazione.

Esse seguono i principi di apprendimento validi per tutti i problemi di Machine Learning (i.g. supervised learning)

Per come sono costruite, le deep neural networks lavorano in parallelo, quindi sono capaci di trattare molti dati. Si tratta di un sofisticato sistema di tipo statistico, dotato di una buona immunità al rumore.

A differenza di sistemi algoritmici dove si può esaminare la generazione dell'output passo dopo passo, nelle reti neurali si possono avere anche risultati molto attendibili ma talvolta senza la possibilità di capire bene la motivazione di tali risulatati.

Non sono presenti teoremi per la generazione di reti neurali ottime: la probabilità di ottenere una buona rete sta tutta nelle mani di chi la crea, il quale deve avere dimestichezza con concetti di statistica e particolare attenzione deve dare alla scelta della variabili predittive.

Infine osserviamo che, per essere produttive, le DNN's necessitano di un addestramento che sintonizzi in modo corretto i pesi sinaptici. L'addestramento può richiedere molto tempo se i dati da esaminare e le variabili in gioco sono elevate, come spesso accade quando si desiderano ottenere risultati ottimali.

4.2.1 Le CNN's: Convolutional Neural Networks

Tra le varie tipologie di deep neural networks, in questa tesi si ci si è concentrati nella analisi di una struttura particolarmente rivolta a problemi di image classification: le *Convolutional Neural Networks.* Questo tipo di rete neurale può raggiungere livelli di accuratezza nella

categorizzazione di oggetti molto elevata, su dataset che contengono milioni di immagini e migliaia di classi [1].

4.2.1.1 Architettura di una CNN

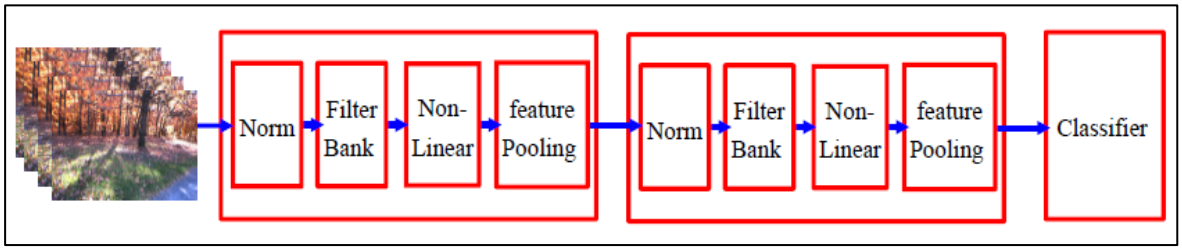


Figura 27: schema di architettura per CNN [17]

Una convolutional neural network è organizzata come in figura: si possono distinguere diversi strati, *convolutional layers*, che fungono da features extractors, e un classificatore finale. Nello schema a blocchi mostrato in figura, gli strati sono due e ognuno implementa quattro operazioni fondamentali:

- Normalizzazione dei dati in input
- Operazioni di *filtering* e *subsampling*, per l'estrazione delle features dalle immagini o dagli output di ogni layer
- Applicazione di una *funzione non lineare* per rafforzare le caratteristiche salienti e indebolire quelle meno importanti.
- *Pooling* delle caratteristiche estratte al fine di rendere più compatto lo spazio delle features che ogni layer deve analizzare.

Se ricordiamo l'architettura di una rete MultiLayer Perceptron, possiamo osservare che le operazioni di filtering e l'applicazione di operatori non lineari ai dati, realizzano la stessa funzione di elaborazione dei neuroni artificiali.
In base alla connessione dei neuroni, si potranno distinguere CNN

- fully connected
- locally connected.

Nelle primo caso, ogni neurone acquisisce informazioni su tutta l'immagine, nel secondo caso, un neurone processa solo alcune parti dell'immagine. La architettura fully connected è

sicuramente più complessa rispetto alla seconda ed è anche quella meno utilizzata. Diversi esperimenti infatti hanno dimostrato [19] che si preferisce la ricchezza di features alla densità delle connessioni. Se i filtri che agiscono sono completamente connessi si peggiorerà la specificità del loro lavoro sulla immagine. Inoltre la alta densità di connessioni è un motivo di maggiore tempo di esecuzione per la rete neurale.

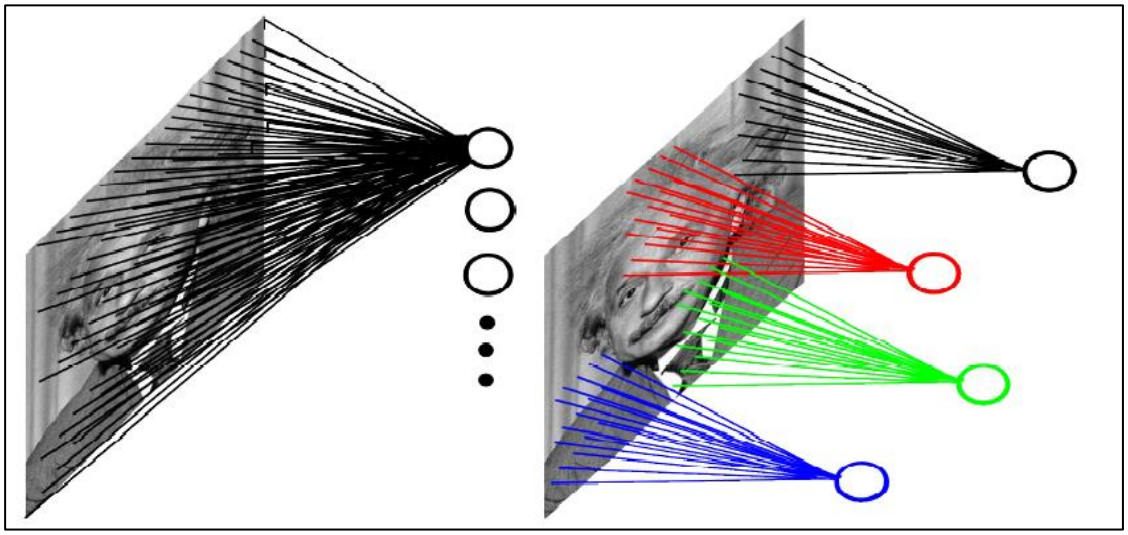


Figura 28: Reti fully connected e local connected [17]

Il passo di elaborazione che una CNN svolge nel primo strato, dopo aver normalizzato l'immagine in ingresso, è quello di una operazione di *convoluzione* tra l'immagine e un certo numero di filtri appartenenti ad un *bank filter*. Tramite la convoluzione tra immagine e filtri del banco, si ottiene in uscita il riconoscimento di determinate caratteristiche salienti. Con l'aumento della profondità della rete, i filtri passeranno dall'essere fortemente locali al diventare globali. Le prime operazioni di filtraggio infatti cercano di estrarre feautures da piccole porzioni di spazio, fino a diventare, negli strati più profondi, sensibili ad una regione più ampia degli input.

I filtri sono maschere, matrici di dimensione tipicamente 3x3, 5x5, 7x7, che contengono valori in grado di "pesare" i pixel in maniera diversa, a seconda dell'operazione che il filtro stesso svolge. Tali pesi sono all'inizio scelti casualmente, e sono poi migliorati ad ogni iterazione mediante l'algoritmo di back propagation, accennato nel capitolo 3. Così facendo la rete addestra i suoi filtri ad estrarre le features più significative degli esempi del training set; cambiando training set i valori dei filtri saranno diversi. Il suddetto addestramento accade per i filtri di ogni strato della rete.

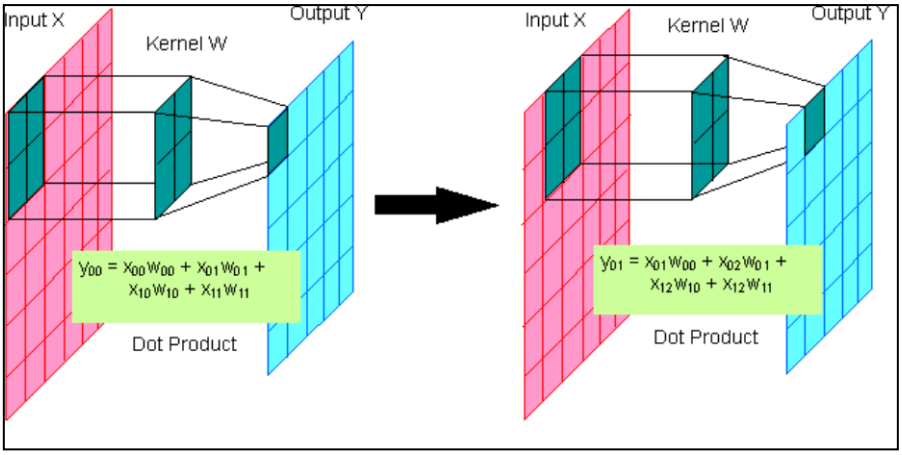


Figura 29: esempio di convoluzione con filtri

La convoluzione non avviene solo con i filtri del banco, ma anche con la funzione non lineare. Tale funzione, tipicamente *Relu, Rectified Linear Unit* [13], serve a rettificare i risultati ottenuti, mettendo in evidenza quelli salienti e trascurando gli altri. Il primo strato di convoluzione si occupa di estrarre features direttamente dai pixel dell'immagine e li memorizza nelle feature maps, che sono le mappe ottenute dopo la convoluzione con i filtri e l'applicazione della non linearità. Questo output diverrà poi l'input di un successivo livello di convoluzione il quale andrà a fare una seconda estrazione delle caratteristiche. Prima di passare al layer successivo, lo strato precedente si conclude con l'applicazione di un determinato numero di filtri di pooling. Questa operazione è molto importante e ha lo scopo di ridurre lo spazio delle feautures che il livello successivo dovrà elaborare. Il pooling, nella versione utilizzata spesso per le CNN ripartisce l'immagine o la feature map in un set di rettangoli non sovrapposti e, per ogni regione ottenuta, fornisce in output il valore massimo.

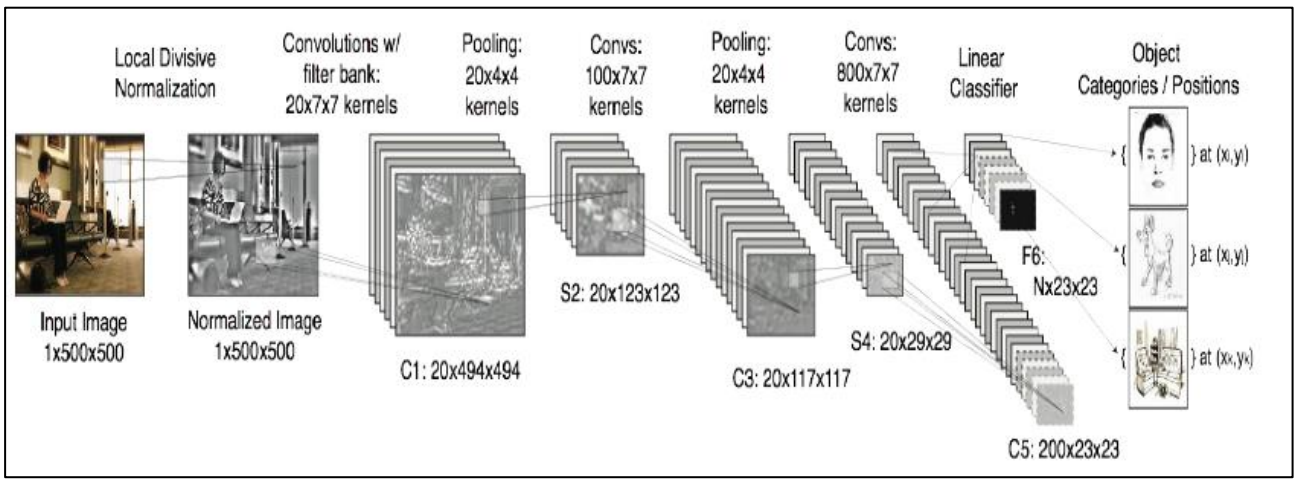


Figura 30: path di elaborazione di una CNN [17]

Il numero di filtri per banco e quello dei filtri di pooling è una scelta progettuale e rientra tra le caratteristiche di una CNN come blackbox. Se ci riferiamo alle dimensioni degli output di ogni operazione per layer, possiamo notare come le dimensioni delle features map cambino rispetto all'immagine di partenza. Per quanto riguarda gli output delle operazioni di pooling è naturale che siano ridotte, visto che il pooling serve proprio per ottimizzare lo spazio delle features; per quanto riguarda le operazioni di filtering, le dimensioni cambiano perché le CNN sono progettate in modo tale da trascurare alcune zone dell'immagine. Un esempio di come agisce una CNN a due layer è rappresentato nello schema seguente:

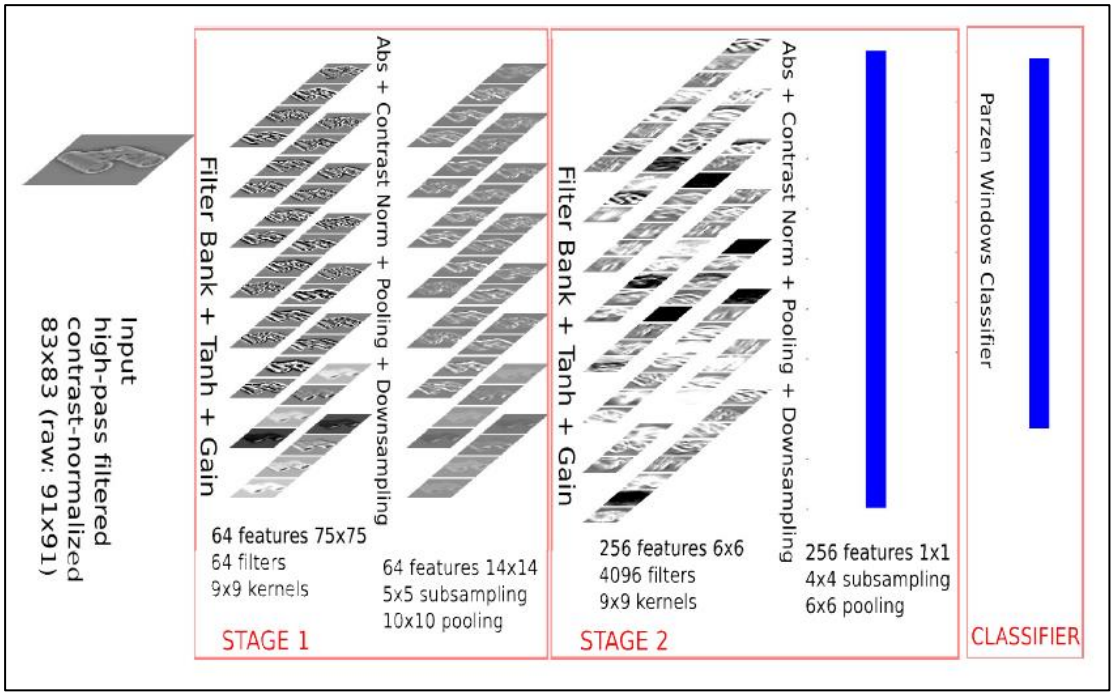
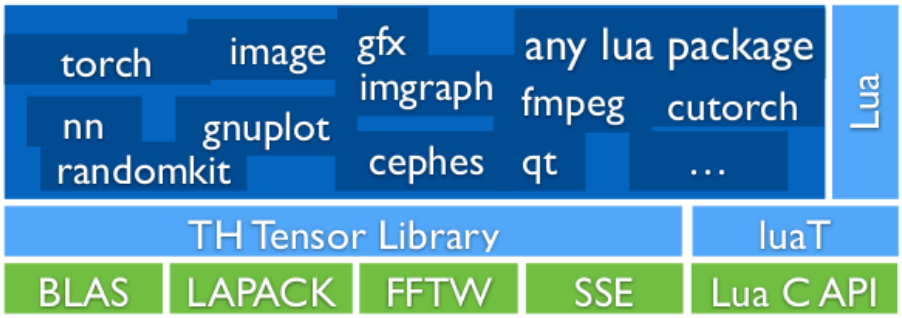


Figura 31: esempio di come lavora una CNN

Notiamo come gli operatori che vengono applicati agli input per ogni layer rientrano in una delle categorie di operazioni elencate all'inizio: filtering, subsampling, non linearità e pooling. L'operazione finale di classificazione può avvenire scegliendo uno tra i classificatori discussi nel capitolo 3. Quelli più utilizzati sono i classificatori matematici: Reti Neurali (i.g. MultiLayer Perceptron) e SVM.

4.3 Il Framework Torch7

In questa tesi l'image classification è stata realizzata attraverso il framework di calcolo Torch7 [28]. Torch è una libreria che estende il linguaggio di scripting Lua [20] e ha l'obiettivo di fornire un ambiente flessibile per la progettazione e l'addestramento di sistemi di Machine Learning. Torch è un framework auto-contenuto e estremamente portabile su ogni piattaforma (i.g. Windows, Mac, Linux, Android) e gli script realizzati riescono ad essere eseguiti su tali piattaforme senza alcuna modifica. Torch fornisce molti package utili per diverse applicazioni



Uno degli strumenti fondamentali definiti in Torch7 è il *Tensore* [28]. Si tratta di un operatore che permette una semplice gestione di vettori N-dimensionali e una efficiente gestione della memoria. Due sono i principali tipi di Tensore che Torch mette a disposizione per le operazioni matematiche: *doubleTensor e float Tensor*

```
1  floatT = torch.FloatTensor(100,100) [[crea un tensore in virgola
2  mobile a precisione singola]]
3  randomT = torch.rand(100,100) -- immette valori random
4  r = floatT + 1/2 --  operatori di base
5  r:add(0.5, floatT) -- operatori in-place
6  floatT:fill(1) --riempe il tensore di 1
7  [[ In Lua i ':' aggiunge semplicemente un primo
8     argomento alla funzione chiamato self.
9     Questo viene utilizzato per usare funzioni sull'oggetto
10    stesso e permette quindi di usare tali oggetti
11    come  classi.
12    in sostanza è come chiamare un metodo ]]
13
14 doubleT = floatT:double() [[casting a tensore a
15 dobbia precisione]]
16 r = torch.FloatTensor(doubleT:size()) --stesso size di doubleT
17 r:copy(doubleT) -- cast automatico double -> float
18
```

Figura 32: Esempio di uso di Tensori in Torch7

Questo lavoro non ha l'obiettivo di studiarne le caratteristiche e di analizzarlo a fondo. Ciò è stato già fatto da [1]. Le finalità con cui si è scelto Torch7 derivano dalla relativa semplicità che questo strumento fornisce per la realizzazione di convolutional neural networks, nonché per il suo allenamento. Si rimanda al capitolo 6 per capire come Torch7 è stato utilizzato e quali risultati sperimentali ha fornito nell'addestramento di CNN per la classificazione di immagini appartenenti a un dataset creato ad hoc per il sistema di ausilio alla mobilità per individui ipovedenti e non vedenti.

Capitolo 5

Algoritmo di segmentazione

In questo capitolo è descritto in dettaglio l'algoritmo di segmentazione delle immagini implementato durante il lavoro di tesi. Come premessa sarà inquadrato globalmente il sistema 3D di assistenza alla mobilità per ipovedenti/non vedenti, per capire il contesto in cui si inserisce tale algoritmo. Questo algoritmo è stato applicato a un dataset di immagini realizzato ad hoc per il sistema di visione per ipovedenti/non vedenti, e contiene alcuni tra i principali ostacoli che si potrebbero incontrare in un parco o più in generale in una situazione all'aperto. Oltre alla descrizione dei principali passi dell'algoritmo, sono analizzati anche i risultati di segmentazione, si evidenziano pregi difetti e possibili soluzioni migliorative dell'algoritmo.

5.1 Introduzione

L'algoritmo che è stato realizzato durante tesi, è un algoritmo di segmentazione di tipo region-based ed è stato scritto in linguaggio C++, utilizzando la piattaforma Visual Studio 2008.
Esso basa il suo funzionamento sulla nota libreria di Computer Vision OpenCV, una libreria open source che possiede più di 2500 algoritmi ottimizzati, utili nel campo dell'Image Processing e della Visione Artificiale.

5.2 Obiettivi

Lo scopo di questo algoritmo è quello di segmentare e isolare l'oggetto che occupa la scena principale di una immagine, in modo da classificare l'immagine ,attraverso gli algoritmi di Deep Learning, riferendosi a quel solo oggetto.
E' un algoritmo che prende in ingresso una immagine, acquisita mediante Stereo Camera, e elabora le informazioni contenute nella mappa di disparità, per individuare le regioni dell'immagine che sono caratterizzate dallo stesso livello di depth (profondità).
I pixel dell'immagine aventi la stessa profondità, idealmente, appartengono alla medesima zona: è possibile dunque individuare lo spazio occupato da un oggetto di interesse, che, nel nostro caso, coincide con l'oggetto più vicino alla telecamera e che costituisce un potenziale ostacolo, per segmentarlo e isolarlo dal resto della scena. Ricordiamo infatti che tutto il lavoro si concentra sulla possibilità di identificare e classificare alcuni degli ostacoli che un ipovedente può incontrare nella quotidianità.

5.3 Il Dataset

Il dataset, ossia la collezione di immagini da processare, utilizzato in questa tesi è stato costruito *ad hoc* attraverso la stereo camera di cui si è parlato nell'introduzione. E' costituito da immagini che appartengono a *8 categorie differenti*, suddivise in base all'oggetto che "domina" nella scena. Le classi di appartenenza sono:

1. Automobili (Cars)
2. Alberi (Tronchi di alberi) (Trees)
3. Pali (Pales)
4. Muri (Walls)
5. Persone (People)
6. Cestini dell'immondizia (Trash Cans)
7. Panchine (Benches)
8. Gradini (Steps)

Queste 8 classi rappresentano alcuni tra gli ostacoli principali che un individuo ipovedente può incontrare durante un qualsiasi tragitto quotidiano. Ogni classe contiene circa 1000 immagini 2D e relativa mappa di disparità calcolata dalla stereo camera.

Figura 32: Esempi di immagini appartenenti al dataset, divisi in 8 classi

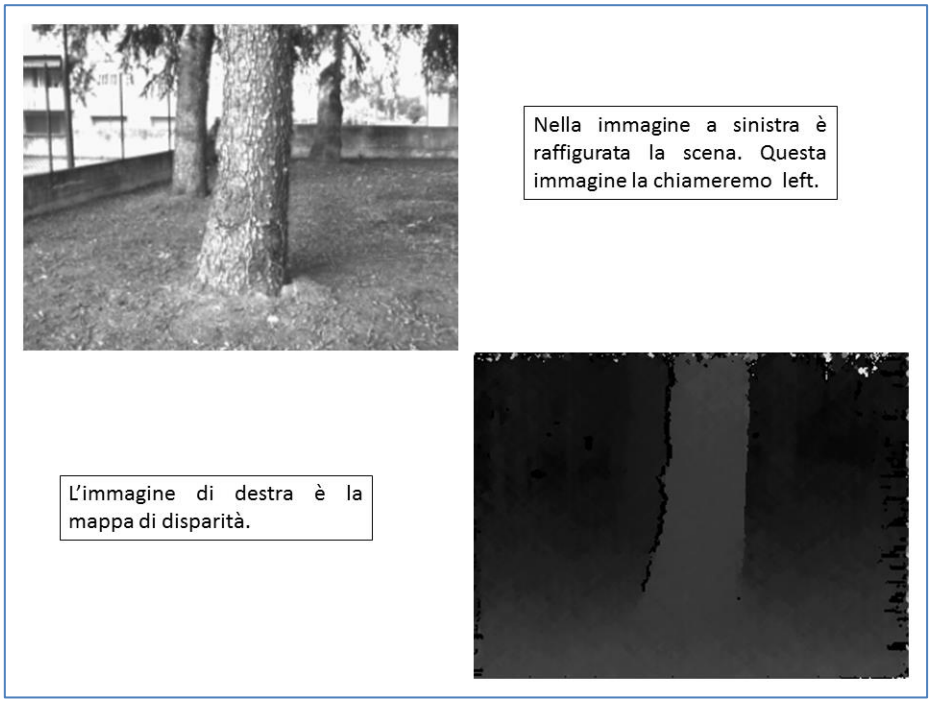


Figura 33: Immagine Left e Mappa di Disparità

Come è stato detto nel capitolo 2 parlando di visione stereo, la stereo camera fornisce come output la mappa di disparità della scena ripresa. Ogni pixel dell'immagine fotografata ha un corrispondente valore di disparità, nella mappa di disparità. Tale livello di disparità, ossia di profondità (depth) del corrispettivo punto nel mondo reale, è codificato da diversi livelli di grigio: gli oggetti più vicini alla stereo camera sono rappresentati con un tono di grigio più tendente al bianco (i.e. maggiore disparità), quelli più lontani con una tonalità più tendente al nero (i.e. minore disparità).

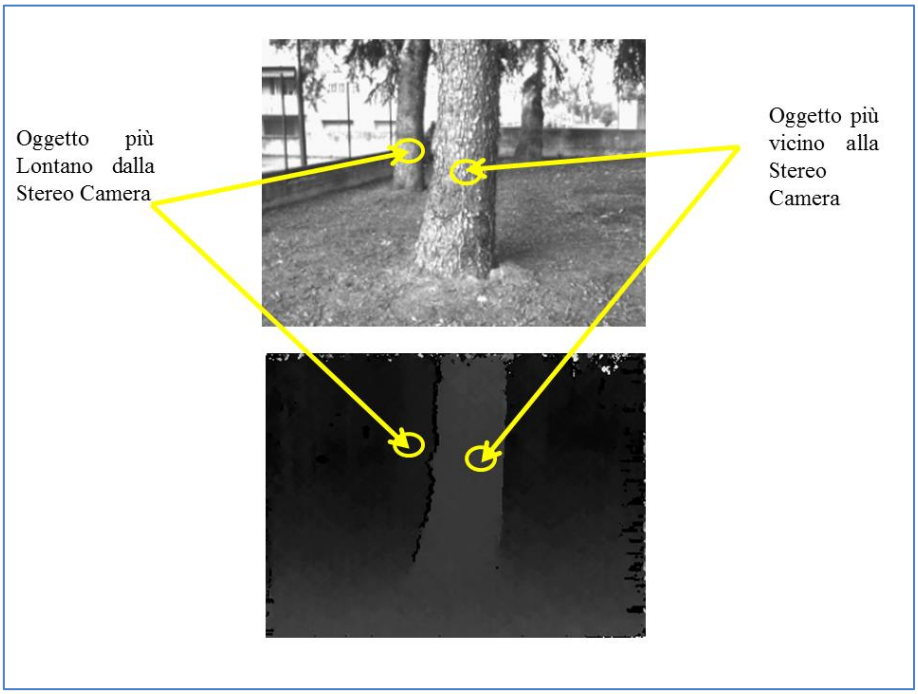


Figura 34: mappa di disparità

D'ora in poi il termine depth e disparità, nonostante siano due concetti differenti, verranno usati in modo interscambiabile, pur sottolineando che la depth è il valore di profondità di un punto nella realtà, la disparità coincide con il delta tra la posizione dello stesso punto nei piani dei due obiettivi della stereo camera.

Posto che, ogni regione uniforme coincide con un oggetto della scena, possiamo notare come la regione grigia in primo piano nella mappa di disparità della figura precedente, sia appunto il tronco dell'albero che si vede nella immagine left.

Capito questo, come possiamo sfruttare le informazioni date dalla mappa di disparità, che sono informazioni 3D sul mondo reale, per elaborare l'immagine left, che invece è una immagine 2D? E' evidente che il carico computazionale per un elaboratore, sia molto ridotto rispetto a quanto lo sarebbe se si elaborassero direttamente dati 3D (i.e. la nuvola di punti).

5.4 Descrizione dell'algoritmo di segmentazione

Una volta compresa la natura delle immagini di input, si procede con la descrizione dei principali step in cui si articola l'algoritmo di segmentazione implementato nella tesi.
Si parte naturalmente acquisendo l'immagine, che si presenta come una immagine a colori, nella quale, in ogni canale (RGB) sono salvate rispettivamente la left e la mappa di disparità:

Figura 35: immagine in input

Su questa immagine viene fatta una operazione di *unpacking*, che permette di separare l'immagine left dalla mappa di disparità. Tale operazione è realizzata applicando la libreria lib_pointcloud.lib sviluppata nell'ambito del progetto della stereo camera.

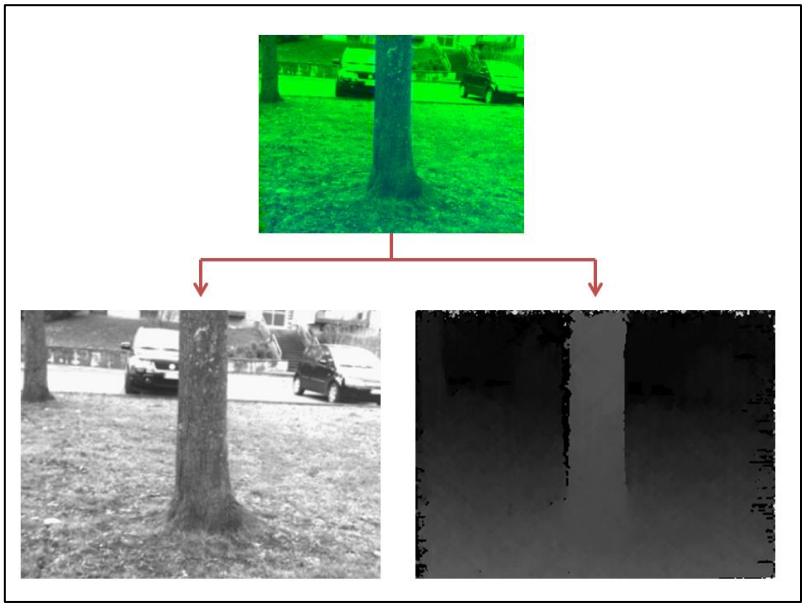


Figura 36: risultati dell'uso di lib_poincloud

La segmentazione come detto in precedenza permette di ripartire l'immagine in regioni aventi le stesse caratteristiche. In questo caso la caratteristica di ogni regione è la depth dei punti. Lo scopo dell'algoritmo è di isolare l'oggetto principale della scena, che in questo caso è il tronco in primo piano, dal resto della scena, per far si che poi l'immagine possa essere classificata in base alla presenza del solo oggetto. L'algoritmo quindi procede per esclusione progressiva delle varie parti dell'immagine, fino a giungere a una scena nella quale è presente solo l'oggetto fondamentale (e.g. l'albero della figura precedente).
In questa ottica, il passo successivo dell'algoritmo è quello di eliminare il piano dalla scena. Per fare questo è stata utilizzata la libreria lib_pointcloud_detection.lib, che permette di effettuare una detection del piano che sia il più possibile indipendente dalla posizione della telecamera e sfrutta un algoritmo real-time, basato su RANSAC [21] per individuarlo.
Ciò che si ottiene dalla operazione di plane detection è visibile in figura; il piano è la parte nera.

Figura 37: operazione di plane detection

A questo punto, ciò che resta nell'immagine di partenza è l'oggetto in primo piano e il background.

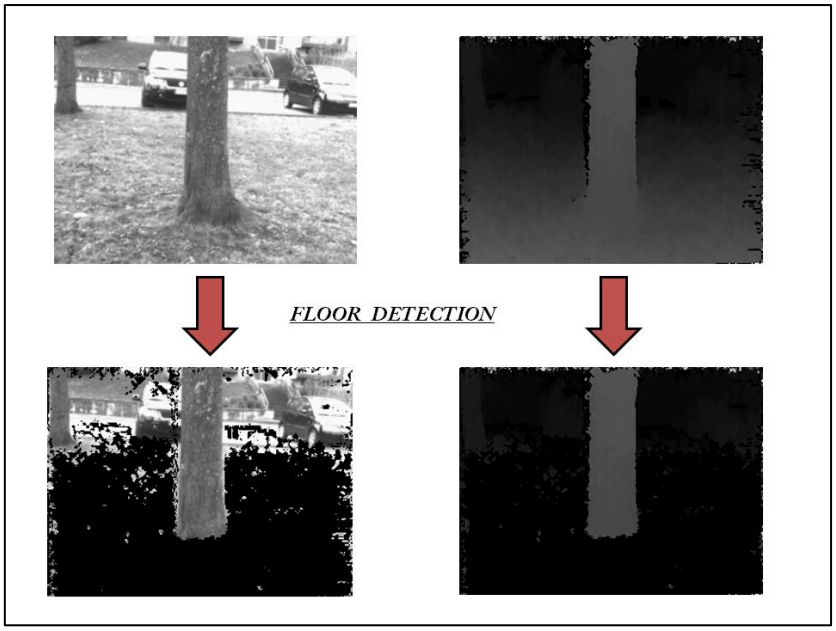


Figura 38: effetti del riconoscimento del piano

Per completare il processo di segmentazione, occorre isolare il tronco di albero dallo sfondo. Dopo aver analizzato varie funzioni della libreria OpenCV, si è optato per la scelta della funzione *cvFloodFill*.

Floodfill [22] è una funzione che agisce partendo da un pixel dell'immagine scelto come *seed point (seme)* e colora di un'unica tonalità i pixel vicini che hanno le stesse caratteristiche del punto di partenza. Per questo lavoro, la caratteristica di riferimento è la depth dei pixel.

Il risultato finale è una regione uniforme coincidente con l'oggetto della scena. Per esempio, partendo da un punto appartenente al tronco di albero in primo piano, si ottiene il seguente risultato:

Figura 39 applicazione di floodfill all'immagine

I pixel della mappa di disparità con la stessa depth sono colorati di bianco. A questo punto, con l'output ottenuto, è possibile applicarlo come maschera all'immagine left iniziale ed estrarre solo l'ostacolo individuato.
Floodfill, che letteralmente significa riempimento, funziona prendendo in ingresso i seguenti parametri:

- Una Immagine di input
- Un seed point, ossia un pixel da cui cominciare l'operazione di riempimento della regione
- Una tonalità con cui colorare la regione da riempire.
- Due parametri di soglia che coincidono rispettivamente con il valore minimo e massimo della grandezza caratteristica con cui discriminare i pixel vicini durante il riempimento. Nel nostro caso tale caratteristica è il valore dei pixel nella mappa di disparità. Quindi i parametri consistono nel livello di profondità minimo e massimo con cui scegliere i pixel.

Il seed point necessario per partire, è stato passato alla funzione selezionandolo direttamente dall'immagine, mediante click del mouse. Per il caso di studio del sistema di ausilio saranno studiate strategie adeguate per il funzionamento real-time sul campo.
Esistono due modalità con cui avviene il riempimento della regione omogenea di pixel:

- *Modalità Floating Range:* si parte dal seed point e si procede confrontando la grandezza caratteristica scelta come criterio di selezione tra i pixel immediatamente vicini, in base alla relazione matematica:

$$src(x', y') - loDiff \leq src(x, y) \leq src(x', y') + upDiff$$

  E' come se il seed point fosse continuamente aggiornato: (x',y') sono le coordinate del nuovo seed point, i parametri *loDiff* e *upDiff* sono le soglie di cui si è detto, (x,y) è il punto da colorare eventualmente. Se esso appartiene all'intervallo di depth stabilito si colora.

- *Modalità Fixed Range*: si parte dal seed point che rimane fisso, e si procede con il medesimo confronto dell'altra modalità , ma con una relazione differente, ossia

$$src(SeedPoint.x, SeedPoint.y) - loDiff \leq src(x, y) \leq src(SeedPoint.x, SeedPoint.y) + upDiff$$

In questo caso si vede che il seed non cambia e il confronto è sempre fatto con esso. Le formule precedenti si riferiscono a una immagine a livelli di grigio. Per le immagini a colori, l'operazione di confronto sarà fatta in ogni canale RGB.

L'algoritmo di segmentazione usa floodfill in modalità floating range, che è anche la modalità di default.

E' utile osservare che, affinché floodfill funzioni in modo ottimale, ci debba essere una notevole omogeneità tra i pixel vicini. Se, nel nostro caso, si ha a che fare con mappe di disparità molto rumorose, i risultati non saranno ottimali, mentre se si ha a che fare con mappe di disparità poco rumorose, si otterranno risultati soddisfacenti. Questa problematica emergerà molto nella parte in cui si discutono i risultati dell'algoritmo.

Prima di scegliere floodfill definitivamente, è stata analizzata anche un'altra funzione messa a disposizione da OpenCV, come alternativa alla prima: *Watershed* [22].

Watershed è una funzione che tratta l' immagine in input, come una superficie topografica: le zone a alta intensità sono paragonabili a picchi e quelle a bassa intensità a valli.

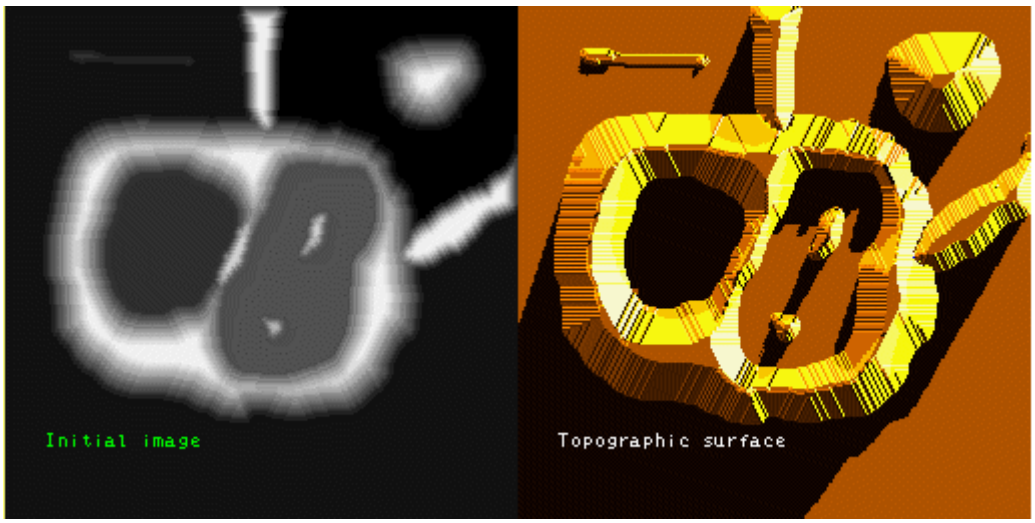


Figura 40: Fasi di filling di una immagine tramite Watershed [23]

Se si cominciano a riempire le zone di minimo (valli) con la stessa tonalità, l'immagine sarà progressivamente divisa in bacini idrografici e linee spartiacque (watershed).

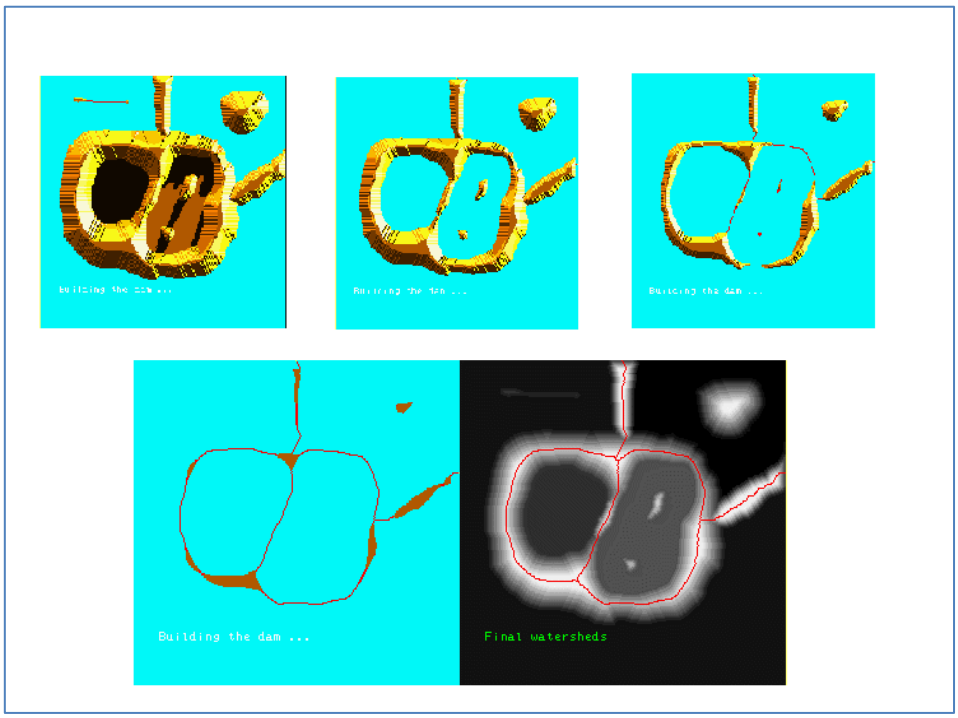


Figura 41: Fasi di filling di una immagine tramite Watershed [23]

In questo modo l'immagine risulterà segmentata in diverse regioni, che corrispondono a diversi bacini idrografici; tali zone saranno delimitate da picchi, ossia da punti in cui c'è un brusco cambio del *gradiente* dell'intensità. I picchi sono individuati applicando ad esempio il metodo del gradiente alla immagine di partenza e creando così una maschera di riferimento per la funzione watershed.

Attraverso un meccanismo di segmentazione interattiva inoltre, è possibile poi stabilire dei *Markers* con cui decidere quale delle regioni colorare di una certa tonalità, riuscendo cosi a distinguere i vari soggetti dell'immagine di partenza.

Questo approccio, benché apparentemente più adatto alla segmentazione di una immagine, si è dimostrato, nel caso di questo lavoro, poco efficace, per due motivazioni fondamentali:

- Il nostro obiettivo è stato quello di isolare un singolo oggetto dal contesto dell'immagine, non associare labels ai soggetti che la compongono. Molte operazioni di Watershed sarebbero quindi state non necessarie, appesantendo il carico computazionale..

- Nel caso di mappe di disparità mediamente rumorose, si hanno molti punti in cui le tonalità di grigio cambiano significativamente; ciò genera delle maschere molto confuse e una eccessiva frammentazione di uno stesso soggetto, che risulterebbe quindi colorato in varie tonalità.

Torniamo all'algoritmo: fino a questo punto abbiamo ottenuto una immagine priva del piano e con l'oggetto in primo piano colorato di bianco.

A partire da questa immagine, non resta che crearne un'altra a due livello di grigio, bianco e nero, nella quale sia presente solo il *blob* bianco corrispondente al tronco e effettuare una operazione di *and logico* tra questa e l'immagine Left di partenza, dove è raffigurata la scena ripresa dalla telecamera. Il risultato finale è il seguente:

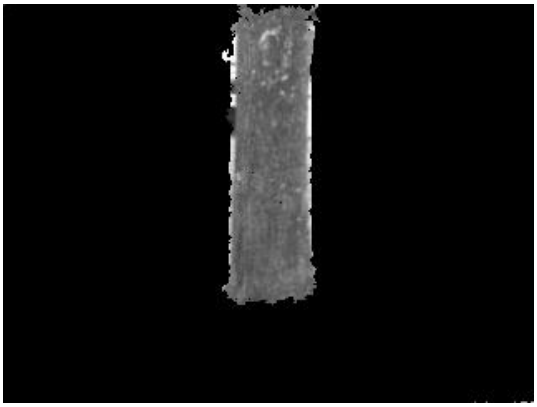


Figura 22: segmentazione finale

Come possiamo osservare dalla figura, si è riusciti ad ottenere una immagine nella quale c'è solo il tronco che noi volevamo individuare all'inizio. Si tratta di una immagine sicuramente meno ricca di particolari rispetto a quella di partenza e ciò rende auspicabile una migliore operazione di classificazione da parte del framework di Deep Learning, torch7.

Nella pagina seguente sono riportati anche altri due risultati, applicati a immagini di Persone e Automobili.

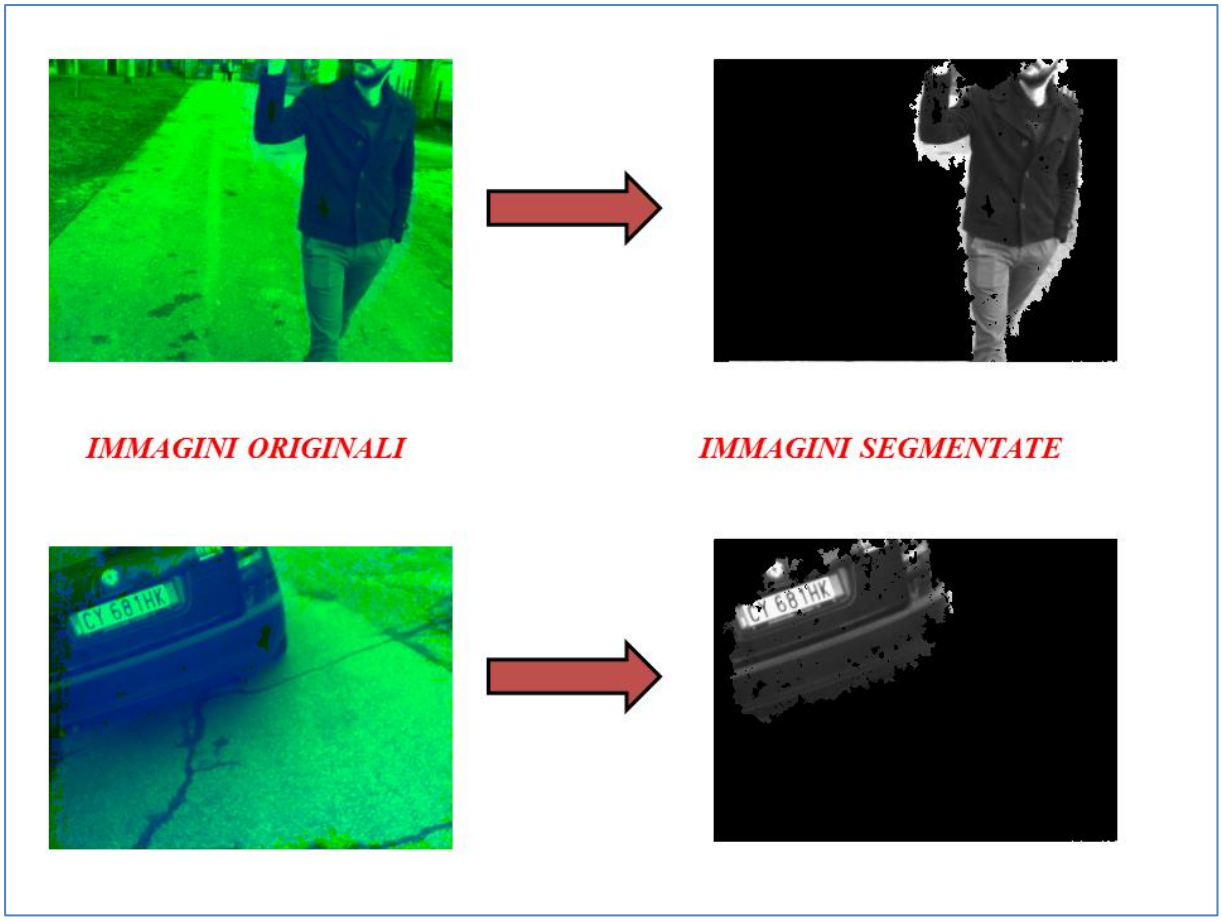


Figura 43: Risultati dell'algoritmo di segmentazione

5.5 Analisi dei risultati

Di seguito alcuni tra gli esempi migliori e peggiori ottenuti durante la segmentazione del dataset, a dimostrazione delle osservazioni fatte subito dopo:

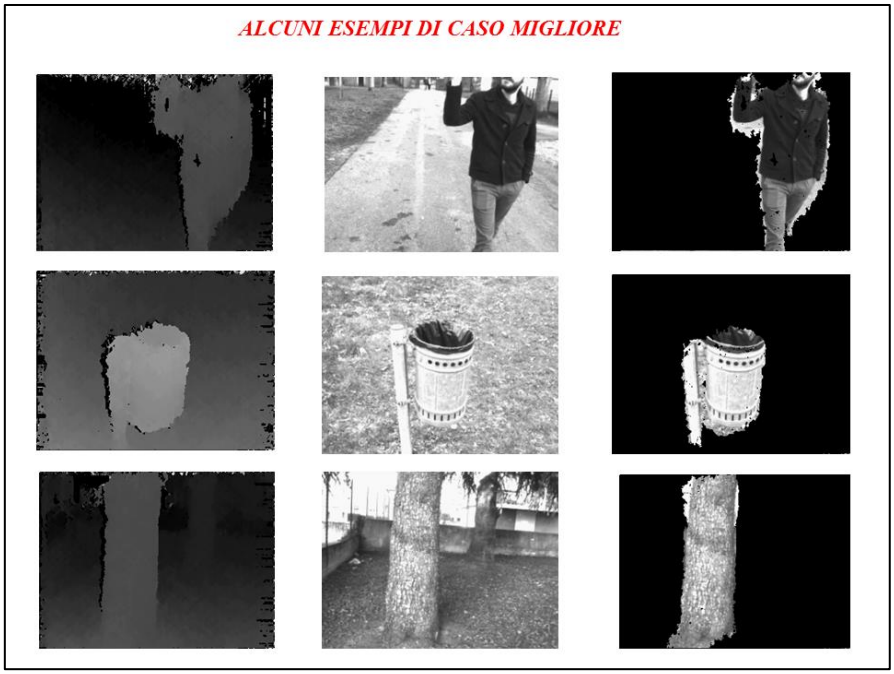


Figura 44: Esempi di risultati soddisfacenti

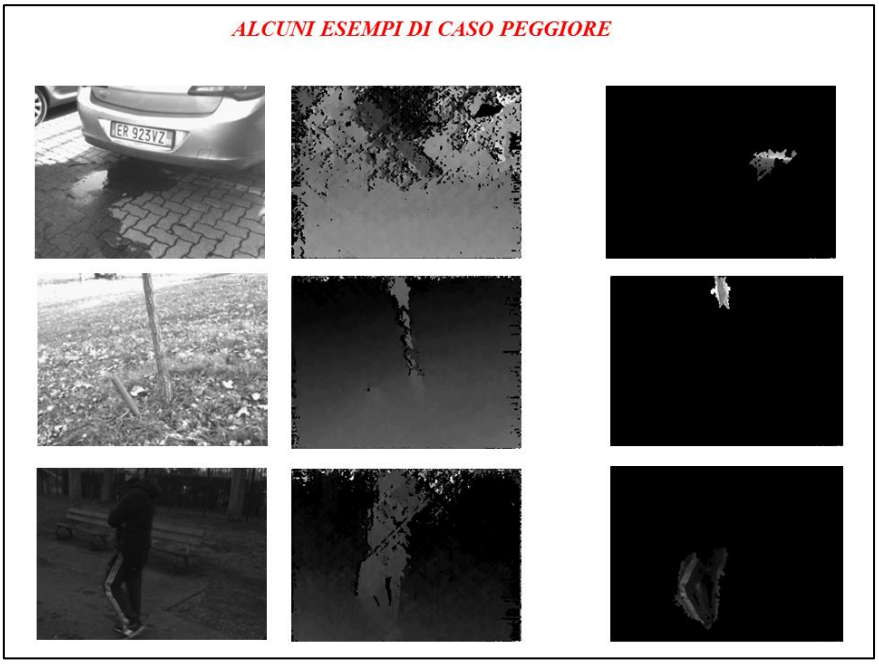


Figura 45: Esempi di risultati non soddisfacenti

Analizzando i risultati ottenuti si può dire che l'algoritmo raggiunge globalmente l'obiettivo per cui è stato realizzato. La percentuale di risultati positivi, per ogni classe, è superiore all'80%.

La causa principale di fallimento è che questo algoritmo è stato implementato basandosi molto sulle caratteristiche della mappa di disparità: a mappe di disparità molto rumorose corrispondono risultati poco soddisfacenti, a mappe di disparità nitide corrispondono risultati molto soddisfacenti. Analizzando i risultati sopra riportati si può notare che, nel caso in cui la disparità è omogena, l'oggetto è stato segmentato correttamente; quando invece la disparità presenta una forti discontinuità, si riesce a segmentare solo una porzione dell'oggetto.

La mappa di disparità riflette le condizioni fisiche in cui la scena è ripresa, e essendo in un contesto molto naturale gli accorgimenti che si possono prendere per migliorarla sono pochi. In uno scenario all'aperto, la luce , i riflessi degli oggetti ripresi, le condizioni atmosferiche, sono poco controllabili, nonostante la robustezza degli algoritmi di acquisizione montati nella stereo camera. E' stata questa la principale causa che non ha permesso di ottenere, in alcuni casi, risultati di segmentazione ottimali.

Nonostante ciò, anche nelle condizioni peggiori, l'algoritmo ha dimostrato di segmentare con un livello di accuratezza sufficiente per la successiva operazione di image classification, come dimostreranno i risultati illustrati nel capitolo 6.

5.6 Possibili strategie di miglioramento dell'algoritmo

Le strade per migliorare la segmentazione di immagini, sempre rispettando quello che è l'obiettivo iniziale, ossia l'isolamento dell'oggetto più vicino alla Stereo Camera, possono essere due:

- Si possono applicare operazioni successive di Dilatazione e Erosione all'output della funzione cvfloodfill, in modo da eliminare quasi completamente eventuali *buchi* presenti nei soggetti, dovute alla rumorosità della mappa di disparità e implementare una operazione di *bounding box*, per aumentare i bordi delle figure.
- Si potrebbe utilizzare una funzione, progettata da Alessandro Muscoloni in una precedente tesi [24], che proietta nel piano il soggetto in 3D, ottenendo una immagine completamente nera tranne che per lo spazio occupato dal soggetto stesso. Dopodiché si potrebbe applicare cvfloodfill all'output ottenuto, riconoscere il blob nel piano e

successivamente riproiettare nello spazio solo i punti che appartengono al blob, che sarà costituito da pixel con la stessa tonalità.

Capitolo 6

Risultati sperimentali con Torch7

In questo capitolo sono presentati e discussi i risultati sperimentali ottenuti con il Framework Torch7 applicato al dataset di immagini descritto nel capitolo 5.

6.1 Introduzione

L'obiettivo di questo lavoro è stato quello di capire quali siano gli effetti della operazione di segmentazione sulla categorizzazione di oggetti presenti in una immagine. In questo lavoro si è tentato di sfruttare le informazioni 3D che la visione stereo permette di acquisire, per ridurre il numero di dati in ingresso ad un metodo di classificazione basato su Deep Learning. L'operazione di segmentazione infatti, permette di isolare gli oggetti di nostro interesse dalla scena ripresa e di circoscrivere le regioni dell'immagine in cui gli algoritmi di image classification devono agire per estrarre features. Ciò può ridurre il tempo e il costo di computazione degli algoritmi, aumentando in molti casi l'accuratezza della classificazione.
Torch7 carica tutti i dati in memoria RAM; ciò implica il bisogno, per l'addestramento di una CNN, di elaboratori che abbiano a disposizione minimo 4G di RAM. Si è tentato inizialmente di eseguire il training/testing della rete in una macchina virtuale (*Vmware Player*) con Ubuntu 14.04, 2 GB di Ram e un processore single core. La VM era installata in un laptop dotato di OS Windows 8.1, con CPU Intel Core i3 a 1.40 GHz. Le simulazioni su questa piattaforma hanno portato sistematicamente ad un blocco del sistema, dovuto alla dimensione insufficiente di memoria a disposizione. I risultati quindi sono stati ottenuti usando un PC con CPU Intel Core i3 a 2.30 GHz con OS Ubuntu 14.04 nativo e 4GB di Ram.

6.2 Descrizione del procedimento

Descriviamo in breve gli strumenti che sono stati utilizzati nel processo di classificazione delle immagini. I passi che sono stati seguiti nella fase sperimentale di questo lavoro sono:

1. Realizzazione del dataset
2. Segmentazione delle immagini appartenenti al dataset
3. Analisi degli script necessari al corretto training e test di una CNN implementata mediante l'uso di Torch7
4. Esecuzione degli script
5. Raccolta e analisi dei dati sperimentali

Le prime due fasi sono state ampiamente discusse nel capitolo 5.
La terza fase ha richiesto una analisi approfondita degli strumenti che Torch7 mette a disposizione per risolvere problemi di categorizzazione, mediante apprendimento automatico. Ricordiamo infatti che Torch è un framework di ampio supporto al Deep Learning. Per l'analisi ci si è riferiti alla tesi di Alessio Salman [1] e ad un tutorial molto accurato, che si basa sugli articoli e gli studi fatti dai creatori di Torch stesso [19], Clement e Farabet. Il lavoro [1] è stato preso come riferimento per le scelte fatte nella implementazione degli script e per i criteri di analisi dei dati. Per definire un modello di CNN, allenarla e testarla, seguendo un approccio supervised, cinque sono le fasi da seguire:

- Fare un preprocessing dell'immagine per facilitare l'apprendimento
- Descrivere un modello che risolva il problema di classificazione
- Scegliere una loss function da minimizzare
- Definire una fase di training
- Stimare la performance del modello attraverso la fase di testing

In questo lavoro, le cinque fasi precedenti, sono state realizzate attraverso gli script Lua:

- DataAcquisition.lua
- model.lua
- loss.lua
- train.lua
- test.lua
- forwarding.lua

Gli script sono eseguiti tutti dalla stessa cartella, che contiene anche la cartella in cui è salvato il dataset. E' importante sottolineare che gli script elaborano immagini a tre canali di tipo .png o .jpg. Inoltre, prima di richiamare il primo script, DataAcquisition.lua, è necessario eseguire lo script bash *data_labeling.sh*, che ha la funzione di rinominare le immagini di ogni classe in base al criterio seguente:

Nome_Classe.numero progressivo.estensione Es: Cars.1.png o Trees.4.png

Lo script *DataAcquisition.lua* ha la funzione di acquisire il dataset di immagini che servono alla rete per le operazioni di training e di testing e di implementare su esse un preprocessing.
Lo script inizia con l'acquisizione del dataset. La prima operazione è di scaling delle immagini: le immagini sono infatti scalate a 32x32 pixel. Una volta definito il numero di classi, sono dichiarati due floatTensor: *imagesAll e labelsAll*. Il primo tensore è utilizzato per caricare tutte le immagini del dataset e nel secondo si definiscono le *labels* da associare ad ogni immagine. Durante l'acquisizione vengono anche definite le percentuali di dataset a disposizione per training e testing della rete. La parte di acquisizione si conclude con la creazione del training set e del testing set.

```
107 torch.setdefaulttensortype('torch.FloatTensor') -- preprocessing requires a floating point representation
108
109 --NB: questo script funziona UNICAMENTE con immagini a 3 canali! --> in ogni caso la conversione di un immagine
110 --monocale a 3 canali è semplice e andrebbe fatta prima di eseguire lo script!
111 local imagesAll = torch.Tensor(numeroTotImmagini,3,32,32) --tensore di tutte le immagini del dataset
112 local labelsAll = torch.Tensor(numeroTotImmagini)         --tensore di tutte le labels del dataset
113
114 local count = 0
115 print("Acquisizione delle immagini")
116 for i in ipairs(folders) do
117         local prefix = "./"..root[1].."/"..folders[i].."/"
118         local numImgFolder = getNumber(prefix)
119         for j=1,numImgFolder do
120                 print(prefix..folders[i].."."..j.."."..opt.img)
121                 local img = image.load(prefix..folders[i].."."..j.."."..opt.img)
122                 img = checkImg(img) --resize a 3x32x32
123                 img:double()
124                 imagesAll[count + j] = img
125                 labelsAll[count + j] = i --Indice della label che indica la classe di appartenenza
126         end
127         count = count + numImgFolder
128 end
129
```

```
--CREAZIONE DEL TRAINING SET
trainData = {
        data = torch.Tensor(trsize,3,32,32),
        labels = torch.Tensor(trsize),
        size = function()
                return trsize
               end
}

--CREAZIONE DEL TESTING SET
testData = {
        data = torch.Tensor(tesize,3,32,32),
        labels = torch.Tensor(tesize),
        size = function()
                return tesize
               end
}

for i=1,trsize do
   trainData.data[i] = imagesAll[labelsShuffle[i]]:clone() -- shuffled training imgs
   trainData.labels[i] = labelsAll[labelsShuffle[i]]  -- shuffled lables for training
end

for i=trsize+1,tesize+trsize do
   testData.data[i-trsize] = imagesAll[labelsShuffle[i]]:clone() -- same as above
   testData.labels[i-trsize] = labelsAll[labelsShuffle[i]]
end
```

La seconda e ultima fase dello script esegue il preprocessing del dataset, equalizzando le immagini del tensore ImagesAll localmente e globalmente. L'immagine è convertita dallo spazio RGB a quello YUV, nel quale si distingue il canale della *luminanza Y* e quelli del *colore U,V*. In particolare la normalizzazione avviene prima globalmente e poi localmente, come mostrato in seguito:

```
-- Name channels for convenience
channels = {'y','u','v'}

-- Normalize each channel, and store mean/std
-- per channel. These values are important, as they are part of
-- the trainable parameters. At test time, test data will be normalized
-- using these values.
print '==> preprocessing data: normalize each feature (channel) globally'
mean = {}
std = {}
for i,channel in ipairs(channels) do
   -- normalize each channel globally:
   mean[i] = trainData.data[{ {},i,{},{} }]:mean()
   std[i] = trainData.data[{ {},i,{},{} }]:std()
   trainData.data[{ {},i,{},{} }]:add(-mean[i])
   trainData.data[{ {},i,{},{} }]:div(std[i])
end

-- Normalize test data, using the training means/stds
for i,channel in ipairs(channels) do
   -- normalize each channel globally:
   testData.data[{ {},i,{},{} }]:add(-mean[i])
   testData.data[{ {},i,{},{} }]:div(std[i])
end
```

```
-- Local normalization
print '==> preprocessing data: normalize all three channels locally'

-- Define the normalization neighborhood:
neighborhood = image.gaussian1D(13)

-- Define our local normalization operator (It is an actual nn module,
-- which could be inserted into a trainable model):
normalization = nn.SpatialContrastiveNormalization(1, neighborhood, 1):float()

-- Normalize all channels locally:
for c in ipairs(channels) do
   for i = 1,trainData:size() do
      trainData.data[{ i,{c},{},{} }] = normalization:forward(trainData.data[{ i,{c},{},{} }])
   end
   for i = 1,testData:size() do
      testData.data[{ i,{c},{},{} }] = normalization:forward(testData.data[{ i,{c},{},{} }])
   end
end
```

Una volta acquisito il dataset occorre definire il modello di CNN. Questa operazione è implementata nello script model.lua. Questo script mette a disposizione tre tipologie di reti: lineare (perceptron), MLP, CNN. In questo lavoro ci si è concentrati appunto sul modello CNN. In particolare la convolutional neural network implementata è costituita da due convolutional layers e un classificatore finale, ossia una MLP con due strati nascosti. Per prima cosa si definiscono alcuni parametri:

```
print '==> define parameters'

-- 2-class problem
noutputs = #classes

-- input dimensions
nfeats = 3
width = 32
height = 32
ninputs = nfeats*width*height

-- number of hidden units (for MLP only):
nhiddens = ninputs / 2

-- hidden units, filter sizes (for ConvNet only):
nstates = {16,256,128}
fanin = {1,4}
filtsize = 5
poolsize = 2
normkernel = image.gaussian1D(7)
```

Dopodiché si definisce il modello

```
model = nn.Sequential()

-- stage 1 : filter bank -> squashing -> L2 pooling -> normalization
model:add(nn.SpatialConvolutionMap(nn.tables.random(nfeats, nstates[1], fanin[1]), filtsize, filtsize))
model:add(nn.Tanh())
model:add(nn.SpatialLPPooling(nstates[1],2,poolsize,poolsize,poolsize,poolsize))
model:add(nn.SpatialSubtractiveNormalization(nstates[1], normkernel))

-- stage 2 : filter bank -> squashing -> L2 pooling -> normalization
model:add(nn.SpatialConvolutionMap(nn.tables.random(nstates[1], nstates[2],fanin[2]), filtsize, filtsize))
model:add(nn.Tanh())
model:add(nn.SpatialLPPooling(nstates[2],2,poolsize,poolsize,poolsize,poolsize))
model:add(nn.SpatialSubtractiveNormalization(nstates[2], normkernel))

-- stage 3 : standard 2-layer neural network
model:add(nn.Reshape(nstates[2]*filtsize*filtsize))
model:add(nn.Linear(nstates[2]*filtsize*filtsize, nstates[3]))
model:add(nn.Tanh())
model:add(nn.Linear(nstates[3], noutputs))
```

Osserviamo i due strati di *feature extraction* che realizzano le operazioni discusse nel capitolo 4. In particolare, per i primi due layers il *bank filter* è costituito da 256 kernels 5x5. Anche le operazioni di pooling sono implementate con lo stesso numero di operatori. La funzione `nn.SpatialConvolutionMap` utilizza collegamenti random tra i neuroni artificiai per

migliorare le prestazioni della rete. Infatti è buona norma seguire questo approccio, ossia scegliere collegamenti random rispettando però il *fan-in*, che varia da 1 per il convolutional layer che acquisisce l'immagine, a 4 per lo strato nascosto. L'ultimo blocco della rete è costituito dal classificatore, una MLP a due layer, che utilizza un banco di 128 filtri.

Con il modello a disposizione, il passo successivo consiste nella definizione della funzione di costo da minimizzare durante il training. La loss function viene definita nello script loss.lua

Il training della rete coincide con lo script trin.lua Una dei passi fondamentali di questo script è minimizzare l'errore di predizione di appartenenza di una immagine ad una classe. Questo si traduce nel trovare un minimo della loss function. L'ottimizzazione della rete avviene attraverso l'algoritmo di back propagation citato nel capitolo 3.

Lo script agisce in due fasi: nella prima sono definiti quattro strumenti per l'ottimizzazione della rete di cui l'algoritmo di back propagation si serve: il SGD [25], il ASGD [25], il BFGS [25], il CG [25]. Sono tutti strumenti che si basano sull'analisi del gradiente dell'immagine. Una trattazione completa non rientra negli obiettivi della tesi. Tuttavia possiamo dire che, per ottenere buoni risultati occorre partire da un metodo puramente stocastico, quale il SGD, che non necessita di nessun assunto sull'input, per poi applicare un metodo che invece si basa su set di esempi, detti *batch*, come il BFGS, che è più robusto rispetto al precedente.

```
print '==> configuring optimizer'

if opt.optimization == 'CG' then
   optimState = {
      maxIter = opt.maxIter
   }
   optimMethod = optim.cg

elseif opt.optimization == 'LBFGS' then
   optimState = {
      learningRate = opt.learningRate,
      maxIter = opt.maxIter,
      nCorrection = 10
   }
   optimMethod = optim.lbfgs

elseif opt.optimization == 'SGD' then
   optimState = {
      learningRate = opt.learningRate,
      weightDecay = opt.weightDecay,
      momentum = opt.momentum,
      learningRateDecay = 1e-7
   }
   optimMethod = optim.sgd

elseif opt.optimization == 'ASGD' then
   optimState = {
      eta0 = opt.learningRate,
      t0 = trsize * opt.t0
   }
   optimMethod = optim.asgd
```

Il *learning rate* è un parametro che stabilisce la velocità della rete per ogni esempio in input.
Il secondo blocco di script implementa l'algoritmo di back propagation:

```
-- create closure to evaluate f(X) and df/dX
local feval = function(x)
                -- get new parameters
                if x ~= parameters then
                   parameters:copy(x)
                end
                -- reset gradients
                gradParameters:zero()
                -- f is the average of all criterions
                local f = 0
                -- evaluate function for complete mini batch
                for i = 1,#inputs do
                   -- estimate f
                   local output = model:forward(inputs[i])
                   local err = criterion:forward(output, targets[i])
                   f = f + err
                   -- estimate df/dW
                   local df_do = criterion:backward(output, targets[i])
                   model:backward(inputs[i], df_do)

                   -- update confusion
                   confusion:add(output, targets[i])
                end
                -- normalize gradients and f(X)
                gradParameters:div(#inputs)
                f = f/#inputs
                -- return f and df/dX
                return f,gradParameters
```

Lo script test.lua infine è in grado di testare la rete mentre si è in corso di addestramento. Questo appunto lo si può fare utilizzando una parte di dataset alla fase di verifica di ciò che è stato fatto nel training fino a quel punto. In particolare lo script utilizza il metodo

`model:forward(input)`

E' un metodo che testa il modello uscito dalla fase di training, model, su una immagine di input passata come parametro. In questo modo è possibile, per ogni iterazione dell'apprendimento, testare i risultati ottenuti subito dopo la fase di training della rete.

```
-- test over test data
print('==> testing on test set:')
for t = 1,testData:size() do
   -- disp progress
   xlua.progress(t, testData:size())

   -- get new sample
   local input = testData.data[t]
   if opt.type == 'double' then input = input:double()
   elseif opt.type == 'cuda' then input = input:cuda() end
   local target = testData.labels[t]

   -- test sample
   local pred = model:forward(input)
   confusion:add(pred, target)
end
```

Sia lo script train.lua che test.lua forniscono in output la matrice di confusione e la curva di apprendimento ottenute. Questi due output saranno descritti nella parte di analisi dei dati.
Tutti gli script sono stati fatti girare lanciandone uno unico, che li riassume. E' lo script *doall.lua*. In esso vengono appunto richiamati tutti gli script descritti finora e crea un loop per il train e il test della rete:

```
211  -- semplice loop per allenare fino
212  -- a che non lo si interrompe
213  -- forzatamente
214  while true do
215     train(trainData)
216     test(testData)
217  end
218  --se vogliamo aggiungere una
219  --condizione di uscita
220  while true do
221     complete = train(trainData)
222     test(testData)
223     -- training 100% allora break
224     if complete == 1.0 then
225        break
226     end
227  end
228
```

Questo script è ancora in fase di miglioramento: non è infatti pensabile di arrivare con la fase di test ad una accuratezza pari al 100%, e per terminare le operazioni quando l'accuratezza si stabilizza ad un valore soddisfacente, occorre, allo stato attuale, bloccarlo con un intervento da parte dell'utente.
Prima di passare alla fase di analisi dei dati è opportuno dire che esiste un altro script utilizzabile ed è lo script *forwarding.lua*. Esso semplicemente, dato il modello allenato, classifica una immagine non appartenente al dataset usato per il training e il test in ad un modello di rete già allenata Per motivi di tempo questo script non è stato utilizzato in modo completo. Si sarebbe dovuto infatti creare un ulteriore dataset di immagini diverse da quelle a disposizione per avere un riscontro positivo o negativo. Ciò non è stato possibile per motivi di tempo.
Una volta in possesso degli strumenti di programmazione software utili per segmentare e implementare il modello di Deep Learning, si è passati alla fase di esecuzione e raccolta dei dati. I dati relativi ai risultati di segmentazione sono già stati discussi nel capitolo 5. Riportiamo qui una analisi dei dati ottenuti utilizzando gli script di Torch7.

6.3 Analisi dei dati

Le scelte fatte per il learning della rete, rispecchiano quelli che sono gli obiettivi detti nell'introduzione al capitolo. I risultati sperimentali sono stati raccolti ripartendo il dataset in due modi:

- Nel primo caso si è utilizzato un 80% del dataset alla fase di training e un 20% a quella di testing
- Nel secondo caso invece il 90% è stato utilizzato per il training della rete e il 10% per il testing.

In entrambi i casi gli output analizzati, per capire l'andamento della classificazione e la sua accuratezza, sono stati due:

- *Matrice di Confusione (Confusion Matrix)*: è una matrice che indica per ogni classe del problema, il numero di esempi classificati correttamente (i.e. quelli appartenenti alla diagonale maggiore) e, per quelli classificati in modo errato, mostra con quale classe li si è confusi.

```
ConfusionMatrix:
[[ 244   1   0   0   0   1   0   0] 99.187% [class: Benches]
[   1 203   3   4   1   0   8   5] 90.222% [class: Cars]
[   0   2 194   5   0   0   6   4] 91.943% [class: Pales]
[   1   0   1 217   0   1   3   2] 96.444% [class: People]
[   1   0   1   0 202   0   0   1] 98.537% [class: Steps]
[   0   0   0   0   0 226   0   0] 100.000% [class: TrashCans]
[   2   2   5   2   0   4 192  12] 87.671% [class: Trees]
[   3   5   4   0   1   0   6 187]] 90.777% [class: Walls]
```

Ad esempio, la prima riga e la prima colonna si riferiscono alla classe benches (panchine). Il numero in diagonale maggiore indica quante immagini sono state classificate correttamente. La prima riga indica quante immagini appartenenti alla classe benches siano state categorizzate in altre classi; la prima colonna indica quante

immagini appartenenti ad altre classi siano state categorizzate come Benches. Così vale per ogni riga e colonna.

- *Curva di Apprendimento*: è un grafico che rappresenta in ascissa il numero di iterazioni della rete durante la fase di training o di testing, in ordinata l'accuratezza media per ogni iterazione, ossia con quale percentuale media sono state categorizzate correttamente le immagini.

Questi due strumenti permettono una comprensione più che sufficiente di come sia avvenuto l'apprendimento da parte della rete.
Un'altra scelta fatta è stata quella di utilizzare lo stesso dataset di immagini, sia segmentato che non segmentato, per vedere come la segmentazione influisce sulla classificazione. Per ognuna delle due soluzioni si è proceduto con le percentuali di immagini per il training e per il testing citate sopra, per un totale di quattro casi di analisi:

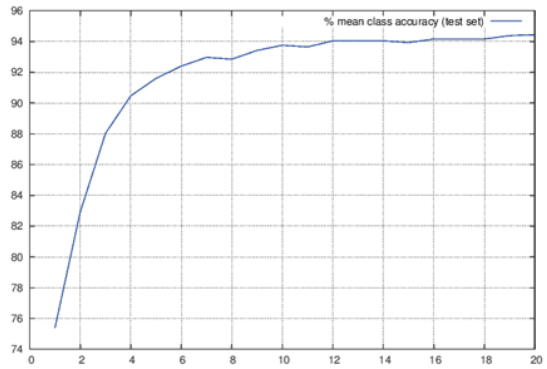
1. Dataset Non segmentato con percentuali 80% per il training/20% per il testing.
2. Dataset Non segmentato con percentuali 90% per il training/10% per il testing
3. Dataset Segmentato con percentuali 80% per il training/20% per il testing
4. Dataset Segmentato con percentuali 90% per il training/10% per il testing.

Ricordiamo che il dataset, in entrambi i casi, contengono circa 1000 immagini per classe, per un totale di circa 8000 immagini.
Nella pagine seguente sono riportati i risultati di questa sperimentazione.

- Caso Dataset Non segmentato
- Percentuali 80%/20%

ConfusionMatrix:

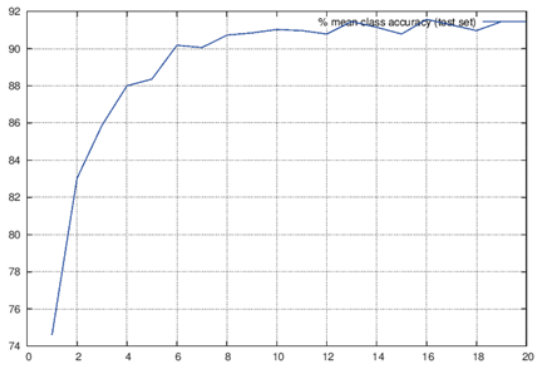
```
[[ 244   1   0   0   0   1   0   0]  99.187%  [class: Benches]
 [   1 203   3   4   1   0   8   5]  90.222%  [class: Cars]
 [   0   2 194   5   0   0   6   4]  91.943%  [class: Pales]
 [   1   0   1 217   0   1   3   2]  96.444%  [class: People]
 [   1   0   1   0 202   0   0   1]  98.537%  [class: Steps]
 [   0   0   0   0   0 226   0   0]  100.000%  [class: TrashCans]
 [   2   2   5   2   0   4 192  12]  87.671%  [class: Trees]
 [   3   5   4   0   1   0   6 187]]  90.777%  [class: Walls]
 + average row correct: 94.347663223743%
 + average rowUcol correct (VOC measure): 89.532358199358%
 + global correct: 94.441293250142%
```



- Caso Dataset Segmentato
- Percentuali 80%/20%

ConfusionMatrix:

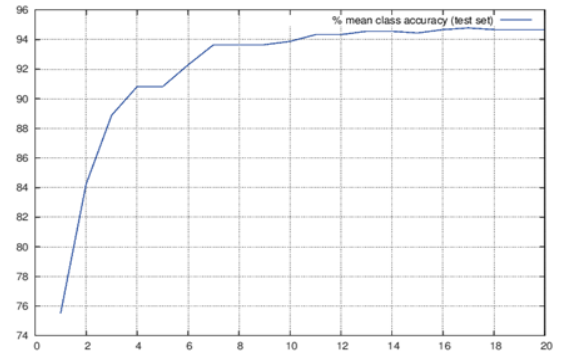
```
[[ 180   1   2   0   1   3   0   2]  95.238%  [class: Benches]
 [   6 166   2   2   5   1   1   9]  86.458%  [class: Cars]
 [   1   4 180   2   0   2   7   0]  91.837%  [class: Pales]
 [   0   2   2 190   0   2   2   0]  95.960%  [class: People]
 [   1   5   0   0 240   0   1   2]  96.386%  [class: Steps]
 [   0   1   2   1   0 190   0   0]  97.938%  [class: TrashCans]
 [   4   6  20   4   0   1 166   4]  80.976%  [class: Trees]
 [   8  11   3   1   4   1   2 197]]  86.784%  [class: Walls]
 + average row correct: 91.4470247924 33%
 + average rowUcol correct (VOC measure): 84.278289973736%
 + global correct: 91.454545454545%
```



- Caso Dataset Non segmentato
- Percentuali 90%/10%

ConfusionMatrix:

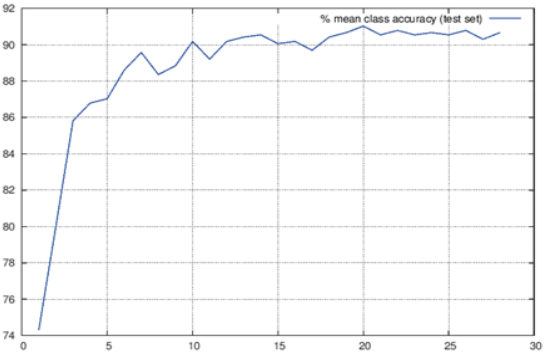
```
[[ 117   0   0   0   0   0   0   0]  100.000%  [class: Benches]
 [   1  97   0   3   1   0   1   4]  90.654%  [class: Cars]
 [   0   1  93   4   0   0   5   0]  90.291%  [class: Pales]
 [   0   0   1 119   0   0   2   0]  97.541%  [class: People]
 [   0   0   1   0 106   0   0   1]  98.148%  [class: Steps]
 [   0   0   0   0   0 119   0   0]  100.000%  [class: TrashCans]
 [   1   1   1   1   0   0  92   8]  88.462%  [class: Trees]
 [   0   2   3   0   1   0   4  92]]  90.196%  [class: Walls]
 + average row correct: 94.411527365446%
 + average rowUcol correct (VOC measure): 89.780031889677%
 + global correct: 94.671201814059%
```



- Caso Dataset Segmentato
- Percentuali 90%/10%

ConfusionMatrix:

```
[[  92   1   2   0   1   2   0   1]  92.929%  [class: Benches]
 [   3  79   2   0   6   0   1   7]  80.612%  [class: Cars]
 [   0   0  87   0   0   0   1   0]  98.864%  [class: Pales]
 [   0   1   4  85   0   0   1   0]  93.407%  [class: People]
 [   1   1   0   0 117   0   0   0]  98.319%  [class: Steps]
 [   0   1   2   1   0  98   0   0]  96.078%  [class: TrashCans]
 [   3   5  11   3   0   0  84   2]  77.778%  [class: Trees]
 [   5   4   1   1   2   0   1 106]]  88.333%  [class: Walls]
 + average row correct: 90.7900795340 54%
 + average rowUcol correct (VOC measure): 82.982764393091%
 + global correct: 90.666666666667%
```



Come primo dato possiamo analizzare la curva di apprendimento. Il numero di iterazioni fatto per ogni allenamento varia da 20 a 30, e rispecchia il numero utilizzato in una simulazione standard. In ogni caso analizzato, i testing sul dataset non segmentato risultato, rispetto a questo grafico, migliori rispetto a quelli ottenuti sul dataset segmentato. Si raggiunge un livello di accuratezza pari al 94% per un dataset non segmentato con percentuali 90%/10%, contro il 91% raggiunto nel caso di un dataset segmentato con le stesse percentuali di ripartizione tra training e testing. Quindi potremmo essere portati a dire che la segmentazione non ha avuto effetto nell'accuratezza globale della classificazione. Tuttavia bisogna tenere conto di due caratteristiche fondamentali del Dataset:

1. Il dataset contiene un numero di immagini molto inferiore rispetto ai dataset tipicamente utilizzati per il Learning di CNN's. Un dataset noto come il CIFAR10 [1] contiene 60000 immagini a colori e 6000 immagini per classe. Creare un dataset ad hoc per il train di reti come quelle implementate su Torch7 è un'operazione lunga e complessa, perché deve tenere conto di varie proprietà (i.g. dimensione di ogni classe, varietà dei soggetti di ogni classe, complessità della scena).
2. Il dataset contiene tipicamente immagini nelle quali è presente un oggetto, al massimo due. Già questo di per se mette la rete nelle condizioni di estrarre feautures con più semplicità rispetto al caso in cui si ha a che fare con immagini nelle quali i soggetti sono molti e appartengono a diverse categorie.

Considerando le matrici di confusione, cercando di capire quanto la segmentazione abbia influito nei risultati ottenuti. Partendo dalla prima tabella del primo blocco di dati, osserviamo che i problemi principali sono legati al fatto che la rete classifica come alberi immagini che invece sono muri. Se passiamo alla matrice di confusione ottenuto per il dataset segmentato, vediamo che il dato corrispondente scende, mentre aumenta la percentuale di immagini contenenti pali che vengono classificate come alberi. Ciò è relativamente logico, vista la somiglianza tra un palo e un albero. Possiamo dedurre che in questo caso la segmentazione ha un effetto positivo, poiché permette di diminuire la percentuale di confusione che si verifica tra due classi poco attinenti come le classi muro/albero. Lo stesso discorso lo si può fare per quanto riguarda le classi albero/automobile. E' altresì vero che aumenta, nel caso di dataset non segmentato, la percentuale di muri classificati come automobili e come panchine. Ciò presumibilmente deriva dal fatto che, alcune auto e panchine, segmentate in modo non ottimale, si presentano come superfici simili ai muri e quindi caratterizzate da feautures simili.

Questo difetto si può tentare di ridurlo applicando all'algoritmo di segmentazione implementato in questo lavoro una delle possibili strategie migliorative che concludono il capitolo 5. Se analizziamo la prima matrice del secondo blocco di dati, la situazione è analoga: alcuni muri sono riconosciuti come alberi, mentre nel resto delle classi c'è una sufficiente omogeneità e accuratezza. Passando alla seconda tabella, quella riferita al caso segmentato, notiamo anche qui che la percentuale di muri riconosciuti come alberi diminuisce, e aumenta invece quella di pali riconosciuti come alberi e di automobili classificate come muri. Anche in questo caso quindi, con una percentuale di dataset per il training più elevata, i risultati sono positivi. Si può concludere che la segmentazione influisce positivamente nella categorizzazione di oggetti. Se si avesse a disposizione un dataset con delle caratteristiche di più ampio raggio, si è portati a dedurre che questa operazione sarebbe sicuramente un elemento molto utile per aumentare l'accuratezza della rete nella categorizzazione di oggetti. Non bisogna dimenticare che la segmentazione riduce dall'inizio il numero di feautures che una rete deve estrarre e ciò influisce sul suo carico computazionale.

Capitolo 7

Conclusioni e sviluppi futuri

In questo lavoro di tesi si è voluto valutare l'influenza della segmentazione per aumentare l'accuratezza di image classification su un dataset di immagini preventivamente segmentate.
Per fare questo è stato implementato un algoritmo di segmentazione di immagini per isolare uno specifico oggetto dal resto della scena. La stereo visione si è dimostrata un ottimo strumento per questo proposito perché ha permesso di segmentare non un semplice oggetto, ma quello in primo piano; ciò è stato possibile grazie all'output della stereo camera: la mappa di disparità.
La dipendenza dell'algoritmo dal livello di rumorosità della mappa di disparità ha permesso di estrarre talvolta solo piccoli dettagli dell'intero oggetto, ma nella maggioranza dei casi il risultato è complessivamente buono e sufficiente per il riconoscimento.
La fase di image categorization è stata realizzata con l'impiego di Torch7. La segmentazione si è mostrata in fase di training e di testing uno strumento importante per aumentare la capacità della rete di distinguere classi molto diverse tra loro. Se da un lato l'algoritmo di segmentazione applicato, ha reso più simili classi costituite di immagini potenzialmente simili (i.g. Alberi/Pali), dall'altra ha distinto nettamente classi costituite da immagini che sono distanti in termini di contenuto (i.g. Alberi/Muri).
Il punto di partenza per ottenere risultati più accurati è legato al miglioramento delle due fasi del metodo: da un lato si potrebbe applicare una delle strategie discusse a fine capitolo 5 all'algoritmo di segmentazione, dall'altro si potrebbe agire testando una CNN a più livelli su un Dataset di maggiori dimensioni sia come classi che come numero di immagini.
La procedura realizzata può essere integrata in altri scenari applicativi che necessitano operazioni di riconoscimento, come ad esempio l'utilizzo di droni da ricognizione o altre applicazioni di SLAM.

Ringraziamenti

Desidero ringraziare innanzitutto il Prof. Stefano Mattoccia per i preziosi insegnamenti e i continui stimoli che ho ricevuto durante il periodo di lavoro alla tesi. Ringrazio Matteo Poggi, che è sempre stato disponibile a dirimere i miei dubbi e a superare le difficoltà incontrate nel tempo. Ringrazio tutti i laureandi/laureati con cui ho condiviso parte del lavoro. Ringrazio gli amici di sempre per i momenti condivisi. Ringrazio Noelia per la pazienza. Infine ringrazio la mia famiglia, che mi ha sostenuto durante questo percorso ed è stata l'unico punto fermo su cui poter sempre contare.

Bibliografia

[1] A.A. Salman, “Classificazione di immagini mediante il framework Torch”, DISI Università di Bologna, Tesi di Laurea AA 2013-2014

[2] D.A. Forsyth, J. Ponce, “Computer Vision: A Modern Approach”, 2nd Edition, Pratience Hall, 2011

[3] S. Mattoccia, “Stereo Vision: Algorithms and Applications”, Department of Computer Science (DISI), University of Bologna, Junuary 12, 2013

[4] S.Mattoccia, I. Marchio, M. Casadio, “A compact 3D camera suited for mobile and embedded vision applications”, 4th IEEE Mobile Vision Workshop, CVPR 2014, Columbus, Ohio, USA, June 23, 2014

[5] S. Mattoccia, “Stereo vision algorithms on FPGAs”, 9th IEEE Embedded Vision Workshop, CVPR 2013, Portland, Oregon, USA, June 24, 2013

[6] D. G. Lowe, “Distinctive image feautures from scale-invariant keypoints”, Int. J. Comput. Vision, vol. 60, pp 91-110, Nov. 2004

[7] S. Lautenegger, M. Chli, R.Y. Siegwart, “BRISK: Binary Robust invariant scalable keypoints”, in Proceedings of the International Conference on Computer Vision-Vol. 2, ICCV ’99, Washington DC, USA, pp. 1150, IEEE Computer Society, 1999

[8] A. Annovi, “Classificazione di oggetti in immagini attraverso il modello Bag of Visual Words”, DISI Università di Bologna, Tesi di Laurea, AA 2012-2013

[9] R. C. Gonzalez, R.E. Woods, “Digital Image Processing”, 2nd Edition, Prentice Hall, 2007

[10] B. Leibe, A. Leonardis, B. Schiele, “Robust object detection with interleaved categorization and segmentation”, International journal of computer vision 77, 2008

[11] C. Bishop, “Pattern Recognition and Machine Learning”, 1th Edition, Springer, 2006

[12] A. Barbieri, A. Messina, “Strumenti per la Classificazione Automatica di Contenuti Audiovisivi”, Tesi di Laurea Politecnico di Torino, AA 2004-2005

[13] Y.S Abu-Mostafa, M. Magdon-Ismail, H. Lin, “Learning from Data: a Short Course”, AMLbook.com, 2012

[14] B. Lazzerini, “Introduzione alle Reti Neurali”, Department of Information Engineering, Università di Pisa, 2014

[15] J.L Castro, C.J. Mantas, J.M. Benitez, “Neural networks with a continuous squashing function in the output are universal approximators”, University of Granada, Neural Networks 13, 2000

[16] A. Saxe, P. Koh et all. On random weights and unsupervised feature learning, International Conference on Machine Learning, 2011

[17] Y. LeCun, M. Ranzato, “Deep Learning Tutorial”, IMCL, Atlanta, USA, June, 2013

[18] C.Szegedy, A. Toshev, D. Erhan, “Deep Neural Networks for Object Detection”, Advances in Neural Information Processing Systems, 2553-2561, 2013

[19] Machine Learning with Torch, http://code.cogbits.com/wiki/doku.php

[20] K. Jung, A. Brown, “Beginning Lua Programming”, Wiley, 1th Edition, 2007

[21] K. Grauman, B. Leibe, “Visual Object Recognition; Synthesis Lectures on Artificial Intelligence and Machine Learning”, 1th Edition, Morgan&Claypool, 2011

[22] OpenCV Documentation, http://docs.opencv.org

[23] Watershed Documentation of Centre de Morphologie Mathématique, http://cmm.ensmp.fr/

[24] A. Muscoloni, S. Mattoccia, “Real-time tracking with an embedded 3D camera with FPGA processing”, International Conference on 3D (IC3D), 9-10 Dec, 2014, Liège, Belgium

[25] Q. Le, J. Ngiam, A. Coates, A. Lahiri, “On optimization methods for deep learning”, Computer Science Depertmente, Stanford University, ICML, 2011

[26] G. Belletti, “Perfezionamento di Sistema di ausilio per non vedenti basato su visione 3d”, Tesi di Laurea, AA 2013-2014

[27] S. Mattoccia, P. Macrì, G. Parmegiani, G. Rizza, “A compact, lightweight and energy efficient system for autonomous navigation based on 3D vision”, 10th IEEE/ASME International Conference on Mechatronic and Embedded Systems and Applications (MESA 2014), September 10-12, 2014, Senigallia, Italy

[28] Torch7 Website, http://torch.ch