Università Politecnica delle Marche

Facoltà di Ingegneria

Corso di Laurea in Ingegneria Informatica

e dell'Automazione

Implementazione di un algoritmo di identificazione della persona che utilizza frame di profondità

Implementation of a depth-based human identification algorithm

Relatore:
Prof. Ennio Gambi

Correlatori:
Prof.ssa Susanna Spinsante

Ing. Enea Cippitelli

Tesi di Laurea di:
Ilario Pierbattista

Anno Accademico 2014/2015

Indice

Capitolo 1

Introduzione

1.1 Human Sensing

L'insieme di tecniche e di soluzioni per il riconoscimento della presenza della persona nello spazio prende il nome *human sensing*.

Vengono utilizzati vari dispositivi di sensoristica per l'acquisizione di informazioni ai fini del riconoscimento. Dall'elaborazione delle informazioni acquisite, i sistemi di human sensing sono in grado di rilevare la presenza e la posizione della persona all'interno dell'ambiente interessato.

I contesti applicativi di sistemi di questo genere sono variegati ed in costante espansione. Sistemi per il rilevamento delle persone si adattano perfettamente in contesti di sorveglianza, sia al fine di garantire la sicurezza di un ambiente in termini di rilevamento delle intrusioni, sia al fine di costruire delle soluzioni di monitoraggio in ambienti assistivi automatizzati. Dispositivi in grado di localizzare corpi umani sono utilissimi nelle attività *search & rescue*, dove è necessario, in caso di catastrofi o calamità naturali, localizzare nel minor tempo possibile i superstiti in condizioni avverse. Vi sono delle applicazioni anche in ambito economico: soluzioni di *people counting* sono sempre più usate nei processi di *retail* per effettuare analisi di mercato.

Molto spesso, la natura dei sensori utilizzati per le acquisizioni, fanno in modo che i problemi di human sensing si intersechino con problemi di *computer vision*: l'utilizzo di sensori ad acquisizione visiva comporta l'insorgere di svariati problemi di visione artificiale.

Lo scopo della visione artificiale è quello di riprodurre la vista umana, intesa non come semplice acquisizione di rappresentazione bidimensionale di una regione di spazio, ma mira ad una reale interpretazione del relativo contenuto.

1.1.1 Stato dell'Arte

Il crescente interesse per le soluzioni di human sensing ha dato luogo a numerosi studi e ricerche sull'argomento. Rimanendo nel contesto del rilevamento legato a problemi di computer vision, spiccano i seguenti risultati.

People Tracking and Counting Papageorgiou et al. [6] nel 1998 proposero una tecnica di riconoscimento e conteggio di pedoni a partire da immagini RGB. Elaborando

opportunamente le informazioni raccolte da comuni telecamere, il sistema era in grado di riconoscere il pattern della figura umana analizzando le variazioni di intensità tra le diverse aree delle immagini. Per l'apprendimento utilizzarono una *macchina a vettori di supporto*[1].

Face Recognition Viola e Jones in [9] descrivono un framework di *object recognition*, applicandolo con successo al caso del riconoscimento dei volti. Ancora una volta, si tratta di un sistema allenabile che utilizza l'algoritmo *Adaboost*[2].

Human Sensing tramite Kinect Anche il dispositivo Kinect, prodotto da Microsoft, costituisce, nel complesso, un valido esempio di sistema di human sensing. La quantità e la qualità dei sensori con cui è equipaggiato, il costo relativamente basso e la disponibilità di *SDK* e toolkit integrabili con ambienti di sviluppo evoluti, stanno accrescendo la popolarità di tale dispositivo al di là delle applicazioni videoludiche. Cresce anche l'interesse per i *serious game*, videogiochi sviluppati con una doppia valenza, quella di assistere e guidare i pazienti in contesti riabilitativi.

1.2 Panoramica Generale

Questo elaborato descrive principalmente il lavoro svolto per l'implementazione del sistema di rilevamento descritto da Zhu e Wong in [11]. È correlato di numerosi approfondimenti teorici, necessari, sia per avere una corretta visione d'insieme del problema, sia per essere in grado di agire efficacemente nella futura richiesta di miglioramenti nell'implementazione proposta.

1.2.1 Il Lavoro di Zhu e Wong

Nel 2013 Zhu e Wong presentano un sistema di riconoscimento della figura umana, ripresa dall'alto di una stanza, con il dispositivo Kinect. Si tratta di una soluzione di human sensing, intercalata in una particolare condizione ambientale, che utilizza il *sensore di profondità* del dispositivo Kinect come sorgente per l'acquisizione di dati.

Come altre soluzioni proposte in precedenza, il sistema deve essere allenato a riconoscere persone. L'algoritmo utilizzato per lo sviluppo dei criteri di riconoscimento è *Adaboost*, lo stesso utilizzato per il framework di *face detection* sviluppato da Viola e Jones. Come si potrà vedere in seguito, in maniera più dettagliata, saranno numerose le somiglianze con quest'ultimo.

1.2.2 Configurazione Hardware

Precisamente la versione del dispositivo Kinect utilizzato è la *V2*. Il sensore di profondità del Kinect V2 fornisce una rappresentazione bidimensionale dello spazio, fondamentalmente sotto forma di immagini particolari. In queste ultime ogni pixel corrisponde il valore in millimetri della distanza dal sensore della superficie dell'oggetto presente nell'immagine. Ci si riferirà ad esse chiamandole *immagini di profondità*.

[1] Consultare la sezione 3.1 ed in generale tutto il capitolo 3 per ulteriori informazioni al riguardo.
[2] Per maggiori informazioni consultare [2] ed il capitolo 3.

Il sensore del Kinect, di cui è disponibile una piccola descrizione più dettagliata all'appendice C, fornisce uno stream di tali immagini ad una frequenza di 30 frame al secondo: è possibile quindi registrare dei *video di profondità*.

Visuale Top-Down

Il dispositivo Kinect viene montato al soffitto di una stanza e l'ambiente viene ripreso da tale prospettiva. Solitamente l'altezza a cui viene montato è di poco inferiore alla distanza del soffitto dal pavimento (poco meno di $3m$), inoltre la linea focale del sensore dovrebbe essere quanto più possibile ortogonale al pavimento della stanza, in modo da ridurre ai minimi termini la presenza di asimmetrie nelle riprese.

Tali distanze sono perfettamente compatibili con le specifiche tecniche del dispositivo stesso. Nel caso in cui vi sia la necessità di montare il Kinect a soffitti più alti di $4m$, si possono utilizzare delle lenti correttive per aumentare il range di affidabilità del sensore.

Molti sistemi di riconoscimento utilizzano il Kinect - e più in generale, qualsiasi dispositivo di acquisizione visivo - in posizione frontale rispetto ai soggetti da riconoscere. Infatti, il dispositivo, concepito per applicazioni videoludiche, è progettato per operare in tale posizione. Tuttavia, la configurazione descritta precedentemente, ha l'enorme vantaggio di eliminare l'occlusione del soggetto: in tali condizioni una persona non può nasconderne dietro di sè un'altra alla vista del sensore (se non in scomode posizioni), cosa invece frequentissima nei i sistemi di rilevamento frontali.

1.2.3 *Head and Shoulders Profile*

Riconoscere un oggetto significa mettere in relazione la sua rappresentazione con un concetto, più o meno specifico, che lo descriva. In questo caso, dovendo riconoscere delle persone, si dovrà mettere correttamente in relazione un'immagine di profondità che raffigura un individuo con il *concetto di persona*, mentre per un'immagine che non la raffigura, si dovrà evitare una relazione di tale natura.

Si definiscono due classi. Il concetto di classe è molto simile a quello delle *classi di equivalenza*, relativo all'algebra astratta, ma anche a quello delle classi come prototipi di *oggetti software*, relativo al paradigma di programmazione ad oggetti. Una classe costituisce un insieme di oggetti che condividono determinate proprietà caratteristiche: per le classi di equivalenza, tutti gli elementi appartenenti ad essa devono essere tra di loro in una relazione di equivalenza; per le classi software, tutte istanze di tali classi sono caratterizzate dalla medesima struttura (stessi attributi, stessi metodi), anche se poi ogni oggetto è ben distinto dall'altro.

Distinguere gli oggetti che rappresentano persone da quelli che non le rappresentano, significa classificare tali oggetti in due classi: la *classe delle persone* e quella delle *non persone*. A questo punto, bisogna iniziare a chiedersi quali debbano essere le proprietà degli elementi dell'una e quali quelli dell'altra.

Bisogna sottolineare il fatto che gli oggetti di una stessa classe, hanno sì delle proprietà in comune tra di loro, ma ne hanno altre per cui differiscono. Individuare le *caratteristiche* - ovvero proprietà osservabili e misurabili di un oggetto - in base alle quali decretare l'appartenenza all'una o all'altra classe, non è un problema banale. Si può intuire quanto sia vasto l'insieme delle caratteristiche valutabili nella rappresentazione

di un oggetto. Ovviamente la natura della rappresentazione influisce nella scelta delle caratteristiche più rilevanti.

Si ricordi che un'immagine di profondità rappresenta la realtà attraverso il valore della distanza misurata in ogni punto dello spazio osservato. È naturale, quindi, considerare tali distanze come caratteristiche misurabili dell'oggetto rappresentato.

In un sistema allenabile, sarà premura del modulo di apprendimento selezionare le caratteristiche più rilevanti per un oggetto. Senza anticipare altro sulla questione dell'apprendimento, è comunque utile fornire una descrizione in linguaggio naturale delle caratteristiche della forma del profilo umano, da tale visuale, che saltano immediatamente all'occhio.

1. L'immagine di una persona è caratterizzata da uno *spazio vuoto*[3] di fronte ad essa e dietro di essa.

2. A sinistra della spalla sinistra ed a destra della spalla destra del profilo dall'alto di una persona, sono presenti degli spazi vuoti.

3. Tra la testa e le spalle vi è una differenza di altezza.

Si segua quindi la seguente convenzione: le immagini che soddisferanno le precedenti proprietà saranno chiamate immagini *Head and Shoulders Profile*, o più brevemente *immagini HASP*.

1.2.4 Flusso di Lavoro

A conclusione di questa premessa, per evitare di perdersi in questo miscuglio di osservazioni eterogenee apparentemente prive di uno scopo preciso, è opportuno definire qual è il *workflow* principale dell'intero sistema di rilevamento. Sostanzialmente quest'ultimo è diviso in due componenti accoppiati, uno delegato all'apprendimento delle caratteristiche di classificazione, l'altro delegato all'effettivo rilevamento. Come è facilmente intuibile, il secondo non può funzionare senza il risultato fornito dal primo, che è il più complesso e corposo dei due.

Allenamento

Per allenare il sistema è necessario creare un insieme di allenamento, ovvero un insieme i cui elementi sono delle immagini di profondità che ritraggono persone (immagini HASP) e non. In fase di creazione, ogni elemento di tale insieme viene dotato di un'etichetta che identifica la reale classe di appartenenza dell'oggetto.

La componente software che si occupa dell'allenamento del sistema implementa l'algoritmo Adaboost. Quest'ultimo riceve in input l'insieme di allenamento, i cui elementi, dotati della rispettiva classificazione reale, sono alla base della scelta delle caratteristiche migliori per la descrizione delle classi di oggetti. Alla fine della sua esecuzione, Adaboost restituisce come output un classificatore.

Nei capitoli successvi si darà una definizione più formale di quello che è un classificatore, per il momento è sufficiente una definizione intuitiva: un classificatore *classifica*

[3] Per *spazio vuoto* si intende una regione di spazio il cui valore della distanza, percepita dal sensore, è molto vicino al quello della distanza del pavimento della stanza.

i vari oggetti, ovvero fornisce una *previsione* della classe di appartenenza relativamente ad ognuno di essi.

La classificazione effettuata da questa componente approssima solamente la classificazione reale. La bontà di tale approssimazione sarà il parametro di valutazione della prestazione generale del sistema.

Nel caso di Adaboost il classificatore risultante sarà simile ad una collezione di test: il risultato di tali test, eseguiti su di un qualsiasi oggetto, fornirà la previsione della classificazione dell'oggetto stesso.

Rilevamento

In questa fase il sistema analizza i frame di profondità delle acquisizioni in ordine sequenziale, alla ricerca di persone al suo interno.

Il classificatore ottenuto al termine dell'esecuzione di Adaboost, sarà in grado di classificare porzioni di immagini di profondità, ma non è in grado di predire direttamente, a partire da un intero frame, la presenza e la posizione di una o più persone al suo interno: le porzioni analizzabili dal classificatore hanno dei vincoli dimensionali da rispettare, che in prima approssimazione si possono immaginare come dei quadrati di dimensione costante. L'attività di rilevamento della persona all'interno del frame, quindi, consterà della sequenziale analisi di tutte le porzioni di frame che rispettano tali vincoli, al fine di coprire l'intera area.

Si vedrà in seguito che nei pressi di una persona nell'immagine di profondità, saranno molteplici le porzioni di frame per le quali il rilevamento darà esito positivo. Si pone quindi l'ulteriore problema di selezionare, delle tante porzioni che hanno dato esito positivo, quella che meglio approssima la reale posizione della persona.

Capitolo 2

Haar-Like Features

È stato già accenato che, parlando di caratteristiche di un oggetto, ci si riferisce alle sue proprietà elementari *osservabili* e *misurabili*. La scelta del tipo di caratteristiche dipende, in primo luogo dalla natura della rappresentazione dell'oggetto stesso, in secondo luogo da ciò che si vuole mettere in risalto.

Il sistema di rilevamento descritto da Zhu e Wong utilizza le *feature di Haar* per la valutazione delle immagini di profondità. L'uso di questo tipo di feature costituisce il primo di molti punti di contatto tra il sistema proposto dai due ricercatori ed il framework di face recognition di Viola e Jones.

2.1 Wavelet di Haar

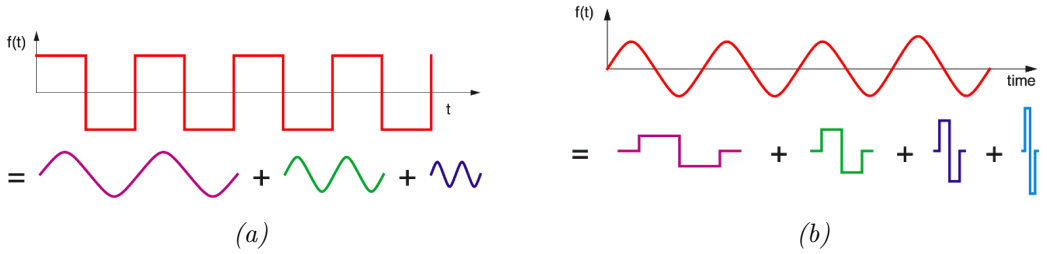
Le feature di Haar sono un costrutto derivante dalle *wavelet di Haar*. Queste ultime costituiscono il primo tipo di wavelet sviluppato, proposte da Alfréd Haar in [3] nel 1910. La wavelet madre è una funzione oscillante di lunghezza finita. L'equazione 2.1 descrive la wavelet madre di Haar.

$$\psi(x) = \begin{cases} 1 & 0 \leq t < 1/2 \\ -1 & 1/2 \leq t < 1 \\ 0 & \text{altrimenti} \end{cases} \tag{2.1}$$

Furono sviluppate come un esempio di funzioni ortonormali di base per uno spazio funzionale ed, in quanto tale, con esse è possibile esprimere un qualsiasi segnale limitato. Inoltre, sotto particolari ipotesi, costituiscono un sistema di rappresentazione duale all'analisi spettrale di Fourier.

Alfrèd Haar propose anche la prima DWT (*Discrete Wavelet Transform*) con la quale, le wavelet che compongono un segnale, vengono campionate discretamente. Le applicazioni sono notevoli e ad ampio spettro, prime tra tutte quelle nell'ambito della codifica dei segnali e nella compressione dei dati. Per fare un esempio, lo standard di compressione JPEG2000 sfrutta la trasformazione DWT [7] per ottenere risultati qualitativamente migliori rispetto allo standard precedente.

Una variante della trasformazione DWT con le wavelet di Haar bidimesionali (figura 2.2) è stata utilizzata da Oren et al. [5] in un sistema in grado di riconoscere la presenza di pedoni in delle immagini.



(a) *(b)*

Figura 2.1: Rappresentazione di un segnale con una serie di funzioni armoniche 2.1a e con una serie di wavelet di Haar 2.1b

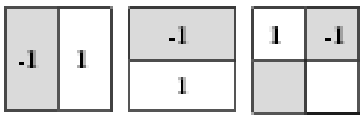


Figura 2.2: Wavelet di Haar bidimensionali utilizzate in [5].

Applicando alle immagini la trasformata DWT in diverse scale, si passa da una rappresentazione dell'immagine in scala dei grigi, ad una rappresentazione in termini di coefficienti delle wavelet di Haar. Tali coefficienti denotano le differenze di intensità tra le aree adiacenti dell'immagine e vengono utilizzati per evidenziare analogie strutturali delle immagini che contengono la figura di un pedone.

Nella descrizione di un framework generale per l'object detection, Papageorgiou et al. [6] utilizzano la trasformata DWT a wavelet di Haar bidimensionali non per l'estrazione di un template dell'oggetto espresso in variazioni di intensità, bensì per la selezione delle wavelet più significative al fine del riconoscimento dell'oggetto.

Una wavelet di Haar bidimensionale di una data dimensione, utilizzata per campionare un'immagine in una posizione specifica mette in evidenza le differenze di intensità dell'immagine nella regione descritta dall'area della wavelet. Tale differenza di intensità costituisce una proprietà osservabile e misurabile dell'immagine che ritrae l'oggetto. Queste sono le *Haar-like feature* o semplicemente *feature di Haar*.

2.2 Definizione

Le feature di Haar riescono a misurare la quantità e il verso delle variazioni di intensità tra due regioni adiacenti di una immagine. La comune rappresentazione grafica mette

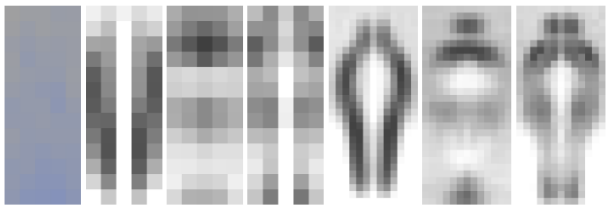
fig 

Figura 2.3: Coefficienti delle wavelet di trasformate DWT di diverse scale applicate alla stessa immagine. I coefficienti vengono codificati utilizzando la scala dei grigi [5, Figura 3].

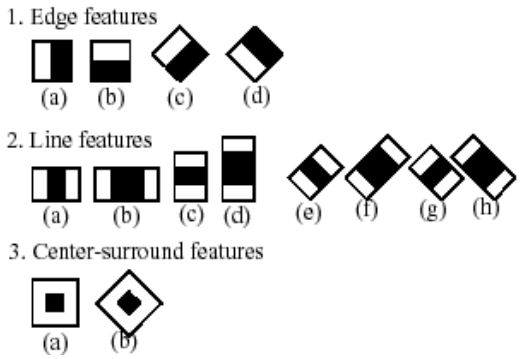


Figura 2.4: Set esteso di feature di Haar messe a disposizione dalla libreria OpenCV. *Le feature inclinate di* 45° *sono state introdotte in [4].*

in evidenza le due regioni colorandone una di bianco e l'altra di nero. La somma delle intensità dei pixel della regione bianca, a cui viene sottratta la somma delle intensità della regione nera, fornisce una misura della variazione media di intensità tra le due regioni.

$$f(Img) = E(Area_{white}) - E(Area_{black}) \tag{2.2}$$

L'equazione 2.2 fornisce una descrizione informale del funzionamento delle feature di Haar. Con $E(Area)$ si intende la somma delle intensità di tutti i pixel che appartengono all'area specificata.

Oltre questa iniziale definizione, una feature può essere composta da più di due aree adiacenti, dando luogo alle forme più disparate. La libreria *OpenCV* mette a disposizione una grande quantità di feature di forme differenti (figura 2.4).

Nonostante le forme differenti, il principio resta lo stesso: si hanno sempre due insiemi di aree, quello delle aree chiare (W) e quello delle aree scure (B). Si estende in tal modo la formula per il calcolo del valore della feature:

$$f(Img) = \sum_{Area_i \in W} E(Area_i) - \sum_{Area_j \in B} E(Area_j) \tag{2.3}$$

Il *segno* del valore di una feature identifica il verso della variazione. Se si prende in esame la forma di feature in alto a sinistra della figura 2.5, un valore positivo denota un'intensità mediamente maggiore nella regione bianca rispetto alla media della regione nera e viceversa.

2.2.1 Applicazioni

Nel framework di riconoscimento dei volti di Viola-Jones, vengono applicate le feature di Haar ad immagini RGB, dopo che queste sono state convertite in scala dei grigi e normalizzate in luminosità. Ciò che viene evidenziato sono le differenze di intensità dei pixel tra le regioni della foto.

A far variare l'intensità di un pixel in una foto concorrono l'illuminazione, il colore e molti altri fattori, alcuni dei quali sono fattori caratteristici del volto, oggetto del riconoscimento, mentre altri sono a tutti gli effetti dei disturbi ambientali che le operazioni di preprocessing mirano ad arginare.

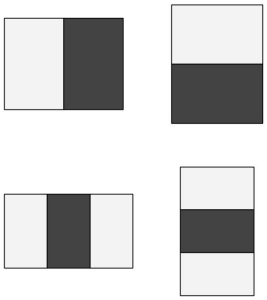


Figura 2.5: Tipi di feature di Haar utilizzati in [11].

Esse divengono il mezzo con il quale si può riconoscere un volto umano. È stato osservato, ad esempio, che nella foto di un volto, l'area che racchiude entrambi gli occhi e l'inizio del naso è caratterizzato da una particolare variazione di luminosità, misurabile e utilizzabile per descriminare le i volti da i non volti.

Zhu e Wong le utilizzano su immagini di profondità. Vengono considerati solamente quattro tipi di feature, contro i cinque utilizzati da Viola e Jones 2.5.

Nelle immagini di profondità, dove il valore di ogni pixel corrisponde alla distanza in millimetri della superficie dal sensore, applicare le feature di Haar ad un'area dell'immagine equivale a misurare la differenza di quota media rispetto ad un'altra adiacente ad essa.

Ciò viene utilizzato ai fini del riconoscimento della persona: sarà premura dell'algoritmo di apprendimento selezionare le feature migliori per descrivere la classe delle persone e tali descrizioni si baseranno su osservazioni di differenze di quota, esattamente come la descrizione che, informalmente, è stata data nella sottosezione 1.2.3.

2.2.2 Invarianza ai Ridimensionamenti

In seguito sarà necessario ridimensionare una feature in modo da coprire un'area più grande, in quanto, dovendo misurare le caratteristiche degli oggetti di interesse, questi ultimi sono variabili in dimensione. Il valore calcolato con la feature in questione, tuttavia, non dovrebbe essere troppo sensibile ai ridimensionamenti.

Al fine di ottenere la massima invarianza ai ridimensionamenti dell'area della feature, il valore di essa viene normalizzato con l'estensione dell'area totale valutata.

$$f'(Img) = \frac{\sum_{Area_i \in W} E(Area_i) - \sum_{Area_j \in B} E(Area_j)}{\sum_{Area_k \in W \cup B} size(Area_k)} \tag{2.4}$$

La tecnica vera e propria utilizzata per ridimensionare una feature verrà presentata nella sezione 5.1.

2.2.3 Vantaggi

Il primo indiscutibile vantaggio delle feature di Haar sta nel fatto che la caratterizzazione dell'oggetto viene effettuata sulla base di osservazioni d'insieme su intere aree dell'immagine e non su ossevazioni locali effettuate su singoli pixel.

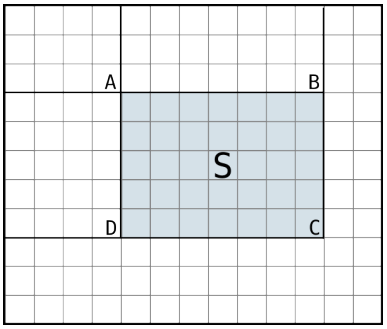


Figura 2.6

Oltre alla grandissima complessità che una valutazione su singoli pixel introdurrebbe, bisogna prendere atto che, con dati soggetti a disturbi e alla presenza di rumore, delle caratteristiche misurate su di essi non sarebbero molto significative.

L'estrazione dei contorni potrebbe essere più significativa della valutazione sui singoli pixel, ma continuano ad essere caratteristiche abbastanza complesse da calcolare ed ottenere. L'estrazione dei contorni, inoltre, è un concetto fortemente legato alle immagini RGB, usarlo con le immagini di profondità è un forzatura.

Il principale vantaggio delle feature di Haar rispetto ad altri tipi più elaborati di feature resta la loro efficienza computazionale. Con una particolare struttura dati di supporto, calcolare il valore di un feature di Haar per un'immagine è un'operazione eseguibile in un tempo costante.

2.3 Immagine Integrale

Il calcolo della somma delle intensità di ciascun pixel appartenente ad un'area, necessario al fine di calcolare il valore delle feature di Haar, è un'operazione il cui costo varia all'aumentare della dimensione complessiva dell'area. Calcolare tali somme infatti ha complessità computazionale $\Theta(m \cdot n)$ con m ed n dimensione dell'area.

La soluzione a tale problema consiste nell'utilizzo dell'*immagine integrale*, altrimenti detta *Summed Area Table*, una struttura dati che permette di calcolare la somma dei pixel di qualsiasi area all'interno di essa in un tempo costante.

Definizione 1. Sia I un'immagine larga w pixel ed alta h pixel. Con la scrittura $I(x, y)$ si identifica il pixel dell'immagine I che si trova alla colonna x e alla riga y. Si definisce *immagine integrale* (altrimenti detta *Summed Area Table*) una seconda immagine SAT delle stesse dimensioni per cui vale $\forall x \in [1, w], y \in [1, h]$:

$$SAT(x, y) = \sum_{i=1}^{x} \sum_{j=1}^{y} I(i, j) \tag{2.5}$$

Calcolare la somma del valore dei pixel contenuti in una regione rettangolare dell'immagine originale, con l'ausilio dell'immagine integrale, è un'operazione velocissima.

Si consideri la figura 2.6: si vuole calcolare la somma del valore dei pixel nella regione S evidenziata. I vertici del rettangolo saranno i punti $A : (x_1, y_1)$, $B : (x_2, y_1)$, $C : (x_2, y_2)$, $D : (x_1, y_2)$, notando che $1 \leq x_1 \leq x_2 \leq w$ e che $1 \leq y_1 \leq y2 \leq h$ [1]. Quindi:

$$\sum_{i=x_1}^{x_2}\sum_{j=y_1}^{y_2} I(i,j) = \sum_{i=1}^{x_2}\sum_{j=y_1}^{y_2} I(i,j) - \sum_{i=1}^{x_1}\sum_{j=y_1}^{y_2} I(i,j) =$$
$$= \left(\sum_{i=1}^{x_2}\sum_{j=1}^{y_2} I(i,j) - \sum_{i=1}^{x_2}\sum_{j=1}^{y_1} I(i,j)\right) - \left(\sum_{i=1}^{x_1}\sum_{j=1}^{y_2} I(i,j) - \sum_{i=1}^{x_1}\sum_{j=1}^{y_1} I(i,j)\right) =$$
$$= SAT(C) - SAT(B) - SAT(D) + SAT(A)$$

Elaborare l'immagine integrale è un'operazione con complessità computazionale $\Theta(w \cdot h)$, cioè il cui costo varia linearmente con la dimensione dell'immagine. Una volta ottenuta però, permette di calcolare qualsiasi somma di pixel in regioni rettangolari con operazioni di complessità $\Theta(1)$.

L'utilizzo dell'immagine integrale è molto vantaggioso nel momento in cui è necessario calcolare molte feature sulla stessa immagine. Volendo essere più espliciti, se la complessità computazionale totale del calcolo di n feature senza l'utilizzo dell'immagine integrale è maggiore di quella per la generazione dell'immagine integrale stessa, allora è vantaggioso utilizzare tale struttura dati.

2.4 Decision Stump

Una volta misurata una caratteristica relativa ad un oggetto, è necessario trovare un meccanismo, da utilizzare per classificare l'oggetto, che si basi esclusivamente sul valore della misura stessa.

2.4.1 Definizione di Albero Decisionale

Un albero decisionale è un modello predittivo, per il quale, definita una variabile obiettivo, grazie ad una serie di osservazioni e valutazioni di alcune proprietà (o caratteristiche), si giunge a predirne il valore.

Valgono le seguenti considerazioni:

1. Ogni nodo dell'albero rappresenta una caratteristica osservabile.
2. Ogni arco dal nodo padre al nodo figlio rappresenta una proprietà, per la caratteristica relativa al nodo padre, che deve essere soddisfatta per percorrere l'arco stesso.
3. Le foglie dell'albero rappresentano i valori ammissibili per la variabile obiettivo.
4. Il percorso dalla radice ad una foglia rappresenta la previsione del valore della variabile obiettivo, a fronte delle osservazioni effettuate sulle caratteristiche relative a ciascun nodo attraversato.

[1] Nelle immagini raster, si iniziano a contare le colonne e le righe dal punto in alto a sinistra.

2.4.2 Definizione di Decision Stump

Il più semplice albero decisionale è il *decision stump* (letteralmente *ceppo decisionale*), il quale ha profondità unitaria e solamente due foglie. Si basa sull'osservazione di una singola caratteristica per le previsioni della variabile obiettivo.

Alla radice del ceppo decisionale è presente una *funzione di valutazione*[2] $v : L \to \{V, F\}$, utilizzata per valutare la caratteristica associata alla radice stessa. A seconda del risultato della funzione di valutazione, il percorso terminerà in una o nell'altra foglia.

Pragmaticamente, intercalando questa iniziale definizione formale nell'attuale contesto applicativo, la radice di ogni ceppo decisionale conterrà un test con cui valutare la feature associata. Essendo le feature delle misure a valori reali (quanto meno in questo caso), tutto ciò che bisognerà verificare sarà l'appartenenza della misura ad un preciso range di valori per la selezione del percorso da seguire.

Si impone un valore di soglia che andrà a partizionare l'immagine della funzione di calcolo della feature: definita la soglia $\theta \in \mathbb{R}$ da associare alla feature $f : Dom(f) \to Img(f) \in \mathbb{R}$, si partizionerà come segue:

$$Img(f) = \{f(x)|f(x) < \theta\} \cup \{f(x)|f(x) \geq \theta\}$$

L'appartenenza della misura ad una o all'altra partizione, decreterà il percorso da seguire verso la foglia. In un problema di classificazione binaria, assumendo una classe come classe di oggetti positivi e l'altra come classe di oggetti negativi, associando alla prima il valore simbolico 1 ed alla seconda 0, un generico test da porre alla radice del ceppo, può essere della forma 2.6 oppure della forma 2.7.

$$h_1(x) = \begin{cases} 1 & \text{se } f(x) > \theta \\ 0 & \text{altrimenti} \end{cases} \tag{2.6}$$

$$h_2(x) = \begin{cases} 1 & \text{se } f(x) < \theta \\ 0 & \text{altrimenti} \end{cases} \tag{2.7}$$

Questi test non fanno altro che discriminare gli oggetti, misurandone una feature, in due classi distinte, a seconda che il valore misurato sia al di sotto o al di sopra del valore di soglia. Volendo trovare una descrizione formale unica, si introduce un ulteriore parametro $p \in \{-1, 1\}$, detto *polarità*, il cui unico compito è quello di invertire il segno della disequazione, permettendo di fondere le formule 2.6 e 2.7 in 2.8.

$$h(x) = \begin{cases} 1 & \text{se } pf(x) < p\theta \\ 0 & \text{altrimenti} \end{cases} \tag{2.8}$$

[2]Il dominio L è costituito da tutte le *formule ben formate*, ovvero da proposizione logiche e dalle relative combinazioni ottenute per mezzo di connettivi logici.

Capitolo 3

L'Algoritmo di Allenamento: Adaboost

3.1 *Ensamble Supervised Learning*

L'algoritmo di *machine learning* utilizzato dal Zhu e Wong in [11] è chiamato *Adaboost*, appartenente alla famiglia degli algoritmi di allenamento supervisionati, in particolare a quella degli *ensamble learner*.

3.1.1 Apprendimento Supervisionato

Definizione

Gli algoritmi di apprendimento supervisionato funzionano attraverso l'utilizzo di un insieme di allenamento, ovvero un insieme i cui elementi sono formati da coppie di oggetti e le relative etichette.

$$T = \{(x_1, y_1), \cdots, (x_n, y_n)\} \text{ con } y_1, \cdots, y_n \in L = \{l_1, \cdots, l_k, \cdots\} \tag{3.1}$$

Esiste una funzione f che associa ad ogni elemento x l'etichetta y corretta, ovvero in modo tale che:

$$f : Dom(f) \to L | \forall x \in Dom(f), (x, f(x)) \in T \tag{3.2}$$

Se l'insieme L delle etichette è costituito da un numero finito di elementi ($L = \{l_1, \cdots, l_k\}$), allora il problema dell'associazione oggetto-etichetta, prende il nome di problema di *classificatore* e la funzione f sarà il relativo classificatore reale.

L'obiettivo dell'apprendimento supervisionato è quello di trovare la *migliore* funzione h che approssima il classificatore reale, essendo sconosciuto.

L'algoritmo di apprendimento cerca la funzione h, chiamata *ipotesi*, all'interno dello *spazio delle ipotesi* $\mathcal{H}$, scegliendo quella in grado di fornire le prestazioni migliori, sia rispetto agli elementi dell'insieme di allenamento, che rispetto ad altri elementi aggiun-

tivi[1]. Si considera una ipotesi h una buona *generalizzazione* di f, quando riesce ad etichettare correttamente anche nuovi oggetti.

Molto spesso la funzione di classificazione reale f non dipende strettamente dall'oggetto x, ma è costituita da un *processo stocastico*: in tal caso, ciò che l'algoritmo deve apprendere è una *distribuzione di probabilità condizionale* $P(Y|x)$.

$$h^* = \arg\max_{h \in \mathcal{H}} P(h|data) \tag{3.3}$$

La scelta del corretto spazio delle ipotesi è fondamentale. È necessario trovare un compromesso tra l'*espressività delle ipotesi* nello spazio $\mathcal{H}$ e la *complessità computazionale della scelta di una buona ipotesi*. Complessivamente, il tempo di calcolo dell'allenamento, tende ad aumentare molto velocemente all'aumento della *dimensione* dello spazio $\mathcal{H}$ e della *complessità computazionale* nel calcolo da ogni singola ipotesi.

Esempi di *Supervised Learning*

Esistono molti algoritmi di apprendimento supervisionato. Alcuni dei più famosi sono l'*apprendimento di alberi decisionali*, l'*addestramento su reti neurali* e le *support vector machine*.

Nell'apprendimento di alberi decisionali, definito un insieme di allenamento (3.1) ed una collezione di *attributi* valutabili sugli oggetti di tale insieme, si costruisce l'albero ramo per ramo, associando alla valutazione di ogni attributo il risultato più probabile. Più nel dettaglio, se l'insieme degli esempi di allenamento non è vuoto, se la classificazione su tali esempi non è sempre la stessa e se l'insieme degli attributi valutabili non è vuoto, allora si seleziona l'attributo più *importante*[2]. Si valuta l'attributo su tutti gli esempi di allenamento e, per ogni valore che esso assume, si partiziona l'insieme di allenamento stesso in base a tale valore. Per ogni partizione, si allena un sottoalbero utilizzando la stessa procedura, prendendo in considerazione però, la collezione degli attributi privata dell'attributo appena valutato.

Una rete neurale è una particolare struttura decisionale che ricorda la struttura neurale del cervello umano. Come una qualsiasi rete è rappresentabile con un grafo, in cui i nodi sono detti *units* e gli archi *links*. Un generico link dall'unità i all'unità j propaga l'attivazione a_i dai due nodi. Ad ognuno di essi è inoltre associato un *peso* w_{ij}, che denota la forza ed il segno della connessione. L'unità j-esima, calcola innanzitutto la combinazione lineare dei segnali in input e dei rispettivi pesi, quindi, in prossimità del valore ottenuto, calcola il risultato della *funzione di attivazione* (3.4).

$$a_j = g(in_j) = g\left(\sum_{i=0}^{n} w_{ij} a_i\right) \tag{3.4}$$

[1] Si costruisce anche un insieme di testing per la valutazione delle prestazioni. Per non appesantire eccessivamente la lettura, i dettagli riguardo al testing vengono omessi ad un livello di trattazione così generale. Si faccia riferimento alla sottosezione 4.1.1 per ulteriori chiarimenti.

[2] Minore è l'*entropia* correlata all'attributo, maggiore è la sua *importanza*. Nella teoria dell'informazione, ci si riferisce al concetto di *entropia di Shannon*, ovvero alla misura dell'incertezza su una variabile aleatoria. Per una variabile aleatoria V con $v_1, \cdots, v_k$ possibili valori, ognuno con probabilità $P(v_i)$, l'entropia relativa a V sarà $H(V) = \sum_{i=1}^{k} P(v_i) \log_2 \frac{1}{P(v_i)} = -\sum_{i=1}^{k} P(v_i) \log_2 P(v_i)$

La definizione delle funzioni di attivazione è un parametro progettuale relativo alla rete da compiere a priori dell'allenamento: a seconda del particolare obiettivo che si vuole perseguire, si sceglie una struttura neurale piuttosto che un'altra. In fase di allenamento, invece, si procede determinare il valore ottimo di ciascun peso relativo agli archi della rete: dato un insieme di allenamento, si cerca di mappare nel modo migliore possibile ciascun input al relativo valore di output.

Le *Support Vector Machine* (SVM) sono, attualmente, il framework di allenamento supervisionato più popolare. Godono di tre caratteristiche:

1. Costruiscono un *margine di separazione massimale* tra gli elementi del dataset di allenamento. Si tratta di un confine che separa gli esempi di allenamento distanziando più possibile gli elementi di classi diverse. Questa caratteristica garantisce un buon grado di generalizzazione.

2. Creano degli *iperpiani lineari di separazione*. Elementi che non possono essere separati linearmente nello spazio originale di separazione, vengono rappresentati in spazi con un numero più elevato di dimensioni, al fine di trovare un iperpiano lineare come separatore.

3. Combinano i vantaggi dei metodi *parametrici* e *non parametrici*[3] risultando molto flessibili nella rappresentazione di funzioni di classificazione complesse, ma poco sensibili a problematiche di *overfitting*.

Overfitting

Le tecniche di addestramento supervisionato potrebbero incorrere in problemi di *overfitting*. Dato un insieme di allenamento, infatti, la ricerca delle ipotesi che approssimano al meglio la funzione obiettivo potrebbe essere influenzata da caratteristiche che sono comuni tra gli elementi dell'insieme stesso, ma che non sono discriminanti nella descrizione più generale della classe di oggetti.

La variabilità *intraclasse* delle caratteristiche degli esempi di allenamento, viene equivocata, in fase di selezione, per variabilità *extraclasse* e utilizzata, erroneamente, per la classificazione.

Ciò porta ad un sequenziale miglioramento delle prestazioni del riconoscimento per gli elementi del dataset di allenamento, ma a prestazioni peggiori nella classificazione di nuovi elementi.

Nella sottosezione 4.1.1, verranno esposte alcune tecniche per evitare l'insorgere di problemi di overfitting.

Ensamble Learning

Gli algoritmi di *ensamble learning* sono una particolare categoria di algoritmi di apprendimento supervisionato.

[3] I modelli *parametrici* caratterizzano gli elementi di un insieme di allenamento utilizzando un numero prefissato di parametri, indipendentemente dalla cardinalità dell'insieme. Ad esempio, nelle reti neurali, il numero di pesi da fissare è una caratteristica strutturale della rete. Di contro, i metodi *non parametrici* non sono caratterizzati da un limite prefissato di parametri con cui caratterizzare gli esempi di allenamento.

L'approssimazione della funzione obiettivo avviene mediante la selezione di una collezione di funzioni dallo spazio delle ipotesi. La predizione della classificazione dell'oggetto avviene, poi, combinando tra loro il risultato delle predizioni formulate dalle singole ipotesi.

Un classificatore allenato con un algoritmo di ensamble learning, intuitivamente, fornisce previsioni più accurate, ampliando il numero di ipotesi vincenti scelte per le predizioni.

3.1.2 *Adaptive Boosting*

Strong e *Weak Learner*

Di qui in avanti, si farà riferimento due tipologie distinte di classificatori, i *weak learner* e gli *strong learner*[4].

I *weak learner* sono costituiti da singole ipotesi, selezionate da un qualsiasi algoritmo di allenamento supervisionato. La loro caratteristica principale è quella di essere meccanismi di classificazione estremamente semplici, anche dal punto di vista computazionale, e nel complesso, la loro attendibilità viene considerata di poco maggiore rispetto ad una previsione casuale.

Gli *strong learner*, invece, sono delle combinazioni di weak learner. Vengono definiti grazie all'implementazione di un metodo di ensamble boosting e garantiscono un'affidabilità molto maggiore rispetto ad i singoli weak learner.

Algoritmi di Boosting

Sono la famiglia più popolare di algoritmi che implementano l'ensamble learning.

Inizialmente, gli algoritmi di questo tipo, associano ad un insieme di allenamento una distribuzione: ogni elemento viene caratterizzato un peso. Iterativamente viene selezionato, dallo spazio delle ipotesi, il *miglior weak learner*, ovvero quello che riesce a classificare gli elementi del dataset compiendo meno errori possibili.

L'idea che è alla base del successo degli algoritmi di boosting sta nel fatto che, tali pesi, vengono aggiornati per ogni elemento ad ogni iterazione in base al risultato della classificazione con il weak learner corrente. Viene diminuito il peso degli elementi classificati correttamente e aumentato quello degli altri, in modo da favorire la selezione, all'iterazione successiva, di un weak learner che vada a compensare gli errori compiuti dal precedente.

Tutti i weak learner selezionati combinano le loro singole predizioni in un'unica predizione molto accurata: lo strong learner risultante sarà la combinazione dei weak learner estratti.

Adaboost

Adaboost è l'algoritmo proposto da Freund e Schapire in [2], ed è valso loro il premio Gödel per il contributo originale apportato al campo dell'informatica teorica.

Il nome *Adaboost* deriva dalla fusione delle parole *Adaptive* e *Boosting* (ed essendo un algoritmo di boosting, implementa le operazioni descritte precedentemente).

[4] Alternativamente, *weak classifier* e *strong classifier*.

Ciò che rende Adaboost un algoritmo di boosting *adaptive* è la politica di aggiornamento dei pesi relativi agli elementi dell'insieme di allenamento: viene decrementato il peso degli elementi classificati correttamente, lasciando invariato il peso degli altri.

Il valore di tale decremento non è costante, ma *dipende dall'errore pesato di classificazione complessivo* per la relativa iterazione. Maggiore è l'errore, maggiore è il decremento dei pesi.

È così che Adaboost, ad ogni iterazione, cambia il proprio comportamento adattandosi perfettamente al contesto specifico.

3.2 Dataset di Allenamento

Prima di procedere ulteriormente nella presentazione dell'algoritmo di allenamento, è necessario fare qualche considerazione preventiva sulla costruzione dei dataset di allenamento.

Innanzi tutto, qualche delucidazione sui termini: è stato utilizzato largamente il termine *insieme* per denotare l'organizzazione di dati in ingresso ad un algoritmo di allenamento, ma di qui in avanti si utilizzerà anche il termine *dataset*. È un termine più legato al gergo tecnico, senza alcuna particolare accezione, se non quella di denotare un insieme organizzato in una reale struttura dati.

3.2.1 Categorie di Classificatori

È stato detto precedentemente che, come tutti gli algoritmi di supervised learning, Adaboost riceve in input un insieme di allenamento e restituisce un classificatore forte.

Potrebbe non essere sufficiente disporre di un unico classificatore forte per l'applicazione di rilevamento che si vuole realizzare.

Variabilità della Forma delle Immagini HASP

La forma dell'immagine HASP della persona nel frame di profondità varia per molti motivi.

La prima causa di variazione della forma HASP è l'*orientazione della persona*. L'immagine di una persona che cammina seguendo una direzione parallela al lato lungo del frame sarà indubbiamente diversa dall'immagine di una persona che cammina seguendo una direzione perpendicolare, così come sarà diversa dall'immagine di una persona che compie una traiettoria obliqua all'interno della stanza.

Un altro fattore importante che concorre nella variazione della forma è la *distorsione prospettica*. L'immagine di una persona ripresa al centro del frame e quella della stessa persona ripresa in una zona periferica è molto diversa, anche se orientata sempre nella stessa direzione.

L'entità della distorsione prospettica della forma è tanto maggiore quanto più il sensore è vicino al pavimento. D'altro canto, le caratteristiche tecniche di quest'ultimo impongono un limite massimo alla distanza.

Considerata la realtà applicativa del sistema, la distanza massima del sensore dal pavimento sarà di poco inferiore all'altezza del soffitto in una comune abitazione. L'altezza a cui viene montato il sensore, quindi, non verrà considerato un parametro progettuale,

bensì una condizione ambientale entro la quale il sistema deve operare: la distorsione prospettica non può essere eliminata.

Un'ulteriore causa della variazione della forma dell'immagine HASP consiste nelle *differenze di corporatura e di statura delle persone*. Tali differenze vengono chiamate *differenze interclasse*, essendo delle caratteristiche variabili di oggetti che appartengono alla stessa classe. Ovviemente un buon sistema di rilevamento dovrebbe essere in grado di riconoscere soggetti di differente statura e corporatura. Per ovviare a questo problema verranno prese delle opportune misure di ridimensionamento delle immagini.

Definizione delle Categorie di Classificatori

Una forte variabilità intraclasse potrebbe portare alla sintesi di classificatori troppo laschi.

Un buon approccio a questo problema consiste nel dividere gli oggetti, appartenenti alla stessa classe, in diverse categorie. Tali categorie devono essere scelte in modo tale che le differenze tra gli oggetti appartenti alla stessa categoria siano lievi, mentre quelle relative agli oggetti di differenti categorie siano più accentuati.

Definite tali categorie, per ognuna di esse verrà sintetizzato un classificatore, allendandolo solo su elementi appartenenti ad una stessa categoria. Così facendo le differenze intraclasse di entità superiore vengono arginate.

L'*orientazione della persona* è il parametro su cui si possono formulare le diverse categorie. In questo sistema vengono allenati due classificatori forti che vanno ad operare in parallelo (equazione 3.5) in fase di riconoscimento: il primo è allenato esclusivamente con immagini di persone che si muovono in direzione parallela al lato lungo del frame (informalmente, direzione *orizzontale*), il secondo con immagini di persone che si muovono in direzione perpendicolare alla prima (direzione *verticale*).

$$F_{final}(x) = F_{hor}(x)||F_{ver}(x) \tag{3.5}$$

Sarebbe possibile anche definire ulteriori categorie basate sulla direzione delle persona, introducendo due classificatori dedicati alle due *direzioni oblique* rispetto alle prime due.

Inoltre si può pensare di porre rimedio alla distorsione prospettica del sensore suddividendo il frame in aree ed allenando, per ognuna di esse, una coppia di classificatori orizzontale-verticale.

Queste ultime due soluzioni introducono una certa complessità aggiuntiva ed in questa fase iniziale di sperimentazione non porteranno grandi vantaggi alla precisione complessiva del sistema.

3.2.2 Preparazione dei Dataset

Acquisizioni

Per la creazione dei dataset di allenamento è stato chiesto a 10 soggetti, di statura e corporatura differente, di percorrere una traiettoria differente per ogni registrazione.

Delimitata fisicamente l'area del pavimento corrispondente all'area di visualizzazione del frame di profondità, sono state individuate tre traiettorie significative:

Orizzontale Seguendo questa traiettoria, i soggetti coprono tutta l'area muovendosi lungo le parallele al lato maggiore.

Verticale I soggetti coprono l'area muovendosi lungo la direzione perpendicolare al lato maggiore.

Random È stato chiesto ai soggetti di seguire una traiettoria casuale, cercando di coprire tutta l'area del frame.

I primi due percorsi corrispondo alle categorie individuate precedentemente: da tali registrazioni vengono estratti gli oggetti per l'allenamento dei classificatori che sono stati denominati, informalmente, orizzontale e verticale.

Le registrazioni dell'ultima traiettoria, invece, vengono utilizzate per la creazione del *dataset di validazione*[5].

Con un software di registrazione[6] apposito vengono sequenzialmente catturati e salvati i frame di profondità elaborati dal dispositivo Kinect, ad una frequenza massima di $30 fps$. Il software è in grado di catturare fino a 1000 frame, corrispondenti a poco più di mezzo minuto, che vengono salvati in file binari grezzi, senza ricorrere a compressione.

La durata delle registrazioni delle traiettorie orizzontale e verticale è subordinata al tempo necessario al soggetto per coprire tutta l'area del frame, mentre le registrazioni riferite alle traiettorie casuali sono tutte costituite da 1000 frame.

Ritaglio dei Samples

È stato sviluppato un software apposito per la gestione dei dataset per ritagliare, organizzare e ridimensionare, le immagini dei dataset di allenamento.

Attraverso un'efficace interfaccia utente, permette di aprire e visualizzare una registrazione e scorrere ciascun frame. Una volta individuato il frame interessato, è possibile estrarre un ritaglio selezionando l'area. La porzione selezionata di frame viene immediatamente salvata in un file binario grezzo nella cartella contenente il dataset correntemente aperto.

Nell'appendice A vi saranno ulteriori spiegazioni al riguardo.

3.2.3 Preprocessing

Resize

In fase di creazione dei dataset è necessario effettuare una prima operazione di preprocessing. In virtù del fatto che, a soggetti di corporatura differente, corrispondo immagini HASP di dimensioni differenti, è necessario normalizzare le dimensioni dei ritagli componenti i dataset di allenamento.

Tutte le immagini vengono quindi ridimensionate a 24×24 pixel, valore già utilizzato in letteratura nel framework proposto da Viola e Jones in [9]. Sono disponibili tre algoritmi per il resize delle immagini: *nearest neighbour*, *bilinear interpolation* e *bicubic interpolation*.

[5] Per ulteriori informazioni, consultare il capitolo 4, nella fattispecie la sottosezione 4.1.1.

[6] Sviluppato al Laboratorio di Telecomunicazioni presso l'Università Politecnica delle Marche (`http://www.tlc.dii.univpm.it/blog/databases4kinect`).

La scelta dell'algoritmo di resize è caduta sul *nearest neighbour*, il più semplice ed il meno invasivo dei tre.

Conversione delle distanze

Un'ulteriore operazione di preprocessing da effettuare sugli elementi del dataset è la conversione delle distanze.

Il valore di ciascun pixel nelle immagini di profondità, esprime la distanza in mm della superficie rilevata dal sensore. È necessario elaborare le immagini di profondità trasformando la distanza della superficie dal sensore in *quota della superficie dal pavimento*.

$$d' = d - d_{pavimento} \tag{3.6}$$

L'equazione 3.6 esplicita la modalità di conversione delle distanze. Bisogna notare che il valore della distanza del pavimento dal sensore, non è uniforme in tutta l'area del frame, a causa della distorsione prospettica: nelle zone periferiche essa sarà maggiore. Si considera valida la distanza misurata esattamente al centro del frame, corrispondente a circa $2850mm$.

3.3 Procedura di Allenamento

3.3.1 Notazione

Sia D un insieme di allenamento.

$$D = \{(x_1, y_1), \cdots, (x_n, y_n)\} \tag{3.7}$$

D è costituito da n coppie (x_i, y_i), dove x_i è un'immagine di profondità e y_i è la relativa etichettatura reale.

Il problema è di classificazione binaria, quindi ogni etichette y_i rappresenta la *classificazione* dell'oggetto x_i. Inoltre, poniamo che:

$$y_i = \begin{cases} 1 & \text{se } x_i \text{ raffigura un umano} \\ 0 & \text{altrimenti} \end{cases} \tag{3.8}$$

Se $y_i = 1$, si parlerà, in riferimento a x_i, di *esempio positivo*, se $y_i = 0$ invece si parlerà di *esempio negativo*.

L'insieme D può essere partizionato come segue:

$$P = \{(x, y) \in D | y = 1\} \text{ e } N = \{(x, y) \in D | y = 0\} \tag{3.9}$$

Si tenga presente che, essendo P ed N partizioni di D, valgono le seguenti[7]:

$$D = P \cup N \tag{3.10}$$

$$P \cap N = \emptyset \tag{3.11}$$

$$\#(D) = \#(P \cup N) = \#(P) + \#(N) \tag{3.12}$$

[7]La scrittura $\#(D)$ denota la cardinalità dell'insieme D.

3.3.2 Definizione dei Weak Learner

È necessario, a questo punto, esplicitare ciò che andrà a costituire i weak learner. Viola e Jones in [9], Zhu e Wong in [11] utilizzano dei *decision stump* equipaggiati per il calcolo delle feature di Haar sulle immagini di profondità.

Come è stato detto nella sezione 2.4, un decision stump è della forma espressa dall'equazione 2.8.

$$h(x) = \begin{cases} 1 & \text{se } pf(x) < p\theta \\ 0 & \text{altrimenti} \end{cases} \tag{2.8}$$

In questo specifico contesto, f rappresenta una singola feature di Haar: esistono tanti weak learner tante sono le possibili feature di Haar per le immagini che costituiscono il dataset di allenamento. Il valore della soglia θ e quello della polarità $p \in \{-1, 1\}$, sono i parametri liberi del weak learner, che verranno fissati in fase di allenamento. Verranno fornite ulteriori spiegazioni nelle sezioni successive (3.3.3 e 3.4).

3.3.3 Procedura di Estrazione dello *Strong Learner*

La seguente procedura, descrive il processo di costruzione dello strong learner come combinazione di weak learner: è il corpo principale di Adaboost ed implementa le tecniche di boosting adattivo sul dataset di allenamento. Di seguito si presenta la sequenza di passi necessari alla combinazione di T weak learner in un unico classificatore forte.

1. Si associa ad ogni elemento $(x_i, y_i) \in D$ un peso w_i iniziale:

$$\forall (x_i, y_i) \in D \; w_i = \begin{cases} \frac{1}{2l} & \text{se } (x_i, y_i) \in P \text{ con } l = \#(P) \\ \frac{1}{2m} & \text{se } (x_i, y_i) \in N \text{ con } m = \#(N) \end{cases} \tag{3.13}$$

   Sia inoltre:

$$n := \#(D) = [\#(P) + \#(N)] = l + m \tag{3.14}$$

2. *For* $t = [1 : T]$

   (a) Si normalizzano i pesi, in modo che la loro somma sia pari ad 1:

$$w_{t,i} \leftarrow \frac{w_{t,i}}{\sum_{j=1}^{n} w_{t,j}} \tag{3.15}$$

   (b) Si estrae il miglior weak learner. La procedura viene esposta nel dettaglio nella sezione 3.4, ma si tenga presente che il miglior classificatore debole è quello il cui *errore pesato* è minimo, in riferimento alla corrente iterazione.

$$\epsilon_t = \min_{f,p,\theta} \left\{ \sum_{i=1}^{n} w_{t,i} \cdot |h(x_i, f, p, \theta) - y_i| \right\} \tag{3.16}$$

   Siano inoltre f_t, p_t, θ_t i parametri del classificatore debole che ne minimizzano l'errore pesato:

$$h_t(x) := h(x, f_t, p_t, \theta_t) \tag{3.17}$$

(c) Si calcola il fattore moltiplicativo per la modifica dei pesi associati agli esempi di allenamento:

$$\beta_t \leftarrow \frac{\epsilon_t}{1-\epsilon_t} \tag{3.18}$$

(d) Si aggiornano i pesi

$$w_{t+1,i} \leftarrow w_{t,i} \cdot \beta_t^{e_i} \tag{3.19}$$

dove:

$$e_i = \begin{cases} 1 & \text{se } (x_i, h_t(x_i)) \in D \\ 0 & \text{altrimenti} \end{cases} \tag{3.20}$$

È da notare che la prima condizione è vera solo nel caso in cui il weak learner h_t ha predetto correttamente la classificazione di x_i.

(e) Definizione del fattore moltiplicativo, ovvero il coefficiente nella combinazione lineare, per il weak learner corrente:

$$\alpha_t \leftarrow \log \frac{1}{\beta_t} \tag{3.21}$$

3. Il classificatore forte è dato da:

$$F(x) = \begin{cases} 1 & \text{se } \sum_{t=1}^{T} \alpha_t h_t(x) > \theta_{str} \sum_{t=1}^{T} \alpha_t \\ 0 & \text{altrimenti} \end{cases} \tag{3.22}$$

dove $\theta_{str} \in [0,1]$ è la soglia dello strong learner.

Si noti che, nell'operazione 2b, l'*errore pesato* non è altro che la somma dei pesi degli esempi che non vengono classificati correttamente. Infatti:

$$|h(x_i, f, p, \theta) - y_i| = \begin{cases} 0 & \text{se } h(x_i, f, p, \theta) = y_i \\ 1 & \text{se } h(x_i, f, p, \theta) \neq y_i \end{cases} \tag{3.23}$$

Al punto 2c, il valore di β_t non è altro che il rapporto tra l'errore pesato del classificatore debole e la somma dei pesi delle immagini classificate correttamente. Tale valore è chiaramente $0 < \beta_t < 1$.

In fase di aggiornamento dei pesi (punto 2d), i pesi relativi ad esempi classificati correttamente vengono moltiplicati per β_t ($\beta_t^1 = \beta_t < 1$) e quindi decrementati, mentre gli altri vengono lasciati inalterati ($\beta_t^0 = 1$). Fare in modo che gli esempi non classificati correttamente abbiano un peso maggiore di quelli classificati correttamente è il modo per influenzare la scelta del classificatore debole successivo che andrà a colmare le lacune del suo predecessore.

La scelta della soglia per il classificatore forte (punto 3) deve minimizzare il numero di esempi classificati in modo errato. La procedura di selezione viene spiegata nel dettaglio al capitolo 4.

3.4 Selezione del *Weak Learner* Migliore

La scelta del miglior weak learner mira ad identificare la feature di Haar, la polarità e la soglia che minimizzano l'errore pesato di classificazione.

Si ricordi che le feature di Haar sono degli indicatori di quanto le intensità dei pixel variano da una regione della feature ad un'altra. Il classificatore debole, quindi, prevederà l'etichettatura dell'immagine a seconda che tale indice sia maggiore o minore di una certa soglia. Il compito della polarità è quello di stabilire il verso della diseguaglianza.

3.4.1 Pool delle Feature di Haar

Il pool delle feature di Haar da testare è costituito da tutte le possibili configurazioni valutabili sulle immagini di allenamento. Viola e Jones utilizzano immagini di allenamento di 24×24 pixel e 5 tipologie di feature differenti ([9, sezione 2.2]).

Contrariamente a quanto potrebbe sembrare, già in dimensioni così limitate il numero di configurazioni possibili raggiunge l'ordine delle centinaia di migliaia[8].

In questa applicazione, come è stato già detto, si utilizzano immagini di allenamento delle stesse dimensioni, adeguatamente ricampionate dopo il ritaglio. Inoltre, si utilizzano solamente quattro feature di Haar, due composte da due rettangoli di egual misura e due composte da tre rettangoli uguali affiancati (confrontare la sottosezione 2.2.1 ed in particolare la figura 2.5): ciò genera un pool di feature possibili di poco più di un centinaio di migliaia di configurazioni differenti, per cui i tempi di calcolo complessivi risulteranno ancora ragionevoli.

3.4.2 Procedura di Selezione

La procedura di selezione del miglior weak learner viene richiamata al punto 2b della procedura di allenamento principale.

Sia $\{f_1, \cdots, f_k\}$ l'insieme di tutte le feature selezionabili, $D = \{(x_1, y_1), \cdots, (x_n, y_n)\}$ l'insieme degli esempi di allenamento e $\{w_1, \cdots, w_n\}$ l'insieme dei relativi pesi[9]. La scelta del classificatore debole avviene come descritto dal seguente algoritmo.

1. Si calcolano T^+ e T^-, rispettivamente, somma dei pesi degli esempi negativi e di quelli negativi:
$$T^+ \leftarrow \sum_{i=1}^{n}(w_i y_i) \ , \ T^- \leftarrow \sum_{i=1}^{n}[w_i(1 - y_i)] \tag{3.24}$$
2. *For each* $f \in \{f_1, \cdots, f_k\}$
   (a) Si inizializza una lista di n elementi per memorizzare i valori della feature i-esima applicata ad ogni immagine di allenamento:
$$values[i] \leftarrow f(x_i) \ \forall x_i \in D \tag{3.25}$$

[8] Questo è un ulteriore motivo per voler ridimensionare la dimensione degli esempi di allenamento: la figura umana in una immagine di profondità catturata con il KinectV2 ha dimensioni comprese tra i 120×120 pixel ed i 160×160 pixel. È impossibile valutare tutte le possibili feature in un'area così grande in tempi accettabili.

[9] Essendo richiamata all'interno del loop principale di Adaboost, saranno disponibili i pesi relativi agli esempi di allenamento in riferimento all'iterazione corrente.

(b) Si ordinano gli elementi della lista in *ordine crescente*. Si tenga in conto che, dopo tale operazione, all'i-esima posizione della lista non corrisponderà più il valore della feature applicata all'i-esima immagine di allenamento.

(c) Si inizializzano S^+ ed S^-, con le quali, *scorrendo gli elementi della lista con un cursore, si indicheranno rispettivamente la somma dei pesi degli esempi positivi e di quelli negativi*:

$$S^+ \leftarrow 0, S^- \leftarrow 0 \tag{3.26}$$

(d) *For* $i = [1 : n]$

i. A causa del rilocamento degli indici, alla posizione i-esima della lista corrisponderà il valore della feature dell'elemento x_j con classificazione y_j e peso w_j tale che $(x_j, y_j) \in D$:

$$x_j, y_j, w_j \Leftarrow values[i] \tag{3.27}$$

ii. *If* $y_i = 1$ *Then*
   A. $S^+ \leftarrow S^+ + w_j$

iii. *Else*
   A. $S^- \leftarrow S^- + w_j$

iv. Si calcola l'errore pesato di classificazione (questo passaggio è critico, seguiranno ulteriori commenti e spiegazioni):

$$e_i = \min\left\{\left[S^+ + (T^- - S^-)\right], \left[S^- + (T^+ - S^+)\right]\right\} \tag{3.28}$$

Inoltre, si determinano la polarità ed il valore di soglia:

$$p_i = \begin{cases} 1 & \text{se } S^+ + (T^- - S^-) > S^- + (T^+ - S^+) \\ -1 & \text{altrimenti} \end{cases} \tag{3.29}$$

$$\theta_i = values[i] \tag{3.30}$$

(e) Si determinano la polarità (p_f) e la soglia (θ_f) per cui l'errore pesato (ϵ_f) di classificazione per un classificatore che utilizza la feature f è minimo:

$$(p_f, \theta_f) \leftarrow (p_i, \theta_i) \mid \begin{cases} \epsilon_f = \min\limits_{i \in \{1,\cdots,n\}} \{e_1, \cdots, e_n\} \\ i = \arg\min\limits_{i \in \{1,\cdots,n\}} \{e_1, \cdots, e_n\} \end{cases} \tag{3.31}$$

(f) Si costruisce il weak learner basato sulla feature f più adeguato per la classificazione, fissando i suoi parametri liberi:

$$h_f(x) \leftarrow h(x, f, p, \theta)|_{p=p_f, \theta=\theta_f} \tag{3.32}$$

3. Ottenuti i weak learner $\{h_1(x), \cdots, h_k(x)\}$ (uno per ogni feature), correlati dei rispettivi errori pesati nella classificazione del dataset di allenamento $E = \{\epsilon_1, \cdots, \epsilon_k\}$, si sceglie quello con l'errore pesato più basso, relativamente all'iterazione t-esima del ciclo principale di Adaboost.

$$h_t(x) \leftarrow h_j(x) = h(x, f_j, p_{f_j}, \theta_{f_j}) | j = \arg\min_{j \in \{1,\cdots,k\}} \{\epsilon_1, \cdots, \epsilon_k\} \tag{3.33}$$

La selezione della polarità e della soglia ottimi per il weak learner costruito sulla generica feature f (punto 2(d)iv) è un passaggio particolarmente critico e necessita di ulteriori spiegazioni. Selezionare un valore di soglia vuol dire trovare il *punto che partiziona al meglio la lista dei valori della feature calcolata sulle immagini di allenamento, al fine di minimizzare gli errori di classificazione.*

Si tenga bene a mente che la lista dei valori delle feature viene, innanzi tutto, ordinata in ordine crescente. La miglior soglia di una buona feature fa in modo che la maggior parte delle immagini appartenenti alla stessa classe assumano un valore minore (o maggiore) della soglia stessa.

Scorrendo una sola volta la lista dei valori ($values[1...n]$) della feature associati ad ogni immagine, si è in grado di trovare il valore di soglia. Bisogna effettuare alcune considerazioni sul significato delle somme presentate al punto 2(d)iv:

1. T^+, (T^-) corrisponde alla somma dei pesi degli esempi positivi (negativi)
2. S^+, (S^-) corrisponde alla somma dei pesi degli esempi positivi (negativi) dalla prima posizione fino all'i-esima della lista (quella in cui è posizionato il cursore)
3. $T^+ - S^+$, ($T^- - S^-$) corrisponde alla somma dei pesi degli esempi positivi (negativi) dalla posizione $i+1$ della lista fino alla fine
4. Per un classificatore della forma 2.8 con $p = 1$, S^+ corrisponde alla *somma dei pesi degli esempi classificati correttamente*, mentre $S^- + (T^+ - S^+)$ corrisponde alla somma dei pesi degli esempi classificati in modo scorretto
5. Per l'osservazione 4 un classificatore con $p = 1$, la quantità $S^- + (T^+ - S^+)$ è *l'errore pesato*
6. Analogamente alle osservazioni 4 e 5, un classificatore con $p = -1$ commetterà un errore pesato pari alla quantità $S^+ + (T^- - S^-)$

Scorrendo la lista, si sceglierà come soglia, il valore che minimizza l'errore pesato di classificazione. Il calcolo dell'errore pesato è specificato dall'equazione 3.28 e deriva dalle osservazioni 5 e 6. Inoltre, per grazie alle stesse osservazioni, si ottiene anche la corretta polarità (equazione 3.29).

3.4.3 Valutazione della complessità computazionale

In definitiva, con uno scorrimento della lista ordinata si ottengono i parametri per la costruzione del classificatore debole. La complessità di tale operazione è fortemente legata all'algoritmo di ordinamento della lista, la quale ha $\Theta(n \log n)$ come limite teorico inferiore [1, p. 167]. Nell'implementazione è stato scelto proprio un algoritmo che avesse complessità $O(n \log n)$ nel caso peggiore.

Ripetendo queste operazioni per ognuna delle feature selezionabili, si ottiene che l'algoritmo di selezione del miglior classificatore debole ha complessità $O(kn \log n)$.

Complessivamente, l'intera procedura di allenamento ha complessità computazionale, nel caso peggiore, pari a $O(tkn \log n)$.

Capitolo 4

Validazione e Testing dei Classificatori

Al termine della fase di allenamento si ottengono dei classificatori della forma:

$$F(x) = \begin{cases} 1 & \text{se } \sum_{t=1}^{T} \alpha_t h_t(x) > \theta \sum_{t=1}^{T} \alpha_t \\ 0 & \text{altrimenti} \end{cases} \tag{3.22}$$

La valore di soglia θ, a questo punto, è ancora ignoto e dovrà essere fissato in modo tale da massimizzare le prestazioni del sistema. È necessario, inoltre, impostare il corretto numero di *weak learner* per ogni *strong learner*, al fine di evitare problemi di *overfitting*.

Queste operazioni sono volte alla *validazione* dei vari classificatori e devono essere affiancate da opportune operazioni di *testing* per la valutazione complessiva del sistema.

4.1 Criteri di Valutazione

Applicando i vari classificator ad un set di elementi si può visualizzare la distribuzione delle istanze degli oggetti rispetto alla classificazione predetta dai classificatori e rispetto a quella reale.

Si utilizza la *matrice di confusione* per descrivere in modo compatto tale distribuzione.

Definizione 2 (Matrice di Confusione)**.** In un generico problema di classificazione ad n classi per il quale è definito un insieme di *label* $L = \{l_1, ..., l_n\}$ ed è stato allenato un classificatore $F(x)$, sia $T = \{(x_1, y_1), ..., (x_m, y_m)\}$ un insieme di elementi per cui è nota la reale classificazione.

Si chiama *matrice di confusione* del classificatore F e relativa all'insieme di elementi T, la matrice $C \in \mathbb{N}^{n \times n}$, per cui ogni elemento c_{ij} è pari al numero di oggetti dell'insieme T la cui reale etichettatura corrisponde a l_i e per i quali il classificatore $F(x)$ prevede l'etichettatura l_j.

$$c_{ij} = \#(\{x | (x, l_i) \in T \wedge F(x) = l_j\}) \tag{4.1}$$

Gli elementi sulla diagonale principale di tale matrice denotano il numero di elementi per cui la predizione fornita dal classificatore è corretta. Infatti, nel caso in cui $i = j$, la

relazione 4.1 diventa:

$$c_{ii} = \#(\{x|(x, l_i) \in T \wedge F(x) = l_i\}) \Longrightarrow c_{ii} = \#(\{x|(x, F(x)) \in T\}) \tag{4.2}$$

Nel caso di classificazione dicotomica, la struttura della matrice di confusione è molto semplice:

$$C = \begin{pmatrix} \text{True Positives} & \text{False Negatives} \\ \text{False Positives} & \text{True Negatives} \end{pmatrix} \tag{4.3}$$

4.1.1 Modalità di Test

La scelta del set di elementi con i quali testare i classificatori, in fase di valutazione, è particolarmente importante. Esistono alcune alternative:

Training Dataset Si utilizza lo stesso dataset impiegato per allenare il costruttore. È la soluzione più semplice, ma presenta notevoli svantaggi, in quanto diventa impossibile rilevare ed evitare problemi di *overfitting*.

Validation Dataset Si crea un dataset da utilizzare esclusivamente per il testing o per la validazione che non contenga elementi dell'insieme di allenamento.

Cross Validation Questa tecnica prevede l'utilizzo di un unico dataset, utilizzato sia per l'allenamento che per la valutazione:

1. Si suddivide l'insieme in k sottoinsiemi, detti *k-fold*[1]
2. Il classificatore viene allenato utilizzando $k - 1$ sottoinsiemi e viene testato utilizzando il sottoinsieme rimanente. Questa operazione viene eseguita k volte.
3. La media delle singole prestazioni ottenute nei k esperimenti costituisce le prestazioni complessive del sistema.

Split Si suddivide l'insieme di dati disponibili in due set distinti, uno da utilizzare per l'allenamento, l'altro - solitamente più piccolo del primo - per la validazione e per il testing.

Per la validazione e per il testing è stato creato un dataset apposito, estratto da registrazioni completamente diverse da quelle utilizzate per i dataset di allenamento. Le porzioni di immagine di profondità che ritraggono la figura della persona, in questo dataset, provengono dalle registrazioni in cui i vari soggetti percorrono delle traiettorie a loro piacimento all'interno dell'area d'interesse.

Il dataset di validazione è costituito da 1500 elementi positivi (immagini che ritraggono persone) e da 1600 elementi negativi.

[1] Si è notato che, sperimentalmente, sono sufficienti una decina di *fold*.

4.1.2 Misure per Valutare le Prestazioni

Scegliere i criteri di valutazione della bontà delle previsioni fornite dai classificatori è un punto cruciale e non banale: l'utilizzo di una misura piuttosto che un'altra dipende dall'obiettivo che si vuole raggiungere.

Per i problemi di classificazione binaria sono disponibili le seguenti misure per valutarne le prestazioni.

Precision Porzione di predizioni positive corrette.

$$PR = \frac{TP}{TP + FP} \tag{4.4}$$

Sensitiviy o Recall Porzione delle istanze realmente positive classificate correttamente. Per P si intende il numero totale di istanze realmente positive.

$$TPR = \frac{TP}{P} = \frac{TP}{TP + FN} \tag{4.5}$$

Specificity Porzione di istanze realmente negative classificate correttamente. Per N si intende il numero totale di istanze negative.

$$TNR = \frac{TN}{N} = \frac{TN}{TN + FP} \tag{4.6}$$

False Positive Rate Porzione di istanze realmente negative classificate erroneamente come positive.

$$FPR = \frac{FP}{N} = \frac{FP}{TN + FP} = 1 - TNR \tag{4.7}$$

Accuracy Porzione di istanze, sia positive che negative, classificate correttamente.

$$ACC = \frac{TP + TN}{P + N} = \frac{TP + TN}{TP + TN + FP + FN} \tag{4.8}$$

F1 measure Media armonica tra *precision* e *recall*.

$$F1 = \frac{2PR \cdot TPR}{PR + TPR} = \frac{2TP}{2TP + FP + FN} \tag{4.9}$$

F1 measure e *accuracy* sono le misure più indicate per la valutazione complessiva dei classificatori, mentre le altre forniscono informazioni parziali.

Solitamente si utializzano *precision*, *recall* e *F1 measure*, nei problemi in cui, tra le due classi, una è molto meno interessante dell'altra, oppure nei casi in cui si vuole studiare l'andamento del classificatore nelle predizioni di una classe di oggetti in particolare.

D'altra parte, l'*accuracy* viene utilizzata quando tutte le classi ricoprono una posizione di egual interesse. In particolare, nei problemi binari, tale misura è una adeguata fonte di valutazione quando le due classi sono bilanciate.

In questa applicazione, poichè le classi e la distribuzione degli elementi di allenamento rispettano i criteri di quest'ultima misura, verrà utilizzata l'*accuracy* come misura di valutazione delle prestazioni.

4.2 Massimizzazione all'*Accuracy*

4.2.1 Parametri liberi del classificatore

I classificatori allenati con Adaboost non sono ancora pronti per poter essere utilizzati, vi sono ancora dei parametri liberi da vincolare. Dal momento che l'obiettivo è ottenere un sistema di rilevamento che sia più accurato possibile, questi parametri devono essere impostati in modo tale che l'*accuracy* sia massima.

Numero di weak learner

Il numero di weak learner[2] per ogni classificatore forte è un parametro libero all'interno della procedura di allenamento. Vista la procedura di Adaboost nel capitolo precedente, potrebbe sembrare che, aggiungendo altri weak learner si possa migliorare progressivamente la precisione del sistema. Ciò generalmente non è vero: troppi classificatori deboli potrebbero generare problemi di *overfitting*. È necessario quindi ricavare il numero ottimale di classificatori deboli a comporre ciascun classificatore forte.

Soglia del classificatore

Il valore di soglia relativo a ciascun classificatore forte allenato non è mai stato precisato. È necessario quindi impostare tale valore al fine di migliorare complessivamente l'affidabilità del sistema.

4.2.2 Algoritmo di ricerca della soglia e del NWL ottimi

I parametri da definire sono quindi il numero di classificatori deboli ed il valore di soglia per ciascuno dei classificatori forti allenati. Entrambi vengono scelti, per ogni classificatore forte, al fine di massimizzarne l'accuratezza in relazione al dataset di validazione. Ciò implica la massimizzazione del numero di oggetti che vengono classificati correttamente e la conseguente riduzione dei falsi positivi e negativi.

L'algoritmo per determinare tali valori è fondamentalmente semplice: ogni strong learner viene utilizzato per classificare gli elementi del *dataset di validazione*. L'attività di classificazione, in virtù dei parametri liberi appena identificati, sarà dipendente dalla coppia di valori (θ, N_{wl}) [3]. Per ogni possibile coppia di parametri, viene misurata l'accuratezza del classificatore nel classificare correttamente l'intero dataset, quindi viene scelta la coppia per cui l'accuratezza è massima.

Una procedura di questo tipo, se implementata senza adoperare dei piccoli accorgimenti, potrebbe richiedere molto tempo per essere portata a termine. Di seguito viene presentato un metodo che permette di classificare molto velocemente il dataset di validazione utilizzando diversi valori per la coppia (θ, N_{wl}).

1. Si definisce il classificatore forte $F(x)$ (equazione 3.22) costituito dai primi T weak learner[4], selezionati da Adaboost.

[2] Abbreviando: NWL

[3] Per θ si intende il valore di soglia del classificatore e N_{wl} il numero di weak learner che lo compongono

[4] In questa applicazione vengono estratti i primi 200 classificatori deboli. Questi ultimi sono il limite superiore al numero di weak learner totali che potranno essere utilizzati.

2. Si definisce l'insieme $V = (x_1, y_1), ..., (x_m, y_m)$ di validazione.

3. Si costruisce la matrice di valutazione dei weak learner W. L'elemento alla posizione (i, j) di tale matrice sarà la valutazione del *j-esimo weak lerner* sul *i-esimo elemento del dataset*.

$$W = \begin{bmatrix} h_1(x_1) & h_2(x_1) & \cdots & h_j(x_1) & \cdots & h_T(x_1) \\ h_1(x_2) & h_2(x_2) & \cdots & h_j(x_2) & \cdots & h_T(x_2) \\ \vdots & \vdots & & \vdots & & \vdots \\ h_1(x_i) & h_2(x_i) & \cdots & h_j(x_i) & \cdots & h_T(x_i) \\ \vdots & \vdots & & \vdots & & \vdots \\ h_1(x_m) & h_2(x_m) & \cdots & h_j(x_m) & \cdots & h_T(x_m) \end{bmatrix} \tag{4.10}$$

   Poichè $h_j(x_i) : V \to \{0, 1\}$, allora tale matrice sarà composta solamente da 0 ed 1 (con un piccolo abuso di notazione: $W \in \{0, 1\}^{m \times T}$).

4. Si costruisce una matrice triangolare superiore con i moltiplicatore dei weak learner, disposti come di seguito:

$$A = \begin{bmatrix} \alpha_1 & \alpha_1 & \cdots & \alpha_1 & \cdots & \alpha_1 \\ 0 & \alpha_2 & \cdots & \alpha_2 & \cdots & \alpha_2 \\ \vdots & \vdots & \ddots & \vdots & & \vdots \\ 0 & 0 & \cdots & \alpha_i & \cdots & \alpha_i \\ \vdots & \vdots & & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & 0 & \cdots & \alpha_T \end{bmatrix} \in \mathbb{R}^{T \times T} \tag{4.11}$$

5. Si calcola il prodotto della matrice W e della matrice A. L'elemento alla posizione (i, j) della matrice risultante, sarà il *valore della combinazione lineare dei primi j weak learner, valutati sul i-esimo elemento del dataset di validazione.*

$$\forall a_{ij} \in (W \cdot A), a_{ij} = \sum_{t=1}^{j} a_t h_t(x_i) \tag{4.12}$$

6. Si definiscono i possibili valori di soglia assumibili dal classificatore forte. Qui sono stati utilizzati tutti i valori da 0 ad 1, con un passo pari a 0.01, estremi inclusi[5].

$$\Theta = \begin{bmatrix} \theta_1 \\ \theta_2 \\ \vdots \\ \theta_l \end{bmatrix} \in \mathbb{R}^l \tag{4.13}$$

[5] Quindi i possibili valori di soglia sono 101: $\{0, 0.01, 0.02, ..., 0.98, 0.99, 1\}$

7. Si definisce un array con la somma cumulativa dei fattori moltiplicativi dei weak learner:

$$\widetilde{A} = \begin{bmatrix} \alpha_1 \\ \alpha_1 + \alpha_2 \\ \vdots \\ \sum_{t=1}^{T} \alpha_t \end{bmatrix} \in \mathbb{R}^T \tag{4.14}$$

8. *For* $i = [1...l]$ (ovvero, per ogni possibile valore di soglia):
   (a) *For* $t = [1...T]$:
      i. Si inizializzano $TP = TN = FP = FN = 0$.
      ii. *For* $k = [1...m]$ (cioè, per ogni elemento del dataset di validazione):
         A. Si classifica velocemente l'elemento utilizzando le strutture dati preparate in precedenza:

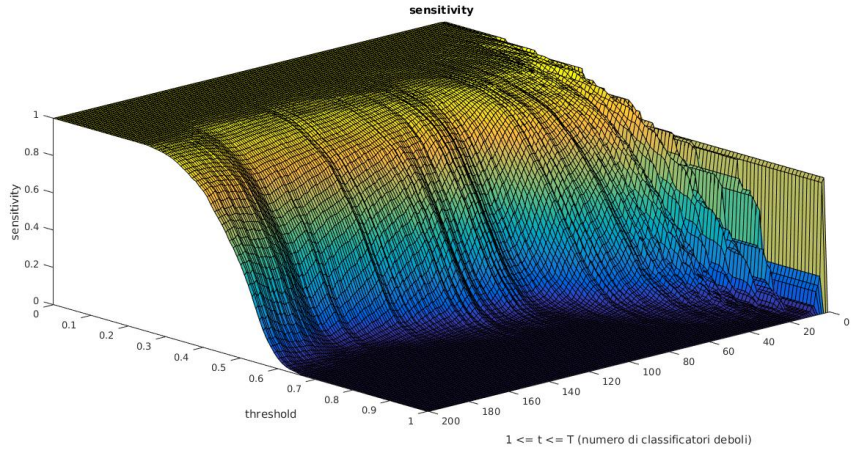
$$F(x_k) = \begin{cases} 1 & \text{se } a_{kt} > \theta_i \widetilde{a_t} \\ 0 & \text{altrimenti} \end{cases} \tag{4.15}$$

         B. Si confronta tale classificazione con la classificazione reale e si aggiornano i contatori:
            - $F(x_k) = y_k = 1 \Longrightarrow TP \leftarrow TP + 1$
            - $F(x_k) = y_k = 0 \Longrightarrow TN \leftarrow TN + 1$
            - $F(x_k) = 1 \wedge y_k = 0 \Longrightarrow FP \leftarrow FP + 1$
            - $F(x_k) = 0 \wedge y_k = 1 \Longrightarrow FN \leftarrow FN + 1$
         C. Si salvano i valori TP, TN, FP, FN relativi alla coppia (θ_i, N_{wl}) (con $N_{wl} = t$), organizzandoli in una struttura dati a piacere, purchè in un secondo momento si sia in grado di risalire esattamente al valore di soglia e al numero di weak learner utilizzati.
            In questo caso, ad esempio, sono state utilizzate 4 matrici distinte, formate da tante colonne quanti sono i weak learner totali e da tante righe quanti sono i possibili valori di soglia.

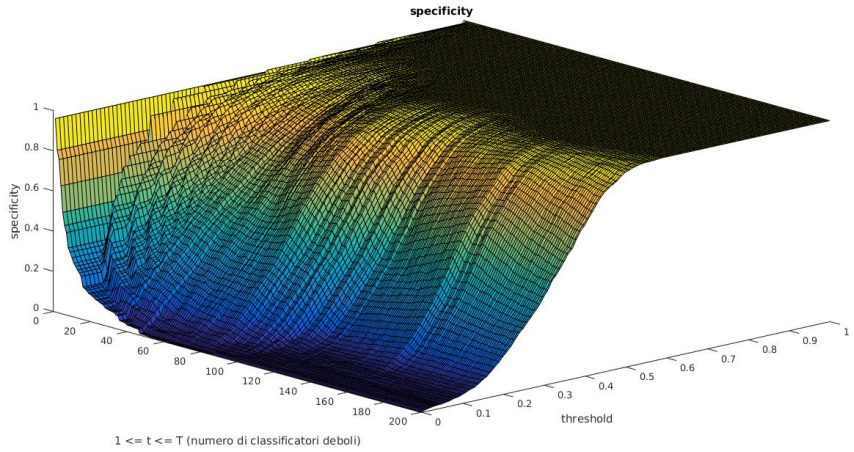
9. Per ogni coppia (θ, N_{wl}) calcolare l'*accuracy* (formula 4.8). Si seleziona la coppia di valori per i quali l'accuratezza del classificatore è massima.

4.3 Analisi dei Risultati

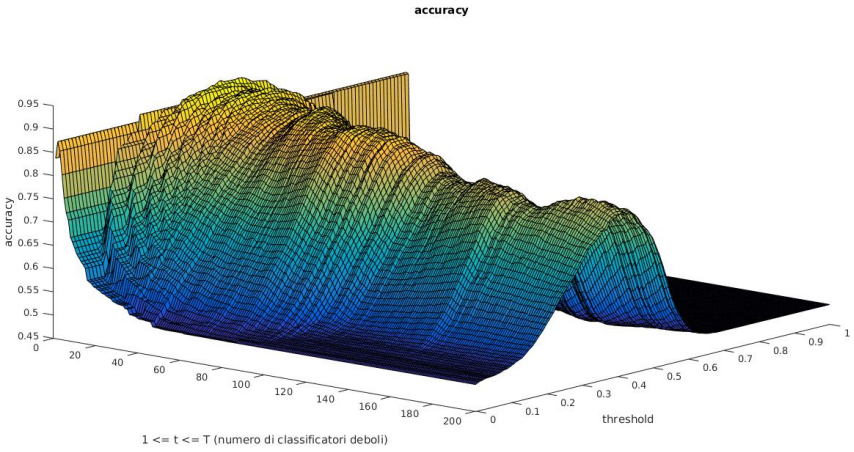
Al termine della procedura di selezione della coppia ottima, si ottiene un classificatore la cui accuratezza è la migliore rispetto a tutte le altre possibili coppie di parametri.



(a)



(b)



(c)

Figura 4.1: Plot tridimensionale dell'andamento di sensitivity *(4.1a),* specificity *(4.1b) e* accuracy *(4.1c) al variare del valore di soglia e del numero di weak learner di* F_{hor}.

4.3.1 Soglia del classificatore

In letteratura il valore di soglia di un classificatore forte si assume essere pari a 0.5, come descritto da Freund e Schapire in [2], Viola e Jones in [9].

Nel lavoro presentato da Zhu e Wong in [11], tuttavia, il valore di soglia non viene esplicitato, ma viene considerato un parametro libero, pur senza fornire alcun metodo esplicativo per la sua derivazione.

Il valore ottenuto tramite l'algoritmo di massimizzazione dell'accuratezza dei classificatori (sezione 4.2) si aggira intorno al valore 0.4 per entrambi (F_{hor} e F_{ver}).

Una prima interpretazione potrebbe essere la seguente: la discordanza, seppur contenuta, tra il valore di soglia presente in letteratura ed il valore calcolato può essere dovuto a problemi di overfitting al dataset di validazione. Senza dover creare un ulteriore dataset esclusivamente per il testing, potrebbe essere utile utilizzare il valore 0.5 come soglia effettiva dei classificatori finali, eliminando l'overfitting, e considerare le soglie calcolate come delle controprova della correttezza dell'algoritmo di allenamento.

La seconda interpretazione, invece, è diametralmente opposta: l'aver ottenuto dei valori di soglia ottimi per accuratezza, potrebbe essere un valore aggiunto rispetto alla formulazione generale classica dei classificatori forti.

Per il momento si assumeranno validi i valori ottimi di soglia appena calcolati, tuttavia sarà solo in fase di rilevamento su frame reali che ci si potrà rendere conto della bontà di tale soluzione.

4.3.2 Numero di Weak Learner

Il numero dei classificatori deboli che compongono ciascun classificatore forte, è molto differente da quello ottenuto in altre applicazioni, prima tra tutte il sistema di riconoscimento dei volti di Viola Jones.

I classificatori forte di questo sistema sono formati da una decina di classificatori deboli, un numero straordinariamente piccolo se confrontato con i migliaia di weak learner necessari a costruire un classificatore di volti.

Tuttavia è necessario considerare la diversa natura dei due problemi e i diversi canali di acquisizione dei dati per il riconoscimento. L'immagine di profondità di una persona ripresa dall'alto è una rappresentazione grezza, ma essenziale, dell'individuo ed è incredibilmente povera di informazione se paragonata all'immagine di un volto umano. La ricchezza di particolari di un volto, l'incredibile variabilità intraclasse dei volti, per non parlare della fortissima sensibilità delle immagini alle variazioni ambientali (come la luce), rende molto più complesso il problema di riconoscimento. Ecco perchè i classificatori di volti sono cento, mille volte più complessi.

Per raggiungere elevate prestazioni con il framework di Viola-Jones, infatti, è necessario dividere un classificatore forte in diversi stadi, creando dei classificatori a cascata in grado di scartare nel minor tempo possibile, il maggior numero di oggetti che non corrispondono ai volti umani.

In questo sistema di rilevamento, tutto questo non è necessario. Un numero così basso di classificatori deboli non è dovuto ad un errore, ma è la conseguenza dell'estrema essenzialità delle immagini di profondità dei profili umani.

Come ulteriore argomentazione, si può pensare alle caratteristiche espresse in linguaggio naturale dell'immagine del profilo della persona: sono solamente tre. Quante ne sarebbero necessarie per esprimere le caratteristiche di un volto umano?

4.3.3 Sensitivity, Specificity, Accuracy

In base a quanto presente in letteratura, un buon sistema di rilevamento presenta dei valori di *sensitivity* intorno al 90% e di *specificity* intorno al 70%.

Ovviamente tali criteri di giudizio dipendono molto dall'applicazione finale del sistema. Esisteranno situazioni più critiche che richiederanno parametri più stringenti, ma in fase sperimentale ci si attiene a questi valori empirici.

I valori di *sensitivity*, *specificity* e *accuracy* complessivi[6], calcolati una volta che i parametri dei classificatori sono stati fissati in condizioni di massima accuratezza, sono i seguenti:

$$TPR = 96.87\% \tag{4.16}$$

$$TNR = 91.25\% \tag{4.17}$$

$$ACC = 93.87\% \tag{4.18}$$

Alla luce dei risultati ottenuti, il classificatore finale, testato sul dataset di validazione, risulta sufficientemente accurato rispetto ai valori empirici di riferimento.

[6]In riferimento al classificatore finale. I singoli classificatori forti, allenati con Adaboost, vengono utilizzati in parallelo in modo da formare uno unico.

Capitolo 5

Rilevamento

I classificatori, ottenuti in fase di allenamento e parametrizzati in fase di validazione, non sono direttamente utilizzabili per il rilevamento delle persone all'interno di un frame di profondità. Basti pensare al fatto che ogni classificatore viene allenato con immagini di 24×24 pixel, di conseguenza, tutti i meccanismi sviluppati dall'algoritmo di apprendimento per il rilevamento, sono indissolubilmente legati a tale dimensione.

Nel capitolo 2, introducendo le feature di Haar, è stato necessario modificarne la formula di calcolo al fine di minimizzare la sensibilità rispetto ai ridimensionamenti della feature stessa. Il motivo per cui è necessario ridimensionare una feature è esattamente questo: i classificatori vengono allenati con immagini di profondità molto piccole, ma vengono trasformati in blocco per riuscire ad elaborare immagini di dimensione maggiore.

Dopo tale trasformazione, i classificatori riusciranno comunque ad elaborare delle porzioni di immagini di profondità che al massimo potranno contenere una persona: bisognerà tener conto anche di tale aspetto.

5.1 Tecnica di Rilevamento

Vengono create delle *detection window* per gestire il rilevamento su frame. La detection window non è altro che una finestra, una porzione dell'immagine di profondità che viene data in pasto ai classificatori per determinare se essa contiene o meno l'immagine di una persona.

Per fare ciò è necessario, innanzi tutto, ridimensionare correttamente l'intero classificatore in modo tale da rendere la dimensione delle immagini in input compatibili con la dimensione delle figure HASP nel frame di profondità.

È stato accennato in precedenza che l'immagine HASP della persona, in un frame ripreso con il Kinect V2, viene circoscritta da un'area quadrata di dimensione variabile tra i 120×120 e i 160×160 pixel, dipendentemente dalla statura e dalla corporatura del soggetto.

Viene presentato, di seguito, l'algoritmo per il ridimensionamento dei classificatori forti. Si tenga presente che, al livello di strutture dati, uno strong learner non è altro che una collezione di weak learner correlata dei relativi fattori moltiplicativi e da un valore di soglia. A loro volta, i weak learner non sono altro che delle feature di haar (tipo della feature, cordinate dell'area rettangolare che racchiude interamente la feature), correlate da un valore di soglia e di una polarità. Alla luce di tali considerazioni, ridimensionare

un classificatore non vuol dire altro che ridimensionare le feature relative ai weak learner a lui associati.

1. *For each* weak learner $w_i \in \Phi = \{w_1, \cdots, w_k\}$:

   (a) Sia f la feature associata al weak learner w_i. Quest'ultima è racchiusa in una regione rettangolare R, esprimibile, rispetto al sistema di coordinate convenzionale delle immagini raster, in base alle coordinate del punto in alto a sinistra (*Top Left Point*) e alle dimensioni del rettagolo (*width* e *height*).

   $$R := (TLP, DIM) \text{ con } TLP = (tl_x, tl_y) \text{ e } DIM = (w, h) \tag{5.1}$$

   (b) Siano $W_w^{'}$ e $W_h^{'}$ le nuove dimensioni desiderate per la finestra di classificazione. Essendo la finestra quadrata, si ha $W_w^{'} = W_h^{'} = W^{'}$.

   (c) Si calcola il rapporto di ridimensionamento (si tenga a mente che il lato della finestra di classificazione originale è di 24 pixel):

   $$\sigma = \frac{W^{'}}{W} = \frac{W^{'}}{24} \tag{5.2}$$

   (d) Si modifica la feature f cambiando la posizione e la dimensione della regione rettangolare associata in modo da mantenere la proporzione con le nuove dimensioni desiderate[1]:

   $$R^{'} := \begin{cases} tl_x^{'} = \lfloor \sigma tl_x \rfloor \\ tl_y^{'} = \lfloor \sigma tl_y \rfloor \\ w^{'} = \lfloor \sigma w \rfloor \\ h^{'} = \lfloor \sigma h \rfloor \end{cases} \tag{5.3}$$

   (e) Ad ogni feature è associato un tipo che ne identifica la forma. Si è parlato dei diversi tipi di feature nella sottosezione 2.2.1: ad ognuno di essi è associato un vincolo di divisibilità rispetto una delle due dimensioni. Bisogna correggere le dimensioni della regione R' affinchè questo vincolo venga soddisfatto (si fa riferimento alla figura 2.4.

      i. se f è del tipo 2.4[1.a], allora:

      $$w^{'} \leftarrow (w^{'} - \mod (w^{'}, 2)) \tag{5.4}$$

      ii. se f è del tipo 2.4[1.b], allora:

      $$h^{'} \leftarrow (h^{'} - \mod (h^{'}, 2)) \tag{5.5}$$

      iii. se f è del tipo 2.4[2.a], allora:

      $$w^{'} \leftarrow (w^{'} - \mod (w^{'}, 3)) \tag{5.6}$$

[1] La notazione $\lfloor x \rfloor$ denota la parte intera di x.

iv. se f è del tipo 2.4[2.c], allora:

$$h^{'} \leftarrow (h^{'} - \quad \text{mod } (h^{'}, 3)) \tag{5.7}$$

La possibilità di ridimensionare i classificatori risulta essere un vantaggio per quanto riguarda la scalabilità dell'applicazione. Attualmente una finestra di classificazione ha una dimensione statica, determinata empiricamente per riuscire a rilevare adeguatamente persone di corporatura e statura differenti. Tuttavia il sistema può essere migliorato andando a far variare la dimensione della finestra per garantire rilevamenti migliori.

Una volta ottenuto un classificatore adattabile alla dimensione desiderata per la finestra di classificazione, è sufficiente far scorrere tale finestra in tutto il frame comprendone tutta l'aria, elaborando sequenzialmente tutto il frame. Ciò permette di effettuare delle rilevazioni in contesti statici, ovvero su singoli frame o su frame non correlati tra di loro.

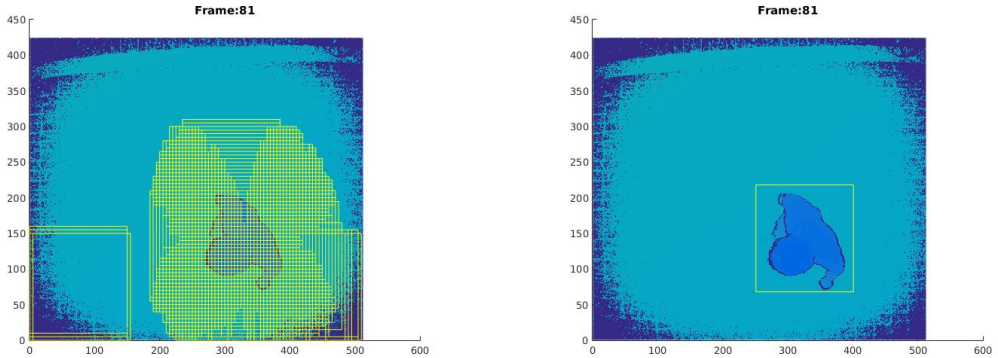
Quando si ha a che fare con frame correlati tra di loro, è possibile velocizzare il processo di rilevamento scartando delle intere aree del frame.

Quando valgono le precedenti ipotesi, infatti, dopo un'iniziale scansione totale del frame, l'algoritmo di rilevamento prende coscenza dell'eventuale presenza di persone e conosce a priori quali sono le zone in cui sicuramente, negli istanti successivi, tutte le finestre di rilevamento daranno esito negativo.

Negli istanti successivi, vengono scansionate solamente le porzioni che si trovano nelle vicinanze della persona rilevata all'istante precedente. Si scansionano inoltre i bordi del frame, alla ricerca di eventuali persone in ingresso. Utilizzando questi accorgimenti, i tempi necessari al rilevamento su frame si accorciano notevolmente.

5.2 Selezione della Finestra Migliore

Solitamente, scansionando un frame alla ricerca di una persona, in prossimità di quest'ultima, saranno numerose le finestre di rilevamento che daranno esito positivo. Bisogna, di contro, elaborare un meccanismo di selezione della finestra migliore, che circoscriva adeguatamente l'area entro cui la persona si trova realmente.



(a) *(b)*

Figura 5.1: Rilevamento senza selezione della finestra migliore 5.1a e con selezione 5.1b

Il problema non è banale, allo stato attuale del sistema rappresenta a tutti gli effetti un collo di bottiglia per quanto riguarda la scalabilità.

In prima approssimazione è stato utilizzato un algoritmo rudimentale basato sull'individuare il centro di una serie di finestre positive. Il risultato, seppur grezzo, è accettabile, grazie al bassissimo numero di falsi positivi prodotti dai classificatori.

In un secondo momento le finestre sono state pesate con valori decrescenti man mano che queste si allontanano dalla posizione della persona allo stato precedente. Finestre positive troppo distanti, difficilmente portano a risultati veritieri, quindi vengono premiate le finestre di rilevamento nelle immediate vicinanze della persona. Viene inoltre effettuata una prima scrematura, non considerando rilevamenti che hanno prodotto un numero di finestre positive troppo basso.

Lo svantaggio delle soluzioni appena presentate sta nel fatto che funzionano quando nel frame è presente una sola persona. Sarà necessario in futuro, per apportare dei miglioramenti, implementare una soluzione simile a quella descritta da Wang in [10], basata sul raggruppamento delle finestre in insiemi di aree connesse tra loro e sulla selezione di quelle con un maggiore indice di sovrapposizione.

Capitolo 6

Conclusioni

A partire da quanto descritto da Zhu e Wong in [11] è stata elaborata una semplice implementazione del loro sistema di human sensing.

Nei capitoli precedenti sono state largamente commentate le molteplici somiglianze con il sistema di *face recognition* di Viola e Jones: l'uso dello stesso algoritmo di allenamento, della stessa famiglia di feature e quello di alcuni parametri caratteristici, come la dimensione delle immagini di allenamento.

L'impiego di immagini di profondità, che forniscono una rappresentazione essenziale della realtà, fondamentalmente priva della ricchezza di dettagli riscontrabile in immagini RGB, comporta l'elaborazione di classificatori molto più semplici rispetto a quelli per il riconoscimento dei volti e quindi, a differenza di questi ultimi, non necessitano di particolari accorgimenti per velocizzare l'esecuzione.

La valutazione dell'accuratezza del sistema ed il confronto dei risultati ottenuti con quelli esposti in letteratura, confermano la correttezza dell'implementazione e la complessiva qualità del sistema.

Rimangono migliorabili le prestazioni in fase di rilevamento. La logica utilizzata per la selezione delle finestre di rilevamento ottimali ha validità limitata al caso in cui è presente solo una persona nell'area del frame, il che costituisce un limite importante, ma raggirabile implementando una logica di selezione migliore.

Appendici

Appendice A

Gestore dei Dataset

Si tratta di un'applicazione desktop che consente di gestire efficacemente i vari dataset. Una volta aperto, nella schemata principale (figura A.1) gli elementi vengono organizzati in una tabella che ne riporta l'id numerico, le dimensioni e l'etichettatura reale.

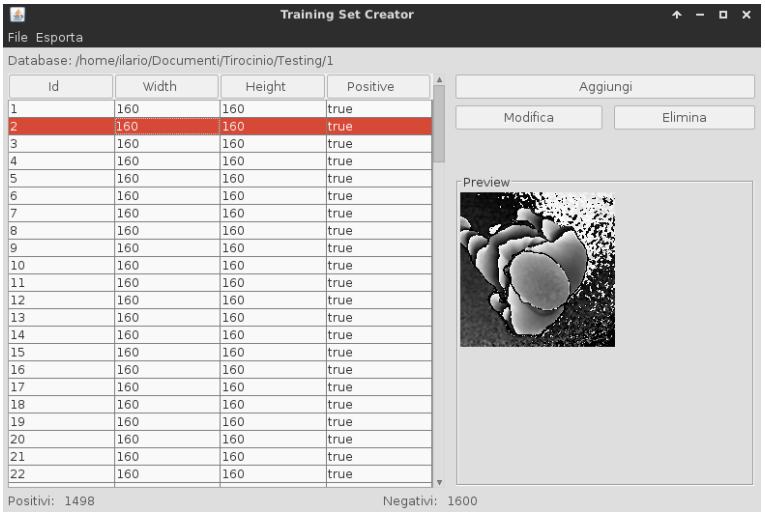


Figura A.1: Schermata principale del software di gestione.

Di ogni elemento, una volta selezionato, è disponibile una piccola preview. Inoltre sono disponibili le elementari azioni di modifica (che consiste nel cambiare l'etichettatura dell'elemento in caso di errore) e cancellazione per ognuno di essi.

Premendo il bottone *Aggiungi* si attiverà la procedura per l'aggiunta degli elementi al dataset. Si aprirà una schermata in cui verrà richiesto di selezionare la cartella contenente le registrazioni effettuate con il Kinect. Poichè ogni frame viene salvato in un file binario privo di qualsiasi metadato, non è possibile risalire alla risoluzione originale del frame e sarà necessario specificarla manualmente.

Inseriti i dati necessari si aprirà l'interfaccia in figura A.2 che permetterà di scorrere tra i frame della registrazione utilizzando la scrollbar in alto e selezionare la porzione di frame desiderata evidenziandola con il mouse, così come si farebbe in un qualsiasi programma di photo editing.

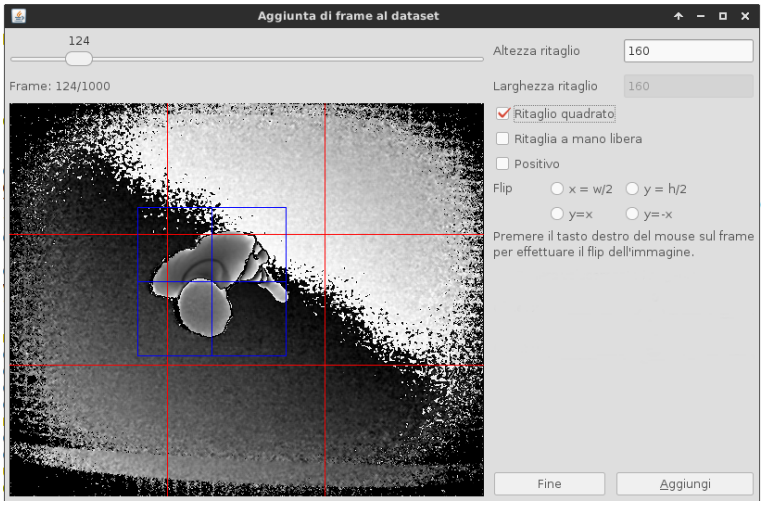


Figura A.2: Schermata per l'aggiunta di un nuovo elemento al dataset.

È disponibile anche la funzionalità per il resize delle immagini, raggiungibile dalla voce *Resize* del menu *Esporta*.

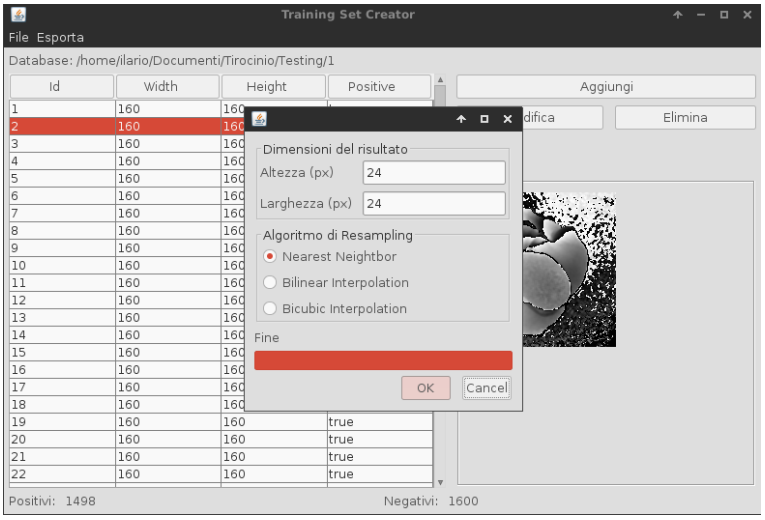


Figura A.3: Schermata per il resize degli elementi del dataset.

Nella schermata in figura A.3 sarà necessario inserire le nuove dimensioni per le immagini e selezionare l'algoritmo di resampling desiderato.

Il software è stato scritto in linguaggio Java, utilizzando il pattern *Model View Controller*. Questo approccio mira a dividere il codice in blocchi funzionali ben distinti, ognuno con un obiettivo ben specifico:

Model Le classi che lo compongono implementano la *business logic* dell'applicazione, ovvero si occupano della modellazione degli oggetti della realtà di interesse in oggetti software e descrivono le operazioni eseguibili su di essi.

View Tutte le interfacce grafiche vengono raggruppate in questo modulo software. Vengono implementati i metodi per il popolamento delle viste e l'estrazione dei valori inseriti nei campi di input dall'utente.

Controller Le classi di controllo si frappongono tra quelle del *model* e quelle della *view*, governando i meccanismi di rappresentazione dei dati all'utente e la loro modifica a fronte di un'azione intrapresa da quest'ultimo.

Questo approccio alla scrittura del software permette la realizzazione di un prodotto modulare, facilmente manutenibile ed espandibile.

Per la persistenza dei dati e soprattutto dei metadati relativi agli elementi dei dataset, si è scelto di non utilizzare alcun meccanismo di persistenza centralizzato (*DBMS* o file *XML*), ma di rappresentare i metadati codificandoli nei nomi dei file.

Ogni dataset, infatti, risiede in una specifica cartella. Al suo interno saranno presenti le sotto cartelle *positives* e *negatives*, che organizzeranno il file con i ritagli delle immagini di profondità in base alla loro reale classificazione. Il nome di ogni ritaglio è composto da una serie di parametri separati dal carattere *underscore* e serializza lo stato del relativo oggetto software che lo modella all'interno dell'applicazione (A.1).

$$nome_{sample} = sample\_ < id > \_ < width > \_ < height > \_ < label > .bin \qquad \text{(A.1)}$$

Appendice B

Implementazione dell'Algoritmo di Allenamento

Il linguaggio Matlab permette di sviluppare software molto velocemente grazie alla miriade di funzionalità messe a disposizione dalle librerie standard, che consentono di tralasciare molti dettagli implementativi e concentrare gli sforzi sullo sviluppo della logica di business. Tuttavia la fase di allenamento è la più pesante dal punto di vista computazionale, sia per la complessità intrinseca dell'algoritmo, che per la mole di dati in ingresso, quindi richiede un approccio differente.

In Matlab alcune funzioni possono essere implementate in un altro linguaggio ad alto livello (come il C++, il Java o il FORTRAN) attraverso l'uso dei *file mex*, al fine di migliorne le prestazioni. Una volta scritto il file sorgente della funzione nel linguaggio scelto, lo si compila utilizzando il compilatore in dotazione con la suite Matlab, generando così il file mex che verrà linkato dinamicamente in fase di esecuzione.

È stata quindi sviluppata una libreria di supporto in C++ che implementa le parti più critiche dell'algoritmo di allenamento. Tale liberia viene poi utilizzata nelle funzioni implementate nei file mex e queste ultime verranno richiamate negli script in Matlab.

```
// [...]
#include "adaboost.h"
#include "mexutils.h"

// [...]

/**
 * Main function
 */
void mexFunction(int outc, mxArray *output[],
    int inc, const mxArray *input[])
{
    // [...]
    bestWeakClassifier = Adaboost::bestWeakClassifier(
        samples, features, values);
    // [...]
```

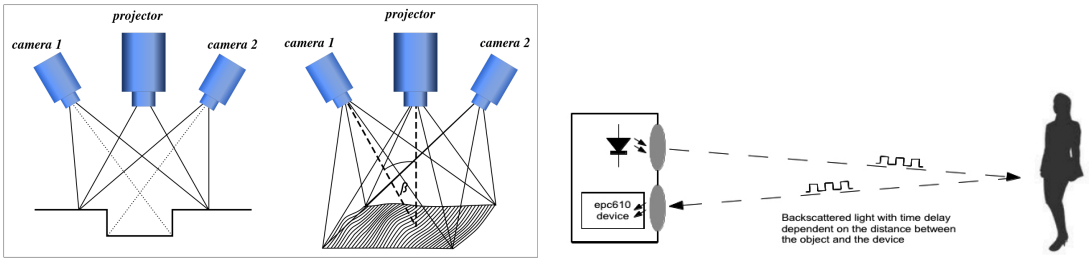
```
}

// [...]
```

In questa piccola porzione di codice, relativa alla funzione compilata per la selezione del miglior weak learner, mostra l'utilizzo del metodo *bestWeakClassifier* della classe *Adaboost* definita in *adaboost.h*.

Appendice C

Accenni sul Funzionamento e le Caratteristiche del Sensore del Kinect

Ciò che viene comunemente chiamato *sensore di profondità* o *sensore di distanza*, trova due differenti realizzazioni, a seconda della versione del Kinect.



(a) *(b)*

Figura C.1: Rappresentazione schematica di uno scanner a luce strutturata (C.1a) e di una TOF camera *(C.1b).*

Il Kinect V1 utilizza uno *scanner 3D a luce strutturata*, ovvero un dispositivo in cui una sorgente di raggi infrarossi proietta una serie di pattern codificati nello spazio. Le superfici colpite inducono una deformazione nella struttura di tali pattern, che vengono quindi catturati da una o più telecamere, che confrontando la deformazione con il pattern originario. Da questo confronto riescono a ricostruire, dalla rappresentazione bidimensionale di ogni punto nello spazio, le sue coordinate tridimensionali.

Il Kinect V2 invece utilizza una *time of flight camera*: vengono inviati degli impulsi luminosi, quindi viene misurato il tempo che intercorre tra l'invio del segnale e la ricezione della radiazione di ritorno. Dalla misura di questo intervallo (che altro non è che il tempo necessario alla luce per raggiungere l'oggetto e tornare indietro per il fenomeno della *riflessione diffusa*) viene ricavata la distanza alla quale si trova il punto. Ripetendo questo procedimento si è in grado di costruire una rappresentazione tridimensionale dello spazio.

Il risultato di un sensore di questo tipo è un insieme di triplette (x, y, z), organizzate in una *immagine di profondità*, una struttura dati che è molto simile ad una semplice immagine in scala dei grigi[1], dove il valore di ogni pixel rappresenta la misura in millimetri della distanza della superficie dal sensore.

La massima affidabilità del sensore del Kinect V2 si ha per distanza comprese tra $50cm$ e $4,5m$. Il dispositivo è montato al soffitto a $2,8m$ da terra e ha un campo visivo di $70° \times 60°$, il quale, all'altezza del pavimento, determina un'area di cattura di circa $4m \times 5m$.

La dimensione di ogni immagine di profondità è di 512×424 pixel. Nativamente non vengono codificate in alcun modo particolare, sono delle semplici matrici di interi. È in grado di catturarne fino a 30 al secondo. Utilizzando un apposito software di registrazione è stato possibile mettere insieme dei video di profondità a 30 fps.

[1] La forte somiglianza con le immagini in scala dei grigi è supportata dal fatto che ogni pixel è codificato utilizzando 16bit.

Bibliografia

[1] Thomas H Cormen. Introduction to algorithms. 2009.

[2] Yoav Freund and Robert E Schapire. A decision-theoretic generalization of on-line learning and an application to boosting. *Journal of computer and system sciences*, 55(1):119–139, 1997.

[3] Alfred Haar. Zur theorie der orthogonalen funktionensysteme. *Mathematische Annalen*, 69(3):331–371, 1910.

[4] Rainer Lienhart and Jochen Maydt. An extended set of haar-like features for rapid object detection. In *Image Processing. 2002. Proceedings. 2002 International Conference on*, volume 1, pages I–900. IEEE, 2002.

[5] Michael Oren, Constantine Papageorgiou, Pawan Sinha, Edgar Osuna, and Tomaso Poggio. Pedestrian detection using wavelet templates. In *Computer Vision and Pattern Recognition, 1997. Proceedings., 1997 IEEE Computer Society Conference on*, pages 193–199. IEEE, 1997.

[6] Constantine P Papageorgiou, Michael Oren, and Tomaso Poggio. A general framework for object detection. In *Computer vision, 1998. sixth international conference on*, pages 555–562. IEEE, 1998.

[7] ITUT Rec. T. 800— iso/iec 15444-1,“. *Information technology—JPEG*, 2000.

[8] Stuart Russell and Peter Norvig. Artificial intelligence: a modern approach. 1995.

[9] Paul Viola and Michael J Jones. Robust real-time face detection. *International journal of computer vision*, 57(2):137–154, 2004.

[10] Yi-Qing Wang. An analysis of the viola-jones face detection algorithm. *Image Processing On Line*, 4:128–148, 2014.

[11] Lei Zhu and Kin-Hong Wong. Human tracking and counting using the kinect range sensor based on adaboost and kalman filter. *Advances in Visual Computing*, pages 582–591, 2013.