ALMA MATER STUDIORUM UNIVERSITÀ DI BOLOGNA

SCUOLA DI INGEGNERIA E ARCHITETTURA

Dipartimento di Informatica – Scienza e Ingegneria - DISI

Corso di Laurea magistrale in Ingegneria Informatica

Classificazione di oggetti in immagini attraverso il modello Bag of Visual Words

Tesi di Laurea in

Logiche Riconfigurabili M

Relatore:
Ing. Stefano Mattoccia

Presentata da:
Andrea Annovi

Correlatori:
Dott. Marco Casadio
Dott. Paolo Macrì
Dott. Ilario Marchio

Anno Accademico 2012/2013
Sessione III

The future belongs to those who believe in the beauty of their dreams.
"Eleanor Roosevelt"

Abstract

Generic object recognition is an important function of the human visual system and everybody finds it highly useful in their everyday life. Humans are able to categorize and classify different object classes in their surrounding environment in an easy way. For an artificial vision system it is a really hard, complex and challenging task because instances of the same object category can generate very different images, depending of different variables such as illumination conditions, the pose of an object, the viewpoint of the camera, partial occlusions, and unrelated background clutter. Computers must go through a series of steps in order to classify a single image.

The purpose of this thesis is to develop a system that is able to classify objects in 2D images based on the context, and identify to which category the object belongs to. Given an image, the system can classify it and decide the correct categorie of the object. This is a highly important area of Computer Vision and there are a lot of researches about this topic. Furthermore the objective of this thesis is also to test the performance and the precision of different supervised Machine Learning algorithms in this specific task of object image categorization. The most important, reliable and recent algorithms are being used. At the end the accuracy of different Features Extractor and Descriptor techniques are compared.

After a long research through different methods to represent images, it has been decided to use the Bag of Visual Words (proposed for the first time in this article [1]), one of the most frequently used algorithm for category recognition, owing to its simplicity and good performance. The approach is similar to the method used in document classification techniques in which a document is represented as a bag of the most potentially meaningful words. The "bag of words" contains no information about the order or position of the words in the book, but simply how often each of it occurs. This concept is applied to image classification using "visual words" which are a meaningful section of the image.

The implementation proposed in this project is able to deal with multi-class problems because the real world contains a lot of different object categories and recognizing different object classes is really important. Through different experiments the implemented application reveals good categorization performances despite the difficulty of the problem. However this project is open to future improvement; it is possible to implement new algorithms that has not been invented yet or using other techniques to extract features to make the system more reliable.

This application can be installed inside an embedded system and after trained (performed outside the system), so it can become able to classify objects in a real-time. The information given from a 3D stereocamera, developed inside the department of Computer Engineering of the University of Bologna, can be used to improve the accuracy of the classification task. The idea is to segment a single object in a scene using the depth given from a stereocamera and in this way make the classification more accurate.? The image classification topic has a lot of important applications, for instance for robotic and automated systems it can be a really useful program.

Indice

Elenco delle figure

Elenco delle tabelle

Capitolo 1

Introduzione

1.1 Il contesto

In Computer Vision, l'*Object Recognition* ha come obiettivo principale quello di riuscire a trovare un oggetto desiderato in una collezione di immagini o video digitali.

L'occhio umano riconosce e interpreta un oggetto appartenente ad un'immagine in maniera immediata, con poco sforzo, nonostante l'immagine possa variare in angolazione, formato, scala e rotazione. Inoltre gli oggetti possono essere riconosciuti anche quando sono parzialmente esclusi dalla vista. Questo processo di identificazione da parte di un calcolatore è molto più elaborato ed è oggi una grande sfida in *Computer Vision*.

Per ogni oggetto in un'immagine, esistono caratteristiche interessanti (*features*) che lo contraddistinguono da altri oggetti. Queste possono essere estratte in modo da fornire una descrizione caratteristica e distintiva dell'oggetto. È importante che, l'insieme di caratteristiche estratte dall'immagine campione, sia insensibile alle variazioni di scala delle immagini, ai disturbi, all'illuminazione e alle distorsioni geometriche, in modo da rendere affidabile il riconoscimento.

Il Riconoscimento di immagini è generalmente suddivisa, dai ricercatori in ambito di Computer Vision, in due tipi: il riconoscimento di un oggetto *specifico* e il riconoscimento *generico*, come si puó vedere in Figura 1.1.

Nel caso di riconoscimento specifico, la scopo è quello di identificare un istanza di un particolare oggetto, persona, luogo (per esempio la faccia di Carl Gauss, la

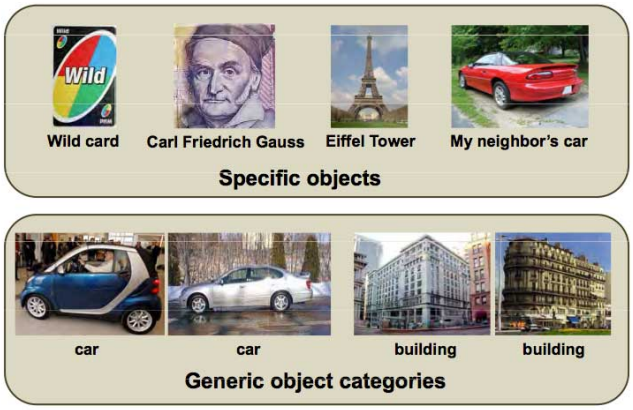

Figura 1.1: Differenti metodi di riconoscimento[4]

Torre Eiffel o la copertina un una rivisita famosa). In contrasto, il riconoscimento a livello di categoria, si basa sul riconoscere differenti istanze di una generica categoria, che appartiene alla stessa classe concettuale (per esempio edifici, pedoni, macchine, biciclette, motociclette).

1.2 Obiettivi del progetto

Lo scopo di questa tesi è quello di classificare un'immagine in base al suo contesto/contenuto e identificare a quale categoria o classe l'oggetto appartiene (*Image Classification* o *Object Categorization*). L'obiettivo non è quello di riconoscere un oggetto particolare, per esempio un modello specifico di una macchina, ma verificare se nell'immagine è presente un oggetto appartenente alla classe automobile.

Riuscire a identificare un oggetto appartenente a una determinata classe non è compito semplice. Occorre generalizzare ed astrarre dalla singola immagine, identificando parti salienti dell'oggetto che appartengono a tutta la classe. Attraverso degli esperimenti cercherò, inoltre, di identificare l'algoritmo di classificazione ideale per questa tipologia di problema, valutando le performance e il livello di accuratezza.

1.3 Descrizione dell'approccio impiegato

Dopo aver analizzato lo stato dell'arte studiando differenti modelli di rappresentazione dell'immagine, ho scelto di utilizzare in questo lavoro di tesi il modello Bag of Visual Words proposto per la prima volta da Sivic nell'articolo [34] in ambito di "video retrieval". Grazie alla sua efficienza ed efficacia, è diventato molto popolare in materia di categorizzazione e classificazione di immagini. Alcune delle migliori tecniche di classificazione di immagini sono in questo momento basati su questo schema. In visione artificiale, il modello Bag of Words (BoW) può essere applicato alla classificazione di immagini, trattando le caratteristiche dell'immagine come parole. In Computer Vision, Bag of Visual Words è un vettore di occorrenza conteggiate basandosi sulle informazioni di un vocabolario in cui sono presenti caratteristiche delle immagini locali. Questa tecnica si basa su algoritmi di apprendimento supervisionati e non supervisionati.

1.4 Problema

L'apprendimento di una particolare categoria di oggetti può essere molto complicata. L'oggetto ricercato all'interno di un immagine può subire svariati tipi di trasformazioni, come ad esempio cambiamenti di scala o di illuminazione, rotazioni, deformazioni, inversioni o sfocature. Inoltre, è spesso possibile che l'oggetto all'interno della scena non sia visibile interamente, ma parzialmente ostruito, rendendone difficile la localizzazione. Dati questi problemi è necessario sviluppare un sistema robusto che tenga conto di ogni possibile variazione dell'oggetto ricercato.

1.5 Organizzazione della tesi

Di seguito viene presentata la struttura della tesi:

Capitolo 1 Introduzione: vengono presentate le motivazioni che hanno portato allo sviluppo di questa tesi, l'obiettivo del progetto e i problemi esistenti in questo ambito.

Capitolo 2 Caratteristiche dell'immagine: si analizza il concetto di caratteristica dell'immagine, concentrando l'attenzione sui punti salienti. Viene presentato il concetto di estrattore di features e descrittore. Nella seconda parte del capitolo si presentano, nei dettagli, tre descrittori molto utilizzati.

Capitolo 3 Strumenti di apprendimento automatico: in questo capitolo sono descritti i fondamenti teorici sui quali si basa il lavoro della tesi. Si parte dalla definizione del problema della classificazione, vengono presentate le tecniche più importanti di apprendimento supervisionato e non supervisionato, tra cui quelle che ho utilizzato nella tesi.

Capitolo 4 Modelli di rappresentazione: in questo capitolo viene affrontato il problema di come rappresentare un immagine in modo strutturato, per essere automaticamente classificata. Si analizzando tre modelli di rappresentazione: Bag-of-Visual Words, Constellation Model e Implicit Shape Model.

Capitolo 5 Soluzione implementata: viene spiegato quali sono state le fasi della progettazione dell'applicazione, strumenti, librerie e Dataset utilizzati. Viene affrontato il problema di classificazione multiclasse e viene spiegato nei dettaglio la realizzazione del progetto, utilizzando il modello Bow. Vengono altresì analizzati vantaggi e svantaggi di tutti i più importanti algoritmi di classificazione supervisionata. Infine è stata descritta una possibile applicazione di questo progetto di tesi, integrandola sulla sterecamera e utilizzando il descrittore binario BRISK.

Capitolo 6 Analisi dei risultati e confronto: si presentano e si analizzano gli esperimenti effettuati, confrontandoli e motivandone le scelte effettuate. Nella prima parte vengono valutate le prestazioni degli algoritmi di classificazione con lo scopo di identificare quello ottimo, con il minore margine di errore. Nella seconda parte si valuta il modello Bag Of Visual Words sperimentando diversi tipi di classificazione. Nella parte finale ho confrontato le prestazioni dei descrittori BRISK, ORB, SURF rispetto a SIFT.

Capitolo 7 Conclusioni e sviluppi futuri: si conclude la tesi con una sintesi del lavoro svolto, riportando i risultati degli esperimenti e facendo un commento

finale su quanto ottenuto. Sono infine presentati eventuali sviluppi futuri che possono essere intrapresi.

Capitolo 2

Features in immagini

In *Computer vision* il concetto di feature è usato per denotare una parte di informazione che è rilevante per risolvere un obiettivo computazionale in relazione a una certa applicazione.
La *caratteristica* scelta quindi ha un ruolo fondamentale, negli ultimi anni la ricerca si è concentrata su un approccio basato sulle parti, nel quale utilizza solo parte dell'immagine e una *caratteristica* su cui eseguire l'analisi desiderata, eliminando i dati superflui.

Per poter descrivere un immagine o un oggetto contenuto in essa, occorre pertanto estrarre delle caratteristiche salienti che la rappresentino, in modo tal da poter fare confronti con altre immagini e cercare di classificare l'oggetto. Esistono due approcci:

- proprietà globali dell'immagine - *features globali*
- punti particolari dell'immagine - *features locali*

Features globali Permettono di rappresentare un oggetto nella sua totalità, cioè informazioni che non dipendono dal dettaglio delle forme, ma cercano di individuare l'andamento generale dell'intera scena, come ad esempio generazione dell'istogramma del colore o la texture dell'immagine.
Questo tipo di caratteristiche risultano interessanti se vogliamo descrivere la scena nel suo complesso, ma difficili da utilizzare quando vogliamo trovare una correlazione tra oggetti presenti nell'immagine e oggetti precedentemente osservati. Inoltre questo metodo è poco robusto in caso di occlusione parziale dell'oggetto,

visualizzazione dell'immagine da diversi punti di vista, o se l'oggetto cercato non è separato dal background.

Features locali Al contrario questa rappresentazione cerca di descrivere un oggetto attraverso rilevazione di punti particolari dell'immagine o punti di interesse (keypoints) detti *features locali*. In questo modo riusciamo ad avere maggiore robustezza, non solo dal punto di vista della occlusione, ma anche invarianza a particolari trasformazioni, come rotazione o cambiamento di scala, e quindi possono essere impiegati in maniera più efficace come nell'ambito della classificazione di immagini in base alla semantica.

2.1 Punti di interesse e descrittori locali

Il concetto di *feature detection* (riconoscimento di caratteristiche) racchiude una serie di metodi per estrappolare informazioni da una immagine e per prendere decisioni locali sull'esistenza o meno di una caratteristica in quel determinato punto. Le caratteristiche risultanti saranno un sottoinsieme del dominio dell'immagine, spesso in forma di punti isolati, curve continue o regioni connesse.

Esistono diversi tipi di *image features* come per esempio *line* (Trasformata di Hough), edge (Canny edge detector), *corners* (Harris corner detector), *creste*, *punti di di interesse* [35].

In questo capitolo concentrerò l'attenzione sui *punti di iteresse* in quanto si adattano meglio al mio progetto di tesi. Questo tipo di rappresentazione cattura proprio l'aspetto disordinato e sparso dei concetti contenuti, senza imporre vincoli specifici di ordinamento o concatenazione. Un insieme di keypoints può essere visto come una "patch" saliente dell'immagine, caratterizzata da un alto contenuto informativo locale della stessa.

Selezionati i punti salienti, il passo successivo consiste nel descriverli in maniera robusta, possibilmente invarianti ai vari cambiamenti di vista e di luce.

2.1.1 Estrazione dei keypoints

Campionamento denso

Questo metodo prevede di descrivere ogni immagine secondo un numero fisso di punti. L'immagine viene segmentata da linee orizzontali e verticali che individuano i punti da estrarre. Tali tecniche sono dette di campionamento denso, esistono due categorie

- *campionamento casuale*: un numero prefissato di punti è scelto in modo casuale tra tutti i pixel dell'immagine.
- *campionamento a griglia regolare*: fissato un passo di campionamento, i punti descritti corrispondono ai nodi di una griglia sovrapposta all'immagine;

Nonostante la semplicità, questo metodo fornisce dei buoni risultati per la rilevazione di textures, scenari naturali e tutte quelle situazioni in cui l'oggetto o la persona da rilevare è messa in evidenza da uno sfondo uniforme. Questo è dovuto essenzialmente al fatto che la griglia è capace di catturare informazioni globali, cioè distribuite uniformemente nell'immagine. Lo svantaggio è che tralascia la maggior parte dell'informazione saliente dei concetti in esame. Esperimenti [36] hanno mostrato che per applicazioni di classificazione basate su bag-of-words di scene naturali le tecniche di campionamento denso basato su una griglia regolare offrono ottime prestazioni.

Campionamento sparso

Caratteristiche particolari sono rilevate con appositi detectors che sono in grado di selezionare zone precise (come egdes, corners, blobs) dell'immagine.
Vi sono svariati algoritmi di estrazione che usano questa tecnica, tra cui *Harris corner detector* [37], *Hessian-Laplace e Harris-Laplace* [38], *Difference of Gaussian (DoG)* il quale viene impiegato all'interno del SIFT (Lowe), *affine covariant patches*.

Il vantaggio di usare il campionamento sparso risiede nel fatto che cattura le informazioni più rilevanti dell'immagine, ma di contro l'individuazione di tali punti è più complessa e dipendente dal detector utilizzato.

2.1.2 Descrittori

Per descrittore si intende un vettore numerico che identifichi quel determinato punto dell'oggetto univocamente, indipendentemente dalle condizioni di acquisizione dell'immagine.

Negli ultimi anni, come si nota in letteratura, uno dei descrittori più usati, in applicazioni sia sperimentali che in sistemi commerciali, è SIFT (Scale Invariant Feature Transform) ideato e sviluppato da David G. Lowe [7], il quale ricerca intelligentemente i punti chiave (o keypoints) all'interno dell'immagine e associa ad essi vettori di 128 elementi. Questo tipo di descrittore ha avuto un grande successo perchè risulta particolarmente robusto rispetto alle principali alterazioni che le immagini posso subire, e in aggiunta a queste proprietà, si caratterizza anche per la sua semplicità di individuazione e computazione, che ne giustifica la larga diffusione che ha ottenuto recentemente.

Esistono diverse tipologie di algoritmi di ricerca di feature locali salienti in immagini: ad esempio l'articolo di Mikolajczyk e C. [39] offre un'ampia descrizione e un confronto tra i principali.

Nei due capitoli seguenti, descriverò nel dettaglio l'aspetto teorico di due metodi molto utilizzati, che offrono ottime performance e che si adattano bene alla realizzazione del mio progetto di tesi: *SIFT* [7], *SURF* [8]. Infine nell'ultimo capitolo tratterò un metodo molto recente (2011) di estrazione keypoints e descrizione dell'immagine chiamato BRISK [40].

2.2 SIFT: Scale Invariant Features Transform

SIFT è una algoritmo che consente l'estrazione di features locali da un'immagine, ideato e proposto per la prima volta da David G. Lowe [41] nel 1999, e migliorato nella localizzazione dei keypoints nel 2004 [7]. Tali features estratte, sono particolarmente robuste a cambi di illuminazione, rumore ed a variazioni del punto di vista. Inoltre sono invarianti a trasformazioni geometriche, come per esempio cambiamenti di scala (dovute ad esempio ad un'operazione di zoom) o rotazioni.

Le feature trovate con questo metodo sono altamente distintive nel senso che una singola feature può trovare la sua corretta corrispondenza in un ampio

database di feature estratte da diverse immagini della stessa scena.
L'algoritmo puó essere riassunto sinteticamente nelle principlai fasi:

1. individuazione degli estremi locali nello scale-space;
2. localizzazione dei keypoints;
3. assegnazione di una (o più) orientazioni canoniche;
4. generazione dei descrittori.

2.2.1 Individuazione estremi locali nello scale-space

Il primo stadio dell'algoritmo consiste nella ricerca di tutti i possibili punti di interesse sull'intera immagine a tutte le possibili scale. Questa fase si effettua filtrando ripetutamente l'immagine originale, attraverso convoluzioni Gaussiane, creando così uno "spazio delle scale". Per ognuna di queste scale vengono calcolate le differenze tra le funzioni gaussiane adiacenti *DoG (Difference of Gaussian)*: i massimi e minimi della funzione differenza vengono identificati come punti di interesse.

La rappresentazione descritta risulta essere utile, in quanto l'immagine originale presenta molti punti da esaminare: sottocampionare e sfocare l'immagine, raffinando la ricerca nelle versioni piú dettagliate, rende l'analisi molto più affidabile.
Data un immagine $I(x, y)$ e un filtro Gaussiano di varianza σ, lo **Scale Space** associato all'immagine è dato dalla funzione $L(x, y, \sigma)$:

$$L(x, y, \sigma) = G(x, y, \sigma) * I(x, y)$$

dove $*$ è l'operatore di convoluzione, e

$$G(x, y, \sigma) = \frac{1}{2\pi\sigma^2} e^{-(x^2+y^2)/2\sigma^2}$$

Figura 2.1: Filtraggio con kernal Gaussiani di varianza σ crescente [5]

La costruzione dello scale-space avviene filtrando iterativamente l'immagine ad intervalli regolari, raddoppiando σ, dando origine a un insieme di *s* immagini dette *ottave*, della stessa dimensione (esempio Figura 2.1).

Per trovare punti interessanti all'interno dello scale-space, viene determinata la funzione *DoG*, calcolata come la differenza tra due funzioni Gaussiane, considerate a scale consecutive:

$$D(x, y, \sigma) = (G(x, y, k\sigma) - G(x, y, \sigma)) * I(x, y) = L(x, y, k\sigma) - L(x, y, \sigma)$$

In Figura 2.2 notiamo a sinistra, le immagini convolute con i kernal Gaussiani che

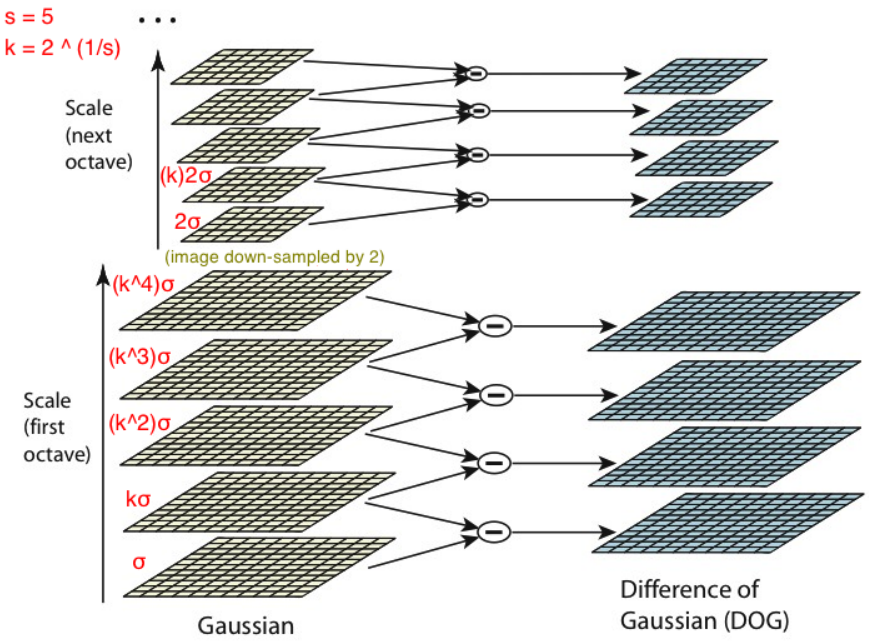

Figura 2.2: Difference of Gaussian (DoG) [6]

formano insiemi di ottave diverse, ciascuna corrispondente ad un valore di θ. Ogni ottava è divisa in scale. A destra, le immagini che rappresentano le Differenze di Gaussiane (DoG), ottenute sottraendo le Gaussiane adiacenti. Al termine di ogni ottava l'immage Gaussiana viene sotto campionata di un fattore 2 e il processo riparte.

La funzione DoG risulta essere estremamente conveniente dal punto di vista computazionale e rappresenta allo stesso tempo una approssimazione molto buona del Laplaciano della funzione gaussiana normalizzato rispetto alla scala.

2.2.2 Localizzazione dei keypoints

I punti individuati sono scelti solo qualora i minimi e i massimi locali della DoG calcolati confrontando ogni punto con gli otto vicini individuati nell'immagine corrente e con i nove vicini che si trovano alla scala superiore e a quella inferiore risultino essere anche minimo o massimi assoluti.

Gli estremi locali della funzione $D(x, y, \sigma)$, rappresentano i punti di interesse da ricercare. Per determinarli ogni punto viene confrontato con gli otto vicini individuati nell'immagine corrente e con i nove delle due scale immediatamente superiore ed inferiore, con notiamo in Figura 2.3 e vengono scelti qualora risultino essere anche minimo o massimi assoluti.

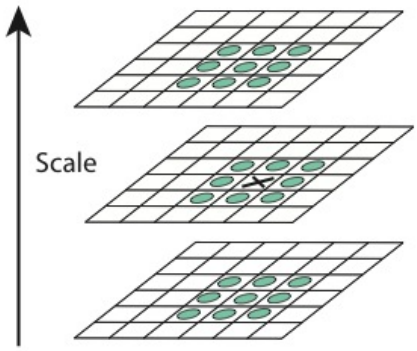

Figura 2.3: Gli estremi locali della DoG sono individuati confrontando il pixel (contrassegnato dalla X) con i 26 adiacenti in una regione 3x3 alla scala corrente, ed alle due adiacenti [7]

Quanto presentato corrisponde alla prima versione di SIFT, l'attuale versione di SIFT (proposta da Lowe nel 2004 [7]) introduce un metodo più accurato per la selezione dei keypoints. Prevede che vengano eliminati i punti con basso contrasto, valutando la presenza di bordi e linee, inoltre viene effettuata una interpolazione tra essi, per aumentare la precesione, la stabilità e il matching dei punti.

Sinteticamente l'ultima versione dell'algoritmo di Lowe, utilizza l'espansione di Taylor, al termine del secondo ordine, della funzione scale–space $D(x, y, k\sigma)$, traslata in modo che l'origine sia collocata alle coordinate del campione dell'immagine, per ulteriori dettagli rimandiamo il lettore al documento originale [7].

Per evitare di avere keypoints non stabili, essi vengono selezionati solo qualora gli estremi della funzione gaussiana risultino superiori ad una soglia, scelta opportunamente in modo da tener conto del contrasto. In Figura 2.4 viene illustrato un esempio di eliminazione del basso contrasto e dei punti di bordo.

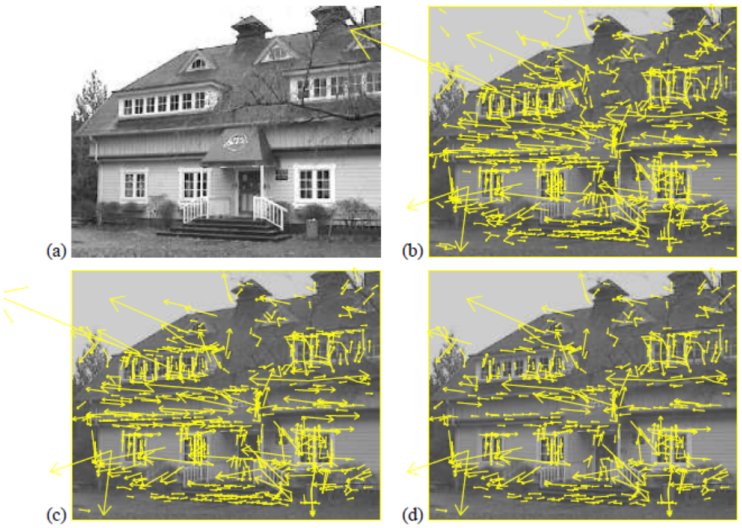

Figura 2.4: Dall'immagine originale (a) vengono estratti tutti i Keypoints con il relativo orientamento (b). In immagine (d) sono indicati i keypoints mantenuti perchè considerati i più stabili [7].

2.2.3 Assegnazione di orientazioni canoniche

Una volta individuate le coordinate e la scala per ogni keypoint, si procede assegnando una orientazione, in questo modo è possibile rappresentare ogni keypoint in relazione al suo specifico orientamento, raggiungendo così l'invarianza rispetto alla rotazione. La scala dei keypoint è poi usata per selezionare la particolare immagine, nella piramide di immagini Gaussiane, dalla quale il punto stesso è stato estratto: procedere secondo questo schema permette di stabilire anche l'invarianza di scala. Quindi viene calcolato per ogni campione dell'immagine scelta $L(x, y)$, l'ampiezza $m(x, y)$ e la direzione $\theta(x, y)$ del gradiente locale, applicando la differenza tra pixels:

$$m(x, y) = \sqrt{(L(x+1, y) - L(x-1, y))^2 + L(x, y+1) - L(x, y-1))^2}$$
$$\theta(x, y) = \tan^{-1}((L(x, y+1) - L(x, y-1))/(L(x+1, y) - L(x-1, y)))$$

Per ogni immagine, ognuna alla propria scala di riferimento, sono quindi calcolati degli istogrammi di orientamento, basati sui gradienti individuati in una regione attorno al punto scelto. L'istogramma è formato da 36 bin, in modo da coprire tutti i 360 gradi delle possibili direzioni.

I picchi di questo istogramma permettono di definire le direzioni dominanti del gradiente locale. Il picco più alto, così come tutti gli altri picchi locali che siano superiori all'80% del picco massimo, individuano l'orientamento. Un keypoint può quindi essere caratterizzato anche da orientamenti multipli.
Una volta calcolato, l'orientamento dominante viene impiegato per ruotare l'immagine, normalizzandone così la posizione e mantenendo una invarianza rispetto alle rotazioni [42].

2.2.4 Generazione dei descrittori locali

Finora le operazioni effettuate sono servite per assegnare ad ogni punto individuato delle coordinate immagine, una scala e un orientamento.
L'ultimo passo consiste quindi nella generazione del descrittore che sia altamente distintivo e il più possibile invariante a variazioni di illuminazione e cambi di prospettiva.

Il descrittore viene creato calcolando la magnitudo in una regione attorno al punto di interesse. Esso viene rappresentato con una finestra gaussiana (rappresentata in Figura 2.5 dal cerchio blu): questo permette di evitare cambi repentini all'interno del descrittore a seguito di piccole variazioni nella posizione della finestra, ma anche di dare minore enfasi ai gradienti che sono più lontani rispetto al centro del descrittore.

Il descrittore SIFT finale viene costruito dividendo i 16 pixel vicini al keypoint in esame in sottoregioni di dimensioni 4x4 pixel; per ognuna di queste regioni vengono individuati 8 orientamenti: ciò comporta che ogni descrittore generato sia un vettore composto da 128 elementi (4 x4 x 8). Ogni elemento del vettore, corrisponde quindi al valore dell'istogramma delle orientazioni della relativa sub–regione 4 x 4 dell'intorno del keypoint.
Infine, per ridurre gli effetti dovuti alle variazioni di illuminazione, il vettore delle caratteristiche viene normalizzato.

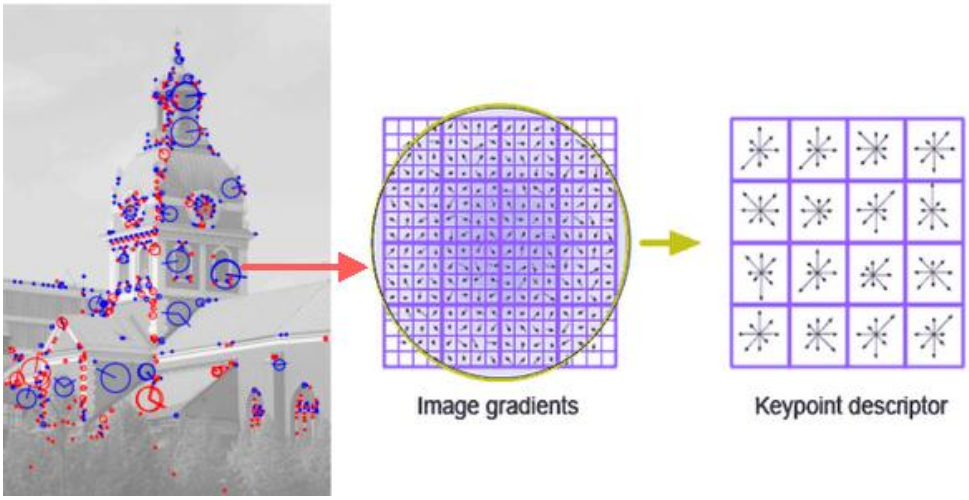

Figura 2.5: Creazione di un descrittore di dimensioni 4x4, costruito a partire da un set di 16x16 campioni dell'immagine L [7]

2.3 SURF: Speeded Up Robust Features

L'algoritmo SURF, implementato nel 2006 da Herbert Bay, Tinne Tuytelaars e Luc Van Gool [8] con l'obbiettivo di migliorare performance e robustezza dell'algoritmo SIFT ed essere più veloce computazionalmente. Grazie a una serie di approssimazioni e accorgimenti sui filtri usati, sulla modalità di costruzione della piramide scale-space e sulla struttura del descrittore utilizzato, lo scopo viene raggiunto.

Per rivelare keypoints all'interno dell'immagine, SURF utilizza la matrice Hessiana, mentre il descrittore raccoglie la distribuzione della risposta alla wavelet di Haar all'interno di un intorno del punto di interesse.
Come il SIFT, l'algoritmo SURF utilizza unicamente le informazioni fornite sulla luminanza dell'immagine (cioè sui toni di grigio) senza utilizzare informazioni legate al colore.
. Questo algoritmo può essere suddiviso in tre fasi:

- costruzione dell'immagine integrale;
- ricerca dei punti di interesse;
- assegnamento di un orientamento;
- calcolo del descrittore.

2.3.1 Costruzione dell'immagine integrale

L'algoritmo SURF sfrutta una rappresentazione intermedia dell'immagine da processare, detta *immagini integrale*, definita da Viola e Jones, 2001 [43], questo fa si che si raggiungano tempi computazionali inferiori se paragonati a quelli degli altri algoritmi per l'estrazione dei punti di interesse che usano invece *Box filter*.

Data in ingresso un'immagine I e un punto di coordinate (x, y), l'integrale dell'immagine $I_\Sigma(x, y)$ è calcolata tramite la somma dei valori dell'intensità dei pixel compresi tra il punto dato e l'origine. Formalmente è definito dalla formula:

$$I_\Sigma(x, y) = \sum_{i=0}^{i \leq x} \sum_{j=0}^{j \leq y} I(x, y)$$

Utilizzando questa tecnica, il calcolo si riduce alla sola esecuzione di semplici somme e alcuni accessi in memoria. Il grande vantaggio di tale metodo è che tale calcolo risulta, inoltre, essere indipendente dalle dimensioni dell'area considerata.

2.3.2 Punti d'interesse basati sulla matrice Hessiana

Il rivelatore dei keypoint è basato sulla *matrice Hessiana*, la quale viene approssimata mediante l'utilizzo delle immagini integrali, permettono ottime performance computazionali ed elevata accuratezza. Il detector così modificato prende il nome di **Fast-Hessian**.

Il SURF si serve di un filtro Gaussiano del secondo ordine normalizzato, che permette un'analisi spaziale e su diversi fattori di scala. Infatti, dato un punto $x = (x, y)$ in un'immagine I, la matrice Hessiana $\mathcal{H}(x, \sigma)$ alla scala σ è definita come:

$$\mathcal{H}(x, \sigma) = \begin{bmatrix} L_{xx}(x, \sigma) & L_{xy}(x, \sigma) \\ L_{xy}(x, \sigma) & L_{yy}(x, \sigma) \end{bmatrix}$$

dove $L_{xx}(x, \sigma)$ è la convoluzione della Gaussiana derivata al secondo ordine $\frac{\partial^2}{\partial x^2} g(\sigma)$ con l'immagine I nel punto x, e similmente per $L_{xy}(x, \sigma)$ e $L_{yy}(x, \sigma)$.

Le funzioni Gaussiane risultano essere ottime per le analisi di tipo spaziale (scale space), ma nella pratica devono essere necessariamente discretizzate e approssimate. L'unico inconveniente, per quanto riguarda i detector basati sulla matrice Hessiana, è che questi potrebbero andare incontro ad una perdita di

reperibilità nel caso di rotazioni di angoli che siano multipli dispari di $\frac{\pi}{4}$, per questo esse debbono essere discretizzate. Inoltre i keypoints devono essere individuati a diverse scale perchè la ricerca delle corrispondenze spesso richiede il loro confronto in immagini dove essi sono visti con scale differenti.

Spingendo l'approssimazione della matrice Hessiana all'estremo, si può utilizzare un *box filter* il quale può essere calcolato ad un costo computazionale estremamente basso, utilizzando le immagini integrali. Inoltre, le sue performance sono paragonabili o, in alcuni casi, addirittura migliori di quelle di una Gaussiana discretizzata e approssimata.

Per esempio, i box filter 9×9 in Figura 2.6 sono le approssimazioni di una Gaussiana con $\sigma = 1.2$ e rappresentano la scala più bassa per calcolare le risposte di convoluzione.

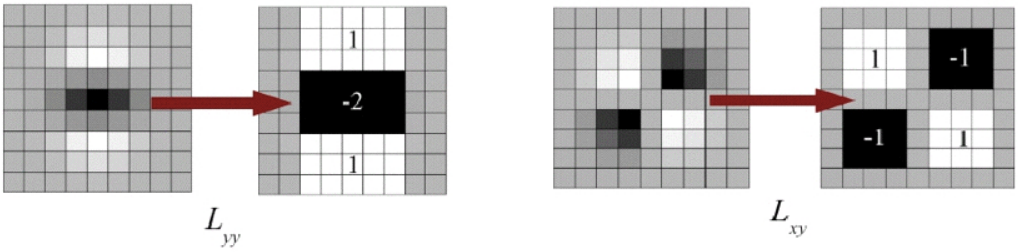

Figura 2.6: Rappresentazione dell'approssimazione apportata dal filtro box 9x9 [8]

Indicando le approssimazione con D_{xx}, D_{xy} e D_{yy} e si calcola il determinante dell'Hessiana con:

$$\det(\mathcal{H}_{approx}) = D_{xx}D_{yy} - (0.9D_{xy})^2$$

.

Rappresentazione Scale-space

E' necessario che i punti d'interesse siano rilevati con fattori di scala differenti, se non altro perchè la ricerca delle corrispondenze spesso richiede il confronto di immagini in cui le feature sono viste a distanze diverse.

Lo *Scale-Space* è generalmente implementato come una piramide di immagini, ottenuta applicando funzione di tipo Gaussiano alle diverse immagini. Grazie all'impiego dei *Box-filter* e l'integrale dell'immagine, non è più necessario applicare

il medesimo filtro all'immagine filtrata e scalata, ma si può applicare il filtro a ogni dimensione direttamente sull'immagine originale. L'analisi *Scale-Space* viene quindi effettuata con l'*up-scaling* del filtro piuttosto che il ridimensionamento iterativo dell'immagine filtrata (Figura 2.7). I vari *layer* della piramide sono ottenuti quindi filtrando l'immagine con filtri via via sempre più grandi.

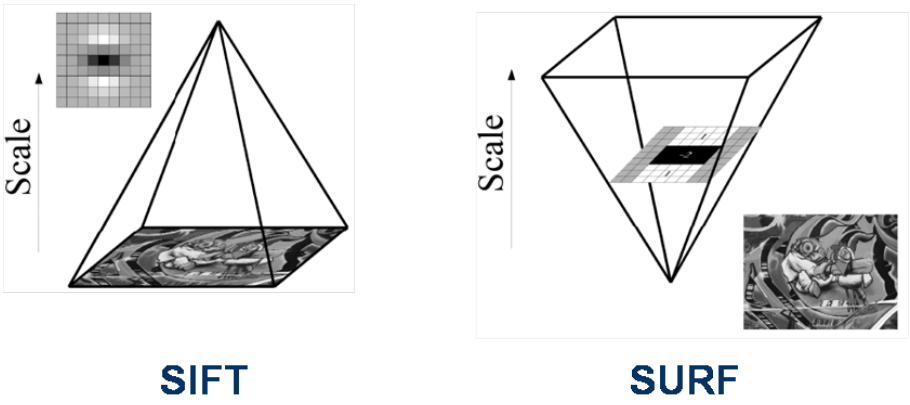

Figura 2.7: A sinistra si pùo notare come nel SIFT l'immagine viene successivamente sottocampionata e scalata. A destra invece l'uso delle immagini integrali permette di incrementare le dimensioni del filtro e mantenere costanti quelle dell'immagine [8]

Per ogni scala i massimi del determinante dell'Hessiana approssimata vengono confrontati, come per il SIFT, con i 26 punti (9x9 meno il punto centrale) connessi al punto candidato. I massimi vengono poi interpolati con i punti vicini per trovare la vera posizione del massimo. Lo spazio delle scale viene diviso in ottave, ognuna delle quali rappresenta una serie di mappe di risposta ottenute convolvendo la medesima immagine di input con un filtro di dimensione crescente con un passo di almeno 2n pixel, in modo da mantenere sempre un pixel centrale.

2.3.3 Il descrittore SURF

Il descrittore SURF è basato su alcune proprietà del SIFT, il primo passo per la creazione del descrittore è fissare un orientamento principale valutando un area circolare nell'intorno del punto di interesse. Successivamente viene costruita una griglia quadrata allineata con l'orientamento e si estrae da essa il descrittore SURF.

Assegnamento dell'Orientamento

Per rendere il descrittore invariante alla rotazione viene identificato un orientamento riproducibile, per ogni *keypoints* estratto nella precedente fase.
L'orientamento deve essere assegnato in modo da raggiungere l'invarianza rispetto alla rotazione dell'immagine.

Il primo passo è quello di calcolare la risposta di Haar wavelet nelle direzioni x e y in un intorno pari a 6 volte s attorno al punto di interesse (dove s rappresenta la scala alla quale il punto è identificato). Ancora una volta si usano le immagini integrali per calcolare la risposta, poichè le wavelet di Haar sono dei Box filter.
Le risposte calcolate sono quindi pesate con una funzione gaussiana (con $\sigma = 2s$) centrata nel punto di interesse. Le risposte vengono quindi rappresentate nello spazio: la risposta orizzontale viene rappresentata sull'asse delle ascisse, mentre quella verticale sull'asse delle ordinate.

La direzione dominante è stimata calcolando la somma di tutte le risposte con una finestra scorrevole, avente apertura $\frac{\pi}{3}$ radianti, come illustrato in Figura 2.8 e viene assunta come orientamento locale. Il vettore con modulo maggiore, tra tutti quelli calcolati con le diverse finestre, è l'orientamento principale del punto di interesse.

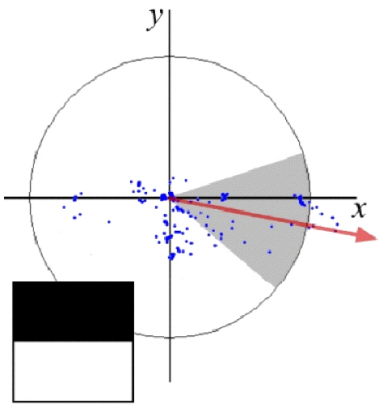

Figura 2.8: Rappresentazione del calcolo dell'orientamento. In grigio la finestra mobile all'interno della quale viene sommato il valore di tutte le risposte alla wavelet di Haar (basso a sinistra) [8]

Le componenti del Descrittore

Una volta individuato l'orientamento, si passa alla determinazione del descrittore che si affronta costruendo una griglia quadrata centrata sul punto di interesse e orientata lungo l'orientamento principale assegnato al passo precedente, come mostra la Figura 2.9. La regione viene poi divisa in sotto regioni regolari di

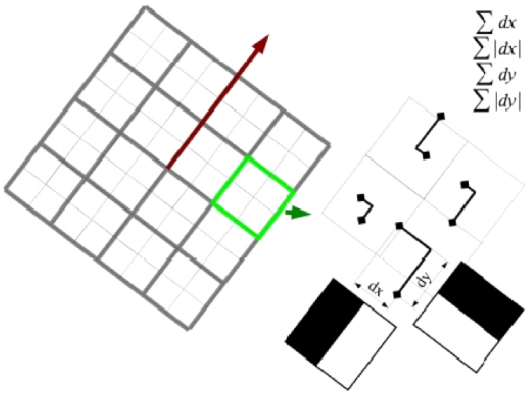

Figura 2.9: Schema del calcolo del descrittore SURF [8]

dimensioni 4x4 in modo da preservare un maggior numero di informazioni spaziali. Per ognuna di queste regioni viene calcolata la risposta di Haar Wavelet con una spaziatura regolare in direzione orizzontale (d_x) e verticale (d_y). Con lo scopo di incrementare la robustezza alle deformazioni geometriche, le risposte d_x e d_y sono pesate con una funzione gaussiana, centrata nel punto di interesse.

Successivamente le risposte d_x e d_y vengono sommate su tutta la sub-regione, formando così il primo set di valori del vettore del descrittore. grandezze da inserirsi nel descrittore. Per mantenere anche le informazioni in merito alla polarità delle variazioni di intensità sono estratti anche i valori assoluti delle somme ($|d_x|$) e ($|d_y|$). In questo modo ogni sub regione fornisce un descrittore di 4 dimensioni, avente la seguente struttura:

$$v = (\sum d_x; \sum d_y; \sum |d_x|; \sum |d_y|;).$$

Concatenando tale descrittore per tutte le sub-regioni di dimensione 4x4 considerate si ottiene un vettore di lunghezza 64. L'invarianza rispetto al contrasto viene raggiunta normalizzando il vettore.
In Figura 2.10 sono rappresentati alcuni esempi dei valori delle componenti del vettore SURF ottenuti per diverse regioni considerate.

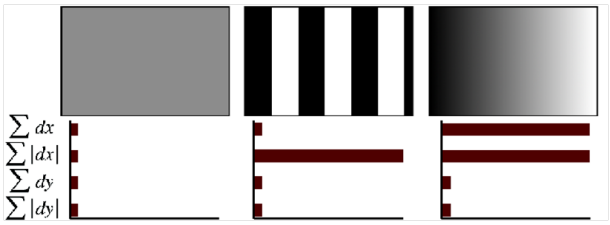

Figura 2.10: Esempio componenti del vettore SURF [8]

L'algoritmo del SURF può essere applicato anche costruendo descrittori di lunghezza diversa. Per molte applicazioni per le quali non è necessario assicurare l'invarianza rispetto alle rotazioni è possibile impiegare l'U-SURF, più rapido da calcolare, e comunque robusto a rotazioni di $\pm 15^{\circ}$ e altamente distintivo. Nel caso si necessiti, invece, di un matching molto rapido è possibile impiegare il SURF-36. Esso è caratterizzato da un descrittore più piccolo, con sotto regioni di dimensione 3x3. È contraddistinto da performance inferiori rispetto a quelle del SURF: i risultati ottenibili sono comunque accettabili se paragonati a quelli conseguibili con altri descrittori noti in letteratura.

Ultima variante dell'algoritmo è il SURF-128. Esso sfrutta il medesimo principio presentato per l'algoritmo nella sua versione originale, ma il descrittore risulta essere maggiormente raffinato poiché le somme d_x e $|d_x|$ sono calcolate separatamente per $d_y < 0$ e $d_y \leq 0$ (e in modo analogo anche per d_y e $|d_y|$ al variare del segno di d_x). In questo modo il numero di features raddoppia, dando così vita ad un descrittore più distintivo e non molto più lento da calcolare. Risulta invece notevolmente rallentata la fase di matching, a causa della notevole dimensione del descrittore stesso [44].

2.4 Descrittori Binari

SIFT e SURF che sono basati sugli istogrammi dei gradienti, questo comporta che i gradienti di ciascun pixel nella "patch" devono essere calcolati. Inoltre i descrittori tradizionali sono formati da un vettore di un certo numero di valori a virgola mobile. Per creare questo vettore ed eseguire in seguito i confronti 'e necessario tempo ed 'e normalmente un processo lento. Questi calcoli costano tempo. Nonostante SURF accelera il calcolo utilizzando immagini integrali, in alcune applicazioni non è abbastanza veloce. Inferiorioltre, SIFT e SURF sono

protetti da brevetto commerciale.

L'alternativa è di codificare maggior parte delle informazioni di una "patch" come una stringa binaria usando solo il confronto della intensità nelle immagini. Questo può essere fatto molto velocemente, inoltre il matching tra due "patch" può essere fatto utilizzando una singola istruzione, come la distanza di Hamming pari alla somma della operazione XOR tra le due stringhe binarie. Questa è esattamente quello che viene fatto nei descrittori binari.

Il vettore binario occupa meno spazio in memoria e per eseguire il confronto è sufficiente calcolare la distanza di Hamming tra le rispettive stringhe binarie. La distanza di Hamming si calcola in maniera molto efficiente calcolando lo XOR delle stringhe binarie e contando i bit attivi.

In generale, i descrittori binari sono composti da tre parti:

- **Un modello di campionamento**: dove campionare i punti salienti nella regione intorno al descrittore.
- **Compensazione Orientamento**: alcuni meccanismo per misurare l'orientamento del punto chiave e ruotarla per compensare le variazioni di rotazione.
- **Campionamento delle coppie**: confrontare le coppie per costruire il descrittore finale.

Consideriamo una piccola "patch" incentrata su un punto chiave, vorremmo descrivere il punto esaminato come una stringa binaria. Per prima cosa, occorre identificare un *modello di campionamento* intorno al punto chiave (per esempio i punti si sviluppano su una serie di cerchi concentrici), in Figura 2.11 due rappresentazione di modello di campionamento di BRISK e FREAK.

Quindi, scegliamo (per esempio 512), coppie di punti su questo modello di campionamento. Ora vengono riesaminate tutte le coppie e confrontato il valore di intensità del primo punto nella coppia con il valore di intensità del secondo punto della coppia. Se il primo valore è più grande della seconda allora la seconda, scrivere '1' nella stringa, altrimenti scrivere '0'. Dopo avere esaminato tutte le 512 coppie, avremo una stringa di 512 caratteri, composto da '1' e '0' che codifica le informazioni locali intorno al punto chiave. Come notiamo questo procedimento risulta essere molto semplice.

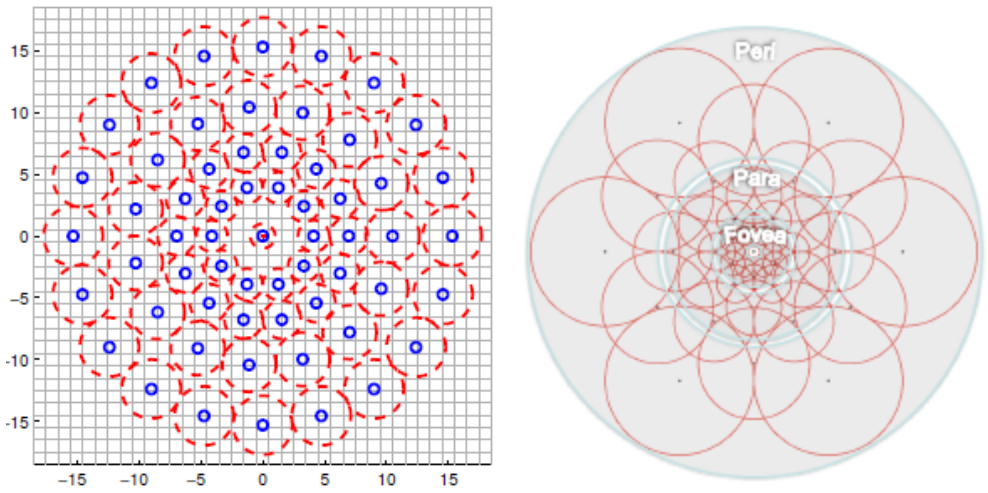

Figura 2.11: BRISK e FREAK descriptor – sampling pattern [9]

La fase di *orientazione* è una fase in cui l'angolo di orientamento della "patch" viene misurata e le coppie vengono ruotate di tale angolo a garantire che la rotazione dell'immagine sia invariate. Supponiamo di prendere una "patch" e descriverla come una stringa binaria utilizzando la procedura descritta sopra. Ora ruotiamo la "patch" di un valore θ e di nuovo estraiamo una stringa binaria che descrive la "patch" ruotata. Confrontiamo le stringhe e vediamo se sono simili, molto probabilmente risulteranno esser diverse dato che il modello è stato ruotato, ma le coppie sono rimaste uguali.

La fase di *matching* tra due corrispondenze avviene nel seguente modo: dati due keypoint, prima si applica la procedura di confronto vista sopra su entrambi, utilizzando ovviamente, lo stesso schema di campionamento e la stessa sequenza di coppie . Una volta che avrà due stringa binarie, il confronto è facile e veloce, basta contare il numero di bit in cui le stringhe differiscono, o equivalentemente applicare somma $(\mathrm{XOR}(\mathrm{stringa1}, \mathrm{stringa2}))$.

Ogni descrittore binario ha un proprio modello di campionamento, proprio metodo di calcolo dell'orientamento e propria serie di coppie di campionamento .

Esistono diverse tecniche di estrazione e descrizione di features binarie, tra cui proposte in letturatura troviamo BRIEF [45], FREAK [46], BRISK [40] , ORB [47]. In seguito analizzerò nei dettagli la tecnica BRISK.

2.5 BRISK: Binary Robust Invariant Scalable Keypoints

L'algoritmo BRISK presentato per la prima volta all'ICCV2011 (13th International Conference on Computer Vision) da Leutenegger, Chli e Siegwart in questo articolo [40] è un metodo per la rilevazione dei keypoints, identificazione descrittori e matching. BRISK è stato proposto con lo scopo di ridurre drasticamente i costi computazionali, rendendo la ricerca dei punti saliente e relativi descrittori, veloce, robusta, affidabile a rotazioni e invarianza.

Secondo quanto dimostrato dagli autori, dopo diversi esperimenti, in termini di rapiditià di esecuzione è molto veloce, mentre per quanto riguarda la accuratezza è paragonabile a SIFT e SURF. In alcuni casi funziona addirittura meglio.

BRISK è un descrittore binario a 512 bit e calcola la media ponderata Gaussiana su un modello di selezione di punti vicino al punto chiave. Confronta i valori di specifiche coppie di finestre, portando ad un 1 o uno 0, a seconda di quale finestra nella coppia era maggiore.

Questo algoritmo puo' essere suddiviso in tre fasi:

- Identificazione dei Keypoint nello Scale-Space;
- Descrizione dei Keypoint;
- Matching dei descrittori.

2.5.1 Identificazione dei Keypoint nello Scale-Space

Con l'obiettivo di ottenere invarianza di scala, che è cruciale per keypoints di alta qualità, la ricerca dei massimi avviene non solo nel piano immagine, ma anche nello scale-space usando il punteggio FAST [10] come misura dell'importanza.

Nel framework BRISK, i livelli della piramide dello scale-space consistono in n ottave c_i e n intra-ottave d_i per $i = \{0, 1, ..., n-1\}$ e solitamente $n = 4$. Le ottave sono formate campionando a metà progressivamente, l'immagine originale (corrispondente a c_0). Ogni intra-ottava d_i è posizionata tra il layer c_i e $c_i + 1$ come illustrato in Figura 2.12

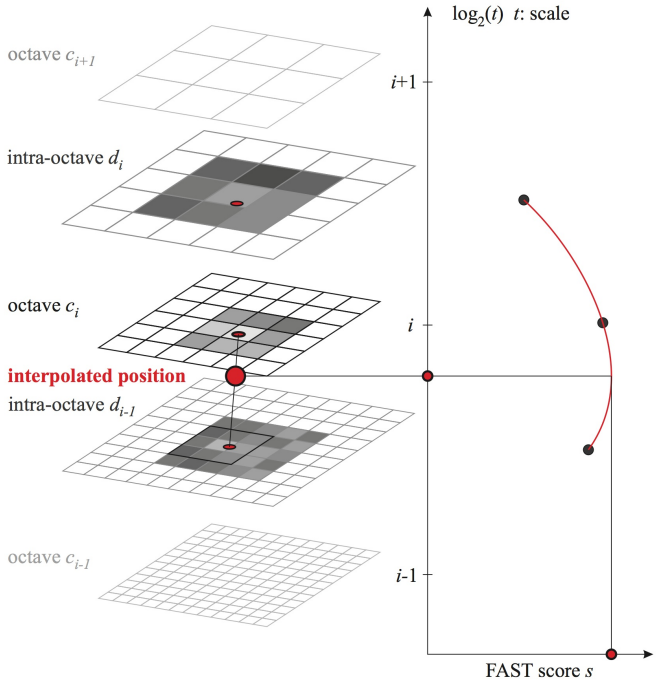

Figura 2.12: Rilevamente dei punti di interesse nello scale-space: un key-points è identificato nell'ottava c_i analizzando gli 8 punti adiacenti, nonché nei corrispondenti scores-patches e negli strati immediatamente contigui sopra e sotto [10].

La prima intra-ottava d_0 si ottiene sottocampionando l'immagine originale c_0 di un fattore di $1,5$, mentre il resto degli altri strati intra-ottava sono derivati da successivi sottocampionamenti.

2.5.2 Descrizione dei Keypoint

Dato un insieme di punti chiave (costituiti da sub-pixel locali di immagini raffinate e relativi valori di scala a virgola mobile), il descrittore BRISK è composto come una stringa binaria concatenando i risultati di semplici test di confronto della luminosità. In BRISK viene identificata la direzione caratteristica di ogni punto saliente per consentire orientamento-normalizzati dei descrittori e quindi raggiungere la invarianza di rotazione, questo è fondamentale per la robustezza generale. In Figura 2.11(destra) è rappresentato il modello di campionamento con $N = 60$ punti: i piccoli cerchi blu indicano i punti di campionamento, mentre i cerchi più grandi tratteggiati rossi sono disegnate in un raggio σ corrispondente

alla deviazione standard del kernel Gaussiano utilizzata per "smussare" i valori di intensità nei punti di campionamento. Il modello esposto è riferito alla scala di $t = 1$.

Campionamento Pattern e rotazione Stima

Il concetto chiave del descrittore BRISK fa uso di un modello utilizzato per il campionamento della zona del punto chiave, illustrato in Figura 2.11(destra). Cercando di stimare l'orientamento del punto chiave e la rotazione del modello di campionamento da tale orientamento, BRISK diventa alquanto invariante alla rotazione.

Per calcolare l'orientamento del punto chiave, BRISK utilizzi gradienti locali tra le coppie di campionamento che sono definiti da:

$$g(\mathbf{p}_i, \mathbf{p}_j) = (\mathbf{p}_j - \mathbf{p}_i) \cdot \frac{I(\mathbf{p}_j, \boldsymbol{\sigma}_j) - I(\mathbf{p}_i, \boldsymbol{\sigma})}{\|\mathbf{p}_j - \mathbf{p}_i\|^2}$$

Dove $g(p_i, p_j)$ è il gradiente locale tra le coppie campionate (p_i, p_j), I è l'intensità livellata (da una gaussiana) nel corrispondente punto di campionamento per la deviazione standard appropriata

Considerando l'insieme $\mathcal{A}$ di tutte le coppie dei punti di campionamento:

$$\mathcal{A} = \{(\mathbf{p}_i, \mathbf{p}_j) \in \mathbb{R}^2 \times \mathbb{R}^2 \mid i < N \wedge j < i \wedge i, j \in N\}$$

definiamo un sottoinsieme di coppia *distanza-breve* $\mathcal{S}$ e un altro sottoinsieme di coppia *distanza-lunga* $\mathcal{L}$:

$$\mathcal{S} = \{(\mathbf{p}_i, \mathbf{p}_j) \in \mathcal{A} \mid \|\mathbf{p}_j - \mathbf{p}_i\| < \delta_{max}\} \subseteq \mathcal{A}$$

$$\mathcal{L} = \{(\mathbf{p}_i, \mathbf{p}_j) \in \mathcal{A} \mid \|\mathbf{p}_j - \mathbf{p}_i\| > \delta_{max}\} \subseteq \mathcal{A}$$

Le distanze di soglia sono impostate a $\delta_{max} = 9.75t$ e $\delta_{min} = 13.67t$ (t è la scala di k). Scorrendo le coppie di punti in $\mathcal{L}$, stimiamo il pattern caratteristico di direzione generale del punto chiave k in questo modo:

$$\begin{pmatrix} g_x \\ g_y \end{pmatrix} = \frac{1}{L} \cdot \sum_{(\mathbf{p}_i, \mathbf{p}_j) \in \mathcal{L}} \mathbf{g}(\mathbf{p}_i, \mathbf{p}_j)$$

Le coppie di *distanza-lunga* sono utilizzate per questo calcolo, basandosi sul presupposto che i gradienti locali si annullano a vicenda e non sono quindi necessari alla determinazione gradiente globale. Per calcolare l'orientamento normalizzata, BRISK applica il modello di campionamento ruotato di $\sigma = \arctan^2(g_y, g_x)$ intorno al punto chiave k.

Si noti che BRISK utilizzare solo coppie a *distanza-lunga* per calcolare l'orientamento; si basa sul presupposto che gradienti locali si annullano a vicenda quindi è necessario determinare un gradiente globale.

Costruzione del descrittore

Come con tutti i descrittori binari, la costruzione del descrittore viene fatto eseguendo il confronti tra le intensità.

Il descrittore (vettore di bit) d_k si forma confrontando le intensità dele coppie di punti a *distanza-breve* $(\mathbf{p}_i^\alpha, \mathbf{p}_j^\alpha) \in \mathcal{S}$ (ruota le coppie dall'orientamento calcolato in precedenza), tale che ogni bit b corrisponde a:

$$b = \begin{cases} 1 & \text{se } I(\mathbf{p}_j^\alpha, \sigma_j) > I(\mathbf{p}_i^\alpha, \sigma_i) \\ 0 & \text{altrimenti} \end{cases}$$

dove $\forall(\mathbf{p}_i^\alpha, \mathbf{p}_j^\alpha) \in \mathcal{S}$.

Il che significa che per ogni coppia a *distanza-breve* si prende l'intensità livellata dei punti di campionamento e verificando se l'intensità del primo punto nella coppia è maggiore di quella del secondo punto. Se è così, allora si scrive '1' nel bit corrispondente del descrittore e '0' altrimenti. Ricordiamo che BRISK utilizza solo i brevi coppie per costruire il descrittore.

2.5.3 Matching dei descrittori.

Il matching tra due descrittori è una semplice e veloce computazione della loro distanza di Hamming. Il numero di bit diversi nei due descrittori indica la misura della loro diversità.

Queste operazioni si riducono ad un XOR bit a bit, e possono essere calcolati molto rapidamente e in modo efficiente sulle architetture odierne attraverso le istruzioni speciali SSE (SSE hardware instructions).

Capitolo 3

Strumenti di apprendimento automatico

L'*apprendimento automatico* (noto in letteratura come Machine Learning) è un branca della computer science che si occupa di realizzare sistemi e algoritmi che possono *imparare*, basandosi su degli esempi dati in *input*. Questi sistemi utilizzano una *base di conoscenza* appresa precedentemente, per processare input simili, ma non uguali, in futuro. Il *Machine Learning* possiamo considerarla una delle aree fondamentali dell'intelligenza artificiale; uno degli obiettivi principali della ricerca in questo ambito, è quello di imparare a riconoscere automaticamente modelli complessi e prendere decisioni intelligenti basate su dati (provenienti da vari media) in ingresso.

La capacità di dare un nome, di riconoscere e di classificare oggetti in tipologie o categorie che si conformano a modelli teorici (pattern) è una delle caratteristiche principali dell'intelligenza umana.

Con il termine *classificazione*, in Machine Learning, si intende l'operazione che determina l'appartenenza di una osservazione ad una classe sulla base dei valori degli attributi che caratterizzano l'osservazione stessa. L'approccio consiste nell'etichettare ciascuna osservazione con un certo valore discreto tra un insieme ben definito di possibilità: data una nuova osservazione il classificatore (ossia il modello ottenuto con tale tecnica) indicherà l'etichetta che ritiene più plausibile, sulla base dei valori degli attributi.

La costruzione di un classificatore si basa su un insieme di informazioni ri-

guardanti alcuni tratti salienti (feature, caratteristiche salienti) di un insieme di osservazioni provenienti da specifici oggetti (immagine, video, pattern) dei quali conosciamo già l'appartenenza. A partire da tali esempi, il modello apprende le regole implicite che determinano l'appartenenza o meno a una certa classe.

Le informazioni (*features*) che si possono estrarre da una immagine per permetterne la classificazione sono molteplici. In genere l'utilizzo diretto di toni di grigio o colore dell'immagine, è raramente usato in applicazioni pratiche, perchè tali valori sono normalmente influenzati dalla luminosità della scena e anche perchè rappresenterebbero uno spazio di ingresso molto vasto, difficilmente gestibile. È necessario pertanto estrarre dalla parte di immagine da classificare delle informazioni essenziali (*features*) che ne descrivano l'aspetto al meglio. Questa parte sarà descritta ampiamente nel Capitolo 4 ed è molto importante per lo scopo di questa tesi, nonché in *machine learning*.

Il compito di *classificare* oggetti, cose, persone è molto ricorrente nelle attività della vita di tutti i giorni delle persone umane, basandosi su una conoscenza appresa durante il corso della vita, una persona è in grado di prendere decisioni; tuttavia questo compito resta molto difficile per un computer.

In diversi contesti applicativi è interessante riuscire a costruire un sistema che "impari" a classificare correttamente una serie di campioni, deducendo quali sono i parametri più significativi che li caratterizzano. Ad esempio:

- In campo medico, un sistema che, ottenuti un numero sufficiente di dati clinici, sia in grado di scoprire se un paziente è affetto da una certa patologia, o quale sia la gravità della stessa.

- Un "classificatore di testi" che sia in grado di dire se un testo è attinente o no a un argomento prestabilito in base solo ai termini che vi compaiono.

- In ambito di sicurezza informatica, "classificatore di email" : dato un insieme di email contraddistinte come spam e non-spam, sia in grado di classificare nuove email e decidere se sono spam o no.

- Un "classificatore di immagini" che, dati alcuni punti appartenenti a un immagine, sia in grado di riconoscere gli oggetti rappresentati.

3.1 Tipologie di classificatori

I metodi di classificazione sono numerosi in letteratura e tali classificatori possono essere anche combinati tra loro. Brevemente illustrerò alcuni dei modelli di classificazione:

- **Statistici (es: Naive Bayes)** memorizzano i parametri delle varie distribuzioni di probabilità relative alle classi ed agli attributi. Per classificare un generico oggetto si possono stimare le probabilità di appartenenza alle varie classi;
- **Basati sugli esempi: (es: Nearest neighbor)** memorizzano tutti gli esempi del training set ed assegnano la classe ad un oggetto valutando la "somiglianza" con gli esempi memorizzati (la cui classe è nota);
- **Matematici (es: Reti Neurali Artificiali, SVM)** attraverso gli esempi di training si crea una funzione matematica di classificazione che modella i dati. Questa viene utilizzata sui nuovi dati di test per individuare l'etichetta corretta.
- **Logici (es: Alberi e Regole di Decisione)** la funzione di classificazione è espressa mediante condizioni logiche sui valori degli attributi estratti dai dati.

3.2 Classificazione dei problemi di learning

I problemi di learning vengono divisi in due tipologie principali: *supervised learning* e *unsupervised learning*. Inoltre si può distinguere tra modelli parametrici e non parametrici.

3.2.1 Supervised learning

Il supervised learning è il problema più studiato nel machine learning. Esso si pone l'obiettivo di prevedere, dato un elemento di cui si conoscono un insieme di parametri (features), il valore di un diverso parametro di output relativo al nuovo elemento. Per fare questo viene utilizzato un *modello*, attraverso l'apprendimento effettuato da un insieme di esempi, come già detto precedentemente.

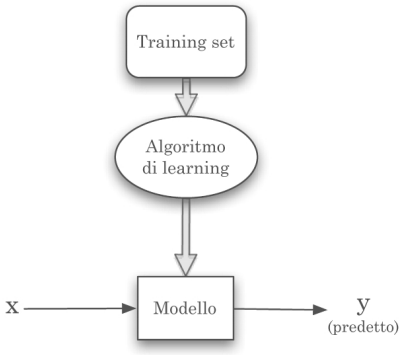

Figura 3.1: Rappresentazione supervised Learning

Il paradigma utilizzato è quello di utilizzare un insieme di "dati di allenamento" (*training set*) già correttamente classificati e di costruire una "funzione di classificazione" basandosi sui tali dati. Deve essere in grado di classificare in modo corretto, entro una certa soglia, dati (*test set*) che potrebbero risultare al di fuori di questo insieme.

Il problema è definito a partire da un universo di elementi, descritto attraverso un insieme di **x** *features* considerate come input del problema, ad ogni elemento è poi associato un valore **y** di output (*etichetta*). Partendo dalla conoscenza di un insieme $\mathcal{T}$ di elementi (*training set*), ciascuno descritto da una coppia (x_i, y_i), dove x_i è il vettore dei valori delle d features $x^{(1)}$,$x^{(2)}$,...,$x^{(d)}$ e y_i è il relativo output, derivare un *modello* della relazione (sconosciuta) tra features e valori di output, che consenta, dato un nuovo elemento **x**, di predire il corrispondente valore di output **y**.

Un approccio comune al problema è quello di derivare dal training set una funzione *h* tale che $y(x)$ sia una buona previsione del valore sconosciuto *y*. Un approccio alternativo è derivare, a partire dal training set, una distribuzione di probabilità $p(y|\mathbf{x})$ per l'output, che consenta ad esempio di ottenere anche dei valori di confidenza. Quella che in effetti verrà utilizzata è la distribuzione a posteriori $p(y|\mathbf{x}, \mathcal{T})$, dove "a posteriori" indica successivamente all'osservazione del training set.

Una volta derivato un modello, una possibile misura della sua qualità nell'effettuare predizioni è la *verosimiglianza a posteriori*, che misura, su un insieme di test di elementi (features e output), la probabilità che il modello assegna ai valori

corretti, a partire dalle features

$$L(\mathcal{M}|\mathcal{T}) = \prod_{i=1}^{n} p(y_i|\mathbf{x}_i, \mathcal{M}, \mathcal{T})$$

Una misura più semplice della qualità del modello è naturalmente la percentuale di errore.

3.2.2 Unsupervised learning

L'apprendimento *non supervisionato* è una tecnica di apprendimento automatico che consiste nel fornire al sistema informatico una serie di input (esperienza del sistema) che egli riclassificherà ed organizzerà sulla base di caratteristiche comuni per cercare di effettuare ragionamenti e previsioni sugli input successivi.

I dati consistono soltanto di un insieme di features $\mathbf{x}_i$, senza variabili di output y_i. Quel che si vuole è individuare un modello che si adatti ai dati, al fine di scoprire proprietà interessanti di essi. La caratteristica distintiva di questo metodo, al contrario dell'apprendimento supervisionato, è che durante l'apprendimento vengono forniti solo esempi *non è etichettati*, e quindi non sono note a priori, ma devono essere apprese automaticamente.

Un esempio tipico di questi algoritmi lo si ha nei motori di ricerca. Questi programmi, data una o più parole chiave, sono in grado di creare una lista di link rimandanti alle pagine che l'algoritmo di ricerca ritiene attinenti alla ricerca effettuata. La validità di questi algoritmi è legata alla utilità delle informazioni che riescono ad estrarre dalla base di dati, nell'esempio sopracitato è legata all'attinenza dei link con l'argomento cercato.

Le tecniche di apprendimento non supervisionato lavorano confrontando i dati e ricercando similarità o differenze. Sono molto efficienti con elementi di tipo numerico, dato che possono utilizzare tutte le tecniche derivate dalla statistica, ma risultano essere meno efficienti con dati non numerici.

Questi algoritmi lavorano correttamente in presenza di dati contenenti un ordinamento o un raggruppamento netto e chiaramente identificabile, al contrario possono rischiare di fallire.

Uno dei principali algoritmi è il **clustering** in due varianti: *gerarchico* e *partizionale* che verranno affrontati nel Capitolo 3.11.

3.2.3 Modelli parametrici e non parametrici

Nei modelli *parametrici* si conosce, o si ipotizza un modello parametrico per le $p(\mathbf{x}|y)$, il problema diventa quello di stimare i parametri di tale modello (ad esempio stimare il valore medio e la deviazione standard di un modello Gaussiano). Essendo preventivamente caratterizzato da un vettore θ di parametri: quel che quindi accade è che viene ipotizzato che la relazione tra features e input sia rappresentabile all'interno di una famiglia di relazioni (modelli) parametrici rispetto a θ, e quindi tali che una assegnazione di valori a θ definisca uno specifico modello della famiglia. Gli elementi nel training set sono successivamente utilizzati per derivare tale assegnazione di valori ai parametri (o una distribuzione di probabilità per tali valori), dopo di che non sono più utilizzati.

Nei modelli *non parametrici* non vengono fatte ipotesi a priori sulle distribuzioni delle classi, ma vengono fatte assunzioni sulle distribuzioni delle probabilità. Si distinguono due famiglie di metodi:

- quelli denominati "density estimation" tentano di derivare le densità di probabilità delle classi dai campioni del training set, per poi utilizzare la regola di Bayes.
- altri metodi sfruttano direttamente le informazioni dei pattern del training set per la classificazione (es: regola Nearest neighbor).

I metodi del secondo tipo sono spesso preferibili, in quanto il problema generale della stima accurata delle distribuzioni è un problema più complesso della classificazione stessa.

Oltre a Nearest neighbor, in seguito analizzeremo altri approcci non paramentri: *Support vector machines* e *Neural Network* attraverso l'algoritmo *Multi-Layer Perceptron*.

3.3 Modelli lineare per la Classificazione

Lo scopo della classificazione è quello di prendere un vettore di input $\mathbf{x}$ e assegnarlo a una della K classi discrete $\mathcal{C}_k$ dove $k = 1, ..., K$. Nello scenario classico, le classi sono prese disgiunte, in modo tale che ogni input sia assegnato a una e una sola classe. Lo spazio di input viene quindi suddiviso in *regioni di decisione*.

Per *modello di classificazione lineare* si intende che i confini di decisione sono funzioni lineari dell'input x (i quali sono definiti da iperpiani a $D-1$ dimensioni nello spazio D-dimensionale nelle features). Insiemi di dati le cui classi possono essere separate in modo esatto per mezzo di superfici lineari sono detti linearmente separabili.

Per modelli probabilistici, il modello più conveniente, nel caso di *classificazione* di due classi di problemi, è la rappresentazione binaria, nel quale c'è una sola variabile target $t \in \{0, 1\}$, dove $t = 0$ indica la classe C_0 e $t = 1$ indica la classe C_1. Nel caso di di problemi con $K > 2$ classi, è conveniente usare soluzione "1 su K". Viene associato ad ogni classe un vettori di K bit, tale che per ogni classe C_j tutti i bit hanno un valore 0, eccetto il j-esimo, pari a 1. Per esempio, se abbiamo $K = 5$ classi, il pattern per la classe 2, sarà il vettore dato: $\mathbf{x} = (0, 1, 0, 0, 0)^T$[48].

3.3.1 Approcci alla classificazione

Possiamo individuare tre approcci generali al problema della classificazione:

- determinare la funzione $f : \mathbf{X} \mapsto \{1, ..., K\}$ chiamata *funzione discriminante* che mappa ogni input x in una classe C_i (con $i = f(\mathbf{x})$)
- determinare le distribuzioni a posteriori $f(C_j|\mathbf{x})$ (fase di inferenza); utilizzare queste distribuzioni per assegnare gli input alle classi (fase di decisione).
- determinare le distribuzioni degli elementi condizionate alle classi $f(\mathbf{x}|C_j)$, oltre alle distribuzioni a priori $f(C_j)$. Utilizzare il teorema di Bayes per derivare le distribuzioni a posteriori $f(C_j|\mathbf{x})$ e utilizzare queste distribuzioni nella fase di decisione, per assegnare gli input alle classi [48].

I primi due modelli sono **discriminativi**, infatti effettuano la classificazione cercando di individuare, sulla base del training set, delle condizioni (ad es. delimitazioni di regioni) che facciano sì che un elemento appartenga all'una o all'altra classe. Viene detto *discriminativo*, perchè, a partire da $\mathcal{T}$, viene derivata una caratterizzazione dell'output in funzione delle features, in modo tale da discriminare, dato un elemento, il più probabile tra i possibili valori dell'output. Questo è l'approccio utilizzato nel modello Bag of Words, utilizzato in questo lavoro di tesi, che illustrerò nel dettaglio nel Capitolo 4.2.

Il terzo approccio è di tipo **generativo**, partendo da $\mathcal{T}$, viene derivata una caratterizzazione delle features in funzione dell'output, in modo tale da generare,

dato un possibile output, un elemento che con buona probabilità potrà essere associato a quell'output. Sostanzialmente, viene derivato, per ogni possibile output, un modello (sotto forma di distribuzione di probabilità) degli elementi associati a quell'output.

Il teorema di Bayes consente, a partire da $p(\mathbf{x}|y)$, e nota la distribuzione a priori *p(y)*, di ottenere $p(y|\mathbf{x})$, in quanto

$$p(y|\mathbf{x}) = \frac{p(\mathbf{x}|y)p(y)}{p(\mathbf{x})}$$

Nel modello di regressione lineare il modello di predizione viene dato da una funzione lineare di parametri $\mathbf{w}$, attraverso la seguente equazione $y = (\mathbf{w}, \mathbf{x})$. Nell'ambito dei problemi di classificazione, tuttavia occorre predire etichette di classe discrete, o più generalmente probabilità a posteriori nel range [0,1]. Per raggiungere questo, consideriamo la generalizzazione di questo modello nel quale trasformiamo la funzione lineare $\mathbf{w}$ usando una funziona non lineare $f(\cdot)$ in questo modo

$$y(\mathbf{x}) = f(\mathbf{w}^T\mathbf{x} + w_0).$$

In letteratura nell'ambito del Machine Learning, la funzione $f(\cdot)$ viene chiamata *activation function*, mentre la sua inversa in Statistica viene chiamata *link function*.

Formalizziamo il problema di classificazione in presenza di un problema a due classi, nel seguente modo: sia X un generico insieme (solitamente $X = \mathbb{R}^n$); abbiamo un insieme di coppie

$$T = \{(x_i, y_i), x = 1, 2, ..., m\} \subset X \times \{-1, +1\},$$

detto *training set*, e vogliamo trovare una funzione $f : X \to \{-1, +1\}$ all'interno di una classe $\mathcal{F}$ il cui grafico passi per le coppie date, o si avvicini ad esse il più possibile. Una volta trovata questa funzione, possiamo applicarla per classificare nuovi punti al di fuori del *training set*: a seconda che $f(x)$ valga $+1$ o -1, classificheremo il punto $x \in X$ in una delle due classi.

Il caso più semplice è quello della *classificazione lineare*, come notiamo in Figura 3.2, cioè il problema di determinare un iperpiano $\{x|\langle w, x\rangle + b = 0\}$ in $X = \mathbb{R}^n$ in modo che un insieme di punti dati P stia nel semispazio positivo delimitato da esso e un altro insieme di punti N stia nel semispazio negativo: in questo caso allora $f = sgn(\langle w, x\rangle + b)$.

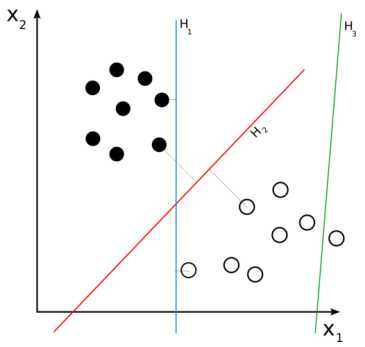

Figura 3.2: Un problema di classificazione lineare: trovare il piano che separa punti bianchi e punti neri [11]

3.4 Classificatori bayesiani

Il classificatore bayesiano, come si intuisce dal nome stesso, si basa sul Teorema di Bayes, per calcolare la probabilità a posteriori, stimando i dati statistici necessari a partire dal training set. Si parla in questo caso di addestramento supervisionato. È fondato sulle proprietà che legano le probabilità congiunte e condizionate di due eventi distinti A e B.

3.4.1 Apprendimento bayesiano

L'apprendimento bayesiano si pone come obiettivo il problema di fare delle previsioni e ritiene il problema della formulazione di ipotesi a partire dai dati, come un suo sottoproblema. Un modo per specificare che cosa intendiamo per *la migliore ipotesi* è quello di affermare che la migliore ipotesi è quella più probabile, avendo a disposizione dei dati ed una certa conoscenza iniziale sulla probabilità a priori delle varie ipotesi . Le ipotesi elaborate dai dati e combinate in modo opportuno, portano alla formulazione di una previsione.

Supponiamo di avere un insieme di possibili classi C e una serie di attributi (o feature) $A_1, A_2, ..., A_k$ usati per discriminare tra le classi, e indichiamo con lettere minuscole $(c, a_1...)$ particolari valori assunti dalle variabili. La classificazione ottimale sarà allora quella per cui la probabilità di una certa classe, data una serie di valori assunti dagli attributi, è massima:

$$\max \ \mathbf{P}(C = c | A_1 = a_1 \wedge A_2 = a_2 \wedge ... \wedge A_k = a_k)$$

Per il teorema di Bayes si ha che questa probabilità è uguale a:

$$\max \frac{\mathbf{P}(A_1 = a_1 \wedge A_2 = a_2 \wedge ... \wedge A_k = a_k | C = c)\mathbf{P}(C = c)}{\mathbf{P}(A_1 = a_1 \wedge A_2 = a_2 \wedge ... \wedge A_k = a_k)} \tag{3.1}$$

La probabilità a priori di avere una data classe c si può stimare facilmente dal training set (a patto che sia composto in maniera da essere rappresentativo), mentre il denominatore dell'equazione è irrilevante, dato che è lo stesso per tutte le classi dell'insieme C in quanto da esse indipendente. Si tratta quindi di riuscire a stimare la probabilità condizionale a numeratore per ciascuno degli attributi; fatto questo, si otterrebbe un classificatore Bayesiano ottimo, in grado di fornire la classificazione migliore in assoluto. Il problema è che tale stima è molto complicata a causa delle dipendenze tra attributi, e anche con poche feature rischia di diventare computazionalmente irrealizzabile.

Per ovviare al problema si può usare un Naive Bayes Classifier che, come dice il nome, fa un'ipotesi piuttosto "naive" sulle feature, ovvero che siano completamente indipendenti.

3.4.2 Naive Bayes Classifier

Un classificatore *Naive Bayes* secondo la Statistica Bayesiana è un semplice classificatore probabilistico basato sull'applicazione del teorema di Bayes sotto ipotesi di indipendenza "strong", *le caratteristiche sono indipendenti dal modello*.

Naive Bayes presuppone che la presenza (o l'assenza) di una particolare caratteristica di una classe, *non sia correlato* alla presenza (o l'assenza) di qualsiasi altra caratteristica. Ad esempio, un frutto può essere considerata una mela se è rosso, rotondo, e circa 4cm di diametro. Anche se queste caratteristiche dipendono dall'esistenza delle altre caratteristiche, un classificatore Naive Bayes considera tutte queste proprietà indipendentemente. Tutti gli attributi che descrivono una certa istanza sono tra loro condizionalmente indipendenti data la categoria a cui appartiene l'istanza.

Questa affermazione viene detta assunzione del Naive (ingenua) Bayes. L'assunzione di indipendenza permette di apprendere separatamente i parametri di ogni attributo, semplificando molto l'apprendimento, specialmente in quelle situazioni in cui il numero di attributi è molto elevato ed in cui i dati a disposizione non sono molto numerosi.

Descriviamo di seguito il modello probabilistico che sta alla base dei *classificatori Bayesiani naive*. Si ha che

$$\mathbf{P}(a_1, a_2, .., a_n|c_k) = \mathbf{P}(a_1|c_k) \cdot \mathbf{P}(a_2|c_k) \cdots \mathbf{P}(a_n|c_k) = \prod_{i=1}^{n} \mathbf{P}(a_i|c_k)$$

Sostituendo quest'ultima equazione alla equazione 3.1, si ottiene:

$$\arg \max_{c_j \in C} \mathbf{P}(c_j) \prod_{i=1}^{n} \mathbf{P}(a_i|c_k)$$

I classificatori *naive* sono piuttosto semplici da realizzare e non richiedono un grande carico computazionale, nonostante le istanze possono essere descritte mediante un insieme di attributi di cardinalità anche molto elevata. Le assunzioni sull'indipendenza delle feature sono il più delle volte irrealistiche, ma ciononostante le prestazioni di questi classificatori sono piuttosto buone se ben addestrati. Questo metodo si è rivelato utile in molte applicazioni, tra cui la classificazione di documenti testuali.

3.4.3 Normal Bayes Classifier

Il modello *Normal Bayes* presuppone che i vettori di feature di ogni classe sono distribuiti secondo la Distribuzione Normale (anche se non necessariamente distribuite in modo indipendente). Il classificatore Naive Bayes stima una distribuzione normale separata per ogni classe calcolando la media e la deviazione standard dei dati di training in quella classe. Per classificare un esempio, si sceglie la classe sotto la cui densità di classe condizionale, ha l'istanza con più alta probabilità.

3.4.4 Reti bayesiane

Per trovare una via di mezzo tra questi due estremi (dipendenza totale tra tutte le feature / indipendenza totale) si può usare l'approccio delle *reti bayesiane*, un particolare tipo di grafi diretti aciclici che consentono di rappresentare relazioni di causalità tra feature, riuscendo a tenere in considerazione solo le dipendenze effettive tra attributi alleggerendo notevolmente il calcolo rispetto al caso di classificatore ottimale.

3.5 Classificazione basata sulle istanze

La classificazione basata sulle istanze (*instances-base learning*) è una tecnica molto particolare perché a differenza delle altre tecniche che mirano alla creazione di un modello dei dati di stima, essa basa la procedura di previsione direttamente sui dati già classificati, mediante il confronto delle caratteristiche delle osservazioni.

Per questo motivo tale tecnica viene detta *lazy learning*: è pigra proprio perché nella fase di training non fa assolutamente nulla per comprendere i dati. Risulta completamente in antitesi con l'approccio classico, che concentra lo sforzo maggiore nella fase di *training*, al fine di generare un'astrazione dei dati stessi.

3.5.1 Nearest Neighbor classifiers

Data una metrica $d(\cdot)$ nello spazio multidimensionale (es. distanza euclidea) il classificarore *nearest-neighbor* (letteralmente "il più vicino tra i vicini"), classifica un pattern $\mathbf{x}$ con la stessa classe dell'elemento $\mathbf{x'}$ ad esso più vicino nel *training set* (TS):

$$d(\mathbf{x}, \mathbf{x}') = \min_{\mathbf{x}_i \in TS} \{d(\mathbf{x}, \mathbf{x}_i)\}$$

Invece di derivare dai dati le distribuzioni condizionali delle classi per poi usare la regola di Bayes per la classificazione, questo classificatore cerca in modo piuttosto pragmatico di massimizzare direttamente la probabilità a posteriori; infatti se $\mathbf{x'}$ è molto vicino a $\mathbf{x}$ è lecito supporre che:

$$P(\omega_i|\mathbf{x}) \approx P(\omega_i|\mathbf{x'}).$$

La regola nearest-neighbor produce un partizionamento dello dello spazio, noto come **tassellazione di Voronoi**.

Ogni elemento $\mathbf{x}_i \in TS$ determina un tassello, all'intero del quale i pattern saranno assegnati alla stessa classi di $\mathbf{x}_i$.

La regola di classificazione *nearest-neighbor* è piuttosto radicale; infatti basta che un elemento del training set non sia molto "affidabile" (outlier) affinchè tutti i pattern nelle sue vicinanze siano in seguito etichettati non correttamente. Un modo generalmente più robusto, che può essere visto come estensione della regola nearest-neighbor (in questo caso detta 1-nearest-neighbor) è il cosiddetto classificatore *k-nearest-neighbor* [12].

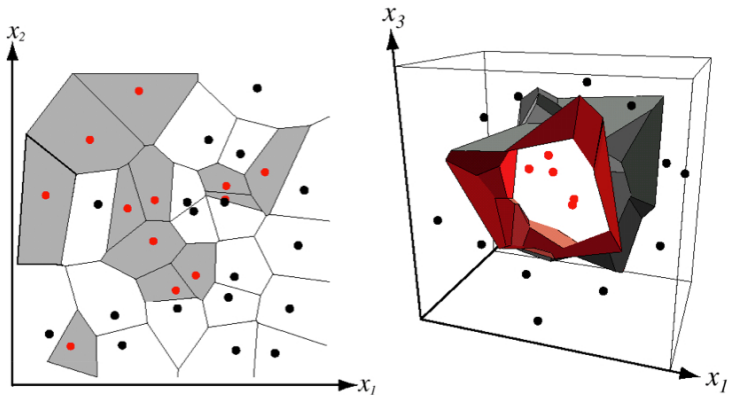

Figura 3.3: Rappresentazione tassellazione di Voronoi in 2D e 3D [12]

3.5.2 K-Nearest Neighbors (KNN)

L'algoritmo *K-Nearest Neighbor* deriva direttamente dall'approccio Nearest Neighbor ed è un algoritmo più generale e flessibile del successivo Nearest Neighbor, infatti, invece di considerare il solo campione più vicino al punto $\mathbf{x}_0$, ne esamina i k più prossimi. Il pattern x viene assegnato alla classe che ha ottenuto il maggior numero di voti. Se $k = 1$, all'oggetto viene assegnata la classe dell'elemento ad esso più vicino e corrisponde all'algoritmo *nearest-neighbor*.

Questo algoritmo può essere definito di tipo semi-supervisionato in quanto non fa uso delle etichette assegnate agli oggetti durante la ricerca dei vicini, ma solo nell'ultima fase di votazione. La fase di addestramento consiste nel semplice salvataggio dei vettori di feature e delle etichette per tutti gli elementi dell'insieme di training. La classificazione di un vettore di feature non etichettato avviene cercando la classe che appare più spesso nell'insieme dei suoi k vicini.

Nella figura 3.4 il classificatore 5-NN, assegna x alla classe "nera" in quanto quest'ultima ha ricevuto 3 voti su 5. Per Training Set infiniti la regola di classificazione k-NN si dimostra migliore di 1-NN, e all'aumentare di k, l'errore P converge all'errore Bayesiano. Nella pratica (Training Set limitati), aumentare k significa estendere l'iper-sfera di ricerca andando a sondare la probabilità a posteriori lontano dal punto di interesse; pertanto il valore di k migliore (solitamente < 10) deve essere determinato sperimentalmente [12], ad esempio usando la tecnica di *k-fold cross-validation*, illustrata in dettaglio nel Capitolo 6.1.2.

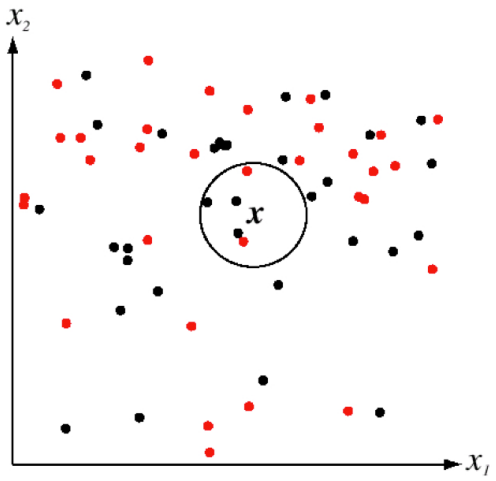

Figura 3.4: Rappresentazione 5-Nearest-Neighbor [12]

Stima della vicinanza

Per stimare la "vicinanza" tra due elementi è necessario stabilire quale criterio di distanza impiegare. In genere viene impiegata la *distanza euclidea*, detta anche *distanza L2* in quanto pari alla norma-2 del vettore differenza:

$$d_{L2}(x, y) = \sqrt{\sum_{i=1}^{n} (x_i - y_i)^2} = ||x - y||$$

Una formula alternativa per il calcolo della distanza è la cosiddetta χ^2 (*chi-square*), una forma particolare di distanza L2 quadratica pesata: in essa gli elementi del vettore devono rappresentare delle frequenze relative e le differenze quadratiche tra gli elementi sono pesate da un parametro associato al loro valore medio. In formule:

$$\chi^2(\bar{x}, \bar{y}) = \sum_{i=1}^{n} \frac{1}{2((x_i + y_i)} \cdot (x_i - y_i)^2$$

$$\bar{x} = [\bar{x}_1, ..., \bar{x}_n] \quad \bar{x}_i = \frac{x_i}{\sum_{i=1}^{n} x_k} = \frac{x_i}{||x||_1}$$

con $\bar{x}$ ad indicare i vettori normalizzati secondo la norma-1.
La scelta del valore di k dipende dai dati a disposizione per l'addestramento ed il test: valori più grandi riducono l'effetto del rumore nella classificazione, ma rendono i confini tra le classi meno distinti. Un buon valore di k può essere scelto eseguendo più prove modificando i dati del training-set.

3.6 Neural Network

Le *reti neurali artificiali* sono modelli matematici che rappresentano l'interconnessione tra elementi, per la simulazione di una rete di neuroni biologici (es. sistema nervoso umano), definiti *neuroni artificiali*. Questi modelli possono essere utilizzati per risolvere problemi ingegneristici di intelligenza artificiale come quelli che si pongono in diversi ambiti tecnologici (in elettronica, informatica, simulazione, e altre discipline). Può essere composta sia da programmi, che da hardware dedicato.

Le *reti neurali artificiali* simulano i diversi aspetti legati al comportamento e alle capacità del cervello umano:

- elaborazione intelligente delle informazioni;
- elaborazione distribuita;
- elevato livello di parallelismo;
- facoltà di apprendimento, di generalizzazione e di adattamento
- tolleranza ad informazioni poco precise (o sbagliate)

Esse si rifanno ad un modello connesso di calcolo, in cui la computazione è effettuata per mezzo dell'interazione tra unità elementari (i neuroni) che effettuano, singolarmente, operazioni molto semplici e si scambiano informazioni per mezzo dei collegamenti che li uniscono.

Le applicazioni sono molto varie, qui elenchiamo le più comuni: sistemi di filtraggio del segnale (es. eliminare il rumore da un segnale), sistemi di classificazione, previsione di serie temporali (Time Series Forecasting), riconoscimento (es. volti o delle espressioni facciali).

3.6.1 Percetron

Il caso più semplice di rete neurale è il *percettrone* (Figura 3.5), di fatto un singolo neurone, gli ingressi sono moltiplicati per dei *pesi*, rappresentativi dell'entità delle connessioni sinaptiche, e la somma algebrica pesata degli ingressi viene confrontata con un valore di *soglia*; il neurone fornisce l'uscita 1 se la somma pesata degli ingressi è maggiore del valore di soglia, e l'uscita -1 (oppure 0) altrimenti. Assegnato un vettore di ingresso $x \in \mathbb{R}^n$, indicando con $w \in \mathbb{R}^n$, il vettore dei

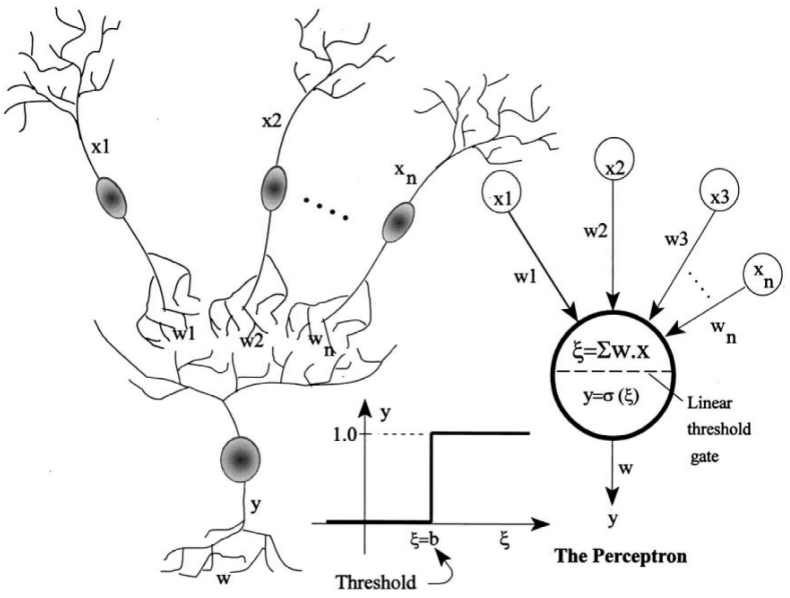

Figura 3.5: Confronto tra neurone biologico e artificiale [13]

pesi, con $\theta \in \mathbb{R}$ il valore di soglia, e con $y \in \{+1, -1\}$ l'uscita del neurone, si può porre

$$y(x) = g(\sum_{i=1}^{n} w_i x_i - \theta) \equiv g(w^T x - \theta),$$

dove g, detta *funzione di attivazione* del neurone, può essere definita, ad esempio, attraverso la *funzione segno*:

$$g(x) \equiv \mathrm{sgn}(t) = \begin{cases} +1 & \text{se } t \geq 0 \\ -1 & \text{se } t < 0 \end{cases} =$$

Un'interpretazione molto più significativa del neurone è tuttavia quella di considerarlo come un *classificatore lineare*, che attribuisce a un generico ingresso x (le cui componenti scalari definiscono le *caratteristiche* dell'oggetto da classificare) il valore $y(x) = 1$ oppure $y(x) = -1$ (e quindi effettua una classificazione binaria), in base al segno della *funzione discriminante* lineare $w^T x - \theta$. L'aspetto più rilevante e innovativo rispetto agli altri modelli di calcolo consiste nel fatto che i valori dei pesi e della soglia possono essere determinati attraverso un processo di *apprendimento* a partire da un *insieme di addestramento* costituito da coppie ingresso-uscita, in cui all'ingresso viene associata la classificazione corretta. Il neurone addestrato usando i campioni dell'insieme T può essere utilizzato

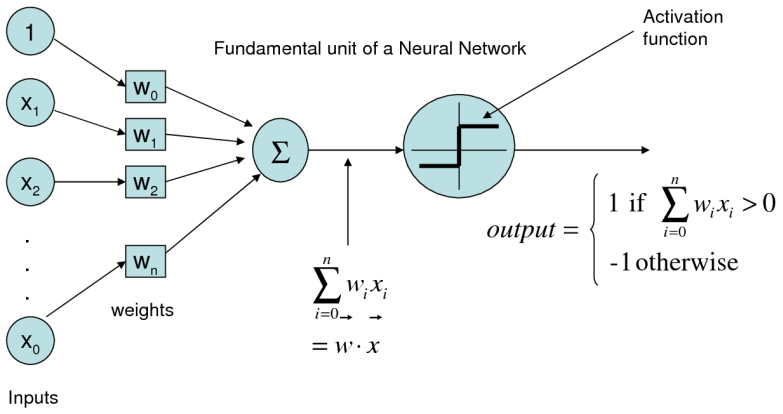

Figura 3.6: Rappresentazione del neurone [14]

successivamente per classificare nuovi ingressi x non appartenenti a T (*generalizzazione*). Si definisce così un strumento per il riconoscimento automatico di configurazioni che è stato denominato *Perceptron*.

Come è facile intuire, però , il percettrone ha potenzialità piuttosto limitate, è dimostrano l'impossibilità di risolvere molte classi di problemi, ossia tutti quelli non caratterizzati da separabilità lineare delle soluzioni. Questo tipo di rete neurale non è abbastanza potente, infatti non è in grado di calcolare neanche la funzione logica *or esclusivo (XOR)*, in quanto non linearmente separabile.

Si passa quindi all'utilizzo di topologie di rete più complesse, che prevedono uno strato di neuroni di input, uno strato di output e uno o piu' strati nascosti (Figura 3.7); sono poi presenti varianti ricorsive, in cui cioè gli output vengono reindirizzati verso lo strato di input o gli strati nascosti.

3.6.2 Multilayer perceptron

L'architettura di una rete neurale *multi-layer*, mostrata in Figura 3.7, può essere descritta definendo:

- un insieme di n nodi di ingresso, sprovvisti di capacità di elaborazione, associati agli *n ingressi* della rete: $x_i \in R, i = 1, ..., n$;
- un insieme di *neuroni* organizzati in $L \geq 2$ diversi *strati* di cui:
  - L-1 *strati nascosti (hidden layers)*, costituiti ciascuno da neuroni le cui uscite contribuiscono agli ingressi dei neuroni dello strato successivo;

  - uno *strato di uscita* costituito da $K \geq 1$ neuroni le cui uscite costituiscono le *uscite* della rete $y_i \in R$, $i = 1, ..., K$;
- un insieme di archi orientati e pesati che rappresentano le connessioni interneurali e le connessioni con i nodi di ingresso; si suppone che non esistano connessioni tra i neuroni di uno stesso strato, né connessioni in feedback tra le uscite dei neuroni di uno strato e gli ingressi dei neuroni degli strati precedenti.

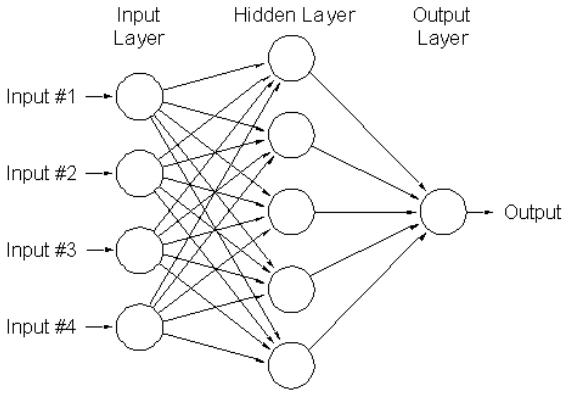

Figura 3.7: Rappresentazione Multilayer perceptron [15]

Ciascun neurone è caratterizzato da una funzione di attivazione, che opera su una combinazione pesata degli ingressi e di un valore di soglia, e che può essere scelta in base alle esigenze (step, tangente iperbolica, sigmoide...). Inoltre ciascun collegamento ha un peso che moltiplica il valore di input di quel collegamento, e proprio sui pesi si concentra la fase di addestramento della rete.

Prendiamo il caso più semplice, l'addestramento supervisionato: in questa situazione possiamo stabilire tutti i valori di input della rete e i rispettivi valori di output desiderati: ciò che resta da determinare sono i vari pesi, in modo da minimizzare l'errore tra l'output ottenuto e quello desiderato.

3.6.3 Addestramento di una rete neurale: Backpropagation

L'algoritmo più noto per l'addestramento è quello della *Backpropagation*: si parte dall'errore all'uscita della rete, e si calcola di quanto vanno corretti i pesi in uscita dall'ultimo strato di neuroni per avvicinarsi al risultato ideale, usando una

stima di massima verosimiglianza (minimizzare l'errore equivale a massimizzare la "probabilità dei dati"). Questa correzione può essere vista come l'errore dallo strato precedente, che quindi ripete la procedura e cerca di aggiustare a sua volta i propri pesi per avvicinarsi all'output richiesto: in sostanza l'errore si "propaga" per tutta la rete. Il procedimento viene ripetuto iterativamente, finchè non viene raggiunto un errore finale ritenuto tollerabile.

È importante notare che ottenere un errore eccessivamente piccolo non è un risultato positivo, perché in questo caso significa che la rete ha "imparato a memoria" i valori del training set, e se le verranno presentati nuovi dati difficilmente riuscirà a lavorare con successo perché ha perso in generalità nel corso dell'addestramento. Questo problema è noto con il termine di *overfitting*, ed esistono alcuni metodi per ridurne l'impatto, tra cui i più comuni sono l'early stopping e il weight decay.

3.7 Support Vector Machines (SVM)

Support Vector Machine è una tecnica di apprendimento supervisionato che permette di risolvere il problema della classificazione di pattern. Essa ha radici nella teoria di learning statistico, è stata sviluppata negli anni *'90* da *Vladimir Vapnik*[49] presso i laboratori Bell AT&T.
Date due classi di pattern multidimensionali, SVM determina l'iper-piano di separazione, *Maximal Margin Hyperplane - MMH*, in grado di separare al meglio le classi (ovvero che classifica correttamente pattern di entrambe con il maggior *margine* possibile). Quando le classi non sono linearmente separabili (non esiste un iper-piano in grado di separare le classi), viene eseguito a priori un *mapping* dei pattern su di uno spazio di dimensione superiore, dove i maggiori gradi di libertà permetto probabilmente di separare le classi.

Si consideri la Figura 3.8a che mostra un insieme di dati appartenenti a due classi diverse. Il dataset è *linearmente separabile*, dalla figura si vede che esistono infiniti iperpiani che possono dividere i due insiemi, ma anche se tutti hanno un training error nullo non è detto che si comporteranno bene anche con esempi non visti. La scelta deve essere fatta in modo da ottenere il miglior risultato su un nuovo test set. In Figura 3.8b sono mostrati due iperpiani diversi, a ciascun decision boundary B_i è associata una coppia di iperpiani b_{i1} e b_{i2}, la distanza tra

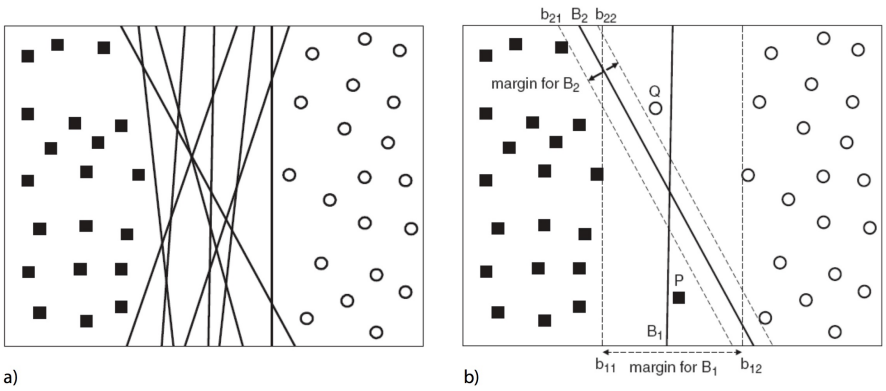

Figura 3.8: a) Possibili iperpiani di separazione per un dataset linearmente separabile b) Margine di un decision boundary [16]

di essi è identificata come *margine* del classificatore.
Intuitivamente se il *margine* è molto sottile ogni piccola perturbazione sugli iperpiani può avere forti ripercussioni sulla classificazione, inoltre classificatori con un margine minore sono più soggetti all'overfitting.
Solo alcuni degli esempi del training set sono importanti per definire il margine (e quindi l'iperpiano), mentre gli altri possono essere ignorati. Questi esempi sono detti *vettori di supporto* (support vectors); da cui deriva il nome dell'algoritmo.

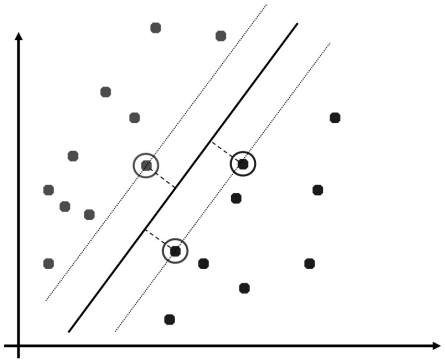

Figura 3.9: Rappresentazione dell'iperpiano e dei Support Vectors [17]

Le SVM hanno alcune interessanti proprietà

- overfitting altamente improbabile;
- possibilità di gestione di dati multidimensionali e classificazione multiclasse;
- possibilità di individuare un sottoinsieme di esempi di training effettivamente necessari alla classificazione, detti vettori di supporto.

In seguito descriveremo il funzionamento di SVM per un problema di classificazione binaria; nella pratica questa metodologia viene impiegata anche per la classificazione multiclasse.

3.7.1 Linear SVM: Caso separabile

La classificazione binaria può essere vista come un problema di separazione di classi nello spazio delle *feature*. Consideriamo un problema di classificazione binaria con N esempi di training. Ogni esempio è una tupla del tipo (x_i, y_i) dove $x_i = (x_{i1}, x_{i2}, x_{i3}, ..., x_{id})^T$ è un vettore che identifica i valori degli attributi del record. Convenzionalmente y_i rappresenta la classe $C = \{+1, -1\}$ a cui appartiene l'esempio.
Il confine di classificazione (decision boundary) di un classificatore lineare può essere scritto come: $y(\mathbf{x}) = \mathbf{w} \cdot \mathbf{x} + b$, dove $\mathbf{w}$ e b sono i parametri del modello. Guardando la Figura 3.10, se si etichettano i quadrati con la classe $+1$ e i cerchi con -1 allora si può predire la classe y per qualsiasi esempio di test z nel seguente modo

$$y = \begin{cases} +1 & \text{se } \mathbf{w} \cdot \mathbf{z} + b > 0 \\ -1 & \text{se } \mathbf{w} \cdot \mathbf{z} + b < 0 \end{cases}$$

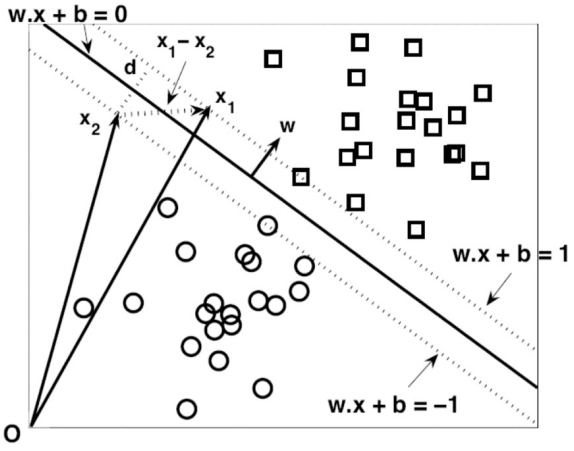

Figura 3.10: Margine per un decision boundary [16]

È possibile riscalare i parametri $\mathbf{w}$ e b dell'iperpiano in modo tale che i due iperpiani b_{i1} e b_{i2} possano essere così definiti:

$$b_{i1} : \mathbf{w} \cdot \mathbf{x} + b = 1$$

$$b_{i2} : \mathbf{w} \cdot \mathbf{x} + b = -1$$

Il *margine* del decision boundary è dato dalla distanza di questi due iperpiani. Sia x_1 un punto che si trova su b_{i1} e x_2 uno che sta su b_{i2} come mostrato in Figura 3.10, allora sostituendo i punti nelle due equazioni sopra il margine può essere calcolato sottraendo la prima con la seconda:

$$w \cdot (x_1 - x_2) = 2 \quad \|w\| \times d = 2$$

$$d = \frac{2}{\|w\|}$$

Massimizzare il margine corrisponde ad individuare l'iperpiano ottimo, e occorre minimizzare $\|w\|$. Questo è l'obiettivo di SVM.

3.7.2 Apprendimento di un modello SVM

La fase di learning di una SVM implica la stima dei parametri $\mathbf{w}$ e b dell'iperpiano a partire dai vettori del training set. I parametri devono essere scelti in modo tale che siano garantite due condizioni

$$\mathbf{w} \cdot \mathbf{x}_i + b \geq 1 \text{ se } y_i = 1$$

$$\mathbf{w} \cdot \mathbf{x}_i + b \leq -1 \text{ se } y_i = -1$$

Esse impongono che tutte le istanze della classe $y = 1$ siano collocate al di sopra o sull'iperpiano mentre quelle della classe -1 siano collocate al di sotto o sull'iperpiano stesso. Si possono sommare le due condizioni in una unica

$$y_i(\mathbf{w} \cdot \mathbf{x}_i + b) \geq 1 \quad i = 1, 2, 3, ..., N$$

Oltre a questo bisogna imporre che il margine dell'iperpiano sia massimale, cioè è equivalente a minimizzare la seguente funzione obiettivo:

$$f(\mathbf{w}) \frac{\|\mathbf{w}\|^2}{2}$$

La fase di learning in SVM è definita dal seguente problema di ottimizzazione vincolata:

$$\min_w \frac{\|w\|^2}{2}$$

soggetto a $y_i(\mathbf{w} \cdot \mathbf{x}_i + b) \geq 1 \;\; i = 1, 2, 3, ..., N$

La risoluzione del seguente problema di ottimizzazione convessa avviene con il metodo dei *moltiplicatori di Lagrange*, in questo modo si possono determinare i parametri per l'equazione dell'iperpiano e la classificazione di un qualsiasi record z avviene sulla base di:

$$f(z) = \text{sign}(\mathbf{w} \cdot \mathbf{z} + b = \text{sign}\left(\sum_{i=1}^{N} \lambda_i y_i \mathbf{x}_i \cdot \mathbf{z} + b\right)$$

Se $f(z) = 1$ allora il record di test è classificato come positivo, altrimenti come negativo.

3.7.3 Linear SVM: Caso non separabile

Non tutti i dataset sono linearmente separabili; in questo caso l'idea è quella di apprendere un iperpiano che sia tollerabile ad un numero limitato di errori rispetto al training set, si tratta di un metodo noto come approccio *soft margin*.
I vincoli presentati precedentemente vengono rilassati introducendo variabili slack[1] ξ_i positive:

$$\mathbf{w} \cdot \mathbf{x}_i + b \geq 1 - \xi_i \text{ se } y_i = 1$$

$$\mathbf{w} \cdot \mathbf{x}_i + b \leq -1 + \xi_i \text{ se } y_i = -1$$

dove $\forall i : \xi_i > 0$. In linea di principio si può usare la funzione obiettivo del caso seperabile ed imporre le stesse condizioni, ma dato che non vi sono vincoli sul numero di errori che possono essere commessi si può trovare un iperpiano che ha un margine elevato ma commette molti errori. Per evitare questo si modifica la funzione obiettivo in modo da penalizzare gli iperpiani con un elevato valore per la variabile di slack:

$$f(\mathbf{w}) \frac{\|\mathbf{w}\|^2}{2} + C\left(\sum_{i=2}^{N} \xi_i\right)^k$$

dove C e k sono parametri specificati dall'utente e rappresentano la penalità nel commettere errori di classificazione. C può essere scelto sulla base delle performance del modello valutate su un validation set, controlla il tradeoff tra errori di training e massimizzazione del margine ($C = \infty$ porta ad una situazione di *hard margin*, abbassando il valore aumentano gli errori sul training set).

[1] In un problema di ottimizzazione, una variabile slack è una variabile che viene aggiunta in un vincolo disuguaglianza per trasformarlo in una parità. In questo modo viene sostituito un vincolo di disuguaglianza con un vincolo di uguaglianza.

3.7.4 Non Linear SVM

Qualora si presenti una situazione come quella di Figura 3.11 è necessario applicare una nuova tecnica, che prevede di proiettare i punti in uno spazio a dimensione maggiore dove i punti possano essere separati più facilmente (come nei caso lineare). I dati dal loro spazio di coordinate originali $\mathbf{x}$ vengono rimappti in un nuovo spazio $\Phi(\mathbf{x}_i)$ tale che: un iperpiano possa essere usato per separare le istanze di classi diverse. Una volta fatto questo si può applicare la stessa tecnica presentata nella sezione precedente. Per definire il nuovo problema si utilizza la

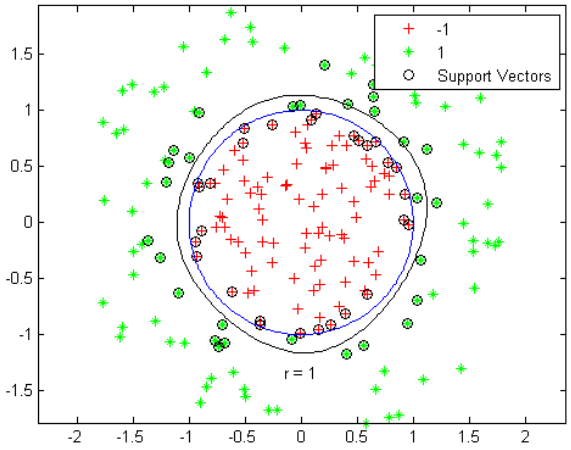

Figura 3.11: Istanze non separabili in $\mathbb{R}^2$ ed iperpiano a massimo margine appreso grazie al kernel

funzione di mapping Φ che porta al nuovo spazio desiderato:

$$\min_{w} \frac{\|w\|^2}{2}$$

soggetto a $y_i(\mathbf{w} \cdot \Phi(\mathbf{x}_i) + b) \geq 1 \;\; i = 1, 2, 3, ..., N$
In maniera analoga al caso lineare un'istanza di test z viene classificata usando:

$$f(z) = \text{sign}(\mathbf{w} \cdot \Phi(\mathbf{z}) + b = \text{sign}\left(\sum_{i=1}^{N} \lambda_i y_i \Phi(\mathbf{x}_i) \cdot \Phi(z) + b\right)$$

Tutte le equazioni utilizzano il prodotto scalare tra una coppia di vettori all'interno dello spazio rimappato: $\Phi(x_i) \cdot \Phi(x_j)$. Quando si hanno molte dimensioni, può

risultare poco maneggevole, per risolvere ciò si fa ricorso a una funzione chiamata *Kernel*. Se esiste una funzione K tale che: $K(x_j, x_i) = \Phi(x_i) \cdot \Phi(x_j)$ allora l'algoritmo di training può utilizzare solo K senza dover conoscere la natura di Φ.
La funzione utilizzata per classificare un nuovo record z diviene allora

$$f(z) = \text{sign}\left(\sum_{i=1}^{N} \lambda_i y_i K(\mathbf{x}_i, z) + b\right)$$

Di seguito si riportano alcune funzioni che rispettano il teorema citato e sono tra le più utilizzate:

- Lineare: $K(x, z) = x$
- Polinomiale: $K(x, y) = (x \cdot y + 1)^p$
- Gaussiana (Radial Basis Function): $K(x, y) = e^{-\|x-z\|^2\gamma}$
- Tangente Iperbolica: $K(x, y) = \tanh(kx \cdot y - \delta)$

Le SVM sono attualmente fra i migliori classificatori in una varietà di problemi (es. elaborazione del linguaggio e genomica). Il buon funzionamento dipende dalla selezione del Kernel, dalla scelta dei suoi parametri e dal parametro di soft margin C. Una comune adozione è quella di utilizzare il *Kernel Gaussiano* che ha un solo parametro γ. La selezione di uno specifico kernel e i parametri viene eseguita in modo empirico: tenta e verifica (trial and test). Nella versione più flessibile delle SVM (con la formulazione per dati non separabili e i kernel) occorre fare diverse scelte: il tipo di kernel, il parametro (o i parametri) del kernel, il parametro C (costo delle mis-classificazioni). Queste scelte portano a superfici di separazione diverse.

I vantaggi di questo tipo di classificatori sono principalmente dati dalla teoria matematica con cui sono costruiti; si rimanda al testo di Cristianini [50] per una trattazione completa. Il fatto che il problema di minimo delle SVM, enunciato in precedenza, abbia un unico ottimo globale ha preferito questo tipo di algoritmi di apprendimento rispetto ad altri piu' tradizionali (reti neurali).

Un altro vantaggio è la loro natura di macchine a kernel, la quale permette di sfruttare questo algoritmo di apprendimento automatico per varie tipologie di dati. Inoltre una SVM può lavorare in uno spazio a dimensionalità enorme (potenzialmente infinita) utilizzando sempre gli stessi algoritmi, infatti l'iperpiano

nello spazio a dimensionalità grande diventa una superficie complessa nello spazio originale.

Il mapping $\Phi(x)$ non viene mai calcolato esplicitamente, quindi può essere complicato quanto si vuole; non ci sono problemi di curse of dimensionality, perchè l'algoritmo è indipendente dalla dimensione dello spazio finale.

3.8 Alberi Decisionali

Un Albero di Decisione (*Decision Tree*) è un metodo molto semplice ed efficace per realizzare un classificatore. L'addestramento degli alberi di decisione è una delle tecniche attuali di maggior successo. La struttura, come suggerisce il nome, ricorda quella di veri e propri alberi: in analogia con questa somiglianza si sviluppa la nomenclatura prendendo spunto dalla teoria dei *grafi*.

Un albero di decisione è un albero di classificatori (Decision Stump) dove ogni nodo interno è associato ad una particolare "domanda" su una caratteristica (*feature*) o attributo. Da questo nodo dipartono tanti *archi* quanti sono i possibili valori che la caratteristica può assumere, fino a raggiungere le *foglie* che indicano la categoria associata alla decisione.

La classificazione avviene partendo dal nodo radice. In ogni nodo una sola caratteristica dell'oggetto viene valutata. Una valutazione tipica è il confronto con un certo valore x_s cosiddetto di soglia, per esempio:

$$x_i \leq x_s \quad x_s \in \mathbb{R}$$

Sono possibili due risposte in base al valore di verità della disuguaglianza: vero o falso. Questo albero viene detto *binario* dato che ammette al massimo due risposte.

Il processo si ferma nel momento in cui il ramo percorso conduce ad una foglia dell'albero: in tal caso il classificatore attribuisce all'oggetto l'etichetta contenuta nella foglia in cui è arrivato.
Nel caso in cui la classificazione interessi variabili discrete, l'albero è detto *decisionale*, nel caso di variabili continue è detto *regressione*.
Un albero decisionale può anche essere pensato come una tecnica per dividere un problema complesso in un insieme di problemi semplici.

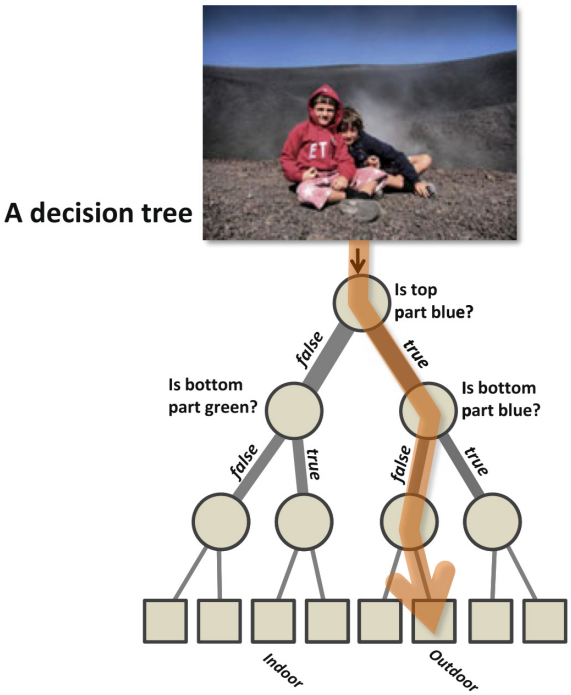

Figura 3.12: Un esempio di utilizzo di albero decisionale [18]

Chiamiamo *data point*, un generico oggetto, definito da un vettore $v = (x_1, x_2, ..., x_d) \in \mathbb{R}^d$, le quali componenti x_i rappresentano gli attributi del *data point*, chiamate *features*. Queste feature variano da applicazione ad applicazione,

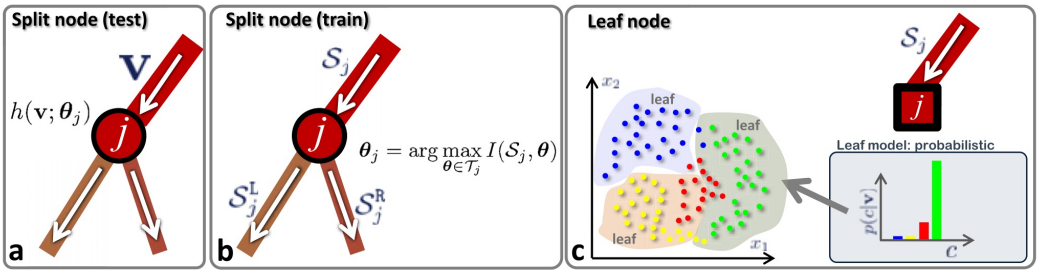

Figura 3.13: Rappresentazione di un nodo di split (fase di traing) e un nodo foglia [18]

per esempio in computer vision *v* potrebbero rappresentare i pixel di un immagine. Il numero delle features dipende naturalmente dal tipo di *data point* e dal tipo di applicazione. In teoria, la dimensione d dello spazio delle features può

essere estremamente largo, anche infinito. In pratica, questo non è necessario e vantaggioso, perciò viene solitamente una piccola porzione di d.

Le features di interesse che vengono effettivamente utilizzate sono un sottoinsieme di tutte le possibili features, è viene definito nel seguente modo: $\boldsymbol{\varphi}\ (v) = (x_{\varphi 1}, x_{\varphi 2}, ..., x_{\varphi d'}) \in \mathbb{R}^{d'}$ dove d' rappresenta la dimensione del sottospazio e $\varphi_i \in [1, d']$ rappresenta la dimensione selezionate. Come detto precedentemente, ogni nodo ha una specifica funzione test (chiamata anche funzione *split*), e la funzione del nodo j con un output binario, viene formalmente definita nel seguente modo: $h(\mathbf{v}, \boldsymbol{\theta}_j) : \mathbb{R}^d \times \tau \rightarrow \{0, 1\}$, dove 0 e 1 possono essere interpretati rispettivamente come "falso" e "vero", $\boldsymbol{\theta}_j \in \tau$ rappresenta il parametro di split associato al nodo j-esimo e τ rappresenta lo spazio di tutti i parametri di split. Il *data point* $\mathbf{v}$, arrivando alla funzione di split viene mandato a destra o sinistra, in base al risultato della funzione di test (domanda).

Una buona *funzione split* divide i campioni di classi eterogenee in dei sottoinsiemi con etichette abbastanza omogenee, stratificando i dati in modo da mettere poca varianza in ogni strato. Per permettere questo è necessario definire una metrica, che misuri questo indice di impurità: solitamente negli alberi decisionali viene utilizzata l'*entropia* che permette di misurare la quantità di incertezza.

3.8.1 Entropia ed Information gain

L'*entropia*, definita da Shannon è una misura dell'ordine dello spazio dei records che si considerano per la costruzione degli alberi di decisione. Un valore elevato di entropia esprime il "disordine", ovvero una maggiore difficoltà nell'assegnare ciascun record alla propria classe sulla base degli attributi che caratterizzano la classe: più l'entropia è alta, meno informazione abbiamo sull'attributo classe. Ad esempio, nel caso delle sole classi "+" e "-" con un valore di entropia pari a 1, corrispondente a $P_+ = 50\%$ e $P_- = 50\%$, sarebbe il caso più difficile da considerare, se l'obiettivo è quello di individuare gli attributi che caratterizzano le due classi.

Formalmente definiamo X come un sottoinsieme di campioni di un particolare insieme di addestramento formato da c possibili classi. X è di fatto una variabile aleatoria, che assume solo valori discreti. È possibile associare ad ogni valore discreto x_i, che può assumere X, la distribuzione di probabilità $p(x_i) = p_i$. X è un

data set formato da c classi e p_i è la frequenza relativa della classe i all'interno dell'insieme X.
In generale, l'*entropia* si definisce come:

$$H(S) = -\sum_{c \in C} p(c) \log(p(c))$$

dove S è l'insieme dei training points e c indicata la classe delle etichette.

L'*information gain*, è definito come la diminuzione di entropia che si ottiene partizionando i dati rispetto ad un certo attributo. Se indichiamo con $H(S)$ il valore iniziale di entropia e con $H(S, A)$ il valore dell'entropia dopo aver partizionato i records con l'attributo A, l'*information gain*, che indicheremo con G, è data da:

$$G = H(S) - H(S, A)$$

Tale quantità è tanto maggiore quanto piú elevata è la diminuzione di entropia dopo aver partizionato i dati con l'attributo A. Dunque un criterio di scelta dei nodi di un eventuale albero di classificazione, consiste nello scegliere di volta in volta l'attributo A che dà una maggiore diminuzione di *entropia* o che analogamente massimizza l'*information gain*.

L'*information gain* ha valori molto elevati in corrispondenza di attributi che sono fortemente informativi e che quindi aiutano ad identificare con buona probabilità la classe di appartenenza dei records. Spesso, però, più gli attributi sono informativi, più perdono di generalità; ad esempio, nel database di una compagnia telefonica, il campo codice fiscale è altamente informativo, ha dunque un alto valore di information gain, dal momento che identifica con certezza l'utente, ma non è per nulla generalizzabile. L'ideale è dunque individuare campi altamente informativi con un buon grado di generalizzazione.

3.9 Metodi di Ensemble Learning

Il concetto di addestramento *Ensemble* richiama l'utilizzo di diversi classificatori, differenti, uniti in un certo modo per riuscire a massimizzare le prestazioni usando i punti di forza di ognuno e limitando le debolezze dei singoli. Alla base del concetto di *Ensemble Learning* ci sono i classificatori deboli (*weak classifier*), cioè poco correlati con la vera classificazione, ma riescono a classificare almeno

il 50% + 1 dei campioni di un problema binario. Al contrario, un classificatore forte (*strong classifier*) è ben correlato con la vera classificazione.
Sommati in un certo modo tra di loro, i classificatori deboli permettono di costruire un classificatore forte, risolvendo allo stesso tempo problemi tipici dei classificatori tradizionali (*overfitting* in primis). Potremmo ad esempio generare cento alberi di decisone differenti partendo dallo stesso insieme di addestramento e farli votare sulla classificazione migliore di un nuovo evento [19].

Lo scopo quindi è di allenare m classificatori con m differenti training sets e combinare i risultati, è illustrato in Figura 3.14.

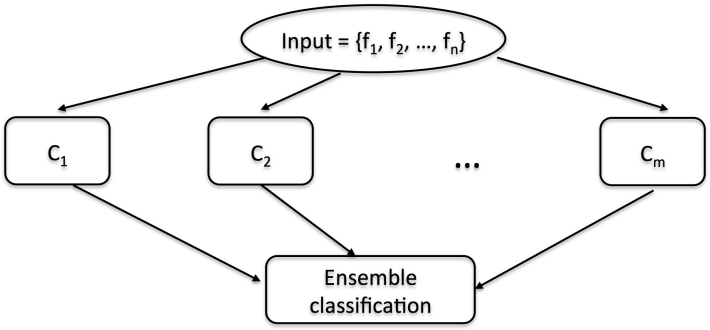

Figura 3.14: Rappresentazione del funzionamento dell'Ensemble Learning [19]

I vantaggi che otteniamo da questo metodo, come si può intuitivamente immaginare, sono molteplici: mediando i risultati di ciascun classificatore si riduce il rischio di utlizzare un singolo classificatore non discriminante; spesso hanno prestazioni migliori rispetto ai singoli classificatori; più resistente al rumore. Mentre l'unico aspetto svantaggioso di questo metodo è che, a volte, richiede più tempo nella fase di training.

Un metodo classico per combinare differenti classificatori, con lo scopo di classificare un nuovo record t, è utilizzare un sistema di voto che conti quanti classificatori base assegnerebbero t ad ognuna delle possibili classi. Naturalmente t viene assegnato alla classe che ottiene più voti.

Esistono diversi algoritmi di classificazione ensemble, la quale differiscono tra di loro in base a come: vengono costruiti training sets, vengono manipolate le features, viene calcolato il risultato finale (media, maggioranza, etc).
Normalmente i classificatori vengono *combinati* insieme attraverso una media

pesata. Supponiamo che le classi siano $\{-1, +1\}$ possiamo formalmente definire:

$$\text{ensemble}(x) = \text{sign}\left(\sum_{i=1}^{N} w_i C_i(x)\right)$$

dove w_i è il peso del classificatore C_i.
Si può estendere a un numero arbitrario classi.
Nel metodo a **Voting semplice** $w_k = w_j$ (per ogni k, j), mentre nel metodo a **Voting pesato** i pesi influenzano la scelta finale.

Di fatto le tecniche di *Ensemble Learning* non forniscono classificatori *general purpose*, ma indicano solo il modo ottimo per unire più classificatori tra loro. Esempi di questa tecnica sono:

- **Decision Tree gli Alberi di Decisione**, essendo costruiti da tanti alberi di decisione in cascata;
- **Bagging** (1996) il BootStrap AGGregatING prova a ridurre i problemi di *overfitting* addestrando diversi classificatori su sottoparti del *training set* ed eseguendo infine una votazione per maggioranza;
- **Boosting** (1990) Invece che prendere sottoparti del *training set* puramente casuali vengono, in parte, usati i campioni che rimangono non classificati correttamente;
- **AdaBoost** l'ADAptive BOOSTing (sezione 3.9.2) è l'algoritmo di *Ensemble Learning* più conosciuto;
- **Random Forest** (2001) è un *BootStrap Aggregating* (bagging) su *Decison Tree*. Composto da diversi alberi di decisione, ognuno creato su un sottoinsieme dei dati di addestramento e delle caratteristiche da analizzare, che votano per maggioranza;

3.9.1 Bagging

Bagging è l'acronimo delle parole inglesi Bootstrap AGGregatING: esso è un metodo per la generazione di versioni multiple di un classificatore al fine di combinarle in un insieme di classificatori. L'idea è quella di addestrare ogni albero all'interno di una foresta su un differente sottoinsieme del training set, campionando casualmente sullo stesso database etichettato [18]. Piú formalmente, si

provvede a campionare ripetutamente il training set di input, con distribuzione di probabilità uniforme e con reinserimento, per costruire ad ogni iterazione un modello sul campione appena prodotto.

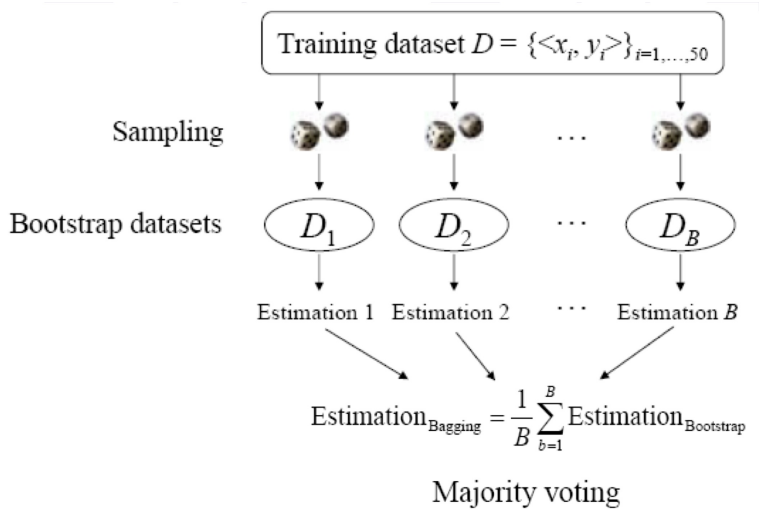

Figura 3.15: Rappresentazione del funzionamento del metodo Bagging [20]

Un campione di *bootstrap* è un campione casuale i cui elementi vengono estratti con reinserimento, per questo la tecnica prende anche nome di *bootstrap aggregating*.

Tutti gli insiemi costruiti ad ogni iterazione sono di taglia uguale a quella del training set di partenza. Visto che si campiona con reinserimento, è quindi possibile che in uno stresso insieme siano presenti elementi ripetuti e che elementi del training set T vengano omessi. Condizione fondamentale affinché tale metodo produca copie dello stesso classificatore di base sufficientemente *diverse* è che la tipologia di classificatore impiegata sia instabile: piccole variazioni dell'insieme d'addestramento devono produrre significative variazioni nella struttura del classificatore. Questa tecnica consente di evitare di specializzare i parametri selezionati ad un unico singolo training set ed ha mostrato di aumentare la generalizzazione. Un altro vantaggio è che la fase di training è molto più veloce, rispetto ad usare l'intero insieme etichettato. Questa è la procedura riassunta:

- eseguo k bootstrap sample dal dataset
- da ognuno creo un classificatore
- per classificare un oggetto mai visto prima
  - raccolgo le previsioni di ciascun classificatore
  - prendo per buona quella più popolare

3.9.2 Boosting

L'idea base delle tecniche *Boosting* è quella di costruire una lista di classificatori (solitamente alberi decisionali con una sola funzione di split) assegnando, in maniera iterativa, un peso ad ogni nuovo classificatore considerando la sua capacità di riconoscere campioni non correttamente identificati dagli altri classificatori già coinvolti nell'addestramento. Ad ogni fase dell'algoritmo, si addestra un nuovo classificatore usando il data set, nel quale i coefficienti pesati sono aggiustati in base alle performance del classificatore addestrato precedentemente, in modo tala da assegnare un peso maggiore ai data points classificati erroneamente. L'algoritmo si concentra sui campioni più "difficili" che vengono quindi pesati maggiormente. Il classificatore finale è ottenuto con una votazione pesata dei modelli costruiti.

Le tecniche di *Boosting* permettono di generare un classificatore nella forma di modello additivo:

$$F_T(x) = f_1(x) + f_2(x) + ... + f_T(x) = \sum_{t=1}^{T} f_t(x)$$

con $f_1, ..., f_T$ singoli classificatori.

Ad ogni esempio è associato un *peso* $w_j \geq 0$, maggiore il peso, più alta è l'importanza dell'esempio durante l'apprendimento di un'ipotesi.

Boosting assegna inizialmente $w_j = 1$ a tutti gli esempi (ovvero, parte da un normale insieme di addestramento). Partendo da tale insieme si genera la prima ipotesi, h_1, che classificherà alcuni degli esempi di training in modo corretto e altri no. Aumentando il peso negli esempi classificati erroneamente, diminuendo nel contempo quello degli esempi classificati correttamente: da questo insieme di addestramento pesato si genera l'ipotesi h_2. Il processo continua finchè non sono state generate M ipotesi, dove M è un parametro di input dell'algoritmo di boosting. L'ipotesi di ensemble finale è una combinazione a maggioranza di tutte le M ipotesi, ognuna pesata in base alla sua prestazione sull'insieme di addestramento.

In modo simile agli altri algoritmi di classificazione, il boosting minimizza in modo approssimato il *training error* su un training set fornito in ingresso. A differenza però di molti altri algoritmi, il boosting procede aggiungendo incrementalmente classificatori base fino a soddisfare un dato criterio di stop (per

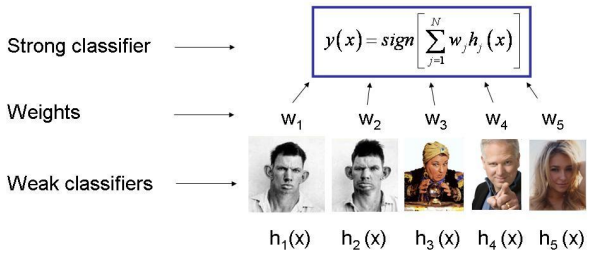

Figura 3.16: Approccio utilizzato da Boost, nell'unire classificatori deboli [21]

esempio, quando il training error è sceso sotto un certo livello oppure quando il numero di classificatori base utilizzati supera un valore predefinito.

Esistono differenti tecniche di Boosting in base a come:

- sono aggiornati i pesi dei record del training set;
- sono combinate le predizioni dei singoli classificatori.

Nell'algoritmo Bagging la costruzione delle basi complementari di apprendimento è lasciata al caso e alla instabilità del metodo. Mentre nel metodo Boosting attivamente si cerca di generare basi complementari di apprendimento, allenando la successiva base di apprendimento, basandosi sugli errori fatti nella fase di apprendimento precedente.

AdaBoost (ADAptive BOOSTing)

Una delle tecniche di Boosting maggiormente utilizzate è sicuramente *AdaBoost*. Partendo dal modello adattivo visto precedentemente, l'obiettivo di AdaBoost è quello di individuare un classificatore $f(x)$ che minimizzi di volta in volta la quantità

$$f_{T+1} = \arg\min_{f} \sum_{i=1}^{m} e^{-y_t(F_T(x_i)+f(x_i))} = \arg\min_{f} \sum_{i=1}^{m} w_i e^{-y_t f(x_i)}$$

Si supponga di avere a disposizione $\mathcal{H} = \{h_1, ..., h_T\}$ classificatori binari, ognuno dei quali, valutando il campione x_i, con $1 \leq i \leq m$, restituisca una opinione $y_i = \{-1, +1\}$.

Sia la funzione $F_T(\mathbf{x};\boldsymbol{\sigma})$, definita come

$$F_T(\mathbf{x};\sigma_1,...,\sigma_T) = \sum_{t=1}^{T} \alpha_t h_t(\mathbf{x})$$

una funzione il cui segno rappresenta l'ipotesi di classificazione e la sua magnitudine riflette la bontà della predizione. Questo modello è chiamato *Extended Additive Model* o *Adaptive Basis-Function Model*

L'obiettivo è ottenere un classificatore forte $H(x_i)$ come somma lineare pesata dei classificatori h_t, il cui segno determini l'ipotesi globale:

$$H(x_i) = \text{sign}\left(\sum_{t=1}^{T} \alpha_t h_t(x_i)\right) = \text{sign} F_T(x_i;\boldsymbol{\sigma})$$

Questa è una votazione per maggioranza: viene scelto come vincitrice l'ipotesi votata da più classificatori, ognuno con peso differente σ_t. Sono proprio le costanti σ_t, i pesi assegnati a ogni classificatore, il risultato fornito da questa tecnica di addestramento.

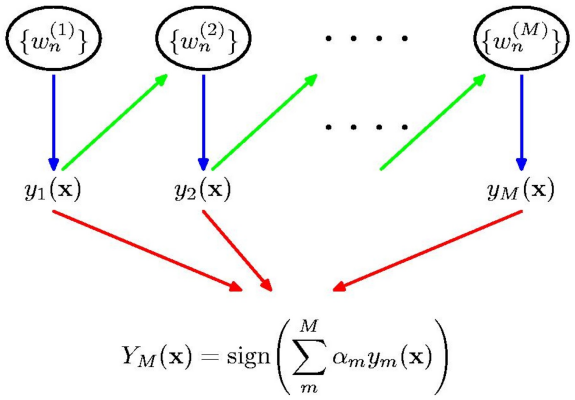

Figura 3.17: Rappresentazione del funzionamento di AdaBooast [20]

Per permettere di assegnare un voto al classificatore, è necessario che a ogni campione in ingresso x_i sia assegnato un certo peso w_i: più il peso è alto più il campione è stato classificato in maniera non corretta fino a questo punto dell'addestramento, mentre più il peso è basso più è stato classificato correttamente. Alla prima iterazione, tutti i pesi sono posti uguali, pari a $w^{(0)} = 1/m$, in modo da avere una esatta distribuzione statistica.

Sia $u_i = y_i h_t(x_i)$ la funzione che esprime il successo $(+1)$ o il fallimento (-1) del classificatore h_t a valutare il campione x_i. Dati i pesi associati a ogni campione, è possibile per ogni classificatore calcolare W_{-1}, la somma dei pesi associati gli insuccessi, e W_{+1}, la somma dei pesi associati alle classificazioni corrette, ovvero attraverso la definizione di u_i, in forma compatta

$$W_b = \sum_{u_i=b} w_i$$

con con $b = +1$, successo, e $b = -1$, insuccesso.

Sia ε_t la misura dell'errore del classificatore h_t calcolata come

$$\varepsilon_t = \sum_{y_i \neq h_t(i)} w_i^{(t)} = \sum_{u_i=-1} w_i^{(t)} = W_-$$

somma dei pesi associati ai soli campioni classificati in maniera errata, e sia

$$r_t = W_+ - W_- = \sum_{i=1}^{m} w_i^{(t)} u_i$$

la media ponderata, usando i pesi w_i, delle performance u_i di classificazione[51].

Le iterazioni dell'algoritmo di *AdaBoost* sono le seguenti:

1. fornito in ingresso un classificatore h_t (la scelta è di fatto lasciata all'utente, cercando di selezionare il classificatore che minimizza l'errore ε_t, ma non è obbligatorio che debba essere per forza il migliore);
2. viene calcolato l'errore ε_t prodotto del classificatore h_t sui campioni in ingresso. Quando non si riesce a trovare un classificatore per il quale $\varepsilon_t > 1/2$, l'addestramento non può proseguire e deve venire pertanto terminato;
3. dato l'errore, al classificatore h_t viene assegnato un peso α_t, calcolato come descritto in seguito;
4. ad ogni campione x_i la distribuzione associata $w_{(t+1)}$ viene aggiornata attraverso la funzione i

$$w_i^{(t+1)} = \frac{1}{Z_t} w_i^{(t)} e^{-\alpha_t u_i} = \frac{1}{Z_t} w_i^{(t)} e^{-y_i f_t(x_i)}$$

Il peso associato ai campioni che hanno avuto successo nella classificazione viene diminuito di una quantità proporzionale a e^{α_t}, mentre ai campioni

che sono stati classificati in maniera errata il peso è aumentato di e^{α_t}. Z_t è un fattore di normalizzazione scelto in modo tale che $\sum w_i^{(t)} = 1$ ma assume anche un significato importante.

Il parametro di normalizzazione Z_t vale

$$Z_t = \sum_{i=1}^{m} w_i^{(t)} e^{-\alpha_t u_i} = e^{-\alpha_t} W_+ + e^{\alpha_t} W_-$$

Questo algoritmo è quello che viene definito in letteratura *AdaBoost.M1* o *Discrete AdaBoost*. Le ipotesi $h_t(x)$ usate da AdaBoost sono *feature* che possono assumere i soli valori $\{+1, -1\}$.

Il funzionamento intuitivo di *AdaBoost* è molto semplice: *AdaBoost* per ogni nuovo classificatore aggiunto alla serie si concentra sui pattern in ingresso che finora sono stati classificati peggio.

AdaBoost, come SVM, ottiene come risultato quello di massimizzare il margine di separazione tra le classi, anche se con metriche differenti. In questo modo, entrambi, riescono ad essere meno sensibile a problemi come l'*overfitting*[51].

AdaBoost ha molti vantaggi: è veloce, semplice e facile da programmare. Non ha parametri da settare (eccetto per il numero di round T). Non richiede una conoscenza pregressa su weak learner e quindi può essere flessibilmente combinato con qualsiasi metodo per trovare ipotesi deboli (*weak*)[52].

Esistono diverse versioni dell'algoritmo AdaBoost:

- Discrete Adaboost;
- Real AdaBoost
- Gentle AdaBoost
- Logit Boost
- Gradient Boosted trees

3.10 Random Trees e Decisione Forest

Random Forests (foreste casuali) è un *Ensemble learning method*: composto da molti alberi decisionali che dà in uscita la classe corrispondente all'uscita delle classi degli alberi presi individualmente. L'algoritmo fu sviluppato da Leo Breiman e Adele Cutler ed è un marchio registrato [53].

Il metodo Random Forest è una variante del bagging ed impiega esclusivamente alberi decisionali come classificatori di base ed è previsto l'utilizzo di vettori casuali per generare alberi. La casualità è un fattore che entra quindi a far parte della costruzione dei classificatori con lo scopo di accrescere la loro diversità, non soltanto nella scelta degli insiemi di addestramento, ma anche nella scelta delle caratteristiche per ogni nodo o nell'alterazione della struttura di alberi già costruiti.
Per esempio, un albero che effettua riconoscimento di oggetti potrebbe avere una lunga lista di potenziali caratteristiche (*features*): colore, texture, pendenza, direzione del gradiente, varianza, magnitudo, e molti altri. Ad ogni nodo dell'albero è consentito di scegliere un sottoinsieme casuale di queste caratteristiche per determinare il modo migliore per dividere i dati, e ogni successivo nodo dell'albero ottiene un nuovo sottoinsieme di caratteristiche, scelto ancora casualmente, su cui dividere i dati.

3.10.1 Randomized Node Optimization (RNO)

Per problemi con dimensioni elevate, la grandezza di $\mathcal{T}$ (spazio di tutti i parametri della funzione split) può essere estremamente grande. Una soluzione è che quando addestriamo il $j-$esimo nodo, utilizziamo solo una piccola parte $\mathcal{T}_j$, scelta in modo random, dei valori dei parametri. Considerando questo modello casuale, addestrare un albero viene fatto ottimizzando ogni *split node* j come:

$$\boldsymbol{\theta}_j = \arg\max_{\boldsymbol{\theta}\in\mathcal{T}_j} I(S_j, \boldsymbol{\theta}).$$

In alcuni casi, potremmo avere $|\mathcal{T}| = \infty$, infatti è conveniente inserire un parametro $\rho = |\mathcal{T}_j|$ che serve a controllare il grado di casualità (randomness) nell'albero $\rho \in \{1, ..., \mathcal{T}_j\}$; solitamente è un valore fisso per tutti i nodi.

Al limite $\rho = |\mathcal{T}|$ tutti i nodi di split usano tutte le informazioni disponibili, la conseguenze è che avremo un sistema senza casualità. Viceversa, quando $\rho = 1$ ogni nodo di split prende in input solamente un piccolo insieme causale per il suo parametro θ; in questo modo non abbiamo una reale ottimizzazione, ma in abbiamo un elevato valore di casualità.

Nel'applicazione pratica dell'algoritmo, si può scegliere liberamente se rendere casuale, nessuno, uno o tutti i parametri $\boldsymbol{\varphi}$, $\boldsymbol{\psi}$ e $\boldsymbol{\tau}$. Per esempio uno potrebbe

voler rendere casuale il parametro $\boldsymbol{\varphi}$ che seleziona i parametri presi dal training set, e il parameto $\boldsymbol{\psi}$ che definisce l'orientamento dell'iperpiano di apprendimento debole, ma ricercare in un predefinito insieme di valori di thresholds $\boldsymbol{\tau}$. Alcune varianti di training, nel quale vengono effettuate determinate scelte sui parametri menzionati precedentemente, sono stati definiti: "totally randomized trees" e "extremely randomized trees"[18].

3.10.2 Combinare alberi in Forest Ensemble

Un Random Decision Forest è costituita dall'insieme (ensemble) di alberi decisionali, addestrati in modo casuale. L'aspetto chiave del modello a foresta è il fatto che gli alberi la compongono sono tutti completamente differenti (in modo casuale) gli uni dagli altri. In questo modo otteniamo non correlazione tra gli alberi individuali e come risultato si incrementa generalizzazione e robustezza. Il

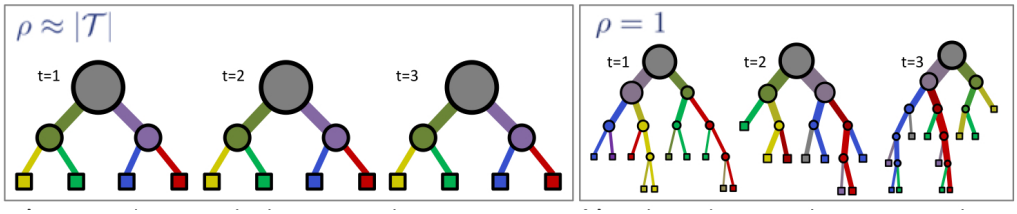

Figura 3.18: Rappresentazione percentuale di casualità e di correlazione [18]

modello a foresta è caratterizzato dalle stesse componenti degli alberi decisionali. L'unico sostanziale cambiamento riguarda il parametro ρ il quale controlla sia la percentuale di casualità di ogni albero, ma anche la percentuale di correlazione tra i differenti alberi della foresta. Infatti, come illustrato in Figura 3.18 quando $\rho = |\mathcal{T}|$ tutti gli alberi saranno identici tra loro. Come ρ diminuisce tutti gli alberi diventano meno correlati (differenti uno dall'altro). Un valore elevato di ρ corrisponde a bassa causalità e grande correlazione tra alberi. La foresta si comporta come se fosse composta da un singolo albero. (**b**) Valore piccola di ρ corrisponde a grande casualità nel processo di addestramento, in questo modo gli alberi che compongono la foresta sono molto differenti uno dall'altro.

In una foresta con T alberi utilizziamo le variabili $t \in 1, ..., T$ per indicizzare ogni albero che compone la foresta. Ogni albero è addestrato indipendentemente (e possibilmente in parallelo) e i valori di ciascun albero dipendono dai valori di un vettore casuale campionato indipendentemente. Durante la fase di test,

ogni test point **v** è simultaneamente inviato su ogni albero (partendo dalla radice) finchè raggiungono le corrispondenti foglie. L'errore generalizzazione dipende dalla robustezza dei singoli alberi e dalla correlazione tra loro.

Anche la fase di test può essere effettuata in parallelo, in modo da poter raggiungere un elevata efficienza computazione nelle moderne CPU o GPU. Per combinare tutte le previsioni degli alberi in una singola foresta, possiamo facilmente usare una operazione di media. Per esempio in classificazione abbiamo

$$p(c|\mathbf{v}) = \frac{1}{T}\sum_{t=1} T p_t(c|\mathbf{v})$$

dove $p_t(c|\mathbf{v})$ rappresenta la distribuzione a posteriori ottenuta sul t-esimo albero. I parametri rappresentativi che maggiormente influenzano il comportamente del modello a foreste decisionali sono:

- il valore massimo di profondità dell'albero D;
- la percentuale di casualità (controllata dal parametro ρ);
- la dimensione della foresta T;
- il modello di apprendimento debole (weak) scelto;
- la funziona obiettivo di apprendimento;
- la scelta sulle features in applicazioni pratiche.

Tali scelte influenzano direttamente la precisione sulla foresta predetta, la qualità della sua affidabilità, la sua generalizzazione e la sua efficienza computazionale.

3.10.3 Il modello Decision Forest per la Classificazione

Nella classificazione ogni valore dell'insieme di training è rappresentato come una coppia (**v**,c). In Figura 3.19 i *data points* sono rappresentati da cerchi, con differenti colori che indicano differenti etichette di training. I test points sono indicati in grigio (infatti l'appartenenza a una particolare classe è inizialmente sconosciuta).

Piú formalmente, durante la fase di test, avendo un dato **v** vogliamo dedurre la classe di appartenenza c tale che $c \in \mathcal{C}$, con $\mathcal{C} = \{c_k\}_{k=1}^{|\mathcal{C}|}$. In generale vogliamo calcolare l'intera distribuzione $p(c|\mathbf{v})$.
Come visto nel caso degli alberi decisionali, l'input è rappresentato da un vettore multidimensionale di features $\mathbf{v} = (x_1, ..., x_d) \in \mathbb{R}^d$.

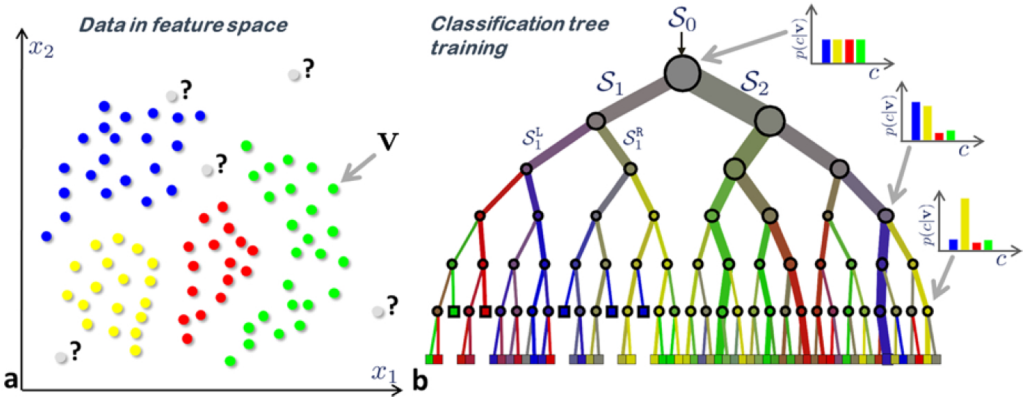

Figura 3.19: Rappresentazione di un Random Decision Forest [18]

Il training della funzione obbiettivo

La fase di training della foresta avviene ottimizzando i parametri della funzione split di ogni nodo j, che consiste nel massimizzare l'information gain della funziona obiettivo

$$\boldsymbol{\theta}_j = \arg\max_{\boldsymbol{\theta}\in\mathcal{T}_j} I(\mathcal{S}_j, \boldsymbol{\theta}).$$

Per il problema della classificazione l'information gain è definito per distribuzioni discrete nel seguente modo:

$$I(S_j, \boldsymbol{\theta}) = H(S_j) - \sum_{i\in\{L,R\}} \frac{|S_j^i|}{|S_j|} H(S_j^i)$$

con i che indicizza i due nodi figli. L'entropia per un generico insieme S dei training points è definita come:

$$H(S) = -\sum_{c\in C} p(c)\log p(c)$$

dove $p(c)$ è calcolato come un istogramma empirica normalizzata di etichette corrispondenti ai training points in S. Come illustrato in Figura 3.19.b addestrando un albero di classificazione massimizzando l'information gain ha la tendenza di produrre alberi i quali l'entropia della distribuzione delle classi associata ai nodi andando dalla radice alle foglie diminuisce. A sua volta, questo permette di aumentare la certezza di previsione.

Randomness

Come visto precedentemente la casualità viene controllata attraverso il parametro $\rho = |\mathcal{T}_j|$. Per esempio prima di iniziare possiamo fissare il valore del parametro $\rho = 1000$ che corrisponde a selezionare in modo random 1000 features, su un insieme di un miliardo. Possiamo generare ogni sottoinsieme random $\mathcal{T}_j$ quando ne abbiamo bisogno, prima di iniziare la fase training del nodo corrispondente.

Le foglie e modello di predizione Ensemble

Il modello Random Forest utilizzato per la classificazione produce un output probabilistico, in quanto non restituiscono un solo pronostico della classe, ma un intero distribuzione di classe. Infatti durante la fase di test, ogni foglia dell'albero porta alla probabilità a posteriori $p_t(c|\mathbf{v})$. Quindi possiamo definire l'output della foresta come

$$p(c|\mathbf{v}) = \frac{1}{T}\sum_{t=1} T p_t(c|\mathbf{v})$$

come visto precedentemente. Si nota molto bene in figura Figura 3.20 infatti notiamo che durante la fase di test ogni nodo da predire $\mathbf{v}$, non etichettato, è mandato su tutti gli alberi che compongono la foresta.

Il metodo appena visto che utilizza la media a posteriori è una scelta possibile di ensemble, che utilizzerò in questo progetto tesi, ma esistono anche altre alternative [18].

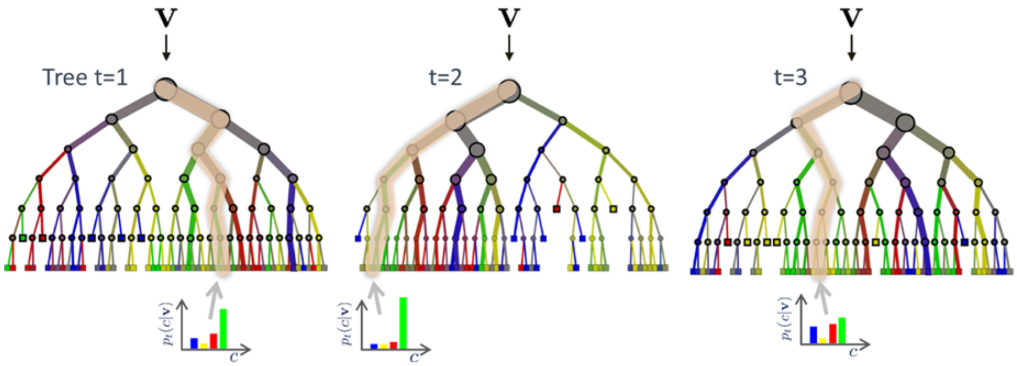

Figura 3.20: Classificazione attraverso il modello Random Forest [18]

3.11 Clustering

Con il termine *clustering* (raggruppare) si denota un famiglia di metodi *non supervisionati* in grado di creare collezioni di oggetti non noti, raggruppandoli in insiemi (cluster) di oggetti tra di loro simili, ma diversi da tutti gli elementi che invece non vi appartengono. Lo scopo è massimizzare la similarità di oggetti appartenenti ad uno stesso cluster (similarità intraclasse) e minimizzare la similarità tra oggetti appartenenti a cluster diversi (similarità interclasse). Le tecniche di clustering possono essere classificate in base a diversi parametri. Uno di questi parametri è il criterio mediante il quale si valuta la somiglianza dei dati da analizzare.

I metodi di clustering possono essere ricondotti a due famiglie:

- **Clustering gerarchico**: organizza i dati, attraverso apposite operazioni, in una sequenza innestata di gruppi che possono essere visualizzati come un albero.
- **Clustering partizionale**: individua (solitamente attraverso algoritmi euristici iterativi) delle partizioni che minimizzano un dato criterio di clustering, ad esempio la somma dei quadrati degli errori.

Comuni a entrambe le famiglie sono i *criteri di clustering* che si possono scegliere per specificare il grado di ottimalità di ogni soluzione ammissibile; a seconda del criterio scelto esistono poi vari *algoritmi di clustering* che forniscono una procedura per determinare le soluzioni che lo ottimizzano.

La maggior parte dei criteri di clustering sono definiti sulla base delle due osservazioni seguenti: i pattern all'interno dello stesso cluster devono essere tra loro più simili rispetto a pattern appartenenti a cluster diversi; i cluster sono costituiti da nuvole di punti a densità relativamente elevata, separate da zone dove la densità è più bassa. Tra i diversi criteri possibili:

- **somma dei quadrati degli errori**: minimizza i quadrati delle distanze dai centri delle classi (detto anche criterio di *minima varianza*).
- altri criteri basati su **varianza intraclasse**: si tratta di una famiglia di metodi tutti tesi a minimizzare la varianza all'interno delle classi.
- criteri basati su **scattering**: tendono a minimizzare la varianza intraclasse e allo stesso tempo a massimizzare quella inter-classe.

3.11.1 Clustering gerarchico

Il clustering gerarchico opera in maniera simile al modo di eseguire classificazione in tassonomia biologica, dove ad esempio gli insetti vengono gerarchicamente classificati specializzandone le specie a partire da famiglie molto ampie fino a famiglie molto più ridotte. Gli algoritmi possono essere *bottom-up (agglomerativi)* o *top-down (divisivi)*.

Nel primo caso si parte cercando di aggregare singoli elementi e ad ogni passo (livello) si fondono in un cluster gli elementi o i sotto-cluster che sono tra loro più simili rispetto a un determinato criterio.
Nel secondo (più complesso e quindi meno utilizzato) si parte con un singolo cluster e ad ogni livello si suddividono in sotto-cluster gli elementi più diversi. In entrambi i casi il risultato può essere rappresentato attraverso un albero.

I metodi gerarchici (tra i più noti possiamo citare Single-Link e Complete-Link) sono nella pratica utilizzati soprattutto quando i pattern sono discretizzati e non appartengono a uno spazio metrico. In tal caso infatti criteri di somiglianza "ad-hoc" possono essere implementati per controllare l'aggregazione e la divisione.

3.11.2 Clustering partizionale

Detto anche non gerarchico, o k-clustering, in cui per definire l'appartenenza ad un gruppo viene utilizzata una distanza da un punto rappresentativo del cluster (centroide, medioide ecc...), avendo prefissato il numero di gruppi della partizione risultato. Gli algoritmi più comunemente utilizzati sono K-means (o la sua variante Fuzzy C-means) e Expectation-Maximization (EM).

K-means

Si tratta di un metodo molto semplice computazionalmente e altrettanto semplice da implementare, che ottimizza il criterio "somma dei quadrati degli errori" . Nella sua versione base, l'algoritmo può essere così descritto:

- genera K cluster e determinane il centro; oppure genera direttamente K punti da usare come centri dei cluster;
- assegna ciascun punto al cluster il cui centro è più vicino, in base al criterio di distanza scelto (euclidea, Mahalonobis...);

- ricalcola i centri dei cluster come media delle coordinate di tutti i punti che vi appartengono;
- ripeti finché il criterio di terminazione non è soddisfatto (in genere il criterio di terminazione prevede che i centri rimangano costanti, e che quindi le partizioni non cambino tra due passi successivi).

Questo algoritmo tende a convergere piuttosto rapidamente (è raro che occorrano più di 10 passi) e a fornire risultati piuttosto buoni, a patto di partire da una soluzione iniziale ragionevole.

In Figura 3.21 possiamo vedere due esempi di clustering ottenuti con l'algoritmo K-means, a sinistra vogliamo un numero k $= 3$ cluster, mentre nella figura di destra un numero k $= 18$. Differenti colorazioni dei dati, indicano a quale cluster sono associati gli elementi.

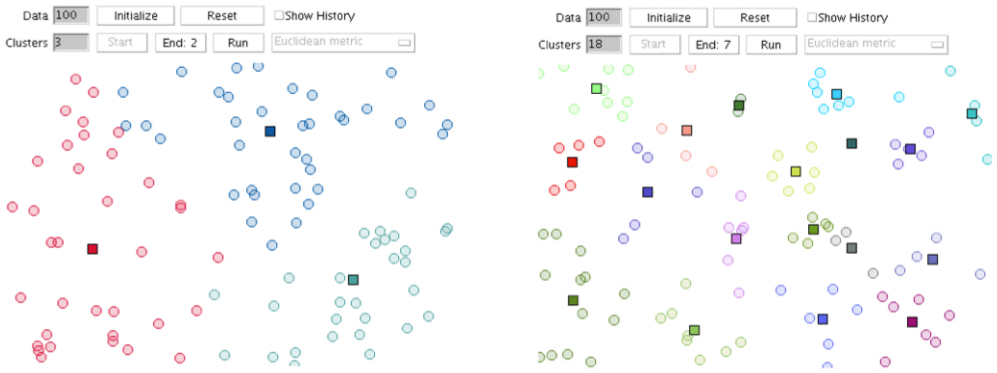

Figura 3.21: esempi di clustering con valori differenti per il parametro k

K-means ha alcuni *svantaggi*: innanzitutto il numero di classi K deve essere noto a priori; inoltre l'ottimizzazione è iterativa e locale, quindi si può avere convergenza su un massimo locale della soluzione.

L'obiettivo è determinare i K gruppi di dati generati da distribuzioni Gaussiane. Si assume che gli attributi degli oggetti possano essere rappresentati come vettori, e che quindi formino uno spazio vettoriale. L'algoritmo raggruppa N vettori o features in K clusters (gruppi) e ritorna il centro dei K gruppi. Ogni cluster viene identificato mediante un centroide o punto medio.

In generale, il clustering k-means è un problema **NP-hard**, quindi sono stati sviluppati una serie di algoritmi euristici per il suo calcolo. Il più comunemente usato è detto **algoritmo di Lloyd**.

Dato un insieme iniziale di K centroidi, $C_1^{(1)}, ..., C_K^{(1)}$, scelti a caso in base al Dataset fornito (Figura 3.22(1)), l'algoritmo procede alternando due passi di elaborazione:

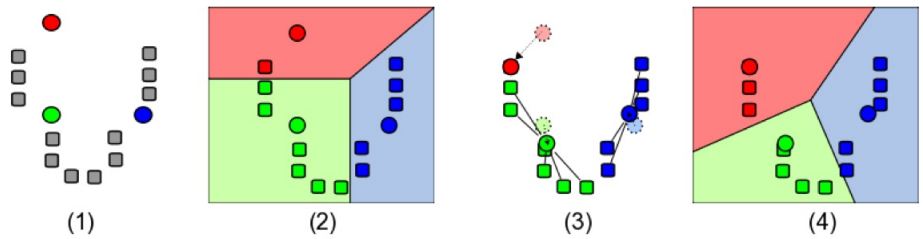

Figura 3.22: Esempio di esecuzione di K-means (K=3) [22]

Assegnazione Assegnazione ciascuno degli elementi da esaminare viene assegnato al cluster il cui centroide risulta più vicino (Figura 3.22(2)):

$$P_i^{(t)} = \{X_j \quad : \quad \|X_j - C_i^{(t)}\| \quad \leq \quad \|X_j - C_{i*}^{(t)}\| \quad \forall i* = 1, ..., K\}$$

Aggiornamento Calcolo dei nuovi centroi di associati a ciascun cluster (Figura 3.22(3)):

$$C_i^{(t+1)} = \frac{1}{|P_i^{(t)}|} \sum_{X_j \in P_i^{(t)}} X_j$$

Si dimostra che l'algoritmo converge quando l'assegnazione non cambia più (Figura 3.22(4)).

Essendo un algoritmo euristico, non c'è garanzia che esso converga ad un ottimo globale ed il risultato finale può dipendere dalla scelta dei cluster iniziali. Dato che l'algoritmo è in genere molto veloce, si tende a ripetere la sua esecuzione piu' volte con differenti condizioni di partenza. È stato dimostrato che esistono certi insiemi di punti per i quali k-means converge in tempo superpolinomiale: $2^{\Omega(\sqrt{N})}$.

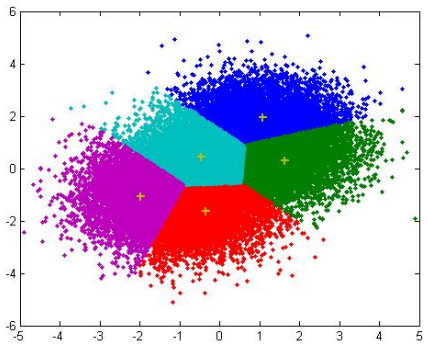

Figura 3.23: Suddivisione di uno spazio 2D utilizzando l'algoritmo K-Means

La *variante fuzzy del K-means* consente a un pattern di appartenere con un certo grado di probabilità a diverse classi; questa variante fornisce a volte una convergenza più robusta verso la soluzione finale, ma soffre in sostanza degli stessi problemi della versione standard di K-means.

Diverse varianti sono state proposte per risolvere questi problemi: ad esempio per minimizzare il rischio di convergenza verso minimi locali l'algoritmo può essere eseguito tante volte a partire da soluzioni iniziali diverse, casuali o magari prodotte da un metodo evoluzionistico (algoritmo genetico).

3.12 Valutazione delle Prestazioni

La valutazione delle prestazioni richiede un attento esame del problema e delle componenti per cui ha senso effettuare una valutazione. E' necessario utilizzare in modo ragionato le misure di valutazione disponibili, giustificare perché utilizzare una misura piuttosto che un'altra o perché complementare una misura con un'altra al fine di avere un quadro completo delle prestazioni.

Analizerò nel dettagli queste tecniche nel Capitolo 6.

Capitolo 4

Modelli di rappresentazione

In questo capitolo tratterò il problema di come rappresentare un immagine in modo strutturato, per essere automaticamente classificata nella opportuna classe, cioè riconoscere un immagine nuova basandosi sulla presenza di una particolare classe di oggetti al suo interno, come un pedone, una macchina, una bicicletta o un camion, ecc. Il problema risulta essere molto interessante e attuale, ma difficile da realizzare come illustrerò nei successivi capitoli. Diversi modelli sono stati sviluppati nell'ambito del riconoscimento e classificazione di immagini. In seguito andrò ad analizzare la struttura di tre modelli studiati in letteratura.

4.1 Part-based Category Models

Quando vogliamo riconoscere un categoria di oggetti, non solo potrebbe cambiare l'aspetto dell'oggetto rappresentato, dovuto alla variabilità della categoria, ma anche la collocazione spaziale dell'oggetto. Così non sempre riusciamo a trovare l'esatta corrispondenza, tuttavia, come possiamo notare nella Figura 4.1, è possibile identificare frammenti dell'oggetto o parti di esso con un aspetto similare e che appaiono nella stessa posizione. Ad esempio le ruote di una macchina, o di una motocicletta, oppure la posizione degli occhi/naso/bocca di una persona.

L'idea alla base di questo modello quindi è quella di apprendere modelli di oggetti basati su parti differenti di uno oggetto che lo contraddistinguono in basa alla loro relazione spaziale.

Figura 4.1: Rappresentazione di parti dell'oggetto con un precisa relazione spaziale [4]

4.1.1 Categorizzazione di oggetti con modello basato sulle parti

Il modello Part Based fu introdotto per la prima volta da Burl *et al.* [54] e Weber *et al.* [55], ma negli ultimi anni sono state proposte in letteratura molte varianti. Lo scopo di questo modello è di rappresentare un oggetto come costituito da differenti parti, ciascuna delle quali possiede delle proprietà, quali *aspetto* o *grandezza relativa*. Ovviamente queste parti devono essere identificate all'interno delle immagini, definendo così le forme tramite la loro posizione nell'immagine, e per questo esse possono essere erroneamente assegnate allo sfondo.

Questo significa che l'algoritmo deve essere in grado di *selezionare* quale regione locale dell'oggetto rappresenta e dovrebbe inoltre *raggruppare* parti simili in una rappresentazione comune.
Una soluzione ottima al problema di selezionamento, implicherebbe una ricerca in uno spazio enorme. Lo sviluppo di *features* invarianti locali, ha fornito una soluzione alternativa al problema, fornendo ottimi risultati in pratica.

Una volta che regioni precise dell'oggetto sono state selezionate, la prossima fase è quella di scegliere un metodo per rappresentare la relazione spaziale che hanno queste parti appartenenti all'oggetto. Questa scelta rifletta le assunzioni di indipendenza reciproca che vogliamo fare relativamente alla collazione delle parti dell'oggetto e che direttamente riguarda il numero di parametri necessa-

ri per specificare interamente il modello risultante, oltre che della complessità dell'inferenza eseguita utilizzando questo modello.

Vari modelli spaziali sono stati proposti negli anni, la Figura 4.2 da un idea delle rappresentazioni più utilizzate.

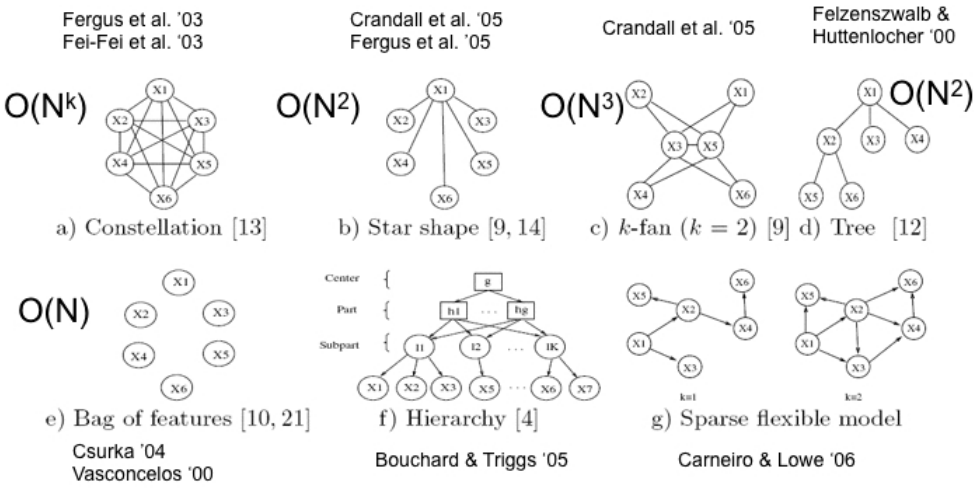

Figura 4.2: Diversi modelli basati sulle parti, proposti in letteratura [4]

Il modello più semplice è *Bag of Visual Words*, mostrato in Figura 4.2 (e). Questo modello non memorizza nessuna relazione geometrica al suo interno, cioè, esso non tiene conto delle relazioni spaziali delle regioni (o visual words) all'interno della immagine. Questo è un limite del modello.

All'estremo opposto notiamo il modello completamente connesso, il quale esprime relazione a due a due tra una qualsiasi coppia di parti. Questo tipo di modello è chiamato *Constellation Model* Figura 4.2 (a) ed è stato usato nell'articolo [56]. Il problema del modello completamente connesso è che tale modello richiede una crescita esponenziale del numero di parametri quando le parti dell'oggetto aumentano, il quale riduce la sua applicabilità a complesse categorie visuali.

Un compromesso è quello di combinare le parti in un modello a stella Figura 4.2 (b), il quale ogni parte è solamente connessa a parte di riferimento e indipendente da tutte le posizioni delle altre parti, dato e una stima per questa parte di riferimento. Il vantaggio di questo modello è la sua efficienza computazionale: puó essere eseguito in $\mathcal{O}(N^2)$, confrontato con $\mathcal{O}(N^k)$ del k-part Costellation Model. L'idea del modello a stella può essere facilmente generalizzato nel *Tree*

Model Figura 4.2 (d), dove la posizione corrispondente a una parte dell'oggetto è solamente dipendente dalla sua posizione. Questo tipo di modello è usato i nel framework *Pictorial Structures* di Felzenszwalb & Huttenlocher [57] a che illustra un efficiente algoritmo per stima della posa umana.

The *k-fan Model* è una via di mezzo tra il modello completamente connesso Constellation Model e il modello connesso singolarmente a stella. Consiste di un insieme di k parti di riferimento e un grande insieme di parti secondarie che sono connesse singolarmente alle parti di riferimento. Conseguentemente la sua complessità computazionale is $\mathcal{O}(N^{k+1})$. Una idea simile è quella impiegata nel *Hierarchical Model* Figura 4.2 (f), il quale contiene un layer di parti dell'oggetto (forma di stella), ognuna delle quali è densamente connessa ad un insieme di local features a livello più basso.

Infine menzioniamo *Sparse Flexible Model* Figura 4.2 (g) il quale la geometria di ogni parte locale, dipende dalla geometria del suo k-nearest neighbours, adatto per configurazioni flessibili di oggetti deformabili.

Nei paragrafi successivi tratterò i tre dei modelli maggiormente studiati in in letteratura: *Bag fo Visual Words*, che utilizzerò in questo progetto di tesi, *Constellation Model* e *Implicit Shape Model (a stella)*.

4.2 Bag-of-Visual Words

4.2.1 Introduzione

Questo modello nasce nell'*Information Retrieval*, utilizzato per la rappresentazione e classificazione di libri testuali in ambito informatico, senza l'aiuto umano. Sostanzialmente prende tutte le parole più importanti contenute in un libro (documento), e le memorizza in un dizionario. Un libro viene rappresentato come un vettore, le cui componenti descrivono la frequenza di ogni parola del dizionario, all'interno del testo stesso.

Questo tipo di rappresentazione viene chiamato *Bag of Words*, ovvero un insieme di termini non ordinato, basandosi unicamente sui termini in esso presenti, senza considerare esattamente la loro posizione o l'ordine delle singole parole all'interno dei libri, ma semplicemente quante volte le parole si verificano (fre-

quenza). La dimensione del vettore che rappresenta un documento, è uguale alla dimensione del dizionario.

Attraverso la fase di *training* è possibile calcolare la probabilità che ogni parola possa apparire in uno specifico libro e di un genere preciso, questa probabilità può essere applicata a tutta la Bag of Words e calcolare quale sarà il genere più probabile di un particolare libro [1].

4.2.2 Descrizione del modello

Il modello Bag of Words appena descritto, può essere applicato in modo similare per la classificazione di immagini, ma anzichè utilizzare *words* si utilizza *visual words*, che sono punti di interesse locali (key-points), estratti dalla immagine. Anche in questo caso il processo prevede, prima della classificazione vera e propria, una fase di apprendimento su un insieme di immagini delle quali è nota la categoria di appartenenza; per ognuna di esse viene costruito un vettore i cui campi indicano la frequenza o, più semplicemente, la presenza delle parole del vocabolario.

Un'immagine in questo modo può essere rappresentata come *Bag-of-Visual Words*, ovvero come un vettore che indica quali parole del vocabolario descrivono l'immagine. Questo tipo di rappresentazione, cattura l'aspetto disordinato e sparso dei concetti contenuti, senza imporre vincoli specifici di ordinamento o di posizione. Proposto per la prima volta in questi articoli: [1] e [58].

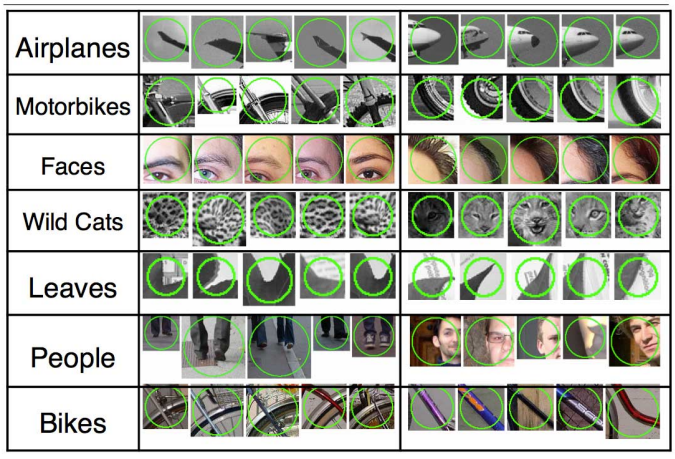

Figura 4.3: Esempio di un dizionario visuale [23]

Per classificare una nuova immagine basterà generarne il vettore di frequenza delle parole rispetto allo stesso vocabolario ed osservare la sua collocazione nello spazio delle categorie.

Il primo passo quindi sta nel costruire un dizionario visuale (codebook of visual Words) Figura 4.3, di dimensioni finite, partendo da caratteristiche locali. Occorre pertanto eseguire una fase di apprendimento su un insieme di immagini delle quali è nota la categoria di appartenenza. Per fare questo utilizziamo *keypoints detectors* che sono in grado di selezionare precise aree specifiche dell'immagine. Un insieme di key-points può essere visto come una patch saliente dell'immagine, caratterizzata da un alto contenuto informativo locale della stessa.

Terminata la fase di identificazione dei punti salienti la fase successiva è quella di descriverli in maniera robusta, questo processo avviene attraverso *feature descriptor*. Per ogni immagine viene costruito un vettore i cui campi indicano la frequenza, cioè la presenza delle parole del vocabolario, come vediamo dalla Figura 4.4. In questo modo abbiamo creato uno spazio delle categorie, utilizzando i vettori generati attraverso le immagini di training. Durante questa fase le immagini sono rappresentate come una collezione non ordinata di feature descriptors, non c'è relazione geometrica tra differenti features [59].

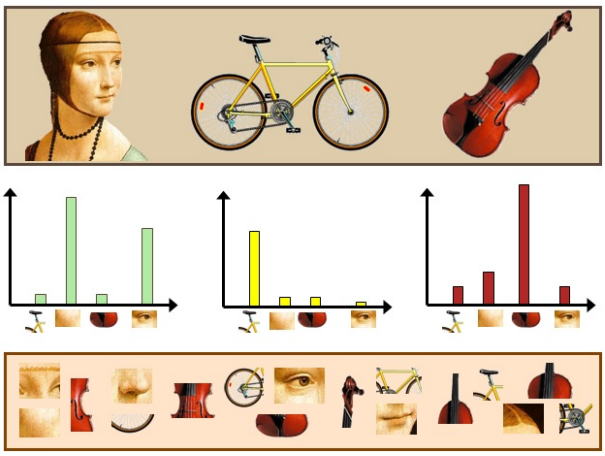

Figura 4.4: Rappresentazione di oggetti come istogramma di Visual Words [24]

Una volta estratti i keypoints dalle immagini di training, le informazioni ottenute vengono raggruppate attraverso un algoritmo di clustering, ottenendo così una rappresentazione compatta dello spazio delle features; proposto da Sivic and Zisserman [34] e Csurka e altri [1] . Questa operazione è fondamentale, in quanto

ci permette di ridurre la dimensione dello spazio delle features ad un valore fissato. Trattando ogni elemento del dizionario come visual words, possiamo pensare di avere un vocabolario capace di descrivere tutti i possibili patterns.

Per classificare una nuova immagine, quindi basterà generarne il vettore di frequenza delle parole rispetto allo stesso vocabolario ed osservare la sua collocazione nello spazio delle categorie. Viene generato un istogramma, che rappresenta il numero di occorrenze di un particolare image patterns estratto da un'immagine e basandosi su queste informazioni, attraverso un classificatore, viene individuata la categoria (pedone, macchina, bici, moto, ecc.) appartenente all'oggetto [25].

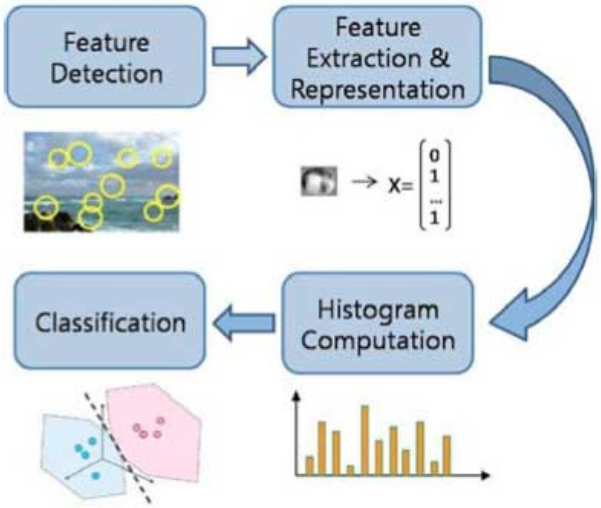

Figura 4.5: Pipeline del processo di classificazione [25]

Riassumendo queste sono le fasi dell'algoritmo:

- identificare automaticamente key-points attraverso un feature detector;
- estrarre descrittori locali per ogni image patches;
- assegnare patch descriptors a un insieme di predeterminati clusters (vocabolario) attraverso un algoritmo di quantizzazione vettoriale;
- costruire bag of key-points, il quale conta il numero di patches assegnati ad ogni cluster (istogramma delle features);
- trovare la corrispondenza nell'immagine per ogni specifica parola nel vocabolario per costruire bag of key-points; [60]
- applica un classificatore multi-classe il quale tratta bag of key-points come un vettore di features, e determina a quale categoria assegnare un immagine.

I vantaggi di questo metodo offre sono: semplicità, efficienza computazionale, invariante indipendentemente da illuminazione, orientamento, scala, prospettiva[1].

La figura 4.6 può essere utile a chiarire ulteriormente la spiegazione.

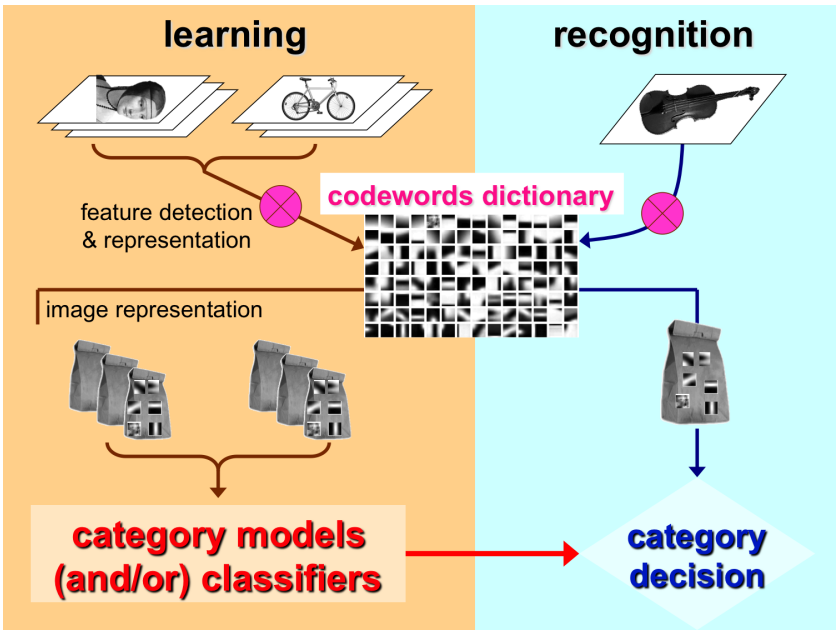

Figura 4.6: Modello BoW - Fase di apprendimento e di testing [24]

4.3 Constellation Model

Questo tipo di modello è chiamato *Constellation Model*, tenta di rappresentare classe di oggetti attraverso un insieme di N parti dell'oggetto sotto vincoli geometrici, mettendoli in relazione tra di loro.

La forma dell'oggetto è rappresentata dalla posizione unica delle parti. L'intero modello è generativo e probabilistico, data la probabilità di assegnare le parti in maniera corretta, e queste probabilità sono modellate con delle Gaussiane (le probabilità rappresentano nel modello aspetto, grandezza, forma e occlusione). L'aspetto, la scala, la forma e l'occlusione sono tutte modellate da una funziona di densità di probabilità, esempio Gaussiane. Il modello è invariante a variazioni di grandezze e a traslazioni (sia nella fase di Addestramento che in quella di Riconoscimento).

Parti dell'oggetto in esame, possono essere caratterizzati o del loro aspetto o dalla posizione nell'oggetto. Come risultato, il modello ottenuto è molto flessibile e può anche essere applicato a oggetti caratterizzati solamente dalla loro texture (per esempio riconoscimento di animali basandosi sulla texture presente nella pelliccia).

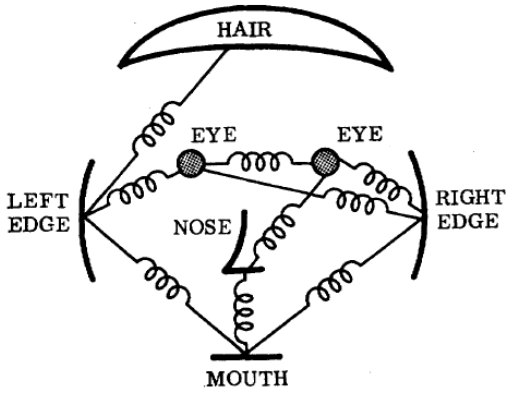

Figura 4.7: Scomposizione in parti di un viso umano [26]

Definiamo alcune alcune variabili tipicamente di input del modello (le parti appartengono al modello, le features appartengono all'immagine/classe vera e propria).

Ogni feature, indipendentemente dal tipo, fornisce al modello le seguenti informazioni:

- **x** - la posizione $(x, y)^T$ della feature nell'immagine, misurata in pixel.
- **s** - la grandezza relativa della feature nell'immagine, presa come raggio della feature e misurata in pixels.
- **d** - la descrizione della feature come coordinata di un spazio di descrizione(per esempio SIFT), espresso tramite un vettore di lunghezza a (SIFT: a=128).

Data un immagine i, assunto che vi siano N_t^i features di tipo t con T tipi in totale, le variabili sopra elencate saranno contenute in grosse strutture X^i, D^i, S^i, che a loro volta saranno racchiuse in tre strutture complessive per tutte le immagini I X, D, S.

Una volta definite le variabili che rappresentano il modello, definiamo come avviene il riconoscimento di una classe all'interno di una immagine:

- *Addestramento*: una categoria è appresa dal modello dopo che, individuate le features e le loro grandezze, i parametri delle densità sopra indicate (X, D, S) vengono stimati, in modo che il modello costruisca una descrizione di maximum-likelihood per tutti i dati di addestramento.

- *Riconoscimento* può essere condotto nel seguente modo: data una immagine di test, le regioni con le relative grandezze vengono individuate, ed i suoi parametri valutati in base alle densità X, D, S create al passo precedente.

In Figura 4.8 parte a) in alto, rappresentazione di parti super imposte su di una immagine di una moto, le ellissi rappresentano la covarianza di ogni parte e la probabilità di essere presente è mostrata a destra della media. In basso sempre nella parte a) viene espressa le relativa scala di densità. b) Campioni delle varie parti estratte dalle immagini il quale sono vicini alla media di densità dell'aspetto.

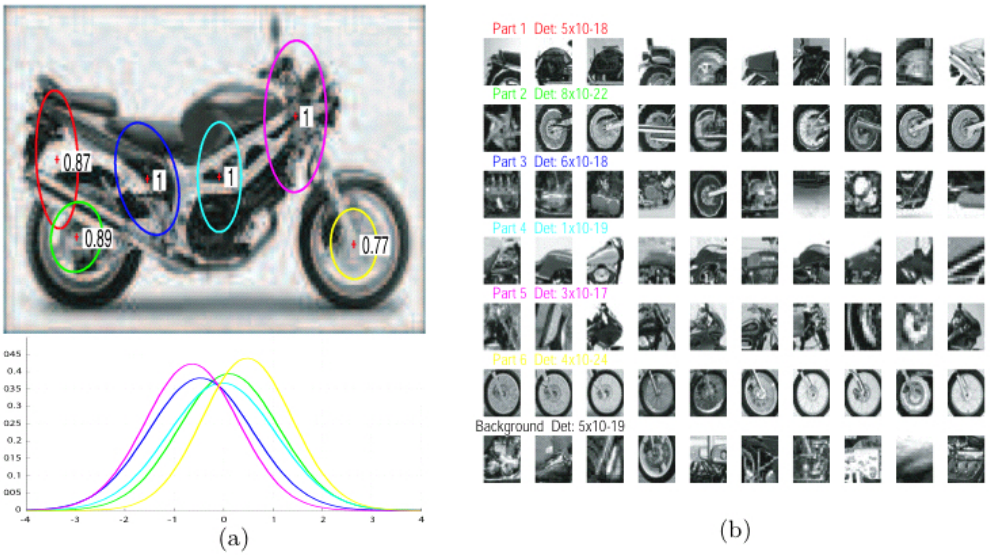

Figura 4.8: Rappresentazione del modello appena descritto [26]

Il modello completamente connesso appena descritto è molto potente, ma soffre del problema di una elevata complessità computazionale dovuto alla sua rapprentazione.

Nel prossimo paragrafo descriverò un modello che si basa su una rappresentazione spaziale molto più semplice, chiamato modello *a stella*.

Figura 4.9: Un tipico esempio di modello di un arereo, completamente connesso a 6 parti [27]

4.4 Implicit Shape Model (ISM)

Nel modello a stella la posizione riferita ad ogni parte dipende solamente da una parte centrale di riferimento, data questa posizione di riferimento, ogni parte è trattata indipendentemente dalle altre. In questo modo la forma dell'oggetto è definita solamente implicitamente, le informazioni quali parti, concordano sullo

stesso punto di riferimento. Questo è il motivo del nome: *Implicit Shape Model (ISM)*, presentato per la prima volta nel 2004 nell'articolo di [29].

L'approccio ISM utilizza una differente filosofia di interpretazione di rappresentazione dell'oggetto, infatti viene visto come una collezione di un numero elevato di *features*, che idealmente dovrebbero fornire una densa copertura dell'area dell'oggetto in esame. Ciascuna features ha un ben definito aspetto compatto, e una distribuzione di probabilità spaziale per le posizioni in cui può apparire relativamente al centro dell'oggetto.

4.4.1 Procedura di apprendimento

In contrasto al Constellation Model, the ISM richiede esempi di training etichettati, inoltre per poter sfruttare a pieno le potenzialità del modello ISM, gli esempi di training dovrebbero essere segmentati.

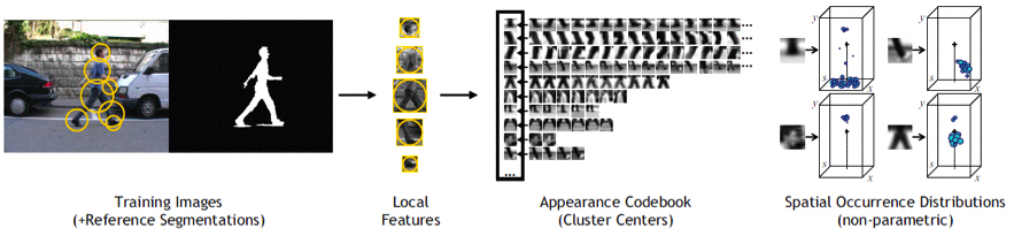

Figura 4.10: Rappresentazione della procedura di apprendimento di ISM [28]

La procedura completa di training ISM è illustrata in Figura 4.10. Il primo passo è quello di creare un vocabolario visuale (*codebook*), come per il modello *Bag of Words*, partendo da features locali invarianti di scala (tipo SIFT) e clusterizzando. Successivamente si fa apprendere a ISM la distribuzione spaziale delle occorrenze per ogni parola visuale, le immagini vengono scandite nuovamente e viene trovata per ogni features il corrispondente nel vocabolario usando uno schema a *soft-matching* (per esempio vengono attivate solamente le parole visuali che superano una certa soglia).

Le visual words, in questo modello non sono più accumulate in un istogramma (come BoW), ma sono processate e associate alla locazione di ogni feature che le ha selezionate. Questa procedura viene definita *voto*: si salva la loro posizione rispetto al centro dell'immagine (che deve essere quindi conosciuto a priori), creando così una lista di associazioni di tipo codeword $\rightarrow$ elenco voti. Per ogni

ogni visual words, ISM memorizza una lista con tutte le posizioni e la scala con cui queste features potrebbero trovare un match (corrispondente), relativamente al centro dell'oggetto. Tale lista non è altro che uno *Shape Model*.

In una immagine di test, la features potrebbe presentarsi in un'altra posizione rispetto alle features presenti nell'immagine originale, se ad esempio l'oggetto in questione risulta essere ruotato. Per questo motivo non si deve salvare la sua posizione assoluta rispetto al centro, ma la posizione del centro rispetto ad un sistema di riferimento locale ad essa.

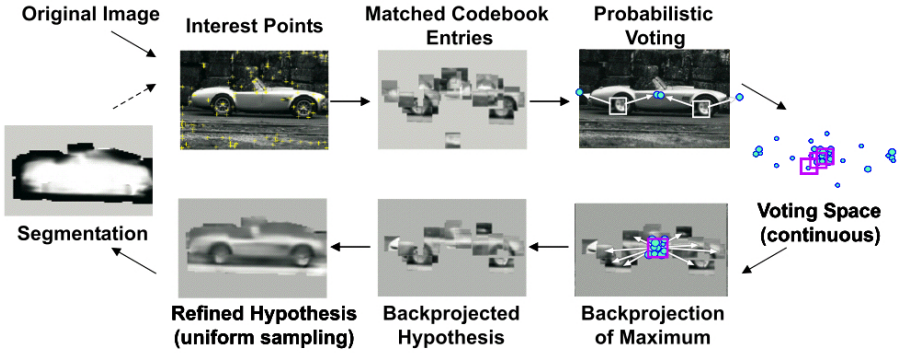

Figura 4.11: Rappresentazione della procedura di riconoscimento [29]

4.4.2 Procedura di riconoscimento

Terminata la fase di creazione degli *Shape Model* per ogni categoria di oggetti, essi possono essere riutilizzati per la fase di test, che avviene in maniera identica a quella di training. Dopo l'attivazione delle codeword si estraggono, per ogni Shape Model le corrispondenze del centro dell'oggetto e si va ad inserire un voto nell'*Hough Space*[1] corrispondente, nella posizione ottenuta ruotando e traslando secondo il nuovo sistema di riferimento della feature appena analizzata. Questa procedura di riconoscimento ISM è descritta in Figura 4.11.

[1]La trasformata di Hough è una tecnica di estrazione di features utilizzata in Computer Vision. Lo scopo della tecnica è quello di trovare istanze imperfette di oggetti all'interno di una certa classe avente una determinata forma, attraverso una procedura di votazione. La procedura del voto avviene in uno spazio parametrico, costruito dalll'algoritmo per calcolare la trasformata di Hough.

Al termine della fase di riconoscimento vi sarà quindi un *Hough Space* per ogni categoria, con all'interno una serie di voti relativi all'interpretazione della stessa immagine di test, secondo i vari *Shape Model*. La categoria finale sarà quella con il punteggio più elevato, secondo una strategia di ricerca nell'*Hough Space*.

4.4.3 Verifica dell'ipotesi

Infine le ipotesi estratte dell'oggetto vengono verificate attraverso un procedura che selezione un insieme di ipotesi che spiegano nel migliore dei modi il contenuto dell'immagine. Brevemente questa procedura esprime il punteggio delle ipotesi come somma dei sui pixel *p(figure)* mappa di probabilità.

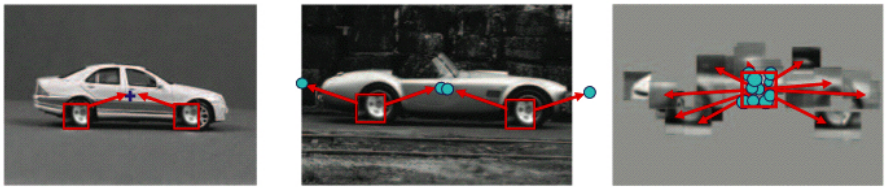

Figura 4.12: Rappresentazione modello ISM *(a sinistra)*. Durante la fase di training, vengono apprese le posizioni di ogni visual words relative al centro dell'oggetto *(al centro)*. Per la fase di riconoscimento viene utilizzata la distribuzione delle occorrenze *(a destra)*. [4]

4.4.4 Valutazioni

Un punto di forza di questo metodo è la creazione di un modello geometrico, ma il punto di debolezza è che manca la possibilità di specificare configurazioni illegali, come ad esempio una faccia con 2 bocche, poiché le feature non vengono contate, ma vengono solo misurate. ISM e alcune sue estensioni sono state applicate con successo a varie categorie di oggetti [28].

Capitolo 5

Soluzione implementata

L'obiettivo del progetto è quello di classificare una determinata immagine o frame video, identificando automaticamente la classe di appartenenza, basandosi sulla presenza di una particolare classe di oggetti al suo interno. La classificazione di immagini è un tema vasto, che trova ampi settori di applicazione. Il suo scopo principale è quello di interpretare le caratteristiche di immagini, per poi poterle classificare in classi di appartenenza.

Ho scelto di realizzare questa applicazione attraverso il modello *Bag of Visual Words (BoW)*, descritto nel capitolo 4.2, per la sua semplicità ed efficienza computazionale, rispetto ad altri modelli che includono informazioni spaziali al loro interno. Nonostante tutto, BoW permette di ottenere risultati molto interessanti, come illustrerò nel capitolo seguente.
Durante gli esperimenti effettuati ho scelto di utilizzare SIFT come tecnica di localizzazione e descrizione di features, dato che nei sistemi che implementano il modello BoW, l'utilizzo di questo descrittore è divenuto lo standard grazie alla sua robustezza. Per la classificazione ho utilizzato diversi algoritmi di apprendimento supervisionato: SVM, AdaBoost, Random forest e Artificial Neural Network. Indipendentemente dal classificatore utilizzato, mi aspetto un calo delle prestazioni all'aumentare del numero delle classi utilizzate nella fase di test.

Nei primi due capitoli descriverò gli strumenti utilizzati; nel Capitolo 5.2 affronterò il problema di classificazione multiclasse. I dettagli implementativi del modello BoW sono illustrati nei Capitoli 5.3 e 5.4. Il Capitolo 5.5 è dedicato all'analisi degli algoritmi di classificazione. Infine nel Capitolo 5.6 ho spiegato come si può integrare questo progetto, sulla sterecamera, utilizzando BRISK.

5.1 Dataset per classificazione

In questo capitolo analizzerò le risorse liberamente fruibili per quanto riguarda i dati su cui operare. Dato lo scopo di questo progetto di tesi è molto importante addestrare il classificatore su un insieme vasto di immagini, per rendere il classificatore robusto. Le immagini considerate dovrebbero contenere solamente un oggetto appartenente a una specifica categoria, dato che in presenza di più oggetti di categorie diverse, il classificatore non riuscirebbe a prendere decisioni.

Con l'avvento di Internet è diventato estremamente facile accedere a collezioni di immagini (Dataset), anche se spesso queste collezioni non si prestano a casi di studio o esperimenti scientifici.

Università e centri di ricerca, mettono a disposizione Dataset per scopi didattici. Questa è una lista (non esaustiva) di alcuni Dataset gratuitamente scaricabili online:

- **LEAR - Learning and Recognition in Vision** centro di ricerca congiunto INRIA Grenoble - RhôneAlpes and the LJK laboratory (indirizzo web: `http://lear.inrialpes.fr/data`). Sono forniti diversi Dataset specifici per determinati scopi per esempio: Soccer Team Dataset, Horse Dataset, Car Dataset, Person Dataset.
- **Beckman Institute - Computer Vision and Robotics** - University of Illinois at Urbana-Champaign - (indirizzo web: `http://www-cvr.ai.uiuc.edu/ponce_grp/`) Sono forniti diversi Dataset specifici per determinati scopi per esempio: Texture Database, Object Recognition Database, Butterflies Dataset, Birds Dataset, Visual Hull Datasets, 3D Object Recognition Stereo Dataset.
- **California Institute of Technology - Computational Vision** (indirizzo web: `http://www.vision.caltech.edu/archive.html`) mette a disposizione diverse tipologie di Dataset:
  - **Caltech-UCSD Birds 200** è un set di dati di immagini con le foto di 200 specie di uccelli (soprattutto Nord America);
  - **Caltech Pedestrian Database** collezione di immagini di pedoni.
  - **Caltech** in cui sono presenti macchine, moto, aerei, volti, foglie di alberi e sfondi;
  - **Caltech-101** 100 categorie di oggetti differenti;
  - **Caltech-256** 255 categorie di oggetti differenti;

  - **Home Objects** questo dataset contiene oggetti casuali da casa. Gli oggetti sono tratte per lo più da ambienti cucina, bagno e living-room.

- **Cognitive Computation Group** University of Illinois at Urbana-Champaign (indirizzo web: `http://cogcomp.cs.illinois.edu/page/resources/data` UIUC Image Database for Car Detection e UIUC Dataset of 3D object categories (S. Savarese and L. Fei-Fei);

- **PASCAL VOC dataset** (indirizzo web: `http://pascallin.ecs.soton.ac.uk/challenges/VOC/voc2009/index.html`);

- **Visual Geometry Group** - Department of Engineering Science, University of Oxford - 17 Flower Category Dataset (indirizzo web: `http://www.robots.ox.ac.uk/~vgg/data0.html`);

- **Computer Vision Lab** - Stanford University - (indirizzo web: `http://vision.stanford.edu/resources_links.html#datasets`) mette a disposizione diverse tipologie di Dataset:
  - **Stanford Dogs Dataset**;
  - **Event Dataset** contenente 8 categorie di eventi sportivi: sport a remi (250 immagini), badminton (200 immagini), Polo (182 immagini), bocce (137 immagini), snowboard (190 immagini), croquet (236 immagini), vela (190 immagini), e arrampicata su roccia (194 immagini). Informazione della distanza degli oggetti in primo piano è prevista anche per ogni immagine;
  - **People-Playing-Musical-Instrument (PPMI) Dataset** Il dataset PPMI contiene immagini di esseri umani che interagiscono con dodici diversi strumenti musicali. Essi sono: fagotto, violoncello, clarinetto, erhu, flauto, corno, chitarra, arpa, registratore, sassofono, tromba e violino;
  - **3D Object Category Dataset**;

- **Computer Vision at Microsoft Research Cambridge** (web: `http://research.microsoft.com/en-us/projects/objectclassrecognition/default.aspx`) Object Recognition Image Database

- **Center for Biological and Computational Learning** (indirizzo web: `http://cbcl.mit.edu/software-datasets/CarData.html`) Car Dataset, StreetScenes Challenge Framework.

5.1.1 PASCAL VOC

PASCAL è una rete di eccellenza finanziata dall'Unione Europea. È un istituto distribuito che riunisce ricercatori e studenti di tutta Europa. PASCAL sostiene e incoraggia la collaborazione tra esperti di Machine Learning, Statistica e ottimizzazione. Promuove inoltre, l'utilizzo di Machine Learning in molti domini applicativi rilevanti tra cui Computer Vision.

The Pascal Visual Object Classes (VOC) Challenge

VOC è un concorso annuale che è stato organizzato dal 2005 al 2012 ogni anno. È un punto di riferimento in visual object category recognition and detection. Fornisce alla comunità di Computer Visione e Machine Learning due componenti fondamentali:

- un set di dati a disposizione del pubblico di immagini e annotazioni, insieme con il software di valutazione standardizzata;
- un concorso annuale e workshop.

La sfida è suddivisa in due task principali: *classificazione* e *detection*, nonché due compiti aggiuntivi sulla *segmentazione* a livello di pixel, e *layout di persona*. L'obiettivo della sfida VOC è indagare le prestazioni dei metodi di riconoscimento su un ampio spettro di immagini naturali. La cosa interessante è che i dataset VOC contengono una significativa variabilità in termini di dimensioni dell'oggetto, orientamento, posa, illuminazione, posizione e l'occlusione. Inoltre i Dataset non presentano errori sistematici, favorendo le immagini con oggetti centrati o buona illuminazione.

Il Dataset VOC è composto da fotografie di vita quotidiana, annotate e raccolti sul sito (Pascal Evaluation Server `http://host.robots.ox.ac.uk:8080/`.

Nel progetto di tesi è stato utilizzato parte del Dataset VOC2010 di valutazione. Grazie alle annotazioni presenti in ogni immagine, ho potuto segmentare i singoli oggetti all'interno dell'immagine e creare un Dataset specifico, attraverso cui valutare i modello Bag of Visual Words implementato.

5.1.2 Dataset Caltech

Questi Dataset sono formati da diverse categorie di oggetti, e ogni immagine è suddivisa nella rispettiva cartella che contraddistingue l'oggetto presente nell'immagine. Ho scelto di utilizzare questo Dataset perché credo sia il più adatto agli scopi della mia tesi, in quanto le immagini presenti sono relativamente semplici: il soggetto solitamente occupa l'intera scena dell'immagine (poco sfondo, ottimi per task di classificazione), la variazione intra-classe è elevata e le prospettive nelle riprese sono simili tra un'immagine e l'altra.

Ovviamente per scopi sperimentali è sconsigliato usare l'intero dataset, prediligendo un approccio che selezioni particolari classi per testare l'efficacia del proprio classificatore. Inoltre questi dataset sono fortemente raccomandati per scopi di classificazione, ma sconsigliati per applicazioni mirate alla localizzazione di classi in una immagine, come suggerito nella guida.

Un sommario dei due principali Dataset Caltech è disponibile in Tabella 5.1 mentre un estratto di alcune delle loro classi sono forniti in Figura 5.1.

Tabella 5.1: Sommario dei DataSet Caltech

DataSet	Released	Categories	Images Total	Images per Category			
				Min	Med	Mean	Max
Caltech-101	2003	102	9144	31	59	90	800
Caltech-101	2006	257	30607	80	100	119	827

Dataset Caltech 101 [2]

Il dataset Caltech-101 è stato realizzato da Fei-Fei Li, Marco Andretto, e Marc Aurelio Ranzato, ricercatori di Computer Vision del California Institute of Technology, nel 2003. Il dataset è stato introdotto per la prima volta in un lavoro presentato al CVPR del 2004 [2] e utilizzato nell'articolo [61]. Inoltre una versione estesa è stata pubblicata nel 2006 [3]. È costituito da immagini di oggetti appartenenti a 101 categorie diverse più una classe di immagini di "background" (esempi negativi), ricavata dalla ricerca tramite la keyword *thing* in Google Image. Ogni categoria ha un numero variabile di immagini tra 40 ed 800, ma la maggior parte delle categorie ha circa 50 immagini catalogate.

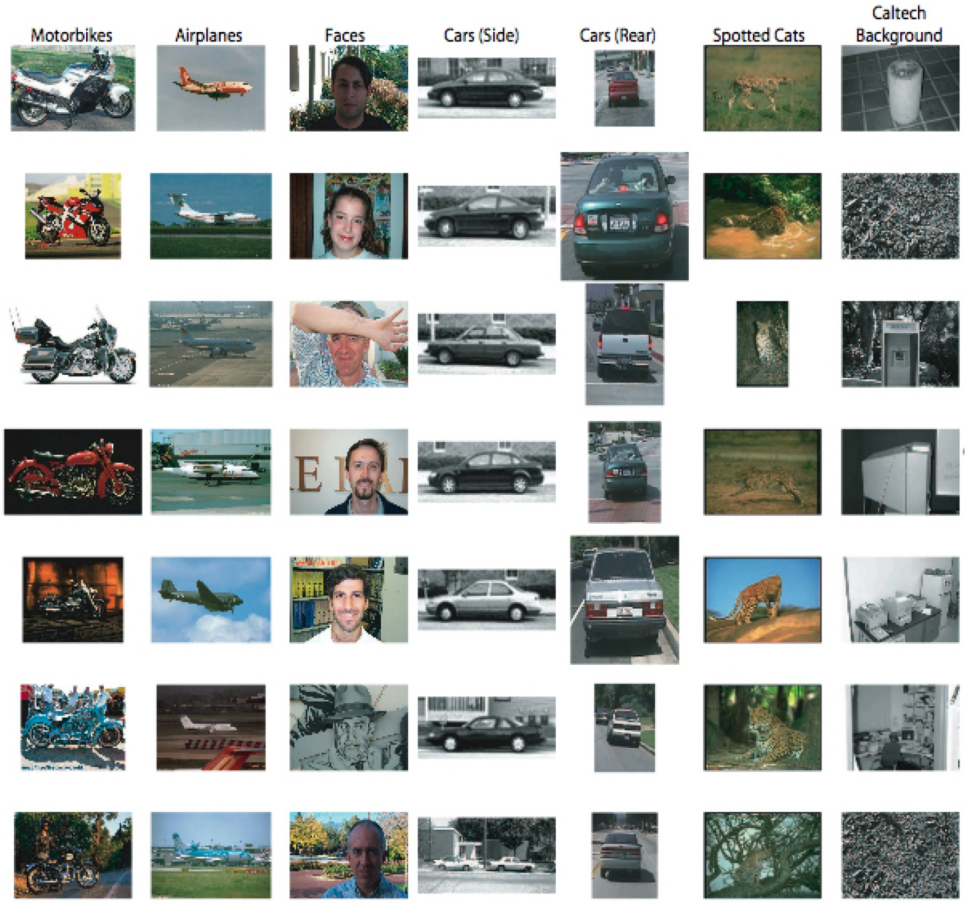

Figura 5.1: Alcuni esempi di immagini da Caltech Dataset (Motorbikes, Airplanes, Faces, Cars (Rear) e Leopards)

Le dimensioni delle immagini sono di circa 300×200 pixels. Tra le varie categorie vi sono: coccodrilli, sedie, aerei, macchine fotografiche, elefanti, motocicli, felini, volti, etc...

L'obiettivo del lavoro presentato da Fei-Fei Li è mostrare che un modello accurato per la classificazione può essere ottenuto a partire da un numero molto ristretto di esempi di training. Per ogni categoria vengono scelte N immagini per l'addestramento in modo casuale, con $N = \{1, 3, 5, 10, 15, 20, 30\}$; i test sono eseguiti con al più altre 50 immagini per categoria scelte sempre in modo casuale tra le rimanenti.

Le immagini scelte sono state sottoposte ad ulteriori passi di elaborazione: quando necessario sono state applicate trasformazioni di ribaltamento e/o ro-

tazione per fare in modo che tutte le istanze di una stessa classe mostrassero oggetti orientati lungo la stessa direzione. Al termine di questa fase di preprocessing tutte le immagini sono state scalate in modo da impostare la dimensione massima su 300 pixel. Questo è stato fatto per rendere il task di classificazione più facile, ma è facile intuire che un tale tipo di rotazione non riproduce appieno quella che potrebbe essere la realtà, e per questo resta limitato l'effetto prodotto.

Dataset Caltech 256 [3]

In questo secondo dataset le classi catalogate sono 256. Quindi in questo caso le classi introdotte sono state più che raddoppiate, ed anche le immagini presenti in ogni classe sono state aumentate, risultando in media circa 40 (a differenza di Caltech 101, dove la media è 31), per un totale di 30.608 immagini (contro 9.144 del precedente dataset Caltech 101). Tra le classi catalogate vi sono, torte, calcolatrici, scacchiere, tastiere di computer, delfini, ecc... (oltre a quelle menzionate in precedenza).

E' importante notare come le immagini, a differenza del precedente dataset, non siano allineate orizzontalmente, e di conseguenza non sono introdotte rotazioni artificiali nelle immagini. Questo Dataset, rappresenta una sfida più difficile rispetto a Caltech 101: infatti gli oggetti, oltre a non essere allineati nelle immagini, non sempre occupano la maggior parte dello spazio totale. In questo Dataset la classificazione delle immagini risulta essere più difficile, questo è dovuto in parte al maggior numero di esempi per classe, ed in parte alle difficoltà introdotte nella struttura delle immagini.

5.1.3 OpenCV Library

La Open Source Computer Vision Library è una libreria scritta in C e C++, che possiede più di 2500 algoritmi ottimizzati, utili nel campo dell'image processing e della computer vision. OpenCV è Sviluppato da Intel, e ora supportato da - Willow Garage e Itseez, è gratuito per uso sotto la licenza open source BSD. La licenza di distribuzione è priva di royalty e ciò consente il suo utilizzo anche in prodotti commerciali a condizione di mantenere le note di copyright. OpenCV è compatibile con la Intel Image Processing Library (IPL). Quest'ultima è una

libreria non open source che esegue solo operazioni di image processing e dalla quale OpenCV ha ereditato originariamente alcune strutture dati.

Se la libreria rileva Integrated Performance Primitives di Intel sul sistema come SSE o MMX, utilizza queste routine di proprietà per migliorare le prestazioni. Attualmente OpenCV sta cercano di sviluppare algoritmi ottimizzati che sfruttano architetture con supporto CUDA e OpenCL.

La libreria è utilizzata da aziende ben consolidate come Google, Yahoo, Microsoft, Intel, IBM, Sony, Honda e Toyota e questo è sintomo di affidabilità e sicurezza.

Lanciato ufficialmente nel 1999, il progetto OpenCV è stato inizialmente una iniziativa Intel Research per migliorare le applicazioni che sfruttano pesantemente la CPU (CPU-intensive). La prima versione alpha di OpenCV è stata rilasciata al pubblico in occasione della Conferenza IEEE sulla Computer Vision e Pattern Recognition, nel 2000, e cinque beta sono stati rilasciati tra il 2001 e il 2005. La prima versione 1.0 è stata rilasciata nel 2006. A metà del 2008, OpenCV ha ottenuto il sostegno sociale da Willow Garage, e ora è di nuovo in fase di sviluppo attivo. La seconda major release della OpenCV è stata rilasciata a ottobre 2009 con la versione 2.0.

OpenCV 2, include importanti modifiche all'interfaccia C++, che mira a modelli più semplici, pattern di tipizzazione sicura, nuove funzioni e miglioramenti in termini di performance su quei sistemi che permettono implementazione ed elaborazione multi-core. Release ufficiali tra 2009 e 2011 avvenivano circa ogni sei mesi, mentre negli ultimi due anni, dalla versione 2.4.0 in avanti (2012 e 2013) sono state rilasciate 9 versioni. Ora lo sviluppo è viene seguito da un team russo indipendente sostenuto da aziende commerciali. Nel mese di agosto 2012, ill supporto di OpenCV è stata rilevata da una fondazione no-profit. L'ultima versione attualmente scaricabile, rilasciata a dicembre 2013, è la 2.4.8

La portabilità della libreria OpenCV è completa, sono infatti disponibili le versioni per i sistemi Windows, Linux, BSD, Unix, Mac OSX, e sistemi embedded o Smartphone come Android, Maemo, iOS, BlackBerry.

Ho scelto di utilizzare nella mia tesi questa libreria, perchè mette a disposizione tutti gli strumenti necessari alla realizzazione del modello: dispone di tutte le tecniche attualmente più importanti di estrazione e descrizione features, tra cui SIFT, SURF, BRISK, ORB, FREAK. Mette a disposizione utili funzioni per

la realizzazione del modello BoW, trai cui la creazione attraverso clustering del vocabolario. Inoltre OpenCV, al suo interno, dispone di un modulo per la gestione di tutti i più importanti algoritmi di Machine Learning di apprendimento supervisionato.

Il fatto di essere liberamente distribuita garantisce sicurezza al codice e la possibilità di apportare modifiche al codice stesso, assicurandone così una continua evoluzione.

5.2 Decomporre un problema Multi-classe

Come illustrato nel Capitolo 3 sull'apprendimento automatico, lo scopo degli algoritmi supervisionati (supervised learning) è di produrre un modello utilizzando i dati provenienti da un *training set etichettato*. Possiamo suddividere i problemi di classificazione in due macro-categorie:

1. **binaria**
2. **multiclasse**

La **classificazione binaria** (o binomiale) ha il compito di classificare gli elementi di un dato insieme in *due* gruppi sulla base di una regola di classificazione.

Alcune applicazioni tipiche di *classificazione binaria* sono:

- spam tagging: decidere se una mail è ritenuta Spam oppure no;
- segmentare una determinata immagine: decidere se una area dell'immagine fa parte del contorno o no;
- test medici: per determinare se un paziente ha una certa malattia o no;
- controllo di qualità nelle fabbriche: decidere se un nuovo prodotto è abbastanza buono per essere venduto, o se deve essere scartato.

La **classificazione multiclasse** (o multinomiale) consiste nel classificare degli esempi in più di due classi. I problemi del mondo reale appartengono quasi tutti alla categoria multiclasse. Ad esempio, l'obiettivo dei sistemi di riconoscimento ottico dei caratteri (OCR) è quello di determinare il valore della cifra $(0, ..., 9)$ rappresentata su un immagine. Il numero di applicazioni che richiedono categorizzazione multiclasse è immenso.

Alcuni esempi di tali applicazioni sono:

- riconoscimento e classificazione di immagini;
- compiti di elaborazione del linguaggio naturale;
- classificazione di testi e categorizzazione;
- riconoscimento di sequenze in ambito biologico;
- riconoscimento di oggetti della visione industriale.

Alcuni *algoritmi di classificazione*, nativamente, consentono l'utilizzo di più di due classi, mentre altri sono per loro natura solo algoritmi di classificazione binari. Tuttavia attraverso tecniche particolari è possibile trasformare problemi multi-classe in problemi binari che possono essere facilmente risolti da classificatori binari.

Un esempio di classificatori che nascono come classificatori binari sono: Supper Vector Machines, che sono in grado di apprendere l'iperpiano ottimale per la separazione di esempi positivi da negativi. Un altro esempio è l'algoritmo del Percettrone, l'algoritmo Boosting e tutti i suoi derivati tra cui AdaBoost. Successivamente sono state proposte alcune varianti degli algoritmi originali che permettono di gestire problemi multi classe; questo è il caso di SVM, K-Nearest Neighbors, Naive Bayes.

Un'altra categoria di classificatori, come ad esempio Artificial Neural Network, si prestano molto bene ad essere estesi per risolvere problemi multiclasse. Come illustrato nel Capitolo 3.6, l'input della Rete Neurale sono N *neuroni* $\{x_1, ..., x_n\}$ che corrispondo alle features da classificare e come output *un solo neurone* che rappresenta l'etichetta assegnata alle features date in input $\{0, 1\}$. In questo modo riusciamo a risolvere problemi di classificazione binaria. L'estensione alla classificazione di problemi multiclasse è molto semplice, anzichè utilizzare un unico *neurone* di output utilizziamo un codice binario che definisce l'output della rete. Per ogni classe K viene definito un codice binario univoco di N bit, ogni bit della "codeword" corrisponde a un singolo neurone di output della rete neurale. Per esempio classe 1 = 01001, classe 2 = 10010, classe 3 = 00101; in questo modo avremo 5 vettori di output nella Rete Neurale. Questo è l'approccio che ho utilizzato nel progetto.

Infine esistono tipologie di classificatori che sono già in grado di risolvere problemi multiclasse, come per esempio Alberi di Decisione, o Random Forest utilizzati recentemente all'interno del progetto Microsoft Kinect.

In seguito descriverò tre tecniche per poter decomporre un problema multi-classe in una serie di problemi binari, per poi essere risolti in modo efficiente con un classificatore binario. Seguirò l'approccio proposto da Allwein, Schapire, Singer nel seguente articolo: "Reducing Multiclass to Binary" [62].

5.2.1 One-vs-all (OVA)

Questo è il caso piú semplice. Supponiamo di avere K classi; il problema di classificazione viene ridotto in K problemi binari, dove ogni problema discrimina tra la classe data e le altre $K-1$ classi. Vengono addestrati K classificatori, ciascuno utilizzando come esempi positivi quelli di una classe e come esempi negativi quelli di tutte le altre. In fase di decisione (testing) su un esempio sconosciuto, il classificatore che produce il massimo output viene considerato il vincitore e l'etichetta che corrisponde a questo classificatore viene assegnata all'esempio. Viene scelta la classe che ottiene il massimo margine dall'iperpiano.

Tabella 5.2: Esempio dell'approccio "One-vs-all"

Training set				
X_1	X_1 $-$	X_1 $+$	X_1 $-$	X_1 $-$
X_2	X_2 $-$	X_2 $-$	X_2 $+$	X_2 $-$
X_3 $\Rightarrow$	X_3 $-$	X_3 $-$	X_3 $-$	X_3 $+$
X_4	X_4 $-$	X_4 $+$	X_4 $-$	X_4 $-$
X_5	X_5 $+$	X_5 $-$	X_5 $-$	X_5 $-$
	$\Downarrow$	$\Downarrow$	$\Downarrow$	$\Downarrow$
	h_1	h_2	h_3	h_4

Nella Tabella 5.2 è illustrato un esempio. Dato un problema con esempi categorizzati in 4 classi (verde, giallo, rosso, blu) trasformiamo questo problema in 4 problemi binari.

Per combinare le predizioni, valutiamo ogni singola ipotesi h_i, sperando che solamente una sia classificata come $+$. Per esempio se $h_1(x) = h_2(x) = h_3(x) = -$ e $h_3(x) = +$ allora il risultato sarà classe .
Gli svantaggi di questo approccio sono:

- se abbiamo anche solo una ipotesi h_i errata, allora la predizione risulta essere errata;

- impiega molto tempo dato che i dataset utilizzati in fase di addestramento sono molto grandi;
- si può incorrere in dataset sbilanciati: ad esempio se abbiamo 6 classi ciascuna con 100 immagini, ciascun classificatore avrà 100 esempi positivi e 500 negativi. Questo problema chiaramente può complicarsi in presenza di dataset già sbilanciati.

Nonostante tutto le performance di one-vs-all sono comparabili alle altre metodologie che andrò ad illustrare in seguito.

5.2.2 All-vs-all (AVA) - All-pairs approach

In questo approccio ogni classe è confrontata con ogni altra classe [63]. Un classificatore binario è costruito per discriminare tra ogni coppia di classi, scartando le rimanenti classi. Questo richiede di addestrare $\frac{K(K-1)}{2}$ classificatori binari, ciascuno addestrato per separare coppie di classi. In fase di decisione (testing nuovo esempio) vengono considerati gli esiti di ciascun classificatore come voti per una classe e viene scelta quella che ne ottiene la maggioranza.

Tabella 5.3: Esempio dell'approccio "All-pairs-approach"

Training set	vs	vs	vs	vs	vs	vs
X_1	X_1 $-$			X_1 $-$		X_1 $-$
X_2		X_2 $-$	X_2 $+$			X_2 $+$
X_3 $\Rightarrow$			X_3 $-$	X_3 $+$	X_3 $-$	
X_4	X_4 $-$			X_1 $-$		X_4 $-$
X_5	X_5 $+$	X_5 $+$			X_5 $+$	
	$\Downarrow$	$\Downarrow$	$\Downarrow$	$\Downarrow$	$\Downarrow$	$\Downarrow$
	h_1	h_2	h_3	h_4	h_5	h_6

Nel caso della strategia one-vs-one il tempo di addestramento può essere minore in quanto, nonostante si debbano addestrare più classificatori, i dataset usati sono di dimensioni di gran lunga minori rispetto a quelli usati nell'approccio one-vs-all.

Lo svantaggio di questo approccio è che non è ovvio come combinare le singole previsioni h_i. Risultati [64], mostrano che questo approccio è generalmente meglio dell'approccio precedente.

5.2.3 Error correcting output codes (ECOC)

Questo approccio incorpora l'idea dei codici a correzione di errore utilizzati nelle reti neurali (ANN) descritto nel paragrafo precedente. Funziona addestrando N classificatori binari per distinguere tra le differenti classi K. Per ogni classe viene creata una parola di codice (*codeword*) di lunghezza N secondo una matrice binaria M predefinita. Ogni riga di M corrisponde ad una certa classe.

La Tabella 5.4 mostra un esempio per k $= 5$ classi codewords di N $= 7$ bit. Ogni classe è data da una riga della matrice. Ogni colonna è usata per addestrare un classificatore binario differente.

Tabella 5.4: Rappresentazione della matrice di "codewords" M

M	1	2	3	4	5
■	+	−	+	−	+
■	−	−	+	+	+
■	+	+	−	−	−
■	+	+	+	+	−

Durante la fase di testing di un esempio non categorizzato, la *codeword* di output che esce dagli N classificatori, viene confrontato con le *codewords* K originali, e quello con la minima *distanza di Hamming* è considerata l'etichetta da assegnare alla classe per questo esempio.

Formalmente, dato x occorre scegliere la riga della matrice M che è più vicina a $\mathbf{h}(x) = \langle h_1(x), ..., h_5(x) \rangle$ Per esempio se l'output degli N classificatori è il seguente: $\mathbf{h}(x) = \langle -, +, +, +, - \rangle$ la predizione sarà classe ■.

Un esempio del funionamento è illustrato in Tabella 5.5.

Diettrich e Bakiri [65] hanno dimostrato che questa tecnica è molto più efficiente e affidabile rispetto alla due descritte precedentemente, in particolare mostra una miglior capacità di generalizzazione.

Diversi esperimenti dimostrano che il metodo OVA è quello che ha performance inferiori, ma in generale il metodo scelto dipende dal problema da affrontare scegliendo il metodo di volta in volta dopo avere effettuato test sul problema specifico.

Tabella 5.5: Esempio dell'approccio "Error correcting output codes"

Training set	1	2	3	4	5
X_1	X_1 $-$	X_1 $-$	X_1 $+$	X_1 $+$	X_1 $+$
X_2	X_2 $+$	X_2 $+$	X_2 $-$	X_2 $-$	X_2 $-$
X_3 $\Rightarrow$	X_3 $+$	X_3 $+$	X_3 $+$	X_3 $+$	X_3 $-$
X_4	X_4 $-$	X_4 $-$	X_4 $+$	X_4 $+$	X_4 $+$
X_5	X_5 $+$	X_5 $-$	X_5 $+$	X_5 $-$	X_5 $+$
	$\Downarrow$	$\Downarrow$	$\Downarrow$	$\Downarrow$	$\Downarrow$
	h_1	h_2	h_3	h_4	h_5

5.2.4 Metodo utilizzato

Come detto in precedenza ho scelto di utilizzare la libreria OpenCV, che al suo interno contiene un modulo con tutti gli algoritmi di Machine Learning maggiormente utilizzati: Support Vector Machine (SVM), AdaBoost, Artificial Neural Network (ANN), Random Forest. Come già discusso precedentemente, non tutti questi algoritmi permettono di poter gestire problemi di classificazione multiclasse.

Per esempio SVM in OpenCV è implementato sia come classificatore binario, sia nella sua variante che permette di gestire classificazione multiclasse. AdaBoost è implementata solo nella versione originale di classificatore binario. In questi articoli [66], [67] sono state proposte varianti per la gestione multiclasse.

In questo progetto di tesi per affrontare il problema multiclasse, ho scelto di utilizzare l'approccio (ECOC) perché dopo avere fatto vari esperimenti, ho constato esser il migliore metodo per la classificazione di immagini. È molto accurato e garantisce una percentuale di errore inferiore.

La libreria OpenCV si basa sulla libreria LibSVM (Open source Machine Learining Library sviluppati presso il Dipartimento di Computer Science della National Taiwan University da Chih-Chung Chang and Chih-Jen Lin) per l'implementazione dell'algoritmo di Support Vector Machine.

5.3 Dettagli implementativi del modello BoW

In questo capitolo illustrerò come ho realizzato il progetto basandomi sul modello Bag of Visual Words e per ogni fase dell'algoritmo descriverò nel dettaglio il funzionamento e le scelte fatte.

5.3.1 Estrazione e descrizione delle feature

In questa prima fase lo scopo è quello di creare un vocabolario visuale, quindi occorre estrarre *key-points* da ogni categoria di immagine. Per fare questo utilizziamo *feature extractor* che sono in grado di selezionare are specifiche dell'immagine (come corners, edge, blobs).

Terminata la fase di identificazione dei punti salienti la fase successiva è quella di descriverli in maniera robusta, questo processo avviene attraverso *feature descriptor*. Per far si che il descrittore sia adeguato al nostro progetto, occorre che rispetti determinate caratteristiche. Prima cosa il descrittore scelto deve essere stabile in presenza di rumore, secondo vogliamo che il descrittore abbia una semplice e giustificabile misure di distanza tra di loro, in quanto sarà utile in fase di costruzione del vocabolario visivo. Infine occorre che il descrittore abbia un numero di features grandi abbastanza da permettere diversità tra differenti patches e piccolo al punto giusto in modo tale che sia in grado di maneggiare le informazioni estratte in modo efficiente.

Come illustrato precedentemente nella introduzione, ho scelto di utilizzare SIFT, per la sua robustezza e affidabilità, dato che soddisfa tutte le caratteristiche elencate precedentemente.

5.3.2 Creazione del Vocabolario

La fase successiva è quella di creare il modello bag of words, che implica la creazione di un *codebook*, il vocabolario visuale discreto. Questa è la fase più importante del metodo BoW. Nel progetto che ho realizzato mi sono basato sull'approccio utilizzato da Zhang et al [68].

La rappresentazione *Visual Words* di un'immagine (noto anche come codebook), consiste nel rappresentare la distribuzione di patterns locali differenti,

all'interno di un immagine. I Keypoints sono estratti da immagini di training e poi raggruppati in un numero definito di cluster. Trattando ogni cluster come *Visual Word*, si ottiene una parola del vocabolario visual che descrive tutti i tipi di patterns locali all'interno delle immagini. L'immagine può quindi essere rappresentato come un vettore di features, contenente il numero di ogni*visual words* all'interno di un immagine.

Un vocabolario nel dominio della classificazione di oggetti/scene può essere calcolato seguendo due approcci:

- *annotazione*: il vocabolario è ottenuto assegnando etichette significative a porzioni dell'immagine (es. cielo, acqua, vegetazione, . . .);
- *data-driven*: i termini del vocabolario sono calcolati come centroidi ottenuti al termine di un processo di clustering delle feature.

In tecniche di creazione del codebook data-driven è necessario eseguire una quantizzazione di un elevato numero di vettori di features, rappresentati in genere in uno spazio a elevata dimensionalità; per fare questo si utilizzano tecniche di clustering che consentono di definire le parole visuali. Lo scopo in questa fase è quello di ottenere una rappresentazione compatta dello spazio delle features, questo avviene attraverso un algoritmo di quantizzazione vettoriale (algoritmo di clustering). L'approccio di quantizzazione comunemente usato è K-means (analizzato in dettagli nel Capitolo 3.11.2), vista la sua relativa semplicità e la velocità di convergenza. Questo ci permette di suddividere un insieme di oggetti in K gruppi sulla base dei loro attributi (features).

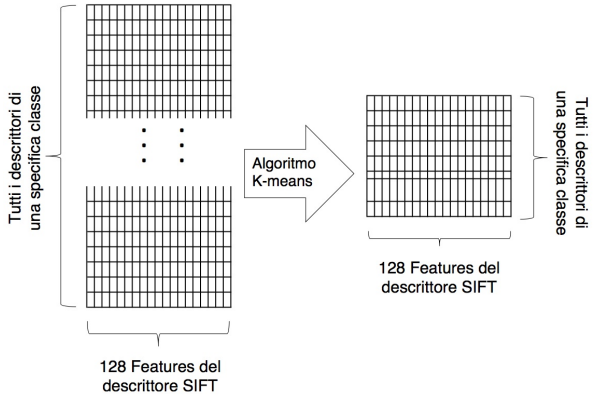

Figura 5.2: Creazione del vocabolario visuale per una classe

Scegliendo SIFT come feature detector, ritorna un vettore di Float con uno spazio di dimensione 128, illustrata in Figura 5.2.

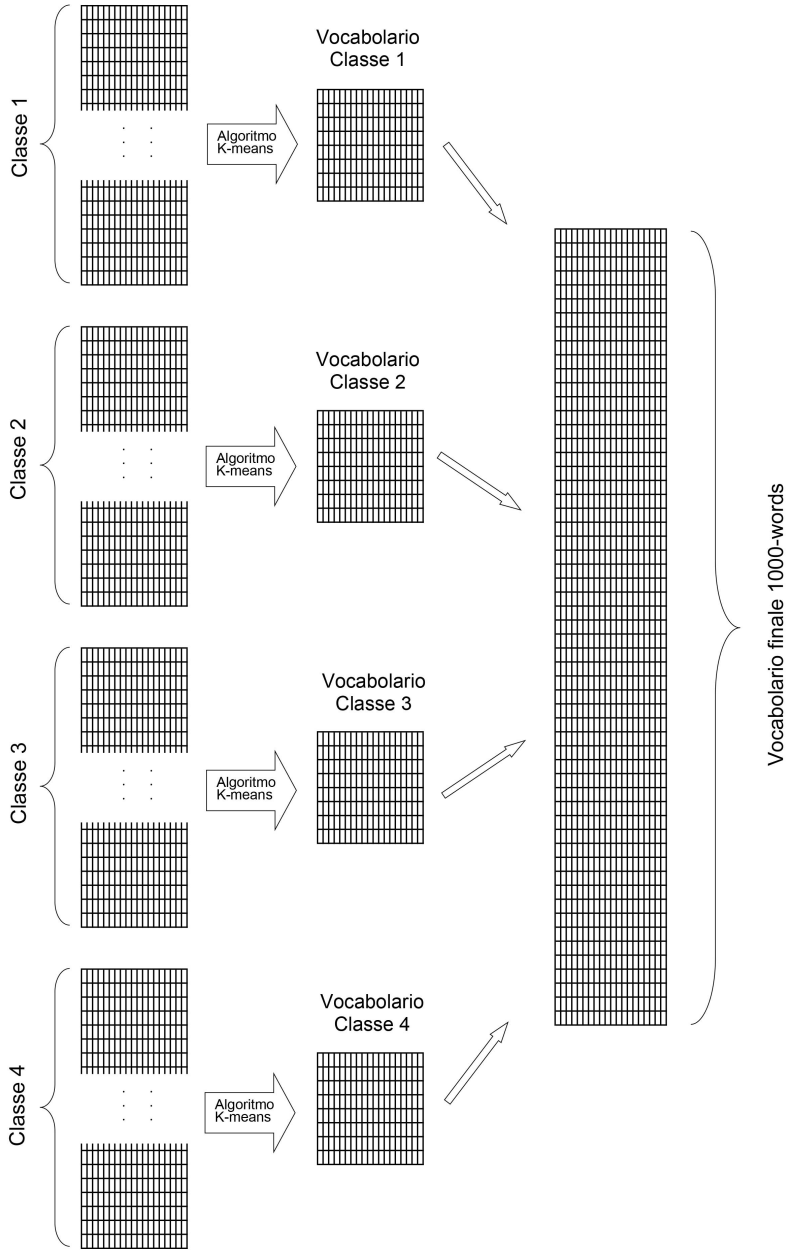

Figura 5.3: Esempio di creazione del Vocabolario visuale Finale (4 classi da 250 codewords per un totale di 1000)

Tutti i descrittori delle immagini di una specifica classe sono memorizzati in una matrice a 128 colonne; ognuna corrispondente a un SIFT feature di descrit-

tori, e questo, come mostrato nell'array di sinistra, ha un numero differente di righe che dipende dal numero di descrittori delle immagini di una specifica classe.

Questa operazione è fatta separatamente per ogni classe e al termine le classi finali vengono nuovamente fuse in un vocabolario finale. In Figura 5.3 si nota bene il funzionamento descritto, in cui a titolo esemplificativo, ho utilizzato 4 classi.

Le performance di questo metodo dipendono dal metodo di quantizzazione scelto e dal numero di parole visuali che compongono il codebook.

5.3.3 Descrizione immagine attraverso istogramma

Una volta che il vocabolario è stato creato, la fase successiva è quella di creare i descrittori delle immagini. Questi descrittori verranno creati usando il vocabolario costruito nello step precedente, attraverso clustering.

Il *descrittore dell'immagine* è un *istogramma* di dimensione pari alla cardinalità del *codebook*, in cui ogni elemento mostra la frequenza della parola visuale corrispondente all'interno dell'immagine, cioè quante volte un punto saliente è stato etichettato con il cluster associato alla parola (*istogramma di codewords*). Un esempio è rappresentato in Figura 5.4 e Figura 5.5.

Nel modello di tipo *bag of words* puro applicato a documenti testuali non si tiene conto dell'ordine in cui compaiono le parole: il documento viene descritto tramite il vettore delle occorrenze dei termini del dizionario all'interno di esso. Analogamente nel caso visuale ogni punto saliente individuato nell'immagine viene assegnato al centro del cluster ad esso piú vicino in base al metodo scelto in fase di creazione del codebook. Proprio grazie a questo processo non si parla piú di punti salienti, ma di *visual words*.

Un esempio di creazione di un vettore di parole visuali di questo tipo è presentato in Figura 5.6.

La scelta che ho fatto nella fase iniziale dell'esperimento, è stata quella di fissare la grandezza del codebook (dizionario) a 1000 codewords, successivamente ho provato ad aumentare la dimensione a 2000, 3000, 4000 e 5000. Come prevedibile, aumentando il numero della classi da riconoscere, occorre aumentare la cardinalià del dizionario.

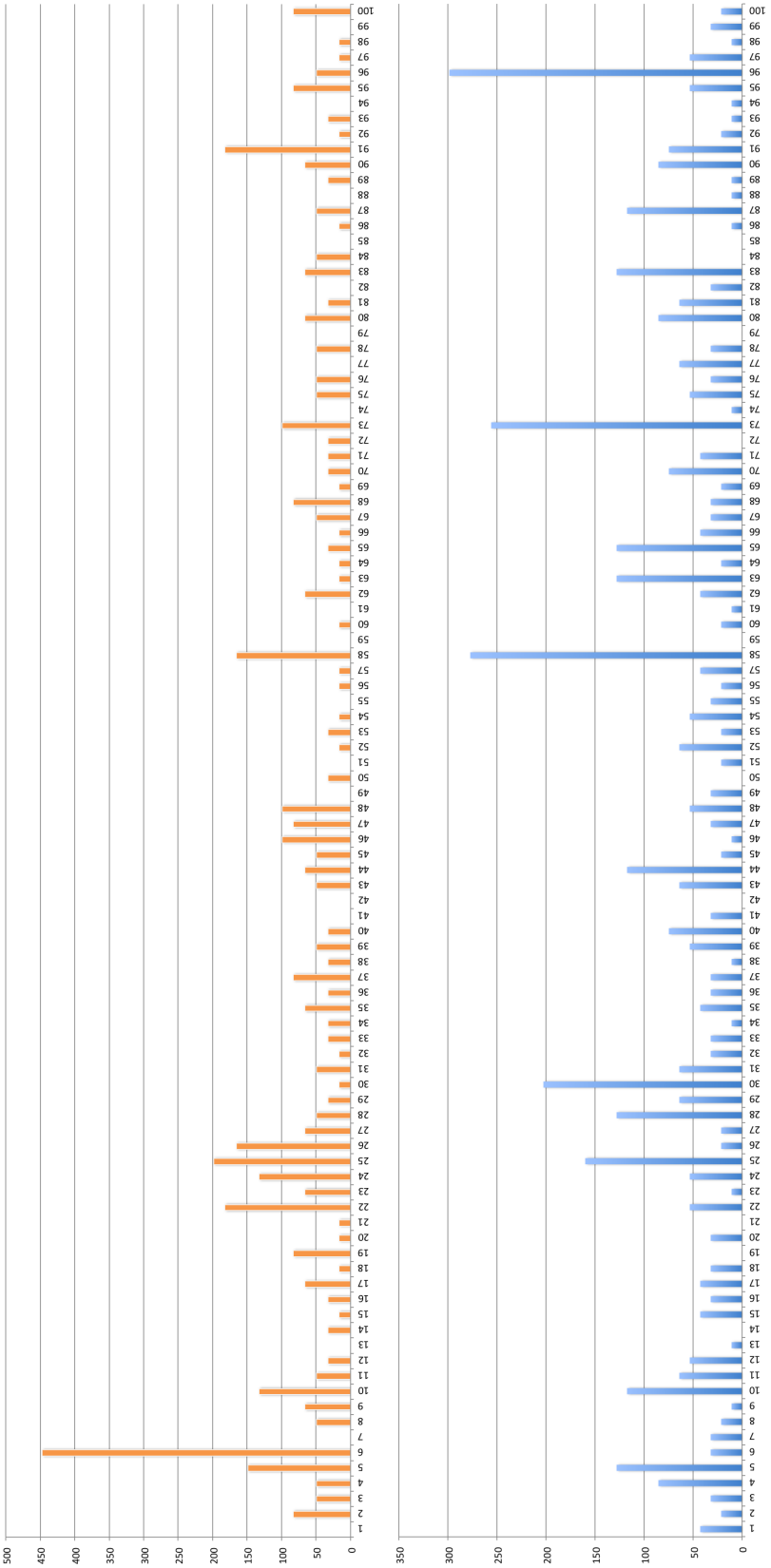

Figura 5.4: Esempio dell'istogramma della classe "airplanes" (in alto) e della classe "Faces_easy" (in basso) Dataset Caltech-101

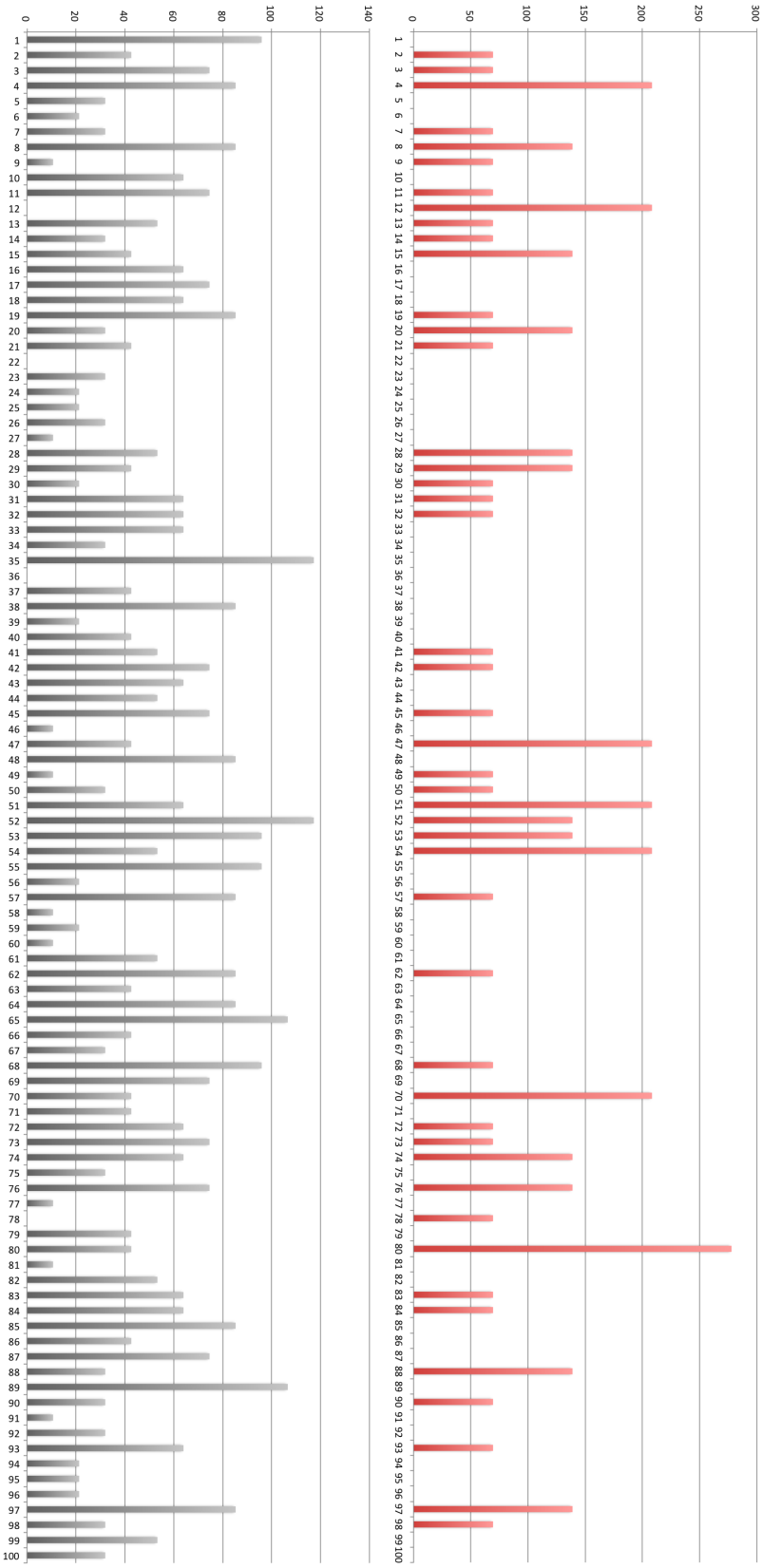

Figura 5.5: Esempio dell'istogramma della classe "Leopards" (in alto) e della classe "Motorbikes" (in basso) Dataset Caltech-101

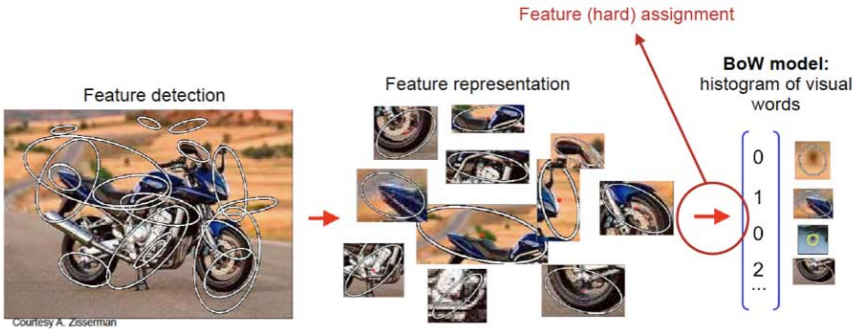

Figura 5.6: Identificazione di un punto saliente e creazione istogramma [30]

5.4 Descrizione del progetto

SIFT descriptors sono stati scelti per la loro robustezza ed invarianza a diverse trasformazioni geometriche e di luminosità. Quello che mi aspetto dalla realizzazione di questo progetto è che per ogni classe un certo numero di SIFT caratteristici vengano ritrovati in ogni altra immagine della stessa classe, indipendentemente da eventuali traslazioni, rotazioni, cambi di dimensione o luminosità.

Il progetto che ho creato per i miei esperimenti, si può suddividere in:

- Identificazione, estrazione e descrizione features;
- costruzione del vocabolario per ogni classe, attraverso l'algoritmo K-means;
- creazione dell'istogramma rappresentativo di ogni immagine;
- apprendimento e/o classificazione.

Questi quattro moduli comunicano tra di loro in modo unidirezionale.

L'aspetto interessante di questo approccio è che una volta addestrato il classificatore, e memorizzato in un file i parametri dell'algoritmo di classificazione, è possibile classificare in tempo reale un oggetto, senza necessità ogni volta di addestrare nuovamente il classificatore.

5.4.1 Algoritmi di classificazione

È possibile riassumere l'intero processo di classificazione di una immagine in due momenti distinti: *Learning* e *Testing*.

Questi due momenti sono molto simili, l'unica differenza è che mentre nella fase di *addestramento* utilizziamo un Training Set (di immagini) solitamente abbastanza grande di immagini già etichettate, nella fase di *testing* utilizziamo un immagine nuova, mai utilizzata prima. Se il classificatore è stato addestrato correttamente, dovrebbe essere in grado di individuare la classe di appartenenza dell'immagine nuova.

Learning

La procedura di *apprendimento* consiste nell'addestrare il classificatore, ad ogni istogramma elaborato nel modello BoW viene assegnata una etichetta che lo contraddistingue e questo caratterizza un singolo esempio di training. Occorre addestrare il classificatore per ogni singola classe di oggetti che vogliamo riconoscere.

Per quanto riguarda il Learning l'algoritmo prevede i seguenti passi:

1. creo il *Cocabolario*: $\forall$ classe i $-$ esima appartenente al Training set:
   - $\forall$ immagine appartenete alla classe i $-$ esima:
     - identifico keypoints SIFT;
     - estraggo i descrittori SIFT dell'immagine;
     - inserisco queste informazioni in un vettore;
   - per ogni classi i$-$esima, attraverso algoritmo *K-means* creo un cluster di descrittori SIFT con un numero fisso di elementi;
   - inserisco il clustering in un nuovo vettore che andrà a formare il vocabolario (codebook).

2. creo il *Training Data Set* e *Label Vector*:
   - leggo nuovamente ogni immagine appartenente al TrainingSet, estraggo *keypoints* e li confronto con il vocabolario, ottenendo in output un *istogramma* rappresentativo per ogni immagine;
   - associo ad ogni istogramma l'etichetta (Label) che contraddistingue la classe di appartenenza dell'immagine

3. addestro il classificatore utilizzando il Training Data Set e Label Vector

Testing

La fase di *classificazione* (o testing) è la stessa utilizzata per la fase di apprendimento, estraiamo keypoints e confrontiamo l'istogramma che rappresenta le codeword con tutti gli istogrammi che abbiamo per ogni categoria.

Per quanto riguarda la fase di Testing l'algoritmo prevede i seguenti passi:

1. per la singola immagine che vogliamo classificare:
   - estraggo i *keypoints* li confronto con il vocabolario, ottenendo in output *l'istogramma* rappresentativo dell'immagine di test;
2. interrogo il classificatore utilizzando l'istogramma estratto.

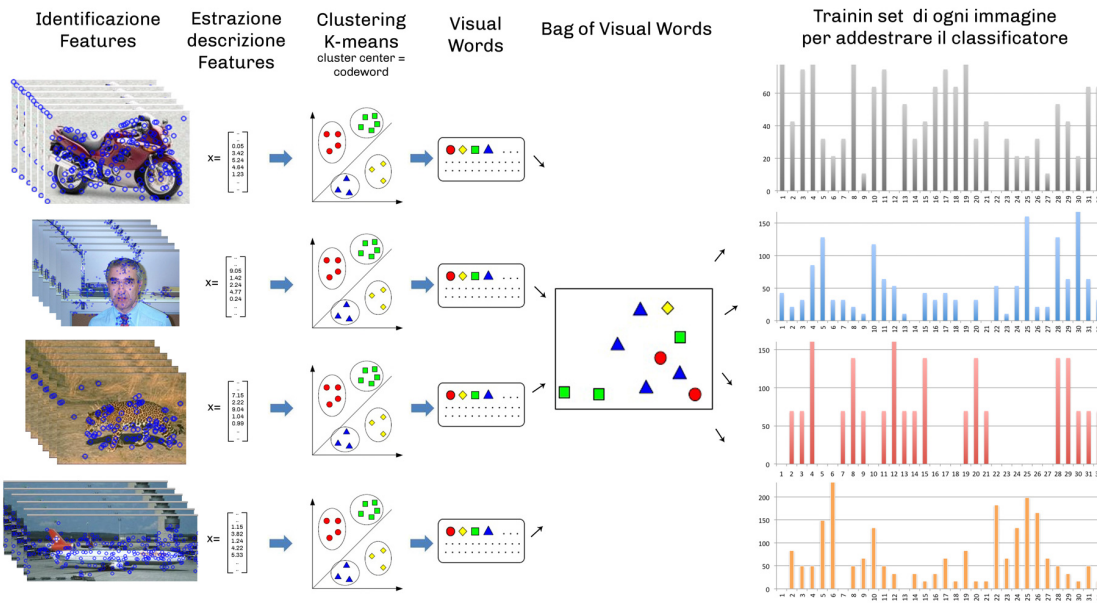

Figura 5.7: Pipeline di training utilizzando il modello Bag of Visual Words

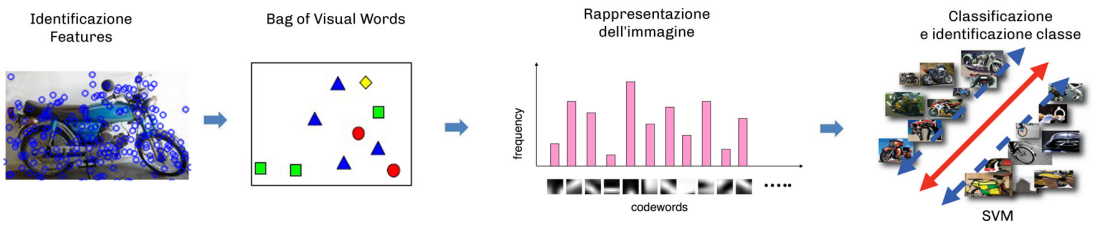

Figura 5.8: Pipeline di testing

5.4.2 Struttura del progetto

Il progetto è strutturato in tre classi principali:

- **Dictionary**: classe che permette di estrarre features/descrittori, creare il vocabolario (clustering), creare trainingDataSet e TestDataSet.
  Al suo interno contiene i seguenti metodi:
  - extractFeaturesAndCluster
  - clustering
  - setVocabularyBOWID
  - setDictionarySize
  - getDictionarySize
  - saveToFile
  - loadFromFile
  - createTrainData
  - createTestData
  - trasform2binaryTrainData
  - trasform2binaryECOC
- **Classifier**: classe che permette di classificare un immagine.
  Al suo interno contiene i seguenti metodi:
  - metodi per *classificatore SVM* : setSVM, trainSVM, getSupportVector, loadTrainedSVM
  - metodi per *Reti Neurali*: setANN, trainANN, loadTrainedANN
  - metodi per *classificatore Random Forest*: setRandForest, trainRandForest, testRandForest, loadRandForest
  - metodi per *classificatori binari*: trainBinaryClassifierECOC, testClassifierHamming
- **Tools**: classe con al suo interno diversi strumenti utili: lettura di file all'interno di una cartella, creazione elenco immagini training, estrazione nome della classe (dato il sui path) e molti altri.
  Al suo interno contiene i seguenti metodi:
  - createTrainingSet
  - createTestSet
  - createTrainingClasses
  - categoryToInt
  - pathToClassName
  - categoryToName
  - getTotclassNum
  - getTotalTrainingImages

5.4.3 Piattaforma di sviluppo utilizzata

Ho realizzato il progetto utilizzando le seguenti risorse:

- Apple MacBook Pro 15" Retina (Early 2013)
  - Processore Intel 2.4 GHz quad-Core i7 (I7-3635QM);
  - 8Gb DDR3L SDRAM (1600 MHz);
  - 256 GB SSD;
  - scheda grafica NVIDIA GeForce GT 650M 1GB.
- Xcode 5.0.2;
- Clang compilatore "LLVM nativo" per C/C++/Objective-C;
- OpenCV 2.4.7;
- Dataset vari per l'addestramento dei classificatori;
- Sterocamera sviluppata dal gruppo di ricerca del Dipartimento Ingegneria - (DISI) dell'Università di Bologna presso il quale ho sviluppato la tesi.

5.5 Classificatori utilizzati

In questo capitolo farò un breve analisi e confronto sui classificatori trattati nel capitolo 3, descriverò vantaggi e svantaggi di ognuno e per quale motivo ho scelto di utilizzarli nel progetto.

5.5.1 Normal Bayes Classifier

In *OpenCV* è implementato solamente il classificatore Bayesiano *Normal Bayes Classifier*. Come specificato nella documentazione le caratteristiche (features) sono distribuiti normalmente e non necessariamente indipendenti.

Questi sono i **vantaggi**:

- è facile da utilizzare, non ci sono molti parametri da decidere o da configurare;
- solitamente si ottengono buoni risultati nei casi reali, senza utilizzare modelli più complessi.

Questi sono gli **svantaggi**:

- la natura del classificatore assume che le features siano fortemente indipendenti tra di loro, questo porta ad a una perdita di accuratezza, infatti nei problemi reali spesso esiste dipendenza tra le variabili. Questo non è il caso del problema di classificazione di immagini, in quanto tutte le features hanno le stessa importanza.

Infine un ultimo problema che ho affrontato, utilizzando la versione implementa nella libreria OpenCV, è che è molto lento nella fase di addestramento. Per addestrare un classificatore Normal Bayes Classifier con un Training set formato da circa 200 righe 1000 codewords per riga ho impiegato circa 48 ore, sul mio calcolatore. Questo purtroppo non mi ha permesso di fare dei test approfonditi ed è per questo che ho scelto di non utilizzare questo modello.

5.5.2 K-Nearest Neighbour

K-Nearest Neighbour come illustrato esaustivamente nel Capitolo 3.5.2 è una algoritmo di tipo non parametrico che prende in considerazione solo le distanze tra gli oggetti nello spazio n-dimensionale, indipendentemente dalla loro distribuzione.

Questi sono i **vantaggi** che l'algoritmo offre:

- la fase di training è molto veloce, dato che non fa assolutamente nulla per comprendere i dati;
- è molto semplice da capire;
- è in grado d apprendere funzioni complesse.

Questi sono gli **svantaggi**:

- a tempo di classificazione, rispetto ad altri algoritmi è abbastanza lento;
- K-NN è molto sensibile ad attributi irrilevanti: in caso siano presenti nel training set features non importanti (immagini con all'interno caratteristiche non distintive) la predizione risulta errata;
- nel caso di problemi molto complicati è spesso preferibile adottare una tecnica che permette di creare un modello di classificazione più affidabile.

Dopo aver fato alcuni esperimenti, ho deciso di non utilizzare questo modello a causa dell'alta percentuale di errori e dei difetti mostrati.

5.5.3 Artificial Neural Network (ANN)

Neural Network è un algoritmo che cerca di emulare le reti neurali umane, solitamente è un ottimo classificatore ed è possibile ottenere risultati accurati. Tuttavia dipende da problema a problema, quindi è sempre consigliato fare dei test sul problema specifico.

Questi sono i **vantaggi** che offre il metodo ANN:

- si presta molto bene a quella tipologia di problemi in cui abbiamo molti dati di ingresso e di uscita da imparare, ma non abbiamo idea di quale sia la funzione di mapping corretta;
- funzionano bene anche con set di dati che sono rumorosi o dove alcuni ingressi hanno variabili mancanti;
- è in grado di risolvere problemi non lineari;
- molto veloce nella fase di predizione;
- non ha molti parametri da configurare, quindi è facile da utilizzare.

Mentre questi sono gli **svantaggi**:

- spesso converge su minimi locali anziché minimi globale, non sempre è in grado di modellare il problema correttamente;
- per sua natura è una "scatola nera" quindi la risposta che emerge dai pesi, può essere difficile da interpretare (può funzionare, ma non sappiamo come);
- abbastanza lento nella fase di training: richiede molte risorse;
- quando la fase di apprendimento diventa lunga, significa che soffre di overfitting: per un dato pattern ANN inizia a considerare il rumore come parte del pattern stesso;
- vulnerabile a valori anomali, dato che si basa sulla somma dell'errore quadratico: per risolvere il problema occorre rimuovere dal training set i valori errati.

Ho scelto di utilizzare questo metodo perchè per la sua natura si adatta molto bene a risolvere problemi multi-classe, come illustrato nel Capitolo 5.2. È ampiamente utilizzato sia nel riconoscimento vocale, sia per il riconoscimento ORC di caratteri o cifre e questo dimostra la sua affidabilià.

Considerando i vantaggi menzionati, posso dedurre che si presta bene per il problema di classificazioni di immagini, ovviamente questa ipotesi è basata su assunzioni teoriche. Ogni problema è diverso quindi occorre fare dei test sul campo per ogni specifico problema.

5.5.4 Support Vector Machines

SVM può modellare complessi problemi del mondo reale, come testo e classificazione delle immagini, riconoscimento della scrittura a mano, e la bioinformatica e analisi biosequence.

SVM offrono diversi **vantaggi**:

- abilità di generalizzare, dato che lo scopo di SVM è quello di massimizzare il margine di separazione tra gli iperpiani;
- sono presenti molti parametri di configurazione, che permettono all'utente di "pensare" ed evitare overfitting;
- attraverso l'utilizzo del kernel, funzionano bene anche se i dati non sono linearmente separabili nello spazio di feature di base; riesce a risolvere qualsiasi tipo di problema;
- è definito come un problema di programmazione quadratica (senza minimi locali), quindi riesce a convergere all'ottimo globale (soluzione ottima);
- si comporta molto bene su insiemi di dati che hanno molti attributi, anche in fase di training sono presenti pochi esempi;
- non esiste un limite superiore al numero di attributi, gli unici vincoli sono quelli imposti da hardware. Reti neurali tradizionali non funzionano bene in queste circostanze;
- è possibile configurare il parametro C per controllare la soglia di valori errati (presenti nel training set) che riesce a sopportare.

Gli **svantaggi** sono i seguenti:

- raramente capita che i modelli a kernel siano sensibili a overfitting e risulta essere complicato trovare il modello corretto;
- a volte il training può essere oneroso in termini di tempo;
- il training non è incrementale (arriva un nuovo oggetto occorre rifare da capo l'addestramento).

Ho scelto di utilizzare questo classificatore perchè uno dei classificatori attualmente più utilizzati. Per il task di classificazione di immagini è molto affidabile e permette di raggiungere un'accuratezza più elevata.

5.5.5 AdaBoost

AdaBoost è un algoritmo di classificazione potente, che ha goduto di un ampio successo. Attualmente viene utilizzato in molte applicazioni commerciali e nei seguenti settori: Biologia, Computer Vision, e l'elaborazione vocale.

I **vantaggi** offerti dal metodo sono:

- è facile da utilizzare, non ci sono molto parametri da decidere o da configurare;
- meno sensibile al problema sovradattamento rispetto alla maggior parte degli algoritmi di apprendimento;
- possibilità di imparare da un piccolo insieme di esempi e di aggiungere in modo incrementale nuove informazioni in fase di esecuzione.

Questi sono gli **svantaggi**:

- AdaBoost può essere sensibile ai dati rumorosi e valori anomali;
- algoritmo per sua natura binario, quindi non facile da adattare a problemi multi-classe.

Per i vantaggi elencati sopra, ho deciso di scegliere questo algoritmo da utilizzare nel lavoro di tesi. Essendo un algoritmo di classificazione binario, ho utilizzato le tecniche illustrate nel capitolo 5.2.3.

5.5.6 Random Forest

Ho scelto di *utilizzare* Random Forest dato che gestisce in modo naturale i problemi multilasse. È inoltre uno dei classificatori più utilizzati negli ultimi anni in ambito Computer Vision, quindi merita di essere valutato in questo progetto sulla classificazione di immagini,

I **vantaggi** di Random Forest sono:

- molto veloce a run-time;
- si tratta di uno dei più accurati algoritmi di apprendimento disponibili, anche in caso di dati sbilanciati;
- funziona in modo efficiente su database di grandi dimensioni, producendo classificazione molto accurate;
- è in grado di gestire migliaia di variabili di input senza cancellazione variabile;
- si presta molto bene a essere parallelizzato su architettura specifiche (CUDA o multicore);
- fornisce stime di quali variabili sono importanti nella classificazione;
- ha un metodo efficace per la stima dei dati mancanti e nonostante tutto mantiene un'accuratezza elevata anche quando una grande proporzione dei dati sono mancanti.

Questi sono gli **svantaggi**:

- in alcuni compiti di classificazione in presenza di rumore, Random Forest può cadere in overfitting;
- quando viene utilizzato per la regressione non riesce a prevedere correttamente, oltre l'ambito di dati utilizzati in fase di addestramento (non è in grado di generalizzare).

5.5.7 Conclusioni

Riassumendo ho scelto di utilizzare Artificial Neural Network, Suppor Vector Machines, AdaBoost, e Random Forest.

Come enunciato nel teorema **No Free Lunch theorem (Wolpert 1996)**: non esiste un metodo di apprendimento automatico "perfetto". Per un problema in cui un certo algoritmo funziona bene, c'è un altro problema per il quale lo stesso metodo fallirebbe orribilmente. Le carenze di questo algoritmo possono essere risolte con altri metodi. Questo dovrebbe essere sempre considerato, quando si utilizzano algoritmi di Machine Learning.

Concludendo quindi occorre fare esperimenti prima di trarre conclusioni, in ogni modo mi aspetto di trovare un algoritmo che sia più adatto e accurato per il problema identificare classi di oggetti all'interno di immagini.

5.6 Utilizzo del classificatore su stereocamera

Questo progetto di tesi è stato ideato per essere installato ed utilizzato su un prototipo di stereocamera realizzata presso il DISI dal gruppo di ricerca del Professor Stefano Mattoccia.

Il funzionamento è il seguente:

- la stereocamera permette di generare dati 3D;
- segmentazione ed estrazione del piano attraverso una libreria creata da Giacomo Parmigiani suo progetto di tesi [69];
- ho creato una una funzione apposita che agisce sulla mappa di disparità (della sterecamera), viene configurato un parametro che permette di segmentare singoli oggetti;
- utilizzando il classificatore (sviluppato in questo progetto), precedentemente addestrato su un ampio training set, è possibile identificare oggetti in tempo reale.

Per mancanza di tempo, non ho descritto, in questo elaborato di tesi, gli esperimenti effettuati utilizzando questa configurazione.

Il descrittore e rivelatore di keypoints SIFT, ben noto per la sua efficacia è utilizzato in molteplici applicazioni, ma richiede uno sforzo computazionale intensivo, specialmente in sistemi real-time, o in dispositivi a bassa potenza come sistemi embedded (stereocamera). Questo ha portato a incrementare la ricerca per identificare e sostituire SIFT, con descrittori più semplici e con esigenze di calcolo inferiori.

Questa tendenza è iniziata con SURF, poi si è cercato di accelerare il calcolo di SIFT, implementando l'algoritmo direttamente su GPU. Recentemente nel 2011 e 2012 la ricerca si è concentrata su descrittori binari, come BRISK [40], FREAK [46], ORB [47] che permettono di alleggerire ulteriormente il carico computazione.

Da qui è nata l'idea di utilizzare un descrittore "leggero" anche sulla sterocamera (per esempio BRISK) il quale permette di estrarre keypoints in molto veloce, ottenendo risultati molti simili a SIFT. Inoltre BRISK non essendo vincolato da copyright è possibile utilizzarlo anche in applicazioni commerciali.

Attraverso un progetto realizzato da alcuni studenti attualmente è possibile estrarre keypoints e descrittori BRISK direttamente dalla stereocamera, ma occorre effettuare delle modifiche al modello descritto precedentemente, in quanto il modello BoW originale, non può essere utilizzato con descrittori binari.

5.7 Utilizzo di descrittori binari (BRISK) nel modello BoW

Il punto centrale su cui si basa il modello Bag of Visual Words è quello di creare il *vocabolario* e successivamente l'istogramma delle features. Come descritto precedentemente l'approccio classico è quello di impiegare k-means con distanza Euclidea tra i vettori di feature. Questo è stato dimostrato essere efficace, anche se computazionalmente impegnative durante la fase di addestramento.

L'algoritmo raggruppa i vari attributi presenti nel vettore SIFT e crea in output un nuovo vettore (ancora SIFT), ma con un numero predefinito di elementi. SIFT essendo composto da vettore di 128 numeri reali, si adatta molto bene al modello BoW in quanto l'algoritmo di clustering K-means utilizzando la distanza Euclide per raggruppare gli attributi.

Un *descrittore binario* (per esempio: BRISK o ORB), come illustrato nel Capitolo 2.4, è formato da una stringa bit. L'algoritmo K-means, utilizzando la distanza Euclidea, funziona correttamente con descrittori come SIFT o SURF, formato da valori numeri (interi o reali), ma non è in grado di gestire la stringa di natura binaria create dal descrittore binario. Pertanto occorre trovare una soluzione questo problema.

In seguito descriverò due soluzioni: una prima soluzione di adattamento del modello BoW, che si ottiene semplicemente modificando alcune linee di codice, convertendo il tipo di dati contenuto nel dizionario, . Mentre una seconda soluzione basata su fondamenti matematici, proposta nell'articolo "A Fast Approach for Integrating ORB Descriptors in the Bag of Words Model' da Grana, Borghesani, Cucchiara [31].

5.7.1 Prima soluzione

Il dizionario (o vocabolario) è composto da un vettore di keypoints. Il tipo di dato corrispondente ad ogni elemento del vettore dipende dal Features Descritptor scelto. Se la scelta fosse ricaduta su SIFT avremo un vettore composto da valori di tipo float. Per risolvere il problema una possibile soluzione, consiste in tre semplice modifiche:

1. Dopo aver estratto tutti i keypoints e i descrittori di ogni immagine, prima di creare clustering per ogni classe (features "non clusterizzate"), occorre convertire il tipo degli elementi in *CV_32FC1* che corrisponde al tipo di dato *Float* in OpenCV. Questo avviene nel modo seguente:

   ```
   Mat descr;
   featuresClass.convertTo(descr, CV_32FC1);
   classVocabulary = clustering( ... );
   ```

2. Convertire il tipo degli elementi contenuti nel vettore vocabolario, in elementi di tipo *CV_8UC1* che corrisponde al tipo di dato *Char* in OpenCV.

   ```
   cv::Mat udictionary;
   vocabularyBuilt.convertTo(udictionary,CV_8UC1);
   bowIDE->setVocabulary(udictionary);
   ```

3. Infine nella fase di matching non viene utilizzato il *Feature Matching* di tipo *FLANN (Fast Approximate Nearest Neighbor Search Library)* previsto dal modello classico Bag of Visual Words, ma viene utilizzato il *Matching* di tipo *Brute-Force Matching* che utilizzata la distanza di Hamming, di conseguenza funziona correttamente con i descrittori binari.

   Normale modello BoW:

   ```
   matcher = new FlannBasedMatcher;
   ```

   Nuova versione:

   ```
   matcher = new BFMatcher (cv::NORM_HAMMING);
   ```

5.7.2 Seconda soluzione

Non potendo usare la distanza Euclidea, questa soluzione prevede di utilizzare una variante dell'algoritmo k-means. Per poter ricavare la distanza tra i vettori binari viene utilizzata la distanza di Hamming. Questa permette di individuare il numero di bit diversi in posizioni corrispondenti, ma il problema rimane, infatti la Distanza di Hamming non restituisce la media (centro di ogni cluster).

Per determinare per ogni elemento il centro, viene utilizzata la regola della maggioranza, con i legami (ties) spezzati a caso. Questa variante dell'algoritmo Lloyd, viene chiamata da *Grana, Borghesani, Cucchiara* **algoritmo k-maggioranza** e viene presentata in seguito.

Algorithm 2 K-majority algorithm

1: Given a collection D of binary vectors
2: Randomly generate k binary centroids C
3: **while** centroids not changed **do**
4: **for** $d \in D$ **do** ▷ Assign data to centroids
5: $c_d \leftarrow \arg\min_{c \in C} \mathit{HammingDistance}(c, d)$
6: **end for**
7: **for** $c \in C$ **do** ▷ Majority voting
8: **for** $d \in D | c_d = c$ **do**
9: v accumulates d votes
10: **end for**
11: $c' \leftarrow \mathit{Majority}(v)$
12: **end for**
13: **end while**

Figura 5.9: Algoritmo k-maggioranza [31]

Per ulteriori dettagli si rimanda all'articolo originale: "A Fast Approach for Integrating ORB Descriptors in the Bag of Words Model" [31].

Purtroppo non sono stato in grado di implementare questo algoritmo, per manca di tempo, quindi resta una soluzione possibile, ma che non ho valutato su questo lavoro di testi. Sicuramente sarà possibile farlo in futuro.

Capitolo 6

Analisi dei risultati e confronto

In questo capitolo vengono presentati e discussi i risultati sperimentali ottenuti con le tecniche illustrate nei capitoli precedenti e ne viene fatto un confronto.

L'analisi di questo progetto si basa su due tipi di valutazioni: prima occorre analizzare le prestazioni dei classificatori scelti, e identificare quello più adatto alla classificazione di immagini. Verrà valutato in base a diversi parametri: accuratezza nella classificazione, velocità di apprendimento e velocità di esecuzione. Come detto precedentemente, ho scelto di utilizzare SIFT come descrittore di riferimento per tutti gli esperimenti. Scelto il classificatore ideale, analizzerò attraverso esperimenti l'affidabilità del modello Bag of Visual Words e come si comporta utilizzando diversi Dataset con all'interno un numero variabile di classi.

Nel Paragrafo 6.1 descriverò gli strumenti utilizzati per valutare le prestazioni dei classificatori. Nel Paragrafo 6.2 è fatto un confronto sui quattro tipi di classificatori che ho scelto di utilizzare: Support Vector Machines, Artificial Neural Network, AdaBoost e Random Forest; mentre nel Paragrafo 6.3 vengono mostrati i test che ho effettuato per valutare le prestazioni del modello BoW. Infine nel Paragrafo 6.4 sono mostrati esperimenti in cui ho utilizzato il descrittore binario ORB, e SURF e l'ho confrontato con l'accuratezza utilizzando SIFT.

6.1 Definizioni delle misure utilizzate

Per analizzare e valutare l'applicazione ho utilizzato le metriche e i parametri solitamente usati in Machine Learning per analizzare problemi di classificazione.

In seguito descriverò brevemente il loro il significato basandomi su quanto proposto nel seguente articolo "A systematic analysis of performance measures for classification tasks" [70].

6.1.1 Prestazioni

Se si considera un problema di classificazione binario (a 2 classi), sono possibili quattro risultati a seconda del valore dell'etichetta assegnata all'esempio in esame:

- **True positive** - **TP**: numero degli elementi correttamente classificati come appartenenti alla classe e appartengono realmente a quella classe;
- **True negative** - **TN**: esempi riconosciuti correttamente come negativi, ma che non appartengono alla classe in esame;
- **False negative** - **FN**: esempi che sono stati etichettati erroneamente come come negativi, ma in realtà sono positivi;
- **False positive** - **FP**: esempi che sono stati etichettati erroneamente come appartenenti a quella classe (positiva), ma che in realtà sono negativi.

Nel caso di classificazione muticlasse, i quattro parametri menzionati sopra vengo modificati nel seguente modo: TP_i, FN_i, TN_i, FP_i. e fanno riferimento ad ogni singola classe C_i.

Questi quattro parametri costituiscono la *Matrice di Confusione*, in tabella 6.1 è rappresentato un esempio che aiuta a chiarire l'idea.

Matrice di confusione

Al termine del processo di classificazione, il risultato ottenuto può essere valutato attraverso l'analisi di una tabella, detta *matrice di confusione* o *matrice di classificazione*. In questa tabella sono visibili gli oggetti realmente appartenenti a ciascuna classe (classe vera) e gli oggetti assegnati a ciascuna classe dal modello (classe assegnata). Questa matrice ha un ruolo molto importante, specialmente nel caso di classificazione muticlasse e rappresenta lo strumento che ho utilizzato maggiormente per mostrare i risultati degli esperimenti nel mio progetto.

Tabella 6.1: Esempio Matrice di Confusione nel caso multiclasse (cat, dogs, rabbit)

		Predict Class			
	Classi	Cat	Dog	Rabbit	Totale
Actual Class	Cat	**5**	3	0	8
	Dog	2	**3**	1	6
	Rabbit	0	2	**11**	13
	Totale	7	8	12	27

Da questa matrice si deducono le percentuali di esempi classificati correttamente **(veri positivi)** totali e all'interno di ogni classe (la diagonale - in grassetto). Inoltre (sottraendo l'elemento$_{ij}$) si deducono molto facilmente i **falsi negativi**, attraverso la somma dell'i-esima riga della classe; mentre attraverso la somma della j-esima colonna della classe ricaviamo i **falsi positivi** (sottraendo l'elemento$_{ij}$). In questo modo è possibile valutare gli eventuali errori di assegnazione degli oggetti alle varie classi. In tabella 6.2 notiamo la rappresentazione finale per la singola classe "cat":

Tabella 6.2: Esempio della Matrice finale riassuntiva per classe C_i

	Classificati come positivi	Classificati come negativi
Positivi	5 True Positive (TP) (gatti correttamente classificati come gatti)	3 False Negative (FN) (gatti che sono stati etichettati erroneamente come cani)
Negativi	2 False Positive (FP) (cani erroneamente classificati come gatti)	17 True Negative (TN) (tutti i rimanenti animali, correttamente classificati come non-gatti)

In seguito definirò i parametri utilizzati nel caso generale, spiegando per ognuno il significato e la formula. Sucessivamente presenterò le formule per il caso muticlasse, che è il caso utilizzato in questo progetto.

Precision e **Recall** sono due metriche molto importanti, utilizzate nelle applicazioni in cui la corretta classificazione dei record della classe positiva riveste una maggiore importanza. La *Recall* misura la capacità di individuare tutti i record positivi mentre la *Precision* misura la capacità di individuare soltanto quelli positivi. Naturalmente quello che si vorrebbe è massimizzarle entrambe.

- **Recall** (*Sensitivity* o *Sensibilità)*: indica la frazione dei record positivi classificati correttamente. Nota anche come TPR *(True Positive Rate)*. Può essere vista come la capacità del classificatore di identificare record correttamente. Se un classificatore ad esempio ha sensibilità pari al 100% allora riconosce tutti gli esempi in modo corretto. Un elevato *recall* inoltre indica che i falsi negativi sono limitati.

$$\text{TPR} = \frac{\text{TP}}{(\text{TP} + \text{FN})}$$

- **Precision** (*o precisione*): indica la frazione dei *True Positive* rispetto a tutti i risultati positivi. Maggiore è la precisione, meno sono i falsi positivi commessi dal modello. È una misura della precisione rispetto a una classe specifica. Essa è definita da:

$$\text{Precision} = \frac{\text{TP}}{(\text{TP} + \text{FP})}$$

- **Accuracy** (*o accuratezza*): indica la percentuale di istanze sia positive che negative classificate correttamente (correttezza generale del modello). Numero di campioni correttamente classificati rispetto al numero totale di campioni classificati.

$$\text{Accuracy} = \frac{(\text{TP} + \text{TN})}{(\text{TP} + \text{TN} + \text{FP} + \text{FN})}$$

- **Specificty** (*Specificità*): Nota anche come TNR *(True Negative Rate)*. Indica la frazione dei record negativi classificati correttamente come negativi. Esprime la capacità del classificatore di identificare risultati negativi. Se un test ha un'elevata specificità allora un risultato positivo dal test indica elevata probabilità che l'esempio sia positivo.

$$\text{TNR} = \frac{\text{TN}}{(\text{TN} + \text{FP})}$$

- **F-SCORE** : rappresenta una media armonica tra *precisione* e *recall*. Se la media armonica è elevata significa che non si sono verificati nè falsi negativi nè falsi positivi. È uguale a:

$$\text{F} - \text{SCORE} = \frac{2 \cdot \text{Recall} \cdot \text{Precision}}{(\text{Recall} + \text{Precision}}.$$

- **Error rate** rappresenta la percentuale di errore totale, ed è:

$$\text{Error rate} = \frac{(\text{FP} + \text{FN})}{(\text{TP} + \text{TN} + \text{FP} + \text{FN})}$$

Nel caso della classificazione di un problema **multiclasse**, come nel caso affrontato in questo progetto, i parametri sono molto simili. Vengono calcolati individualmente per **una singola classe i-esima** $C_i$nel seguente modo:

$$\textbf{Recall}_\textbf{i} = \frac{TP_i}{(TP_i + FN_i)}$$

$$\textbf{Precision}_\textbf{i} = \frac{TP_i}{(TP_i + FP_i)}$$

$$\textbf{Accuracy}_\textbf{i} = \frac{TP_i + TN_i}{TP_i + TN_i + FP_i + FN_i +}$$

$$\textbf{Errorrate}_\textbf{i} = 1 - Accuracy_i$$

Tramite le seguenti formule riassuntive di *Macro*, è possibile misurare **l'intero modello nella sua complessità** [70].

$$\textbf{Recall}_\boldsymbol{\mu} = \frac{\sum_{i=1}(TP)_i}{\sum_{i=1}((TP)_i + (FN)_i)}$$

$$\textbf{Precision}_\boldsymbol{\mu} = \frac{\sum_{i=1}(TP)_i}{\sum_{i=1}((TP)_i + (FP)_i)}$$

$$\textbf{AverageAccuracy}_\boldsymbol{\mu} = \frac{\sum_{i=1}((TP)_i + (TN)_i)}{\sum_{i=1}((TP)_i + (TN)_i + (FP)_i + (FN)_i)}$$

Successivamente l'analisi delle prestazioni del modello sarà effettuata tramite le sopra citate misure, in modo da poter offrire uno strumento di comparazione anche con altri lavori. In questo progetto occorre mettere in risalto le immagini classificate correttamente, quindi i dati positivi rivestono un ruolo fondamentale. Per ogni singola classe "i" evidenzierò : $\textbf{Recall}_\textbf{i}$, $\textbf{Precision}_\textbf{i}$. Mentre il parametro globale che ho utilizzato per valutare il modello è: $\textbf{Precision}_\boldsymbol{\mu}$.

Curve ROC

Come strumento aggiuntivo per la valutazione dei classificatori binari, è possibile utilizzare le curve ROC (Receiver Operating Characteristics). Sono degli schemi grafici in cui lungo i due assi si possono rappresentare la *True Positive Rate* (sensibilità) e sull'ascissa *False Positive Rate* (1-specificità).

In altre parole, si studiano i rapporti fra *falsi allarmi* (false rate) e *allarmi veri* (hit rate). Per esempio, in un classificatore a soglia, si calcola la frazione di veri positivi e quella di falsi positivi per ogni possibile valore della soglia; tutti i punti così ottenuti nello spazio FP-TP descrivono la curva ROC.

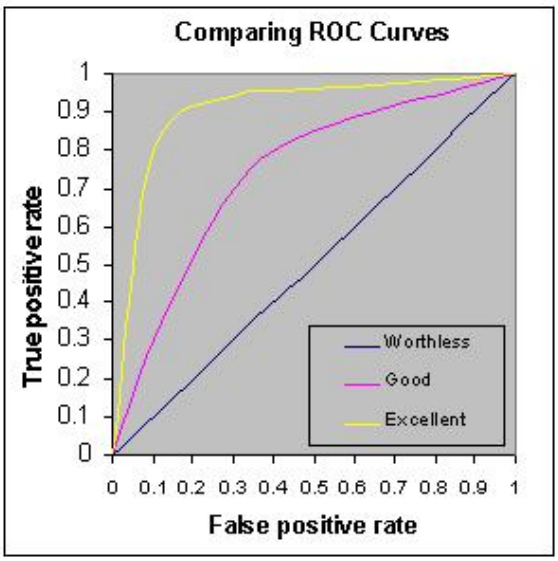

Figura 6.1: Rappresentazione della curva Roc [32]

Per la valutazione dei classificatori multi-classe, solitamente non viene utilizzato questo grafico. Si potrebbe confrontare singolarmente coppie di classi, ma questo risulterebbe poco interessante nel caso specifico del problema che sto analizzando. Sono state proposte varianti per poter rappresentazione curva Roc per problemi muticlasse [71], ma risultano computazionalmente dispendiose, in quanto i dati vengono rappresentate in più dimensioni.

Per le motivazione elencate precedentemente, ho scelto di non utilizzare le *Curve Roc* in questo progetto.

6.1.2 Modalità di test di un classificatore

Disponendo di un *data set* di cui è noto, per ogni tupla di dati, il valore della *class label*, si può dividere questo *Dataset* in modo che una parte venga utilizzata come *Training Set* nella fase di Learning e l'altra parte venga invece utilizzata come *Test Set* per verificare l'accuratezza del classificatore. Utilizzare gli stessi dati sia nella fase di apprendimento che nella fase di verifica della performance di un classificatore è molto pericoloso, perché si rischia l'**overfitting**, cioè di avere stime troppo ottimistiche riguardante le prestazioni a livello di nuovi casi.

Esistono diversi metodi per determinare l'accuratezza di un classificatore, di seguito descriverò brevemente tre metodi molto utilizzati:

Cross-validation - k-fold

Con questo metodo il *Dataset* viene diviso, in modo casuale, in k-folds, cioè in k sottoinsiemi che in maniera esclusiva vengono utilizzati come *Training Set* e *Test Set*.

- **primo passo:** dividere i dati in k sottoinsiemi di pari cardinalità;
- **secondo passo:** usare un sottoinsieme alla volta per il test, ed il resto per il training.

Il ciclo viene quindi ripetuto k volte.

L'accuratezza complessiva viene ottenuta sommando il numero dei casi correttamente classificati nelle k iterazioni e dividendo questa somma per il numero dei Round. Le varie stime dell'errore sono mediate per produrre una stima dell'errore globale.

Per esempio: se viene fatta una cross-validation a 10 folds, il Dataset viene diviso in 10 parti; 9 parti vengono usate come *Training Set* ed 1 parte come *Test Set* e tutto questo viene ripetuto per 10 volte con un fold diverso ogni volta. In Figura 6.2 un esempio aiuta a chiarire l'idea.

Questa è la tecnica che ho utilizzato in questo progetto per valutare le prestazione dei classificatori scelti e decidere l'algoritmo più idoneo al problema di classificazione di immagini con modello Bag of Visual Words.

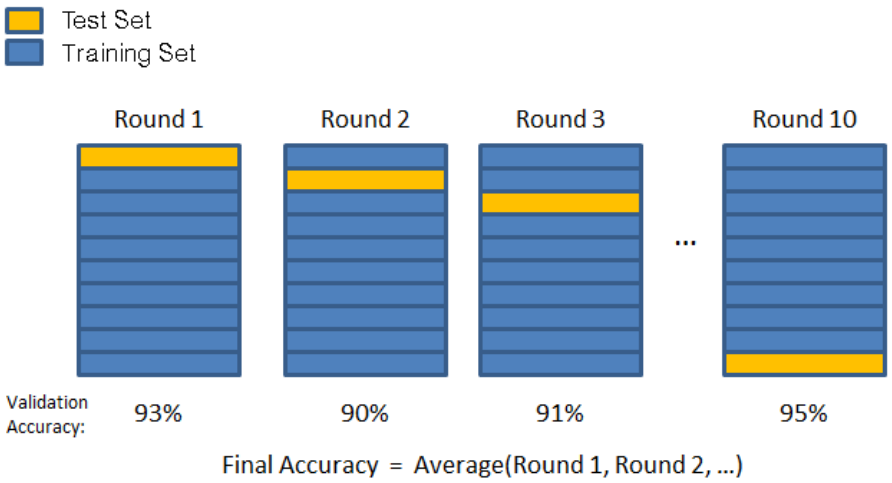

Figura 6.2: Rappresentazione di 10fold cross validation

Dataset train e validation (two folder validation)

Consiste nell'utilizzare un set di addestramento (training-set) costituito da campioni dei quali si conosce a priori la classe di appartenenza, curandosi del fatto che tale insieme sia *significativo e completo*, cioè con un numero sufficiente di campioni rappresentativi di tutte le classi.

Per la verifica del metodo di riconoscimento ci si avvale di un set (validation-set), anch'esso costituito da campioni la cui classe è nota, usato per controllare la generalizzazione dei risultati. È costituito da un insieme di campioni diversi rispetto a quelli del training-set. Volendo esaminare l'accuratezza del metodo su esempi reali, si fornisce al classificatore un set di test (detto "testing-set") le cui classi non sono note. In seguito le predizioni sulle classi di questo insieme vengono utilizzate, dopo la classificazione, per determinare gli errori e quindi l'accuratezza reale del classificatore.

Hold Out o split

In questo caso il *Dataset* iniziale viene di solito diviso in modo tale che $2/3$ di esso vengano usati come *Training Set* ed il resto come *Test Set*; la stima che si ottiene è di solito pessimistica in quanto viene utilizzata una porzione dei dati fissa per ottenere il classificatore. Per esempio si può utilizzare il 66% del data set come *Training set* ed il restante 34% come *Test Set*.

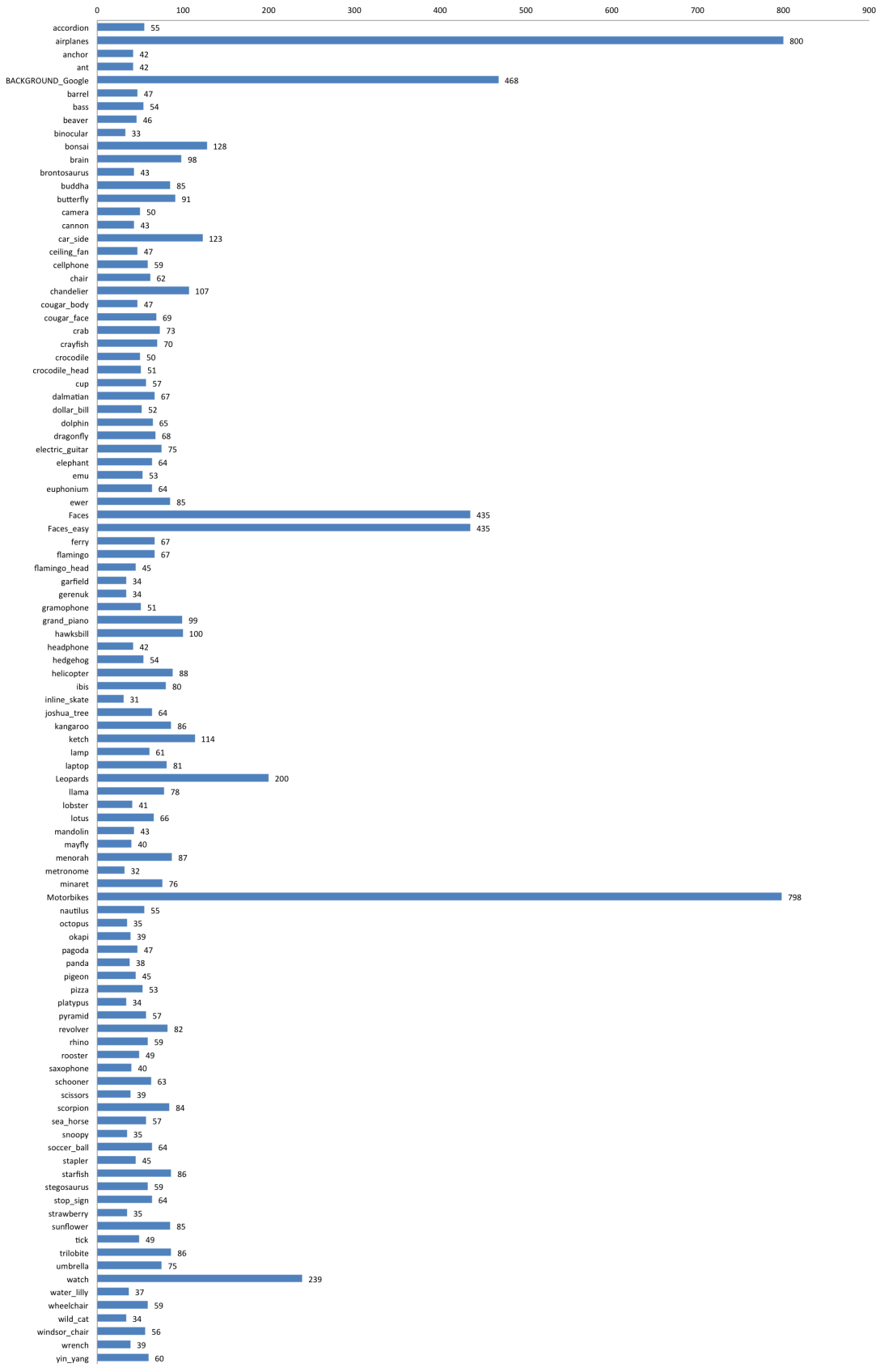

Figura 6.3: Dataset Caltech-101: distribuzione delle immagini per ogni singola classe

6.2 Valutazione prestazioni dei Classificatori

In questo capitolo illustrerò gli esperimenti che ho fatto per valutare le prestazioni dei classificatori, scelti inizialmente per questo progetto. I primo passo consiste nell'identificare il Dataset di ferimento da utilizzare: in questo esperimento ho scelto di utilizzare Dataset Caltech 101. Purtroppo questo Dataset ha un grosso difetto, infatti come possiamo vedere nel grafico 6.3, il numero delle immagini presenti per ogni classe di immagine non è uniforme. Alcune classi contengono 800 immagini (airplanes), mentre altre contengono solamente 33 immagini (binocular). Per questo motivo ho deciso di estrarre in modo casuale (attraverso un generatore di numeri casuali) esattamente 55 elementi per ogni classe di immagini. La grandezza del vocabolario è di 250 codewords per classe.

Come detto precedentemente, nell'esperimento eseguito ho scelto di utilizzare K-fold con $K = 5$ in quanto ho ritenuto fosse la tecnica più affidabile. Inoltre l'esperimento di cross-validation è stato eseguito 3 volte, in quanto le 55 immagini vengono scelte dal dataset in modo random ed eseguendolo più volte mi permette di verificare la precisione in modo più accurato. Il primo esperimento l'ho fatto su una selezione del Dataset con solamente 4 classi di immagini. Successivamente ho eseguito gli stessi test su 10 classi e infine su 25 classi. In questo modo riesco ad avere una panoramica completa del comportamento dei diversi classificatori. Nelle tabelle che mostrerò in seguito, è rappresentata la l'accuratezza globale complessiva di classificazione, per ogni round.

6.2.1 Configurazioni dei classificatori

In questo capitolo descriverò brevemente i parametri che ho utilizzato per ognuno dei classificatori scelti.

Support Vector Machine

- Tipo della SVM: C_SVC (C-Support Vector Classification). Questo parametro indica che la SVM è in grado di effettuare classificazione muticlasse con n classi ($n > 2$), consente la separazione di classi imperfetta, con moltiplicatore di penalità C.

- Tipo di funzione Kernel: RBF (Gaussian) Radial Basis Function Kernel $K(x_i, x_j) = e^{-\gamma||x_i - x_j||^2}, \gamma > 0$.
- Paramentro γ nella funzione kernel: 71.
- Valore C: 300.

Esistono altre tipologie di SVM e differenti Kernel, ma dopo molti test ho identificato questi parametri di configurazione essere quelli ottimali per il caso specifico di classificazione affrontato in questa tesi.

Random Forest

- Numero massimo di alberi nella foresta: 100.
- Profondità massima degli alberi: 50.
- Parametro ρ che indica la dimensione del sottoinsieme di features scelte a caso da testare in ogni nodo. Descritto ampiamente nel Capitolo 3.10.2.
- Precisione della Foresta (OOB error): 0.01f.
- Numero massimo delle categorie (max_categories): 100.
- Numero minimo di esempi (min_sample_count): 50. Se il numero di esempi in un nodo è minoro di questo parametro alllora il nodo non verrà diviso.

Gli altri parametri configurabili, non permettono di migliorare le prestazioni, quindi ho preferito non menzionarli.

Artificial Neural Network

- Numero di layer della rete: 3 (1 input layer, 1 output layer e 1 hidden layer).
- Numero dei neuroni da inserire nell'input layer: corrisponde al numero features presenti per ogni record nel training set.
- Numero dei neuroni da inserire nell'output layer: codice binario che è successivamente decodificato e assegnato come label all'esempio di test.
- Numero dei neuroni da inserire nel hidden layer: 201;
- Tipologia di apprendimento: BACKPROPAGATION;
- Funzione di attivazione per ogni neurone (activation function): Sigmoid Funcion.

Esistono altri parametri che la libreria OpenCv permette di configurare, come per esempio il peso del termine gradiente o del termine momento (la differenza tra i pesi sui 2 iterazioni precedenti). Ho preferito utilizzare i valori di default suggeriti dalla documentazione, in quanto dopo molti test effettuati, questi sono i parametri che danno risultati migliori.

AdaBoost

A differenza di altri algoritmi ha pochissimi parametri da configurare.

- Tipo dell'algoritmo Boost: Discrete AdaBoost (DISCRETE).
- Profondità massima dell'albero: 5.
- Numero dei classificatori deboli (weak_count): 300.
- Soglia $[0 - 1]$ usata per risparmiare tempo di calcolo (weight_trim_rate): 0,95.

6.2.2 Classificazione di 4 classi di immagini

In tabella 6.3 si nota chiaramente come la percentuale di accuratezza più alta è raggiunta nei due classificatori *Support Vector Machine* e *Random Forest*.

Tabella 6.3: Risultati classificazione di 4 classi - Cross Validation 5-Fold

K-FOLD	**SVM**	**R-Forest**	**ANN**	**AdaBoost**
Round 1	93.18	97.73	68.18	84.09
Round 2	93.18	86.36	72.73	77.27
Round 3	88.64	90.91	81.82	68.18
Round 4	95.45	93.18	79.55	77.27
Round 5	90.91	93.18	72.73	63.64
Media	**92.27**	**92.27**	**75.00**	**74.09**

A livello di performance SVM è risultato essere il più veloce nella fase di addestramento (media 150 milli-secondi per ogni round), mentre nella fase di testing impiega 400 micro-secondi. Random Forest nella fase di addestramento ha impiegato circa 4200 milli-secondi su ogni round, mentre nella fase di testing è il più veloce, infatti occorrono solamente 25 micro-secondi per testare un'immagine.

Per quanto riguarda gli algoritmi Artificial Neural Network e AdaBoost, notiamo in tabella 6.3 che sono leggermente meno accurati rispetto ai primi due algoritmi. Le performance per Artificial Neural Network dipendono molto da quanti layer ho nella rete e da quanti neuroni ho scelto per ogni "hidden" layer. Nell'esperimento con i parametri di configurazione scelti, in fase di addestramento ANN impiega in media 2000 milli-secondi, mentre in fase di testing circa 400 micro-secondi.

AdaBoost è molto lento in fase di testing rispetto agli altri algoritmi, a causa della trasformazione del problema in multiclasse, e di conseguenza occorre addestrare un numero maggiore di classificatori. In media nella fase di addestramento impiega circa 62 secondi per ogni round, mentre la fase di testing circa 350 micro-secondi.

6.2.3 Classificazione di 10 classi di immagini

Nel caso di classificazione di 10 classi di immagini, notiamo nuovamente in tabella 6.4 che i risultati più promettenti sono dati da *SVM* e *Random Forest*.

Tabella 6.4: Risultati classificazione di 10 classi - Cross Validation 5-Fold

K-FOLD	**SVM**	**R-Forest**	**ANN**	**AdaBoost**
Round 1	83.64	70.00	48.18	44.55
Round 2	72.73	73.64	42.73	46.36
Round 3	71.82	70.00	39.09	46.36
Round 4	76.36	67.27	49.09	48.18
Round 5	78.18	65.45	41.82	43.64
Media	**76.55**	**69.27**	**44.18**	**45.82**

A livello di performance SVM impiega in questo esperimento in media 2,4 secondi per fare l'addestramento e 3 milli-secondi per testing di un immagini. Random forest per l'addestramento impiega in media 61 secondi, mentre è nuovamente il più veloce nella fase di testing, con in media 67 micro-secondi. Artificial Neural Network è addestrata in 6,2 secondi, mentre impiega per la fase di test in media 2 milli-secondi. AdaBoost si comporta come nell'esempio prece-

dente ed è molto lento in fase di addestramento, mentre come si può notare in tabella 6.6 (in basso), migliora in fase di testing.

6.2.4 Classificazione di 25 classi di immagini

In questo esperimento ho scelto 35 immagini per classe, in quanto alcune classi contenevano meno di 55 immagini. Si può osservare, anche in questo caso,

Tabella 6.5: Risultati classificazione di 25 classi - Cross Validation 5-Fold

K-FOLD	**SVM**	**R-Forest**	**ANN**	**AdaBoost**
Round 1	56.57	42.29	21.71	24.57
Round 2	60.00	40.00	22.86	26.86
Round 3	53.71	48.57	18.29	28.57
Round 4	52.57	44.57	18.86	28.57
Round 5	52.00	44.00	20.00	28.00
Media	**54.97**	**43.89**	**20.34**	**27.31**

il netto divario di accuratezza tra il classificatore SVM e Random Forest e il classificatore ANN e AdaBoost. Questi risultati confermano nuovamente quanto detto precedentemente nel caso a 10 classi. Da notare Random Forest, nonostante le classi siano aumentate, nella fase di testing, il tempo rimane molto basso (0,82 micro-secondi). SVM ha performance abbastanza basse nella fase di testing, mentre AdaBoost nella fase di addestramento impiega circa a 31 minuti.

Tabella 6.6: Confronto tempistiche dei vari classificatori (millisecondi)

Classificatore	**Training**			**Testing**		
	4 classi	**10 classi**	**25 classi**	**4 classi**	**10 classi**	**25 classi**
SVM	150	2400	10600	0.400	3.000	8.150
RFOREST	4200	61000	221130	**0.025**	**0.078**	**0.082**
ANN	2000	6200	33740	0.400	2.000	3.800
BOOST	62000	645700	**1867680**	0.350	0.650	0.680

6.2.5 Classificatore scelto

I test effettuati dimostrano che SVM e Random Forest risultano essere i due classificatori migliori, sia in termini di accuratezza globale, sia in termini di performance. Ho deciso di utilizzare *Support Vector Machines* come classificatore di riferimento per il mio progetto in quanto mi permette di raggiungere maggior accuratezza aumentando il numero delle classi, in particolare 10 o 25. Random Forest risulta essere molto preciso nella classificazione di 4 classi di oggetti, ma aumentando il numero della classi l'accuratezza cala e preferisco utilizzare un classificatore che mi garantisca maggior accuratezza. Random Forest inoltre è molto veloce in fase di testing è questo può essere interessante nel caso di utilizzo di questa applicazione in un sistema embedded in cui è necessario prendere decisioni in tempo real time. Gli ultimi due algoritmi che ha analizzato (AdaBoost e ANN), risultano essere poco accurati in presenza di molte classi.

Da adesso in avanti, in tutti gli esperimenti, utilizzerò sempre come classificatore di riferimento SVM con i parametri di configurazione indicati in precedenza.

6.3 Valutazione sperimentale del modello BoW con SVM

Identificato *Support Vector Machines* come classificatore ideale per il problema di classificazioni di immagini, nel seguente capitolo mostrerò gli esperimenti effettuati con lo scopo di valutare le prestazioni del modello Bag of Visual Words. Inoltre analizzerò come reagisce su diversi Dataset, partendo da classificazioni molto semplici di 4 classi, fino ad arrivare a classificare 25 classi di immagini.

In questa seconda fase dei miei esperimenti, in accordo con quanto detto precedentemente nel Capitolo 5.1, ho scelto come Dataset Caltech-101 e Dataset Caltech-256. Ho effettuato quattro tipologie di test, partendo da classificazioni semplici e crescendo di difficoltà. Infine ho provato ad utilizzare delle immagini segmentate prese dal Dataset Pascal VOC-2010.

Un parametro molto importante in questi esperimenti è quello di scegliere il numero di immagini utilizzare per ogni classe, per formare il *training set* e il *test set*. Negli esperimenti la scelta che ho fatto, è stata quella di prendere in modo

Random circa 55 immagini per ogni classe (senza ripetizioni) da usare come training set, le restanti le ho usate come test set. Mentre per quanto riguarda quelle classi a cardinalità minore, ho scelto le prime 40 immagini e le restanti le ho usate come test set. Infine occorre decidere la grandezza del vocabolario per ogni singola classe di immagini. Questo è un problema abbastanza complesso che tratterò nel paragrafo seguente.

6.3.1 Grandezza del vocabolario

La grandezza del vocabolario visuale, descritta nel capitolo precedente, è molto importante. Sono stati proposti vari articoli in cui si discute la dimensione corretta di codewords da utilizzare nel modello Bag of Visual Words: [1], [25]. Per esempio nell'articolo di Lazebnik [72] si utilizzano da 200 a 400 Visual Words per il vocabolario, Zhang [73] suggerisce 1000, mentre Sivic [34] propone da 6000 a 10000. Questo articolo [74] propone una soluzione generica del problema, che può essere applicato a qualsiasi Dataset.

È stato dimostrato che la dimensione del vocabolario influisce sulla precisione nel trovare le corrispondenti Visual Words. Scegliere la dimensione del vocabolario troppo piccola, non permette di discriminare in modo corretto tra le diverse Visual Words. Sebbene un vocabolario molto grande tende a migliorare l'accuratezza nel trovare corrispondenze (matching), non significa che un vocabolario più ampio sicuramente determina un'accuratezza più elevata. Gli autori [59] mostrano attraverso diversi esperimenti, che aumentando la dimensione del dizionario da centinaia a decine di migliaia, l'accuratezza nel trovare corrispondenze, prima aumenta drasticamente, poi raggunge un picco dopo che ha raggiunto il livello corretto, e infine scende leggermente. Analogo andamento è stato osservato anche in questo articolo [75]. Ciò significa che un vocabolario *molto grande* non solo aumenta il carico computazionale per il calcolo attraverso clustering, ma un impatto negativo sulle prestazioni. Quindi è necessario selezionare un vocabolario appropriato che produce le migliori prestazioni trovando un corretto bilanciamento. Non esiste ancora una teoria definitiva o metodi per guidare la scelta della dimensione ottimale vocabolario per un dato insieme di dati immagine.

In seguito mostrerò attraverso opportune tabelle, quanto influisce la dimensione del dizionario sul modello in esame e cercherò di individuare la dimensione ottimale del vocabolario per un determinato set di dati.

6.3.2 Primo test semplice

Il primo esperimento effettuato è stato quello di classificare 4 classi di immagini, estratte dal Dataset Caltech-101: airplanes, car, Faces_easy, Leopards e motorbikes. Sono tutti oggetti tra loro molto diversi. Ho utilizzato circa 100 immagini per ogni classe scelte in modo random da ogni cartella e la restante parte delle immagini le ho utilizzate per il testing.

In tabella 6.7 sono rappresentate le 4 classi utilizzate e per ogni classe ho indicato il numero di immagini utilizzate.

Tabella 6.7: Selezione di 4 classi del Dataset Catech-101

Classe	Training set	Test set	Recall	Precision
Airplanes	107	398	96.48	94.12
Faces_easy	104	169	97.04	86.31
Leopards	87	33	100.00	97.06
Motorbikes	110	392	90.81	98.89

In tabella 6.9 sono illustrati i risultati ottenuti in questo primo test. Attraverso la matrice di confusione si nota molto chiaramente la bassa percentuali di errore su queste 4 classi e il numero di immagini classificate correttamente. In questo esperimento ho utilizzato un vocabolario di dimensione 200 per ogni classe.

Tabella 6.8: Classificazione di 4 classi di oggetti - dim. vocabolario: 800

Act / Pred	Airplanes	Faces_easy	Leopards	Motorbikes	Total	Recall
airplanes	**384**	11	0	3	398	96.48
Faces_easy	3	**164**	1	1	169	97.04
Leopards	0	0	**33**	0	33	100.00
Motorbikes	21	15	0	**356**	392	90.81
Total	408	190	34	360	992	
Precision	94.12	86.31	97.06	98.89		

Infine in tabella 6.10 è mostrato la variazione dei valori di accuratezza al variare del Codebook. Essendo un test molto facile, variando la variazione della grandezza del vocabolario per ogni singola classe, notiamo che la accuratezza globale del modello rimane invariata.

Tabella 6.9: Classificazione di 4 classi di oggetti - dim. vocabolario: 2000

Act / Pred	Airplanes	Faces_easy	Leopards	Motorbikes	Total	Recall
Airplanes	**369**	24	0	5	398	92.71
Faces_easy	4	**165**	0	0	169	97.63
Leopards	0	0	**33**	0	33	100
Motorbikes	9	24	0	**359**	392	91.58
Total	382	213	33	364	992	
Precision	96.59	77.46	100	98.62		

Tabella 6.10: Confronto con differenti dimensioni di vocabolario - 4 classi

CodeBook Size	Precision_{μ}	Recall_{i} più alta	Recall_{i} più bassa
150	93.65	100	91.58
200	94.45	100	90.81
250	94.45	100	93.11
300	94.15	100	91.58
400	94.35	100	92.09
500	93.34	100	91.58
600	93.04	100	89.75

6.3.3 Test intermedio: Dataset Caltech-101

In questo test ho selezionato inizialmente 10 classi di immagini dal Dataset Caltech-101, successivamente ho scelto 25 classi di oggetti e ho eseguito gli stessi esperimenti. Ho utilizzato circa 55 immagini scelte in modo random (senza ripetizioni) per classi contenenti un numero di immagini superiori a 80, mentre per le classi con un numero di immagini inferiori, ho selezionato le prime 40 immagini contenute nella cartella e le restanti per la fase di testing.

In seguito ho rappresentato attraverso varie tabelle i risultati ottenuti. Non è un test molto complesso, in quanto come detto precedentemente, le immagini presenti in questo Dataset sono state elaborate, quindi risultano abbastanza semplici. Successivamente ho effettuato un test più complesso, sempre sullo stesso Dataset, selezionando 25 classi.

Classificazione di 10 classi di immagini

Ho scelto 10 classi all'interno del Dataset Caltech-101, non facendo nessuna assunzione particolare. Ho selezionato categorie di immagini molto diverse tra loro: cars, leopards, watch, hawksbill (tartaruga), ketch (veliero) dal Dataset.

Notiamo in tabella 6.11 che il valore *Recall*, (che indica la percentuale di immagini classificate correttamente) denota la difficoltà nel riconoscere la classe "helicopter". Questo è dovuto al fatto che le immagini di training non sono molto accurate: l'oggetto nella scena non è ben visibile rispetto allo sfondo, oppure l'immagini è di bassa qualità. Un altra classe difficile da riconoscere è la classe "watch", anche in questo caso per le medesime motivazioni.

Tabella 6.11: Selezione di 10 classi - codebook 150

Classi	Training set	Test set	Recall	Precision
accordion	40	15	80.00	75.00
car_side	46	29	79.31	74.19
cougar_face	40	29	82.75	55.81
dalmatian	40	27	85.18	67.64
hawksbill	40	60	81.66	73.13
helicopter	40	48	37.50	42.85
ketch	47	28	75.00	50.00
Leopards	48	83	86.74	97.29
sunflower	40	45	73.33	91.66
watch	50	109	67.89	84.09

Per quanto riguarda la *Precision* sulla singola classe, (frazione di true positivi rispetto ai positivi - più è alto, meno falsi negativi abbiamo), si può osservare che in alcune classi sono presenti molti falsi positivi: "dalmation", "cougar_face", "helicopter", "ketch". Questo si nota molto bene anche nella tabella 6.12 ed è riconducibile alle immagini non molto definite.

La Recall più bassa viene riscontrata praticamente sempre nella classe "helicopter", mentre la Recall con valore più elevato, si alterna tra la classe: "Leopards", "cougar_face", "accordion" e "dalmatian". L'accuratezza globale (mostrata in tabella 6.13) con vocabolario da 200 codewords per classe è di 74,20%.

L'istogramma illustrato in Figura 6.4, è la rappresentazione grafica della tabella 6.11.

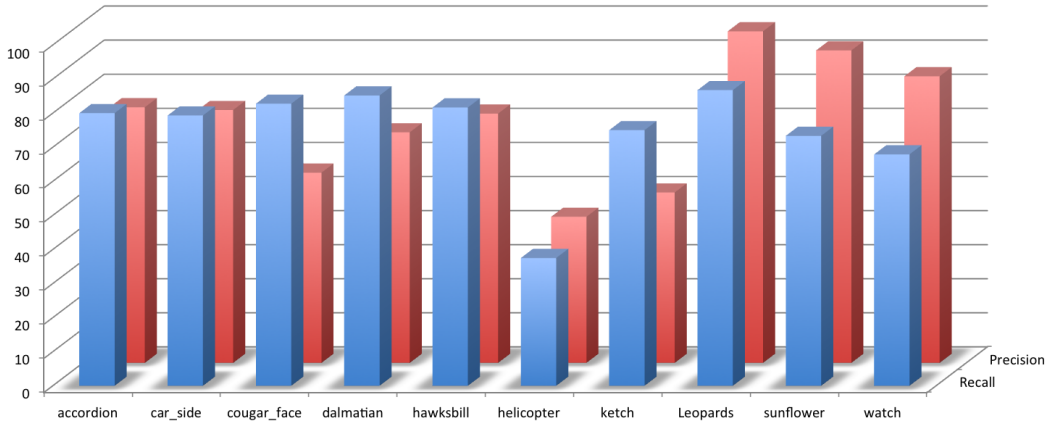

Figura 6.4: Rappresentazione del risultato di classificazione di 10 classi

Tabella 6.12: Matrice di confusione: 10 classi - codebook 150

Act / Pred	accordion	car_side	cougar_face	dalmatian	hawksbill	helicopter	ketch	Leopards	sunflower	watch	Total	Recall
accordion	**12**	0	1	0	0	1	0	0	0	1	15	80.00
car_side	0	**23**	2	1	0	1	2	0	0	0	29	79.31
cougar_face	0	0	**24**	0	2	1	0	0	2	0	29	82.75
dalmatian	0	0	3	**23**	1	0	0	0	0	0	27	85.18
hawksbill	0	1	1	2	**49**	4	0	2	0	1	60	81.66
helicopter	0	3	6	3	1	**18**	8	0	0	9	48	37.50
ketch	0	0	0	0	0	4	**21**	0	1	2	28	75.00
Leopards	0	1	1	0	9	0	0	**72**	0	0	83	86.74
sunflower	0	0	4	3	1	1	2	0	**33**	1	45	73.33
watch	4	3	1	2	4	12	9	0	0	**74**	109	67.89
Total	16	31	43	34	67	42	42	74	36	88	**473**	
Precision	75.00	74.19	55.81	67.64	73.13	42.85	50.00	97.29	91.66	84.09		

In tabella 6.13 notiamo che, in questa tipologia di classificazione, la grandezza ideale del clustering per ogni classe è 200. Aumentando la grandezza del codebook, come prevedibile, l'accuratezza globale diminuisce. Avendo un numero di classi non elevato, aumentando la grandezza si introduce nel vocabolario rumore (cioè parole visuali non caratteristiche dell'immagine) e nel complesso avremo diminuzione delle prestazioni e dell'affidabilità. Inoltre avendo molte codewords il classificatore fa più fatica a discriminare correttamente la classe corretta.

Tabella 6.13: Confronto con differenti dimensioni di vocabolario - 10 classi

CodeBook Size	Precision_{μ}	Recall_i più alta	Recall_i più bassa
100	72.72	85.18	45.83
150	73.78	86.74	37.50
200	74.20	86.67	39.58
250	70.40	92.59	37.51
300	71.24	89.15	91.58
350	71.88	88.88	22.91
400	70.40	86.67	35.41
450	70.61	86.67	29.16
500	70.61	86.20	39.58
600	70.61	82.22	33.33

Classificazione di 25 classi di immagini

In quest'ultimo test ho preso 25 classi dal Dataset Caltech-101. Nella scelta delle classi ho cercato di selezionare una collezione di immagini che comprendesse sia classi distinte (ad esempio: bonsai vs airplanes) che classi simili (ad esempio: wild_cat vs Leopard). In questo modo ho potuto testare le prestazioni del classificatore sia su immagini relativamente simili che su immagini nettamente differenti.

Si nota chiaramente in tabella 6.14 che alcune classi vengono riconosciute molto bene, mentre altre, come per esempio la classe "butterfly", ha un valore di Recall molto basso. Nel complesso però i risultati sono abbastanza interessanti: 11 classi di immagini sono state classificati correttamente con una percentuale maggiore del 70%, mentre altre 8 classi hanno una percentuale tra il 50% e il 60%. Nel complesso la accuratezza complessiva su 25 classi è di 71,33% e credo sia un ottimo risultato.

Tabella 6.14: Selezione di 25 classi

Classe	Training set	Test set	Recall	Precision
accordion	40	15	73.33	64.70
airplanes	52	446	77.57	92.26
bonsai	50	36	66.66	21.24
butterfly	40	51	25.49	30.23
car_side	40	33	45.45	57.69
dalmatian	40	27	74.07	76.92
dollar_bill	40	12	**91.66**	91.66
Faces	51	224	88.83	70.57
hawksbill	40	60	58.33	50.72
headphone	40	2	50.00	4.54
ibis	40	40	47.50	33.33
laptop	40	41	39.02	64.00
Leopards	50	86	76.74	**97.059**
lotus	40	26	53.84	46.67
Motorbikes	52	451	78.05	98.05
pizza	40	13	76.92	15.15
pyramid	40	17	35.29	50.00
rhino	40	19	57.89	34.37
soccer_ball	40	24	50.00	92.31
stop_sign	40	24	75.00	94.74
sunflower	40	45	57.78	89.65
trilobite	40	46	76.08	87.50
watch	49	105	51.43	66.67
wild_cat	27	7	14.28	11.11
yin_yang	40	20	**95.00**	42.22

L'istogramma illustrato in Figura 6.5, sottolinea nuovamente quanto già detto. Le colonne di colore blu mostrano la Recall, mentre le colonne rosse mostrano la Precision.

Nella matrice di confusione (tabella 6.15) notiamo come alcune classi vengono classificate molto meglio di altre, ottenendo una percentuale di classificazione corretta molto alta: "Faces", "yin_yang", "dollar_bill", "airplanes". Questo comportamenti è molto simile all'esperimento fatto precedentemente a 10 classi. Questo è dovuto all'oggetto rappresentato nell'immagine, facilmente separabile dalla sfondo e che contiene pattern rappresentativi di quella classe. Inoltre si nota molto chiaramente che alcuni oggetti vengono classificati in modo scorretto: "wild_cat", "ibis", "butterfly".

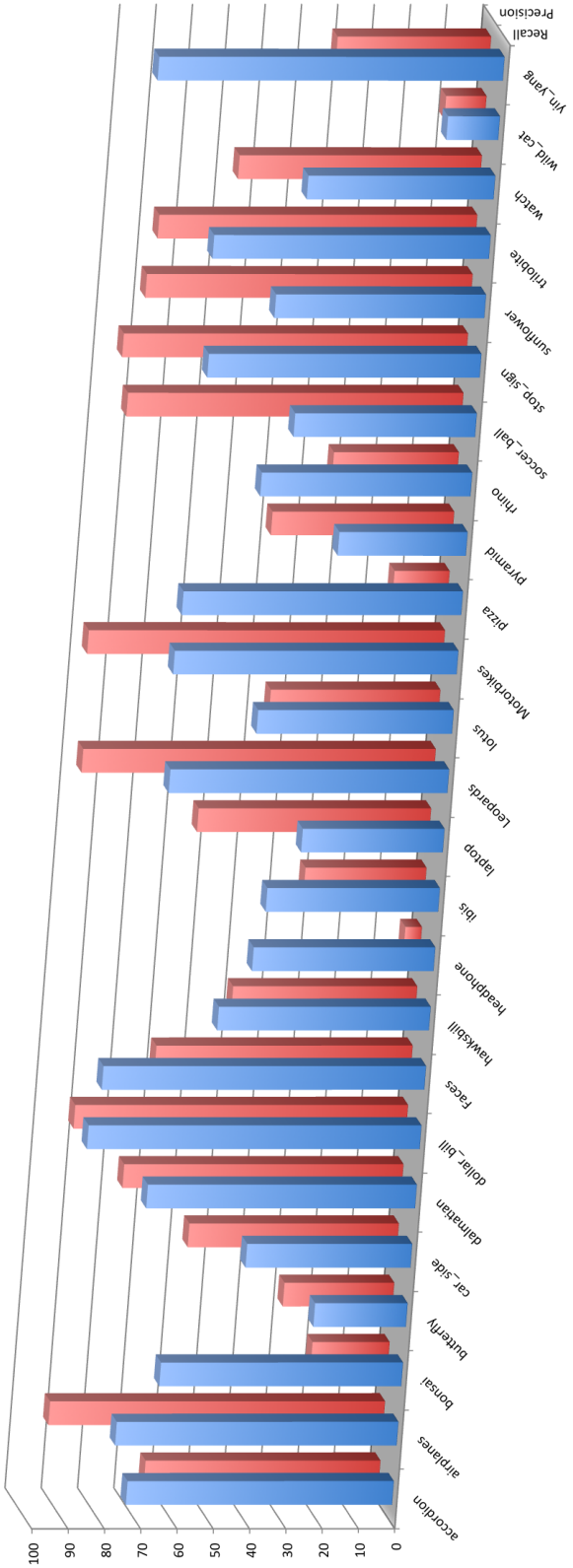

Figura 6.5: Rappresentazione del risultato di classificazione di 25 classi

Tabella 6.15: Matrice di confusione: 25 classi - codebook 200

Act / Pred	accordion	airplanes	bonsai	butterfly	car_side	dalmatian	dollar_bill	Faces	hawksbill	headphone	ibis	laptop	Leopards	lotus	Motorbikes	pizza	pyramid	rhino	soccer_ball	stop_sign	sunflower	trilobite	watch	wild_cat	yin_yang	Totale	Recall
accordion	**11**	0	0	1	0	0	0	0	0	0	1	1	0	0	0	0	0	0	0	0	0	1	0	0	0	15	73.33
airplanes	0	**346**	8	5	5	0	0	25	1	12	4	2	0	6	4	4	1	0	0	0	0	0	12	0	11	446	77.57
bonsai	0	0	**24**	0	2	1	0	1	5	0	1	0	0	0	0	2	0	0	0	0	0	0	0	0	0	36	66.66
butterfly	0	1	6	**13**	0	2	0	2	2	4	6	0	0	3	0	4	1	1	1	0	2	1	0	1	1	51	25.49
car_side	0	0	5	0	**15**	0	0	3	2	0	0	0	0	0	0	7	0	1	0	0	0	0	0	0	0	33	45.45
dalmatian	0	0	2	0	0	**20**	0	1	0	0	0	0	0	0	0	4	0	0	0	0	0	0	0	0	0	27	74.07
dollar_bill	0	0	0	0	0	0	**11**	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	12	91.66
Faces	2	0	3	1	0	1	1	**199**	2	1	0	0	0	1	0	9	0	2	0	0	0	0	1	0	1	224	88.83
hawksbill	0	2	9	0	0	0	0	0	**35**	0	4	0	1	0	1	3	1	2	0	0	0	0	0	1	1	60	58.33
headphone	0	1	0	0	0	0	0	0	0	**1**	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	2	50.00
ibis	0	1	4	2	0	0	0	3	5	0	**19**	0	0	1	1	2	0	0	0	0	0	0	0	1	1	40	47.50
laptop	1	2	0	2	0	0	0	9	0	1	0	**16**	0	0	1	2	2	0	0	0	0	0	5	0	0	41	39.02
Leopards	0	0	1	1	0	0	0	0	10	0	4	0	**66**	0	0	0	1	2	0	0	0	0	0	1	0	86	76.74
lotus	0	1	2	1	0	1	0	2	0	0	1	0	0	**14**	0	0	0	2	0	0	1	0	1	0	0	26	53.84
Motorbikes	0	14	31	7	4	0	0	13	3	0	6	1	0	1	**352**	8	0	4	0	1	0	0	6	0	0	451	78.05
pizza	0	0	1	1	0	1	0	0	0	0	0	0	0	0	0	**10**	0	0	0	0	0	0	0	0	0	13	76.92
pyramid	0	0	3	0	0	0	0	0	0	0	0	1	1	1	0	0	**6**	0	0	0	0	0	0	0	5	17	35.29
rhino	0	0	2	1	0	0	0	0	1	0	3	0	0	0	0	1	0	**11**	0	0	0	0	0	0	0	19	57.89
soccer_ball	0	0	0	0	0	0	0	2	0	0	3	1	0	1	0	0	0	0	**12**	0	0	0	0	2	3	24	50.00
stop_sign	0	0	1	0	0	0	0	0	1	1	0	0	0	0	0	0	0	0	0	**18**	0	0	0	0	3	24	75.00
sunflower	0	0	9	1	0	0	0	3	0	1	0	0	0	2	0	0	0	1	0	0	**26**	1	1	0	0	45	57.79
trilobite	0	1	0	0	0	0	0	1	1	0	2	0	0	0	0	1	0	4	0	0	0	**35**	0	1	0	46	76.08
watch	3	6	2	7	0	0	0	17	0	1	3	3	0	0	0	6	0	0	0	0	0	2	**54**	1	0	105	51.43
wild_cat	0	0	0	0	0	0	0	1	1	0	0	0	0	0	0	2	0	2	0	0	0	0	0	**1**	0	7	14.28
yin_yang	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	**19**	20	95.00
Totale	17	375	113	43	26	26	12	282	69	22	57	25	68	30	359	66	12	32	13	19	29	40	81	9	45	1870	
Precision	64.70	92.26	21.24	30.23	57.69	76.92	91.66	70.57	50.72	4.54	33.33	64.00	97.05	46.66	98.05	15.15	50.00	34.37	92.31	94.74	89.65	87.5	66.67	11.11	42.22		

Tabella 6.16: Confronto con differenti dimensioni di vocabolario - 25 classi

CodeBook Size	Precision_{μ}	Recall_{i} più alta	Recall_{i} più bassa
100	71.39	100	0
150	71.92	100	14.28
200	71.33	95	14.28
250	71.33	100	14.28
300	69.35	100	0
350	68.82	90	0
400	67.54	100	14.28
450	66.57	91.66	0
500	66.73	100	0
550	65.50	90	14.28

Infine in tabella 6.16 notiamo come il classificatore si comporta meglio con codebook di dimensione tra 200 e 250 per classe per un totale di 6250 codewords.

6.3.4 Test difficile: Dataset Caltech-256

In questo esperimento ho utilizzato 5 e 10 classi di immagini, scelte dal Dataset Caltech-256. E' importante notare come le immagini in questo Dataset non sono state elaborate artificialmente, quindi questo tipo di classificazione sarà molto più difficile. Gli oggetti oltre a non essere allineati nelle immagini, non sempre occupano la maggior parte dello spazio totale. Come illustrerò attraverso i risultati ottenuti, l'accuratezza non risulta essere molto alta.

Classificazione di 5 classi di immagini

Se osserviamo la labella 6.17, notiamo che la percentuale di immagini classificate correttamente non è alta come nell'esperimento sul Dataset Caltech-101.

Tabella 6.17: Selezione di 5 classi del Dataset Catech-256 - codebook 300

Classe	Training set	Test set	Recall	Precision
158.penguin	42	50	79.48	37.80
159.people	51	80	54.66	51.89
216.tennis-ball	40	58	51.72	49.18
232.t-shirt	53	185	56.10	75.41
250.zebra	40	56	76.78	89.58

Nella matrice di confusione 6.18 osserviamo inoltre un numero molto elevato di falsi positivi. La classe "158.penguin", nonostante abbiamo un valore Recall abbastanza alto (79,50%), ha un valore molto basso di accuratezza globale (38,70%), dovuto alla grande varietà di immagini presenti in questa classe e alla difficoltà del classificatore nel trovare differenze rappresentative rispetto ad altre classi. Spesso la classe "158.penguin" è confusa con la classe "232.t-shirt" o "159.people", rispettivamente 16 e 17 falsi positivi.

Tabella 6.18: Matrice di confusione - Vocabolario 300 per classe

Act / Pred	**158.penguin**	**159.people**	**216.tennis-ball**	**232.t-shirt**	**250.zebra**	Totale
158.penguin	**31**	6	0	2	0	39
159.people	17	**41**	4	11	2	75
216.tennis-ball	9	2	**30**	17	0	58
232.t-shirt	16	27	26	**92**	3	164
250.zebra	9	3	1	0	**43**	56
Totale	82	79	61	122	48	392

Classificazione di 10 classi di immagini

In tabella 6.19, in cui è mostrata la Recall per ogni classe, si nota come sia difficile identificare la dimensione corretta del dizionario. Questo risultato è confermato anche dalla accuratezza globale ottenuta che è costantemente intorno al 60,02%. Ho evidenziato per ogni classe i valori con percentuale maggiore. Anche in questo caso, osserviamo che la percentuale di immagini classificate correttamente non è alta. Solamente due classi superano il 70% e solamente due il 60%.

I risultati ottenuti su questo Dataset Caltech-256, come era prevedibile, sono abbastanza negativi, ma questo è dovuto alle immagini presenti nel dataset e alla difficoltà del modello a generalizzare su determinate classi.

6.3.5 Test su immagini segmentate: Dataset VOC2010

In quest'ultimo esperimento, leggermente diverso rispetto a quelli precedenti, ho scelto di utilizzare immagini segmentate prese dal Dataset Pascal VOC-2010. Normalmente in un immagine, possono essere presenti contemporanemante più oggetti di classi differenti. All'interno della cartella Pascal Voc (scaricabile dal

Tabella 6.19: Accuratezza di ogni classe al variare della dimensione del vocabolario

			Codebook										
Classe	**Training**	**Test**	**1000**	**1500**	**2000**	**2500**	**3000**	**3500**	**4000**	**4500**	**5000**	**5500**	**6000**
024.butterfly	37	26	34.61	42.31	46.15	46.15	26.92	42.30	50.00	42.30	42.30	**57.69**	34.61
043.coin	46	33	66.66	69.69	63.63	69.69	66.66	66.66	66.66	69.69	60.60	60.60	**72.73**
056.dog	43	26	23.08	26.92	19.23	**30.76**	11.53	19.23	15.38	15.38	11.53	11.53	19.23
105.horse	48	127	28.34	**29.92**	27.55	21.25	16.53	18.89	23.62	14.17	14.17	11.02	14.96
158.penguin	46	49	46.94	46.94	55.10	**61.22**	53.06	59.18	55.10	55.10	55.10	55.10	59.18
159.people	47	85	31.76	34.12	37.64	28.23	32.94	32.94	35.29	36.47	**45.88**	44.71	38.82
185.skateboard	39	29	20.69	20.69	10.34	13.79	13.79	13.79	10.34	17.24	17.24	**20.69**	10.34
216.tennis-ball	40	58	**60.34**	48.28	53.44	55.17	50.00	50.00	48.27	51.72	51.72	53.44	53.44
232.t-shirt	52	170	54.12	58.23	53.52	**55.88**	51.76	54.70	51.76	54.70	50.58	50.00	48.82
250.zebra	40	56	76.78	**80.35**	76.78	78.57	78.57	78.57	76.78	76.78	78.57	73.21	75.00

Server della competizione) c'è un file xml, per ogni immagine, in cui è indicata l'area segmentata "bounding-box" di ogni singolo oggetto presente nell'immagine. $(X_{min}, X_{max}, Y_{min}, Y_{max})$. In questo modo sono riuscito ritagliare dall'immagine differenti oggetti segmentati, che posano essere classificati separatamente attraverso l'applicazione che ho creato.

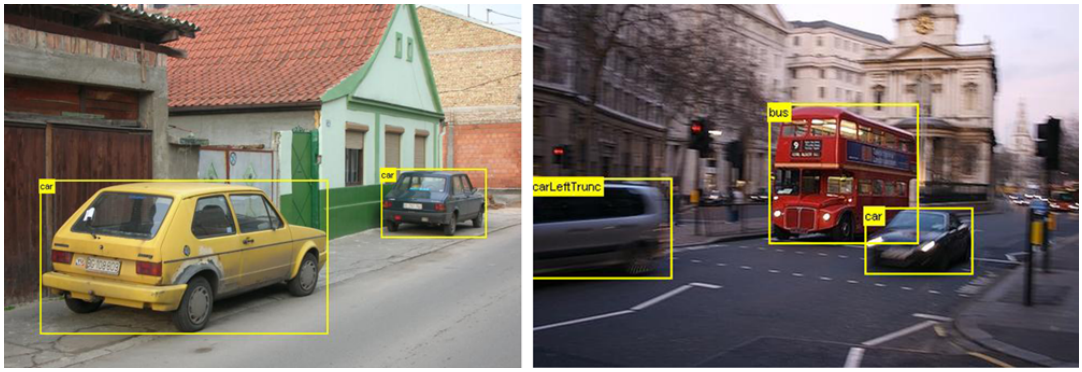

Figura 6.6: Immagini segmentate di più oggetti nella stessa immagine [33]

Classificazione di 4 classi di immagini

Notiamo nella matrice di confusione (6.21) che la classe in cui il classificatore, riscontra maggiore difficoltà è la classe "airplanes", con una percentuale che supera di poco il 50%. La percentuale ottenuta sulle altre classi è abbastanza alta, anche se non raggiunge l'accuratezza ottenuta nel primo esperimento fatto sul Dataset Caltech-101. Questo è dovuto al fatto che nonostante le immagini

Tabella 6.20: Selezione di 4 classi del Dataset VOC2010 - Vocabolario 400

Classe	Training set	Test set
aeroplane	40	37
car	40	55
cat	48	83
person	54	482

Tabella 6.21: Matrice di confusione - Vocabolario 400 per classe

Act / Pred	aeroplane	car	cat	person	Totale	Recall
aeroplane	**19**	8	0	10	37	51.35
car	3	**33**	0	19	55	60.00
cat	0	0	**59**	24	83	71.08
person	15	20	60	**387**	482	80.29
Totale	37	61	119	440	657	
Precision	51.35	54.09	49.58	87.95		

siano segmentate, spesso lo sfondo dell'immagine non è uniforme e il classificatore non riesce a generalizzare rispetto alla classe.

Inoltre non sempre l'oggetto è rappresentato nell'immagine nella stessa posizione. Nel Dataset Caltech-101 l'aereo è raffigurato sempre di lato e interamente, mentre in questo Dataset a volte gli oggetti sono ruotati o tagliati, quindi la classificazione risulta esser più complicata. Un esempio è mostrato in Figura 6.7.

Figura 6.7: Immagini segmentate di aerei, con sfondo non uniforme [33]

Classificazione di 7 classi di immagini

Aumentando il numero delle classi, come era prevedibile, l'accuratezza sulle singole classi diminuisce.

Tabella 6.22: Matrice di confusione - Vocabolario 450 per classe

Act / Pred	aeroplane	bicycle	bird	bus	cat	motorbike	person	Totale	Recall
aeroplane	**10**	3	2	3	0	1	6	25	40.00
bicycle	0	**19**	3	0	0	3	5	30	63.33
bird	7	0	**25**	0	6	2	8	48	52.08
bus	2	2	0	**8**	0	1	0	13	61.54
cat	2	0	12	0	**47**	2	20	83	56.62
motorbike	0	1	0	1	0	**15**	4	21	71.43
person	18	6	60	6	19	33	**341**	483	70.60
Totale	39	31	102	18	72	57	384	707	
Precision	25.64	61.29	24.5	44.45	65.28	26.31	88.80		

Quest'ultimo test, nonostante le immagini siano state segmentate in precedenza, ha dimostrato la difficoltà del modello Bag of Visual Words a classificare classi di oggetti provenienti da immagini con sfondo non uniforme o in cui oggetti della stessa classe sono rappresentati in posizioni diverse, o tagliati.

6.4 Confronto con altri tipi di descrittori

In questo capitolo valuterò le prestazioni di 3 nuovi descrittori: ORB, BRISK e SURF confrontandoli con i risultati ottenuti precedentemente utilizzando il descrittore SIFT. Effettuerò l'esperimento su una selezione di 4, 10 e 25 classi del Dataset Caltech 101, in modo simile a quanto precedentemente (Cap. 6.3.2).

Tabella 6.23: Accuratezza globale utilizzando diversi descrittori

	ORB	BRISK	SURF	SIFT
4 classi	80.42	79.47	94.72	94.32
10 classi	53.53	43.83	80.09	74.20
25 classi	42.01	42.91	73.39	71.33

Osserviamo in tabella 6.23, come era prevedibile SURF e SIFT sono i descrittori che raggiungono percentuali di accuratezza più alta. Questo è dovuto alla strutturato del modello Bag of Visual Words. Come illustrato nel Paragrafo 5.7, i descrittori binari BRISK e ORB sono trasformati e forzatamente raggruppati in cluster attraverso K-means, ma per loro natura non si prestano ad essere utilizzati in questo modo, essendo stringhe di bit. Purtroppo questi test sono stati effettuati con la prima soluzione descritta nel Paragrafo 5.7.1, per mancanza di tempo non sono riuscito ad implementare in questa tesi la seconda versione proposta da Grana, Borghesani, Cucchiara nell'articolo [31]. Sicuramente uno stimolo per essere sviluppata in futuro.

Dal punto di vista delle prestazione SIFT, come sottolineato precedentemente, è molto dispendioso e per l'estrarre features impiega molto tempo. SURF permette di raggiungere una accuratezza abbastanza elevata, e sul calcolatore utilizzato in questo lavoro risulta essere molto veloce, confrontato con SIFT. Queste sono le motivazione che hanno portato Bay, Tuytelaars e Van Gool [8] a creare un nuovo descrittore più leggero.

ORB e BRISK essendo descrittori binari creati appositamente per essere veloci, hanno prestazioni ottime. In tabella 6.24 possiamo osservare il tempo medio impiegato da ogni descrittore per estrarre features da un immagine scelta in modo casuale dal Dataset Caltech-101. Questi valori ovviamente sono indicativi, in quanto dipendono dalla macchina sui cui è stato eseguito questo test e dai parametri di configurazione di ogni singolo descrittore utilizzato in OpenCV.

Tabella 6.24: Tempo impiegato per estrarre features (millisecondi)

	ORB	BRISK	SURF	SIFT
media	22,8654	42,714	39,1735	259,077

SURF BRISK

Figura 6.8: Identificazione keypoints utilizzando diversi metodi

Capitolo 7

Conclusioni e sviluppi futuri

7.1 Sintesi dei risultati ottenuti

In questo lavoro di tesi si è voluto realizzare un'applicazione in grado di classificare classi di oggetti all'interno di immagini o video. L'individuazione e il riconoscimento di oggetti è a mio avviso una sfida molto affascinante, che da molti anni impegna i ricercatori. Le applicazioni possono essere infinite; basti immaginare un sistema in grado di compiere determinate scelte dopo avere identificato l'oggetto per cui è stato addestrato. Robot in grado di muoversi autonomamente o veicoli automatizzati che funzionano senza la supervisione umana. Questo avviene grazie all'unione di due discipline molto diverse tra loro, ma allo stesso tempo correlate: Computer Visione e Artificial Intelligence.

Dopo aver analizzato alcuni metodi presenti in letteratura, ho scelto di utilizzare il modello Bag of Visual Words che si presta molto bene a questo tipo di problema ed relativamente semplice da realizzare.

Sono soddisfatto di aver raggiunto gli obiettivi che mi sono prefissato all'inizio di questo lavoro di tesi. Analizzando i risultati ottenuti attraverso il metodo scelto, l'accuratezza di classificazione raggiunta su una singola classe è abbastanza alta, anche in presenza di molte classi.

L'analisi che ho fatto sui tipi di classificatori è risultata essere molto utile, infatti come ho descritto nel capitolo 6.2, sia Support Vector Machines, sia Random Forest raggiungono una accuratezza elevata. Quello più accurato risulta essere SVM, essendo costruito su forti teorie matematiche, ma Random Forest è molto

veloce in fase di esecuzione. Questa caratteristica lo rende molto interessante, in quanto può esser utilizzato su un sistema embedded con risorse limitate. La fase di addestramento ovviamente deve essere eseguita all'esterno su un sistema più potente.

Se nelle immagini di training sono presenti oggetti appartenenti a classi diverse, come prevedibile, l'applicazione non riesce a classificare in modo corretto l'immagine. La soluzione ideale potrebbe essere quella di separare ogni singolo oggetto presente nella scena e su di esso effettuare la classificazione. Le prestazioni inoltre dipendono molto dalla immagini utilizzate per addestrare il modello: infatti, se le immagini hanno uno sfondo non uniforme, il classificatore è maggiormente portato a commettere errori. Se nell'immagine lo sfondo contiene molte features interessanti, quando l'algoritmo andrà a raggruppare le features comuni, non è in grado di generalizzare ed inserisce dei dati che in realtà non appartengono alla classe, compromettendo l'accuratezza che si ottiene nella fase di riconoscimento di immagini nuove.

SIFT permette di avere un'elevata accuratezza e precisione nell'identificazione delle features, ma purtroppo ha il limite di essere molto lento in fase di esecuzione. Per poter utilizzare questa applicazione in un sistema real-time occorre utilizzare altri tipi di descrittori come SURF o un descrittore binario che permette di raggiungere un risultato similare, ma con un carico computazionale decisamente inferiore.

Nonostante i risultati ottenuti, il modello Bag of Visual Words presenta due limiti: il primo è la presenza di molte codewords rumorose (non inerenti la classe) all'interno del dizionario, che può causare un degrado dell'accuratezza nel processo di classificazione. Inoltre, il vocabolario essendo costruito aggregando i descrittori dei keypoints attraverso l'algoritmo k-means, non porta a costruire cluster ben strutturati poichè utilizza tecniche probabilistiche. Nel caso di dati di grandi dimensioni tali algoritmi richiedono un tempo di calcolo estremamente elevato. Questo ci porta a utilizzare algoritmi approssimativi, che accelerano il processo di clustering, ma che introducono più rumore nelle parole visive. L'altro limite è causato dalla perdita delle informazioni spaziali, dal momento che questo modello si basa su un semplice conteggio delle occorrenze di Codewords nelle immagini. Non esiste un'associazione geometrica precisa, questo porta ad una diminuzione della accuratezza del modello, infatti due oggetti differenti possono condividere parole simili, ma la disposizione di queste parole è specifica per

ciascun oggetto.

7.2 Sviluppi futuri

L'applicazione realizzata è stata progettata per essere utilizzata in futuro su un sistema embedded, in grado di riconoscere in tempo reale gli oggetti presenti nella scena. Attraverso la stereocamera sviluppata dal gruppo di ricerca del Dipartimento di Ingegneria dell'Università di Bologna, presso il quale ho svolto la tesi, è possibile sfruttare le informazioni date dalla profondità per migliorare la classificazione di oggetti. Attualmente è stata implementata su FPGA la libreria in grado di estrarre punti salienti dall'immagine direttamente dalla sterecamera, utilizzando il descrittore binario BRISK. Integrando quest'ultima libreria con l'applicazione che ho realizzato, teoricamente, è possibile riconoscere oggetti presenti nella scena in tempo reale.

Per migliorare l'accuratezza del modello implementato, l'ideale sarebbe utilizzare immagini, o parti di immagini, precedentemente segmentate, estraendo tutto quello superfluo che non corrisponde all'oggetto della classe. Questo è possibile attraverso la stereocamera. Avendo a disposizione immagini in tre dimensioni è possibile utilizzare la profondità per separare oggetti all'interno della scena e poter eliminare le informazioni aggiuntive che andrebbero ad alterare la classificazione.

Un altro approccio che potrebbe migliorare il modello BoW, potrebbe essere quello di associare informazioni geometriche spaziali all'oggetto attraverso Implicit Shape Model, analizzata nel Capitolo 4.4 aggiungendo ulteriori informazioni relative alla profondità. Ogni oggetto rappresentato nella scena ha associato ad esso informazioni precise: ad esempio se prendiamo il viso di una persona, riusciamo ad identificare immediatamente alcune caratteristiche distintive: occhi, naso, bocca, capelli, orecchie. Ad ognuna di queste parti sono associate relazioni spaziali: la bocca si trova nella parte inferiore del viso, disposta in posizione centrale, e sappiamo che si trova sicuramente davanti alle orecchie. Basandosi sulla posizione e sulla profondità è possibile migliorare l'accuratezza e ottenere risultati molto più affidabili.

Infine un modo per migliorare ulteriormente le prestazioni, sarebbe quello di applicare delle tecniche differenti in grado di eliminare dal dizionario le parole (codewords) inutili o ridondanti che portano a un peggioramento delle presta-

zioni. Per esempio potrebbe essere interessante utilizzare una nuova tecnica di classificazione che prenda in considerazione l'aspetto sequenziale delle frasi, utilizzando modelli di linguaggio (LM). Questo strumento è molto efficiente ed efficace in ambito di riconoscimento vocale e analisi del testo. È un modello generativo che permette di spiegare perchè alcune parti dei dati sono simili tra loro. Questa tecnica è chiamata *Probabilistic Latent Semantic Analysis* (pLSA) e il suo obiettivo principale è quello di modellare le informazioni di co-occorrenza in un quadro probabilistico per scoprire la struttura semantica sottostante dei dati. Una possibile soluzione è proposta nel seguente articolo: "Language Modeling for Bag-of-Visual Words Image Categorization" [76].

Il progetto che ho creato è abbastanza versatile in quanto, dopo aver addestrato il classificatore, è possibile utilizzare l'applicazione su un altro sistema e predire nuove immagini senza necessità di addestrare nuovamente il modello. Inoltre in futuro può essere esteso introducendo nuovi algoritmi di Machine Learning o utilizzando nuovi estrattori di features, senza dover modificare la struttura del programma.

L'applicazione realizzata credo sia molto utile, date le innumerevoli applicazioni e i notevoli margini di miglioramento, che possono essere considerati per futuri sviluppi.

Bibliografia

[1] G. Csurka, C. Bray, C. Dance, and L. Fan, "Visual categorization with bags of keypoints," *Workshop on Statistical Learning in Computer Vision, ECCV*, pp. 1–22, 2004.

[2] R. F. L. Fei-Fei and P. Perona, "Caltech-101," tech. rep., California Institute of Technology, 2004.

[3] A. P. Griffin, G. Holub, "Caltech-256," tech. rep., California Institute of Technology, 2004.

[4] K. Grauman and B. Leibe, *Visual Object Recognition*. Synthesis Lectures on Artificial Intelligence and Machine Learning, Morgan & Claypool Publishers, 2011.

[5] M. Schneider. `http://cs.brown.edu/courses/cs143/2011/results/proj1/mmschnei/`.

[6] R. Lei. `http://ryanlei.wordpress.com/2011/03/09/`.

[7] D. G. Lowe, "Distinctive image features from scale-invariant keypoints," *Int. J. Comput. Vision*, vol. 60, pp. 91–110, Nov. 2004.

[8] H. Bay, A. Ess, T. Tuytelaars, and L. Van Gool, "Speeded-up robust features (surf)," *Comput. Vis. Image Underst.*, vol. 110, pp. 346–359, June 2008.

[9] C. Schaeffer, "A comparison of keypoint descriptors in the context of pedestrian detection: Freak vs. surf vs. brisk."

[10] E. Rosten and T. Drummond, "Machine learning for high-speed corner detection," in *Proceedings of the 9th European Conference on Computer Vision - Volume Part I*, ECCV'06, (Berlin, Heidelberg), pp. 430–443, Springer-Verlag, 2006.

[11] Cyc, "Linear classifier." http://en.wikipedia.org/wiki/File:Svm_separating_hyperplanes.png, Feb. 2008.

[12] P. D. Maltoni, "Classificazione supervisionata," (Università di Bologna, Italy), 2011.

[13] J. C. Epidemiol, "Artificial neural network." http://tekhabit.wordpress.com/2013/04/15/artificial-neural-network/p, Apr. 2013.

[14] E. Pasero and L. Mesin, "Artificial neural networks for pollution forecast." http://www.intechopen.com/books/air-pollution/artificial-neural-networks-for-pollution-forecast, Aug. 2010.

[15] T. Ivry and S. Michal, "License plate number recognition using artificial neural network." http://www.cs.bgu.ac.il/~icbv061/StudentProjects/ICBV061/ICBV-2006-1-TorIvry-ShaharMichal/index.php, 2006.

[16] N. Lazzarini, "Tecniche di apprendimento automatico per l'identificazione dell'iperaldosteronismo primario," Master's thesis, Dipartimento di Ingegneria dell'Informazione, Università degli Studi di Padova, Italia, 2012.

[17] MATLAB, "Support vector machines (svm)." http://www.mathworks.se/help/stats/support-vector-machines-svm.html.

[18] A. Criminisi and J. Shotton, *Decision Forests for Computer Vision and Medical Image Analysis*. Springer Publishing Company, Incorporated, 2013.

[19] P. N. Stuart J. Russell, *Intelligenza artificiale vol.2: Un approccio moderno*, vol. 2. Pearson Education Press, 2005.

[20] X. Li, "Boosting - one of combining models." Department of Computer and Information Sciences - Philadelphia, USA - (University Lecture), 2008.

[21] D.-W. blog, "A dictionary of weak classifiers." http://dwave.wordpress.com/2009/08/05/iii-a-dictionary-of-weak-classifiers/.

[22] Chire, "Clustering kmeans." http://it.wikipedia.org/wiki/K-means, Oct. 2010.

[23] M. Marszałek, C. Schmid, H. Harzallah, and J. van de Weijer, "Learning object representations for visual object class recognition," oct 2007. Visual Recognition Challange workshop, in conjunction with ICCV.

[24] L. Fei-Fei, "Bag-of-words models." University of Standford - (University Lecture), 2005.

[25] J. Kim, B.-S. Kim, and S. Savarese, "Comparing image classification methods: K-nearest-neighbor and support-vector-machines," in *Proceedings of the 6th WSEAS international conference on Computer Engineering and Applications, and Proceedings of the 2012 American conference on Applied Mathematics*, AMERICAN-MATH'12/CEA'12, (Stevens Point, Wisconsin, USA), pp. 133–138, World Scientific and Engineering Academy and Society (WSEAS), 2012.

[26] R. Fergus, *Visual Object Category Recognition*. PhD thesis, University of Oxford, 2005.

[27] R. Fergus, P. Perona, and A. Zisserman, "Object class recognition by unsupervised scale-invariant learning," in *Cognitive Vision Systems* (H. Nagel, ed.), Kluwer Academic Publishers, in press – 2006.

[28] B. Leibe, A. Leonardis, and B. Schiele, "Robust object detection with interleaved categorization and segmentation," *International Journal of Computer Vision*, vol. 77, pp. 259–289, May 2008.

[29] B. Leibe, A. Leonardis, and B. Schiele, "Combined object categorization and segmentation with an implicit shape model," in *In ECCV workshop on statistical learning in computer vision*, pp. 17–32, 2004.

[30] C. Schmid, "Bag-of-features for category classification." INRIA Summer School 2010 - (University Lecture), 2010.

[31] C. Grana, D. Borghesani, M. Manfredi, and R. Cucchiara, "A fast approach for integrating orb descriptors in the bag of words model," in *Proceedings of IS&T/SPIE Electronic Imaging: Multimedia Content Access: Algorithms and Systems*, (San Francisco, California, US), Feb. 2013.

[32] M. Thomas G. Tape, "Interpreting diagnostic tests,"

[33] E. funded PASCAL2 Network of Excellence, "The pascal visual object classes homepage." http://pascallin.ecs.soton.ac.uk/challenges/VOC/voc2010/, Oct. 2010.

[34] J. Sivic and A. Zisserman, "Video google: A text retrieval approach to object matching in videos," in *Proceedings of the Ninth IEEE International Conference on Computer Vision - Volume 2*, ICCV '03, (Washington, DC, USA), pp. 1470–, IEEE Computer Society, 2003.

[35] S. J. D. Prince, *Computer Vision: Models, Learning, and Inference*. New York, NY, USA: Cambridge University Press, 1st ed., 2012.

[36] F.-F. Li and P. Perona, "A bayesian hierarchical model for learning natural scene categories," in *Proceedings of the 2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05) - Volume 2 - Volume 02*, CVPR '05, (Washington, DC, USA), pp. 524–531, IEEE Computer Society, 2005.

[37] C. Harris and M. Stephens, "A combined corner and edge detector," in *In Proc. of Fourth Alvey Vision Conference*, pp. 147–151, 1988.

[38] K. Mikolajczyk and C. Schmid, "Indexing based on scale invariant interest points," in *In Proceedings of the 8th International Conference on Computer Vision*, pp. 525–531, 2001.

[39] K. Mikolajczyk, T. Tuytelaars, C. Schmid, A. Zisserman, J. Matas, F. Schaffalitzky, T. Kadir, and L. V. Gool, "A comparison of affine region detectors," *Int. J. Comput. Vision*, vol. 65, pp. 43–72, Nov. 2005.

[40] S. Leutenegger, M. Chli, and R. Y. Siegwart, "Brisk: Binary robust invariant scalable keypoints," in *Proceedings of the 2011 International Conference on Computer Vision*, ICCV '11, (Washington, DC, USA), pp. 2548–2555, IEEE Computer Society, 2011.

[41] D. G. Lowe, "Object recognition from local scale-invariant features," in *Proceedings of the International Conference on Computer Vision-Volume 2 - Volume 2*, ICCV '99, (Washington, DC, USA), pp. 1150–, IEEE Computer Society, 1999.

[42] D. Pagliari, "Rilievo di traiettorie urbane con approccio fotogrammetrico: Una sperimentazione orientata all'automazione," Master's thesis, POLITECNICO DI MILANO, University of Milan, Italy, 2010.

[43] P. Viola and M. Jones, "Rapid object detection using a boosted cascade of simple features," pp. 511–518, 2001.

[44] D. Balducci, "Analisi comparata di algoritmi per l'individuazione di keypoints in immagini: Good features to track, surf," 2011.

[45] M. Calonder, V. Lepetit, C. Strecha, and P. Fua, "Brief: Binary robust independent elementary features," in *Proceedings of the 11th European Conference on Computer Vision: Part IV*, ECCV'10, (Berlin, Heidelberg), pp. 778–792, Springer-Verlag, 2010.

[46] R. Ortiz, "Freak: Fast retina keypoint," in *Proceedings of the 2012 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, CVPR '12, (Washington, DC, USA), pp. 510–517, IEEE Computer Society, 2012.

[47] E. Rublee, V. Rabaud, K. Konolige, and G. Bradski, "Orb: An efficient alternative to sift or surf," in *Proceedings of the 2011 International Conference on Computer Vision*, ICCV '11, (Washington, DC, USA), pp. 2564–2571, IEEE Computer Society, 2011.

[48] C. M. Bishop, *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Secaucus, NJ, USA: Springer-Verlag New York, Inc., 2006.

[49] C. Cortes and V. Vapnik, "Support-vector networks," *Machine Learning*, vol. 20, no. 3, pp. 273–297, 1995.

[50] N. Cristianini and J. Shawe-Taylor, *An Introduction to Support Vector Machines and Other Kernel-based Learning Methods*. Cambridge University Press, 1 ed., 2000.

[51] P. Medici, "Adaptive boosting," (Department of Information Engineering of University of Parma and co-founder of VisLab).

[52] Y. Freund and R. E. Schapire, "A short introduction to boosting," in *In Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence*, pp. 1401–1406, Morgan Kaufmann, 1999.

[53] L. Breiman, "Random forests," *Mach. Learn.*, vol. 45, pp. 5–32, Oct. 2001.

[54] M. Weber, M. Welling, and P. Perona, "Unsupervised learning of models for recognition," in *In ECCV*, pp. 18–32, 2000.

[55] M. C. Burl, M. Weber, and P. Perona, "A probabilistic approach to object recognition using local photometry and global geometry," 1998.

[56] Y. Kamiya, T. Takahashi, I. Ide, and H. Murase, "A multimodal constellation model for object category recognition," in *Proceedings of the 15th International Multimedia Modeling Conference on Advances in Multimedia Modeling*, MMM '09, (Berlin, Heidelberg), pp. 310–321, Springer-Verlag, 2008.

[57] P. F. Felzenszwalb and D. P. Huttenlocher, "Pictorial structures for object recognition," *Int. J. Comput. Vision*, vol. 61, pp. 55–79, Jan. 2005.

[58] E. Nowak, F. Jurie, and B. Triggs, "Sampling strategies for bag-of-features image classification," in *European Conference on Computer Vision*, Springer, 2006.

[59] J. Yang, Y.-G. Jiang, A. G. Hauptmann, and C.-W. Ngo, "Evaluating bag-of-visual-words representations in scene classification," in *Proceedings of the international workshop on Workshop on multimedia information retrieval*, MIR '07, (New York, NY, USA), pp. 197–206, ACM, 2007.

[60] C.-F. Tsai, "Bag-of-words representation in image annotation: A review," *ISRN Artificial Intelligence, vol. 2012, Article ID 376804, 19 pages*, 2012.

[61] L. Fei-Fei, R. Fergus, and P. Perona, "Learning generative visual models from few training examples: An incremental bayesian approach tested on 101 object categories," *Comput. Vis. Image Underst.*, vol. 106, pp. 59–70, Apr. 2007.

[62] E. L. Allwein, R. E. Schapire, and Y. Singer, "Reducing multiclass to binary: A unifying approach for margin classifiers," in *Proceedings of the 17th International Conference on Machine Learning (ICML-2000)*, pp. 9–16, Morgan Kaufmann, 2000.

[63] J. H. Friedman, "Another approach to polychotomous classification," tech. rep., Department of Statistics, Stanford University, 1996.

[64] C.-W. Hsu and C.-J. Lin, "A comparison of methods for multiclass support vector machines," *IEEE Transactions on Neural Networks*, vol. 13, no. 2, pp. 415–425, 2002.

[65] T. G. Dietterich and G. Bakiri, "Solving multiclass learning problems via error-correcting output codes," *J. Artif. Int. Res.*, vol. 2, pp. 263–286, Jan. 1995.

[66] J. Zhu, H. Zou, S. Rosset, and T. Hastie, "Multi-class adaboost," 2009.

[67] D. Benbouzid, R. Busa-Fekete, N. Casagrande, F.-D. Collin, and B. Kégl, "Multiboost: A multi-purpose boosting package," *J. Mach. Learn. Res.*, vol. 13, pp. 549–553, Mar. 2012.

[68] J. Zhang, M. Marszałek, S. Lazebnik, and C. Schmid, "Local features and kernels for classification of texture and object categories: a comprehensive study," *International Journal of Computer Vision*, vol. 73, pp. 213–238, jun 2007.

[69] G. Parmigiani, "Algoritmi di visione per l'identificazione di ostacoli in immagini 3d generate da telecamera stereo," Master's thesis, Dipartimento Di Ingegneria Elettrica e dell'Informazione "Guglielmo Marconi" - DEI, Alma Mater Studiorum - Università di Bologna, Italia, 2014.

[70] M. Sokolova and G. Lapalme, "A systematic analysis of performance measures for classification tasks," *Inf. Process. Manage.*, vol. 45, pp. 427–437, July 2009.

[71] R. M. Everson and J. E. Fieldsend, "Multi-class roc analysis from a multi-objective optimisation perspective," *Pattern Recogn. Lett.*, vol. 27, pp. 918–927, June 2006.

[72] S. Lazebnik, C. Schmid, and J. Ponce, "Beyond bags of features: Spatial pyramid matching for recognizing natural scene categories," in *Proceedings of the 2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognition - Volume 2*, CVPR '06, (Washington, DC, USA), pp. 2169–2178, IEEE Computer Society, 2006.

[73] J. Zhang, S. Lazebnik, and C. Schmid, "Local features and kernels for classification of texture and object categories: a comprehensive study," *International Journal of Computer Vision*, vol. 73, p. 2007, 2007.

[74] J. Hou, J. Kang, and N. Qi, "On vocabulary size in bag-of-visual-words representation," in *Proceedings of the 11th Pacific Rim Conference on Advances in Multimedia Information Processing: Part I*, PCM'10, (Berlin, Heidelberg), pp. 414–424, Springer-Verlag, 2010.

[75] T. Deselaers, L. Pimenidis, and H. Ney, "Bag-of-visual-words models for adult image classification and filtering," in *Proc. 19th International Conference on Pattern Recognition ICPR 2008*, pp. 1–4.

[76] P. Tirilly, V. Claveau, and P. Gros, "Language modeling for bag-of-visual words image categorization," in *Proceedings of the 2008 International Conference on Content-based Image and Video Retrieval*, CIVR '08, (New York, NY, USA), pp. 249–258, ACM, 2008.

Rigraziamenti

Giunto al termine di questo lungo percorso di studi universitari, desidero ringraziare chi, ha contribuito a raggiungere questo ambito traguardo. In poche parole è difficile ricordare tutte le persone che in questi anni mi sono state vicino.

Prima di tutto mi sembra doveroso ringraziare il Professor Stefano Mattoccia che mi ha permesso di svolgere questo elaborato di tesi, mi ha sempre dato fiducia, mi ha supportato e guidato in tutto questo percorso. Inoltre mi ha dato l'opportunità di lavorare su un tema molto affascinante. Vorrei ringraziare Dott. Marco Casadio, Dott. Ilario Marchio e in particolare Paolo Macrì per il loro aiuto nei momenti di difficoltà.

Un doveroso ringraziamento va alla mia famiglia che mi ha sempre supportato in questi anni, dandomi fiducia e aiutandomi: Vasco, Alessandro e Annalisa e Patrizia (che mi segue dall'alto). Per la pazienza e la sopportazione, senza il loro contributo sicuramente non avrei potuto raggiungere questo traguardo. Inoltre un ringraziamento particolare ad Alessandro a Annalisa per avermi aiutato nella correzione di questo elaborato.

Vorrei ringraziare Elise che da due anni e mezzo mi è accanto in ogni istante della mia vita, appoggiando ogni mia decisione, sopportandomi e ascoltandomi nei momenti di sconforto e stimolandomi a dare il massimo ogni giorno.

Un grazie di cuore gli amici che conosco da "una vita" Simona A., Sara, Cecilia, Simona S., Luca C., Luca P., Natascia e Anna che non hanno mai dubitato nelle mie capacità.

Un abbraccio a tutti gli amici che ho conosciuto in questo percorso universitario magistrale, con cui ho condiviso momenti di studio. Un ringraziamento anche a Filippo Cantucci che ha letto la prima parte della mia tesi dandomi preziosi consigli e che sicuramente proseguirà questo progetto.

Un ringraziamento particolare a tutti gli amici che ho conosciuto nei 10 mesi trascorsi a Bergen (Norvegia) in Erasmus. Sono stati momenti indimenticabili: Francesco, Fabrizio, Alessandra, Caterina, Consuelo, Giulia S., Michele, Luka e tutti gli altri ragazzi tedeschi e francesi. Giornate intere passate a studiare nelle aule di Informatica per riuscire a superare gli assignment. Le indimenticabili feste del venerdì nella cucina del blocco 17C e poi il Klubb Fantoft. Le passeggiate sui monti di Bergen.

Un grazie a tutto lo staff dell'Antica Osteria del Borgo (cuochi e camerieri) per le lunghe serate di lavoro trascorse insieme ridendo e scherzando anche nei momenti di difficoltà.

Vorrei ringraziare anche i ragazzi della Consiglio Direttivo della Polisportiva Borzanese per questi 5 anni trascorsi insieme.

Infine un ultimo ringraziamento a tutte le persone che leggeranno questa tesi, scusandomi per eventuali errori.