Università degli studi di Firenze
Facoltà di Ingegneria

Tesi di Laurea in

Ingegneria Informatica

Riconoscimento di loghi TV in sequenze video (TV logo detection in video streams)

Relatore

Prof. Alberto Del Bimbo

Candidato

Giulia Fontanini

Correlatori

Ing. Marco Bertini

Ing. Lamberto Ballan

Anno Accademico 2011/2012

L'informatica non riguarda i computer più di quanto l'astronomia riguardi i telescopi.

Edsger Wybe Dijkstra

Indice

1 Introduzione **4**

1.1 Scenario 4

1.2 Obiettivi della tesi 6

1.3 Organizzazione dei capitoli 7

2 Stato dell'arte **8**

2.1 Trademark Matching and Retrieval in Sports Video Databases 8

2.2 A Robust Method for TV Logo Tracking in Video Streams 12

2.3 Real-Time Opaque and Semi-Transparent TV Logos Detection 13

2.4 Shape Matching and Object Recognition Using Shape Context 15

3 L'algoritmo **19**

3.1 Estrazione delle features 19

3.2 SIFT: Scale Invariant Feature Transform 21

3.2.1 Identificazione degli estremi locali nello scale-space 22

3.2.2 Localizzazione dei keypoints 23

3.2.3 Assegnazione dell'orientazione 26

3.2.4 Generazione dei descrittori 27

3.3 LBP: Local Binary Pattern 28

3.3.1 Confronto tramite istogrammi 29

3.4 Matching tra le features 30

3.4.1 Applicazione di una maschera nelle regioni di interesse 31

3.4.2 K-d-tree 33
3.4.3 Ratio test e RANSAC test 35
3.4.4 Salvataggio dei valori in file CSV 36
3.5 Classificatore SVM lineare 37
3.5.1 Trainer 39
3.5.2 Tester 40

4 Risultati sperimentali **41**
4.1 Presentazione del dataset 41
4.2 Estrazione delle features 42
4.3 Analisi superivisonata dei match 45
4.3.1 Scelta del campionamento dei video 45
4.3.2 Precision e Recall 46
4.3.3 Prestazioni globali 53
4.4 L'addestramento del classificatore 53
4.4.1 Scelta della finestra e overlap 54
4.4.2 Testing dei loghi opachi 55
4.4.3 Testing dei loghi trasparenti 57
4.5 Esempi 59

5 Conclusioni **63**

Bibliografia **65**

Capitolo 1

Introduzione

1.1 Scenario

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. Mentre l'occhio umano riconosce e interpreta un oggetto appartenente ad un'immagine in maniera immediata, il processo di detection da parte di un calcolatore è molto più elaborato.

In particolare, 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. Rispetto a questo tipo di criticità, si rende perciò necessario sviluppare un sistema robusto che tenga conto di ogni possibile variazione dell'oggetto ricercato.

Un'area di interesse molto importante riguarda la ricerca di immagini

artificiali come loghi, icone, marchi, ecc [1]. Un oggetto di questo tipo può essere descritto, nella ricerca per similarità con un altro oggetto, da tre caratteristiche principali:

- forma
- struttura (ovvero la disposizione di ogni singolo elemento dell'oggetto)
- semantica (ovvero il significato o l'interpretazione che si associa all'immagine)

Uno degli intenti principali in questo ambito nasce dal problema dell'infrangimento del copyright e, più in generale, dall'esigenza di monitorare l'utilizzo di tali trademark per differenti scopi.

Si può ad esempio pensare ad un'azienda che voglia creare un proprio marchio, ma senza rischiare che sia troppo simile ad uno già esistente, o anche, inversamente, ad un'impresa che già ne possiede uno e voglia monitorare quelli 'nuovi'.

Analogamente, risulta di non meno interesse il caso in cui un'azienda che opera nel settore del broadcasting voglia assicurarsi che i propri prodotti non vengano trasmessi su siti web di condivisione video senza il proprio consenso. Piattaforme come Youtube o Dailymotion, la cui grande espansione negli ultimi anni ha permesso il proliferare di questi rischi, potrebbero voler monitorare esse stesse i video caricati dai propri utenti, al fine di evitare infrangimenti di proprietà intellettuale.

A quest'ultima problematica in particolare si lega quindi l'interesse di tenere sott'occhio i video che vengono di giorno in giorno caricati sulla rete, tramite operazioni di confronto tra i loghi televisivi che vogliono essere cercati e quelli che potrebbero essere presenti in tali video.

Queste operazioni vengono di norma effettuate tramite approcci supervisionati da parte dell'uomo e, benchè ciò abbia il pregio di essere mediamente infallibile dal punto di vista degli errori (come già detto, l'occhio umano riconosce gli oggetti con poco sforzo), d'altra parte la grande quantità di informazioni presenti sulla rete rende quasi impossibile monitorare tutto in tempi accettabili.

Proprio per tale motivo, può risultare necessario far svolgere questo tipo di operazioni ad un calcolatore, che, in maniera automatica e senza supervisione (se non in fase finale, quando gran parte del dataset è stato scremato), possa sostituirsi autonomamente nella ricerca della similarità tra loghi.

1.2 Obiettivi della tesi

L'intento di questa tesi è quello di automatizzare un sistema capace di effettuare il riconoscimento di un logo qualsiasi (piccola immagine in forma digitale) all'interno di un dataset di video presi dalla rete.

In particolare, il sistema procede all'estrazione di caratteristiche salienti (dette *features*) dell'immagine desiderata, per poi controllare in ciascun frame del generico video analizzato se queste siano presenti o meno. Nel caso in cui una percentuale accettabile di features venga ritrovata in uno o più frame, si parla di *match*, ovvero di corrispondenza.

Successivamente viene stabilito, attraverso un classificatore SVM precedentemente addestrato in maniera supervisionata (in base a risultati già ottenuti), se i match evidenziano una corrispondenza giusta o sbagliata, decretando definitivamente la presenza o meno del logo nel video.

La tipologia di loghi analizzati in questa tesi (appartenenti a Mediaset, Rai, La7, Istituto Luce) presenta diverse variabilità, come forma, colore, trasparenza, grandezza, ecc.., che fanno sperare che il sistema possa funzionare in maniera generale con qualsiasi altro tipo di logo.

1.3 Organizzazione dei capitoli

Nel **capitolo 2** verranno introdotti i problemi, le soluzioni proposte e i risultati sperimentali di lavori precedentemente svolti riguardanti la detection di immagini artificiali.

Nel **capitolo 3** verrà descritto l'algoritmo utilizzato per raggiungere l'obiettivo prefisso di questa tesi, descrivendo le possibili soluzioni implementative, adottate o meno. In particolare farò riferimento al funzionamento dei descrittori SIFT e LBP e del classificatore SVM.

Il **capitolo 4** esporrà i risultati sperimentali ottenuti attraverso l'approccio utilizzato, ai quali si affiancheranno esempi esplicativi.

Nelle **conclusioni** riassumo gli obiettivi raggiunti e le possibili applicazioni future.

Capitolo 2

Stato dell'arte

In questo capitolo vengono introdotti alcuni articoli significativi riguardanti diverse problematiche legate al riconoscimento di loghi, marchi o forme artificiali in generale, con relative soluzioni proposte, sia a livello teorico, che implementativo.

2.1 Trademark Matching and Retrieval in Sports Video Databases

In questo articolo [2], viene presentato il problema del rilevamento e riconoscimento dei trademarks che appaiono nei video sportivi. Una delle ragioni principali per cui sviluppare una metodologia che permetta questo tipo di detection è legata a motivi economici: dato l'alto costo previsto per piazzare cartelloni pubblicitari, banners e ogni altro tipo di pubblicità indiretta in contesti sportivi come campi da calcio e da tennis, piste di formula uno, ecc., le aziende sono fortemente interessate a verificare l'effettiva visibilità dei prodotti sponsorizzati.

Figura 2.1: Esempi di trademarks. In alto la versione grafica, in basso i trademark estratti da video.

Questo problema si lega in qualche modo a quello dell'infrangimento del copyright nella creazione di nuovi loghi, con la differenza che in quest'ultimo caso si suppone che le immagini abbiano una qualità accettabile e non subiscano distorsioni.

I trademark all'interno di un video sportivo, infatti, sono spesso caratterizzati da:

- deformazione prospettica (dovuta al posizionamento della camera)
- sfocatura (dovuta al movimento della camera)
- occlusione (dovuta, ad esempio, dalla presenza di giocatori in campo che coprono parzialmente il banner pubblicitario)

Figura 2.2: Esempi di marchi caratterizzati rispettivamente da cattiva illuminazione (Coca-Cola), deformazione (Starbucks) e occlusione (Pepsi).

Per questo motivo, i trademarks vengono rappresentati tramite descrittori SIFT (Scale Invariant Features Transform), ovvero descrittori robusti

al cambiamento di scala, illuminazione, prospettiva e rotazione. Poichè descrivono caratteristiche *locali*, si prestano bene all'identificazione di un trademark anche in presenza di occlusioni parziali.

Il metodo proposto nell'articolo prevede che ciascun marchio T_i sia quindi rappresentato da un insieme di N_i punti SIFT:

$$T_i = \{(x_k^t, y_k^t, s_k^t, d_k^t, \mathbf{O}_k^t)\}, \text{ for } k \in \{1, \ldots, N_i\},$$

dove x_k^t, y_k^t, s_k^t, e d_k^t sono, rispettivamente, la posizione x e y, la scala e la direzione dominante del k-esimo punto. L'elemento O_k^t è un vettore di 128 elementi che caratterizza il descrittore SIFT in quel punto.

Analogamente, ciascun frame V_i di un video è rappresentato da altrettanti M_i punti SIFT:

$$V_i = \{(x_k^v, y_k^v, s_k^v, d_k^v, \mathbf{O}_k^v)\}, \text{ for } k \in \{1, \ldots, M_i\},$$

dove gli apici t e v differenziano i feature points rispettivamente del trademark e del frame video.

L'algoritmo si sviluppa in questo modo: dopo aver rappresentato adeguatamente trademark e frame video attraverso i descrittori SIFT, si procede alla comparazione tra le features. Per ogni punto k rilevato nel trademark T_j, si trovano i due punti più simili nel generico frame video V_i, calcolati con la distanza euclidea, in questo modo:

$$\begin{aligned} N_1(T_j^k, V_i) &= \min_q ||\mathbf{O}_q^v - \mathbf{O}_k^t|| \\ N_2(T_j^k, V_i) &= \min_{q \neq N_1(T_j^k, V_i)} ||\mathbf{O}_q^v - \mathbf{O}_k^t||. \end{aligned}$$

Dopodichè, viene calcolato il rapporto tra le distanze dal primo e dal secondo punto.

$$M(T_j^k, V_i) = \frac{N_1(T_j^k, V_i)}{N_2(T_j^k, V_i)}, \tag{2.1}$$

I punti candidati ad essere dei 'buoni' match sono quelli il cui rapporto è minore di una certa soglia (secondo i risultati sperimentali, il valore migliore è 0.8), ovvero

$$M_i^j = \{k \mid M(T_j^k, V_i) < \tau_1\}.$$

L'idea di base di questo ragionamento è che 'buoni match' tendono ad avere il primo punto più vicino (corretto) significativamente distante dal secondo trovato (scorretto), mentre falsi match hanno un alto numero di punti simili tra loro, data la grandezza dello spazio in cui vengono estratte le features.

Il test finale per decretare l'esistenza del trademark nel frame video V_i consiste nel verificare che un adeguata percentuale di punti abbia avuto corrispondenza. Ciò si verifica normalizzando, rispetto al numero di features totali del trademark, il numero di punti che hanno passato il 'ratio-test', ponendolo maggiore di una certa soglia (in questo caso a 0.2):

$$\frac{|M_i^j|}{|T_j|} > \tau_2$$

Infine, viene calcolata l'area del marchio in questo modo: si calcolano le coordinate del centro rispetto alla nuvola di punti e si stabiliscono le distanze di ogni punto dal centro per identificarne l'area.

2.2 A Robust Method for TV Logo Tracking in Video Streams

L'obiettivo di questo articolo [3] è quello di proporre un metodo robusto per determinare o meno la presenza di loghi in stream video (senza che vi sia comparazione con uno già esistente: viene semplicemente stabilito se nel video è presente un logo o no), considerando soprattutto il caso più critico in cui il logo sia semi-trasparente. Questa caratteristica rende l'oggetto mediamente più difficile da localizzare, a causa del rumore di background a cui è sottoposto e l'assenza di contorni dettagliati.

La tecnica proposta si basa sulla detection del gradiente multispettrale dell'immagine (chiamato anche gradiente generalizzato) e, più in generale, su un'osservazione temporale del video: invece di un'estrazione fatta sul singolo frame, si prendono in considerazione più frame consecutivi, che permettono di delineare maggiormente il contorno costante del logo.

Il procedimento è il seguente:

1. Viene elaborata l'immagine attraverso il *metodo Otsu*, che consiste nel calcolare una soglia ottima dell'istogramma per poter binarizzare l'immagine (in questo modo, vengono enormemente accentuati i contorni dell'immagine).

Figura 2.3: Un esempio di immagine binarizzata col metodo Ostu.

2. Il concetto di gradiente generalizzato si riferisce all'estensione temporale del calcolo di un tradizionale gradiente da una singola immagine ad una sequenza di immagini. In questo caso viene analizzata la trasformazione del gradiente: nel caso di un logo semi-trasparente, un numero giusto di frame aiuta a sommare i contributi dei contorni e ad eliminare il rumore di background.

3. Infine, viene nuovamente binarizzata l'immagine del gradiente generalizzato ottenuto, per costruire un modello da comparare con finestre scorrevoli sovrapposte del video. La presenza o meno del logo viene stabilita in base a una soglia di match.

2.3 Real-Time Opaque and Semi-Transparent TV Logos Detection

Un requisito spesso fondamentale nell'analisi di stream video è la possibilità di estrarre informazioni real-time. Questo intento è mostrato nell'articolo [4], il cui scopo è il riconoscimento di loghi (opachi, semi-trasparenti e parzialmente animati) da video di trasmissioni televisive.

Proprio per la necessità di elaborare informazioni in tempo reale, l'algoritmo si presenta economico dal punto di vista della memoria utilizzata e richiede una bassa potenza del processore per le operazioni che vengono effettuate.

La tecnica di estrazione viene effettuata in diversi passi:

1. Inizialmente, nei frame che vengono analizzati, viene tolta qualsiasi informazione riguardante il colore: l'immagine passa in scala di grigi,

poichè qualsiasi altra informazione aggiuntiva su ciascun pixel non contribuisce, ad esempio, a rilevare loghi semi-trasparenti, il cui sfondo cambia in continuazione.

2. Da ciascun frame vengono poi estratte 4 regioni di interesse, poste agli angoli, all'interno di una delle quali si presume che si trovi il logo. Quest'operazione, in relazione alle successive, velocizza l'intero processo, in quanto viene ridimensionata la dimensione di frame da elaborare.

Figura 2.4: Estrazione degli angoli da un frame video.

3. Per rilevare i contorni del logo, che rimangono costanti per la durata del video, e rimuovere lo sfondo, invece variabile, viene usata la tecnica del *time averaging*, che enfatizza quei pixel che non variano nel corso del tempo, o variano leggermente. Considerando un periodo di tempo Δt, dato approssimativamente da 30 frames, ovvero 1 secondo, viene calcolato:

$$\bar{F}(j,k,t) = \frac{1}{2}[\bar{F}(j,k,t-\Delta t) + F(j,k,t)], t \geq 0$$

dove $\bar{F}(j,k,t)$ è la media rispetto al tempo dello stream video nel pixel (j,k) del frame t.

Grazie a quest'operazione, la maggior parte degli oggetti presenti nel frame viene sfocata, eccetto il logo e probabilmente altri elementi invarianti nel tempo.

4. Successivamente, si procede all'estrazione dei contorni dell'immagine ottenuta, attraverso il calcolo del gradiente, che fornisce sostanzialmente una misura della trasformazione dei pixel nell'immagine e quindi una *edge-detection*.

5. L'ultimo passo consiste nel comparare i frame via via analizzati con un dataset di loghi già esistenti e precedentemente elaborati nello stesso modo. Inizialmente, per ciascuna immagine, viene tolto il valore medio della scala di grigi dell'immagine stessa (per livellare la comparazione). Successivamente, viene calcolata la correlazione incrociata tra le due immagini comparate, per valutarne la similarità, e in un secondo momento viene normalizzata rispetto alla grandezza delle due. Il coefficiente risultante, compreso fra -1 e 1, indica una stima della somiglianza tra i due segnali: sperimentalmente viene scelto un valore maggiore di 0.73, come soglia per eliminare i falsi match.

2.4 Shape Matching and Object Recognition Using Shape Context

Un altro metodo efficacie per riscontrare la similarità tra immagini artificiali è esposto nell'articolo [5]. Qui viene presentato per la prima volta un nuovo descrittore, lo *Shape Context*, il cui scopo è quello di trovare corrispondenze tra forme simili. E' stato infatti osservato che forme simili, ma

non identiche, possono essere deformate per essere allineate, attuando una semplice trasformazione di coordinate.

L'idea di base dello Shape Context è quella di descrivere il contorno (tramite un edge-detector) di una specifica forma con n punti e creare un descrittore, per ciascuno di essi, che tenga conto della posizione relativa di tutti gli altri punti.

Lo Shape Context di un punto p_i (per $i = 1...n$) è l'istogramma relativo alle coordinate dei rimanenti $n - 1$ punti, dai cui

$$h_i(k) = \#\{q \neq p_i : (q - p_i) \in \text{bin}(k)\}$$

dove bin sono le classi dell'istogramma ricavate dal diagramma polare su cui sono calcolate le distanze.

Lo shape Context è un descrittore robusto invariante a

- trasformazioni/traslazioni (fa parte della natura del descrittore)
- cambiamento di scala (le distanze radiali vengono normalizzate dalla distanza media α di tutte le possibili n^2 coppie di punti)
- deformazioni e rumore (dimostrato sperimentalmente)

I passi per il matching delle forme per un sistema che utilizza lo Shape Context descritti nell'articolo sono:

1. Selezionare i punti di contorno di un'immagine campione e di un immagine test (in cui si vuole ritrovare l'immagine campione)
2. Calcolare lo Shape Context per ogni punto p_i del contorno, per $i = 1...n$

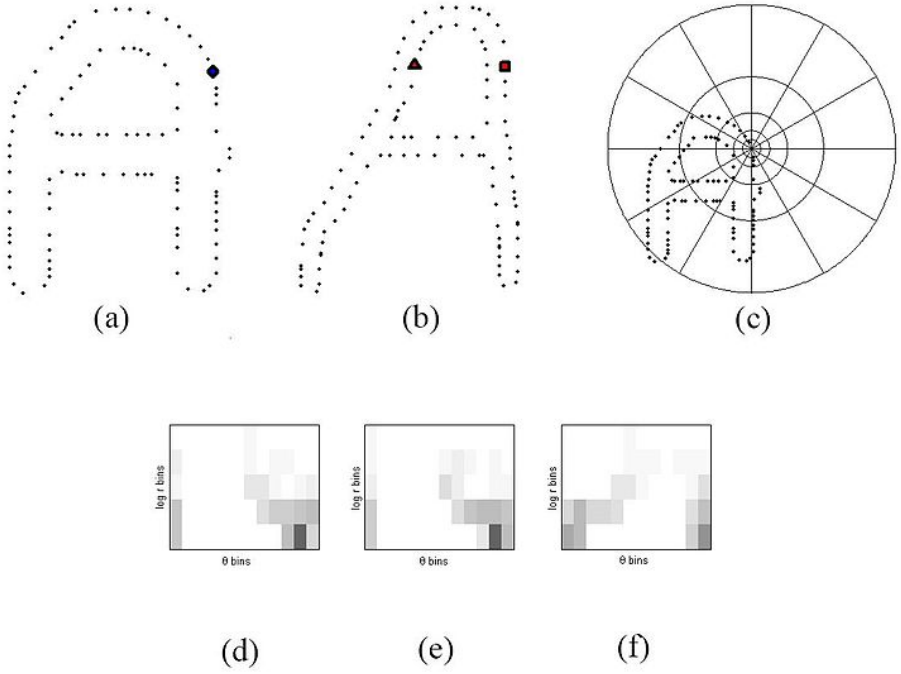

Figura 2.5: Esempi di shape context. a) e b) sono i contorni estratti da due forme simili. c) è il diagramma polare dei bins usato per calcolare il descrittore. Il punto p_i viene posizionato al centro. Gli altri $n-1$ punti si dispongono relativamente a p_i nelle classi bins identificate da modulo $\log r$ e angolo θ. d) è lo shape context del cerchio, e) quello del quadrato e f) quello del triangolo. Si nota che i primi due, nonostante le forme dei due contorni siano leggermente diverse, sono molto simili.

3. Eseguire il matching dei punti: viene scelta una trasformazione $T : R^2 \rightarrow R^2$ che modifica il contorno dell'immagine campione in modo da allinearlo a quello dell'immagine test.

4. Calcolare la distanza tra ogni coppia di punti per trovare i punti più vicini tra loro.

Nonostante questo metodo non sia strettamente collegato alla ricerca dei marchi, offre uno spunto per ripensare la comparazione tra immagini artificiali attraverso la descrizione della loro forma.

Capitolo 3

L'algoritmo

In questo capitolo viene descritta l'implementazione utilizzata nella fase progettuale per arrivare agli obiettivi desiderati. A tale descrizione si affianca una parte teorica sul funzionamento di due principali descrittori utilizzati, il SIFT (Scale Invariant Feature Transform) e l'LBP (Local Binary Patterns), seguita da una parte sul matching tra le features e una sul classificatore SVM lineare.

3.1 Estrazione delle features

Come già detto, nella descrizione di un immagine o di un oggetto, è necessario estrarre delle caratteristiche salienti che lo rappresentino, per poterlo confrontare con altri. La ricerca in questo ambito ha inizialmente proposto dei metodi che individuassero proprietà globali dell'immagine, dette *features globali*, le quali rappresentano l'oggetto nella sua totalità, come ad esempio l'istogramma di colore o la tessitura dell'immagine.

Questo tipo di descrizione però risulta poco robusta rispetto ad alcune criticità come l'occlusione parziale o la visualizzazione dell'immagine da di-

versi punti di vista, per cui in tempi più recenti l'attenzione si è spostata su una rappresentazione che si focalizzasse su punti particolari dell'immagine, detti *features locali*.

La descrizione di cosiddetti *keypoints* permette una robustezza maggiore non solo dal punto di vista delle occlusioni parziali, ma anche rispetto a rotazioni, cambiamenti di scala e deformazioni.

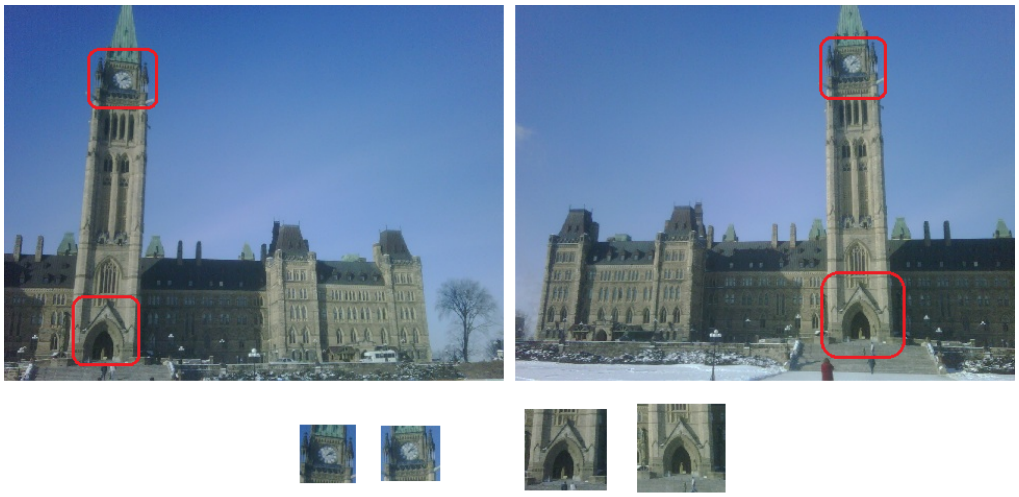

Figura 3.1: Una stessa immagine ripresa da 2 punti di vista diversi. La descrizione globale di ciascuna di esse non permetterebbe un confronto corretto, mentre è possibile notare come alcune caratteristiche locali cambino di poco e possano quindi essere comparate.

In un approccio di questo tipo bisogna distinguere due passi:

1. l'estrazione (o rilevazione) dei keypoints
2. la loro descrizione

In questa tesi abbiamo usato prima il descrittore SIFT e poi l'LBP, con differenti risultati.

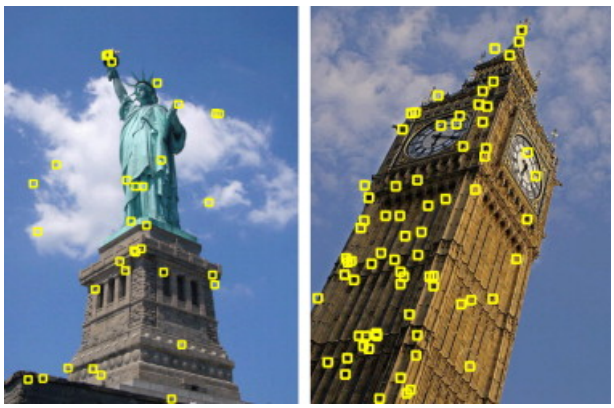

Figura 3.2: Esempio di estrazione di features locali.

3.2 SIFT: Scale Invariant Feature Transform

SIFT [6] è un algoritmo che permette di estrarre features locali da immagini. Tali features risultano essere invarianti al cambiamento di scala, alla rotazione e alla traslazione. Inoltre si presentano robuste al cambiamento del punto di vista, alla variazione delle condizioni di illuminazione, di rumore e all'occlusione parziale. A ciascun punto localizzato corrisponde un descrittore, che permette di individuarlo in immagini diverse se presente.

L'algoritmo si divide nei seguenti passi per l'individuazione e la descrizione dei keypoints da un'immagine:

1. identificazione degli estremi locali nello scale-space
2. localizzazione dei keypoints
3. assegnazione dell'orientazione
4. generazione dei descrittori

3.2.1 Identificazione degli estremi locali nello scale-space

L'individuazione degli estremi locali nello *scale-space*, ovvero in una rappresentazione multiscala dell'immagine, viene ottenuta attraverso la convoluzione del segnale con un insieme di kernel Gaussiani di varianza crescente, dove questa è chiamata parametro di scala (*scale parameter*).

Questo tipo di rappresentazione risulta utile in quanto l'immagine originale presenta molti punti da esaminare, rendendo la ricerca delle corrispondenze inefficente: sottocampionare e sfocare l'immagine, per poi raffinare la ricerca nelle versioni più dettagliate, rende l'analisi più affidabile.

In particolare, data un immagine $I(x, y)$ e un filtro gaussiano $G(x, y, \sigma)$ di varianza σ, lo scale-space associato all'immagine è dato da:

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

dove $*$ è l'operatore convoluzione e

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

La costruzione dello scale-space avviene filtrando iterativamente l'immagine ad intervalli regolari, dando origine a insiemi di immagini detti *ottave*.

Figura 3.3: Filtraggio in cascata dell'immagine originale (la prima a sinistra) con kernel gaussiani di varianza σ crescente.

Per trovare punti interessanti all'interno dello scale-space, viene determi-

nata la funzione *DoG (Difference of Gaussian)*, calcolata come la differenza tra due funzioni Gaussiane considerate a scale consecutive:

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

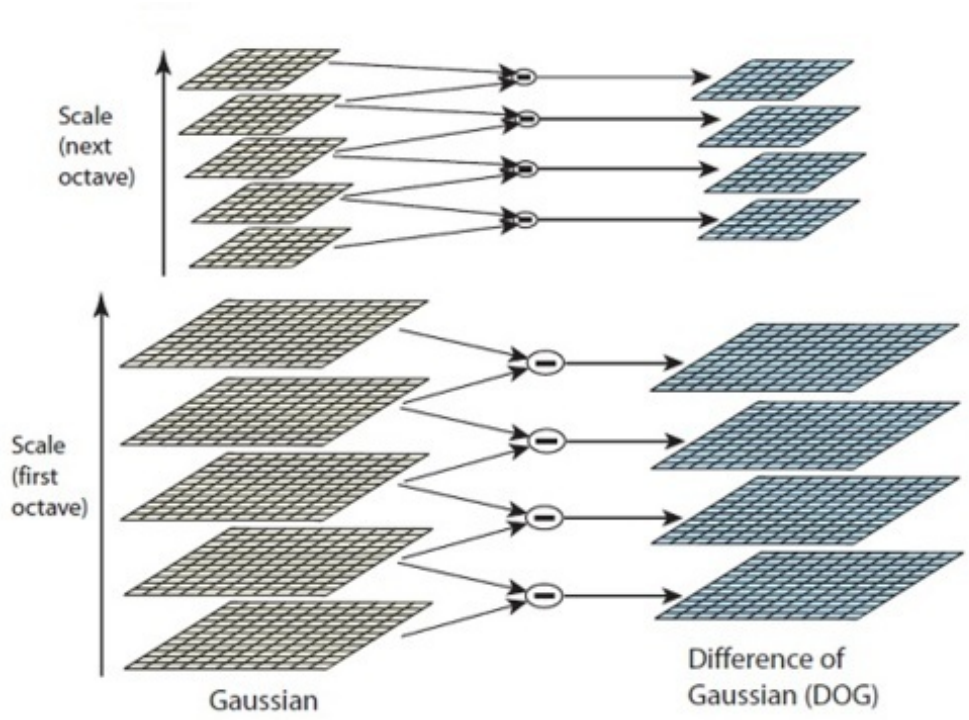

Figura 3.4: A sinistra, le immagini convolute con i kernel Gaussiani che formano gli insiemi di ottave. A destra, le immagini che rappresentano le Differenze di Gaussiane (DoG), ottenute sottraendo le Gaussiane adiacenti. Al termine di ogni ottava, l'immagine Gaussiana viene sotto-campionata di un fattore 2 e il processo riparte.

3.2.2 Localizzazione dei keypoints

Gli estremi locali della funzione $D(x, y, \sigma)$ rappresentano i punti di interesse da ricercare. Per determinarli, ciascun punto viene confrontato con i suoi otto vicini nell'immagine corrente e con i nove vicini delle immagini in

scala superiore e inferiore. Il criterio per cui un punto sia un massimo o un minimo locale è che risulti più grande o più piccolo in tutti i confronti.

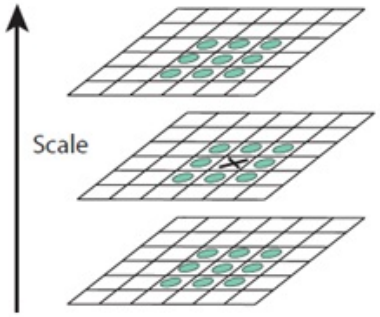

Figura 3.5: Localizzazione dei massimi e minimi locali.

Localizzazione ottimale dei keypoints

Un metodo più accurato per la selezione dei keypoints (che estende il medoto SIFT iniziale) prevede che vengano eliminati i punti con basso contrasto, valutando la presenza di bordi e linee, che vengono interpolate per aumentare la precisione, la stabilità e il matching dei punti.

La tecnica consiste nell'interpolazione quadratica 3D dei campioni nell'intorno di un estremo locale e utilizza l'espansione di Taylor al secondo ordine della funzione dello scale-space $D(x, y, \sigma)$, traslata in modo tale che l'origine sia posta sulle stesse coordinate del campione dell'immagine:

$$D(x) = D + \frac{\partial D^T}{\partial x}x + \frac{1}{2}x^T\frac{\partial^2 D}{\partial x^2}x \tag{3.1}$$

dove D e le sue derivate vengono determinate nel punto campione e $x = (x, y, \sigma)^T$ è *l'offset* da tale punto. L'estremo $\hat{x}$ di tale funzione, calcolato

mettendo a zero le derivate, è dato da:

$$\hat{x} = \frac{\partial^2 D^{-1}}{\partial x^2} \frac{\partial D}{\partial x} \tag{3.2}$$

Il valore assunto dalla funzione nei punti estremi, $D(\hat{x})$, serve per eliminare i punti a basso contrasto, che portano quindi rumore. Ciò si ottiene sostituendo 3.2 in 3.1, da cui:

$$D(\hat{x}) = D + \frac{1}{2} \frac{\partial D^T}{\partial x} x \tag{3.3}$$

A questo punto, si stabilisce una soglia (Lowe propone 0.3), per eliminare tutti i punti per i quali si verifica che $|D(\hat{x})|$ è inferiore a tale soglia.

Rapporto delle curvature principali degli edge

Un ulteriore passaggio per garantire la stabilità dei keypoints consiste nell'eliminare anche i punti collocati sugli edge. Consideriamo la matrice Hessiana data da:

$$H = \begin{bmatrix} D_{xx} & D_{xy} \\ D_{xy} & D_{yy} \end{bmatrix}$$

Gli autovalori di H sono proporzionali alle curvature principali di D. Senza doverli calcolare esplicitamente, ma considerando il loro rapporto, si considerano α l'autovalore più grande e β quello più piccolo. La somma degli autovalori può essere calcolata a partire dalla traccia di H e il loro prodotto a partire dal determinante:

$$Tr(H) = D_{xx} + D_{yy} = \alpha + \beta$$

$$Det(H) = D_{xx} D_{yy} - (D_{xy})^2 = \alpha\beta$$

Se il determinante è negativo, le curvature hanno segni opposti e conseguentemente il punto viene scartato non essendo un estremo. Sia poi r il rapporto tra l'autovalore più grande e il più piccolo ($a = rb$). Quindi,

$$\frac{Tr(H)^2}{Det(H)} = \frac{(\alpha+\beta)^2}{\alpha\beta} = \frac{(r\beta+\beta)^2}{r\beta^2} = \frac{(r+1)^2}{r}$$

La quantità $(r+1)^2/r$ è minima quando i due autovalori sono uguali, mentre cresce al crescere di r. Quindi, per imporre che il rapporto tra curvature principali sia sotto una certa soglia, occorre fissare r tale che:

$$\frac{Tr(H)^2}{Det(H)} < \frac{(r+1)^2}{r}$$

Il valore di soglia proposto è pari a $r = 10$ e determina l'eliminazione di tutti i keypoint che hanno un rapporto tra le curvature principali maggiore di 10.

3.2.3 Assegnazione dell'orientazione

Dopo aver individuato per ciascun keypoint delle coordinate e una scala, si procede al calcolo dell'orientazione per garantire una buona robustezza rispetto alle rotazioni. La scala associata al keypoint viene utilizzata per scegliere, nella piramide Gaussiana, l'immagine con la scala più vicina. Si calcolano quindi, per ogni campione dell'immagine $L(x, y)$ alla scala scelta, il modulo $m(x, y)$ e l'orientazione $\theta(x, y)$ del gradiente locale, applicando la differenza tra pixel:

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

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

Viene quindi costruito un istogramma delle orientazioni, costituito da 36 bin che coprono i 360 gradi assumibili. I picchi nell'istogramma delle orientazioni corrispondono alle orientazioni dominanti dei gradienti locali. Viene identificato il piccho più alto nell'istogramma e poi vengono considerati tutti gli altri picchi che hanno un valore pari almeno all'80% del valore massimo trovato, che vanno quindi a formare un keypoint con quella orientazione.

3.2.4 Generazione dei descrittori

Fin'ora le operazioni effettuate si occupano di localizzare dei keypoints dell'immagine a cui vengono assegnate coordinate spaziali, scala e orientazione. Per ultimo, occorre attribuire ai punti chiave un descrittore che risulti invariante ai cambiamenti di illuminazione e punti di vista 3D.

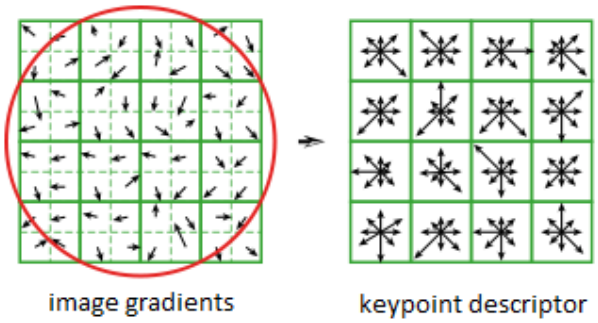

Figura 3.6: Generazione del descrittore a partire dalle orientazioni del gradiente nell'intorno del keypoint.

La tecnica per la generazione del descrittore consiste nel selezionare un'area intorno al keypoint analizzato (generalmente pari a 16x16 campioni) all'interno della piramide di immagini Gaussiane $L(x,y,\sigma)$. Per garantire invarianza rispetto a rotazioni, le coordinate del descrittore e le

direzioni del gradiente locale vengono ruotate rispetto all'orientazione del keypoint.

Successivamente, le orientazioni dei singoli campioni dell'intorno del keypoint vengono raggruppate in sub-regioni 4x4 e per ciascuna regione viene calcolato un istogramma di 8 bin, dove ciascun bin corrisponde a una diversa direzione e copre 45 gradi (figura 3.6).

Il descrittore SIFT è così generato, e avrà una dimensione di 4x4x8=128 elementi. Ogni elemento del vettore corrisponde quindi al valore dell'istogramma delle orientazioni della relativa sub-regione 4 x 4 dell'intorno del keypoint.

3.3 LBP: Local Binary Pattern

L'operatore LBP, presentato in [7], è un altro tipo di descrittore locale utilizzato principalmente per l'analisi della *texture* di un immagine. L'implementazione originale (usata in questa tesi) è molto semplice: preso un pixel centrale utilizzato come sogliatura, vengono confrontati con esso gli otto pixel attigui, dando origine ad un numero binario (1 byte). La codifica LBP avviene moltiplicando i valori sogliati con dei pesi caratteristici dipendenti dalla posizione del pixel vicino. Sommando gli addendi così ottenuti, si ricava il valore dell'LBP per quel pixel.

Una delle proprietà più importanti di questo operatore è la robustezza alle variazioni dei livelli di grigio, dovute ad esempio al cambiamento delle condizioni di illuminazione. Inoltre l'LBP, grazie alla sua semplicità di implementazione, che si traduce in poco sforzo computazionale, si addice in maniera particolare alle applicazioni real-time. Infine, ha anche una

ridottissima dimensionalità: 1 solo byte per pixel.

3.3.1 Confronto tramite istogrammi

I passi utilizzati per poter confrontare due oggetti attraverso l'operatore LBP sono i seguenti:

1. Dividere l'immagine in sub-regioni (tipicamente di 16x16 pixel)
2. Per ogni pixel della sub-regione, comparare il pixel con ciascuno degli 8 pixel vicini a lui (estensioni del metodo prevedono che il pixel venga comparato con n pixel distanti di un raggio R da esso)

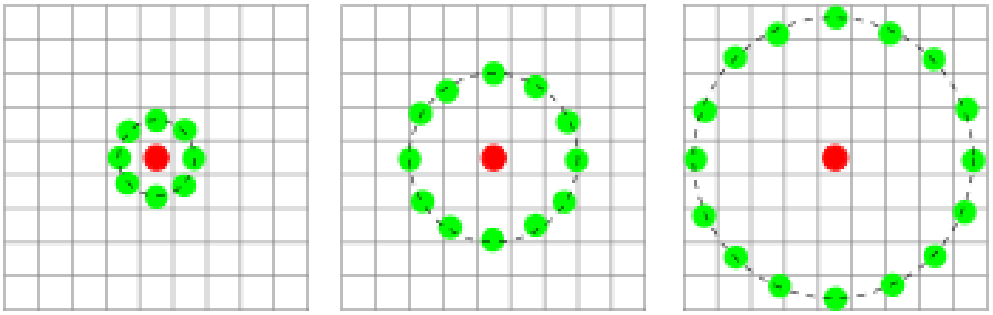

Figura 3.7: Possibile scelta dei vicini di un pixel. Nel primo caso sono scelti gli otto adiacenti. Nel secondo e nel terzo, vengono interpolati i valori che fanno parte di una circonferenza di raggio R dal pixel di sogliatura.

3. Per ogni pixel comparato, viene memorizzato in un vettore di 8 bit, il numero 1 se il pixel vicino ha un valore più alto, il numero 0 altrimenti. Solitamente il vettore di un byte è convertito in decimale, potendo così acquistare un valore compreso tra 0 e 255.
4. Calcolare l'istogramma, per ogni sub-regione, delle occorrenze del valore dell'LBP per ciascun pixel ed eventualmente normalizzare gli istogrammi.

5. Concatenare gli istogrammi e compararli con altri istogrammi ottenuti nello stesso modo da altre immagini. La corrispondenza tra gli istogrammi darà una stima del match tra i due oggetti.

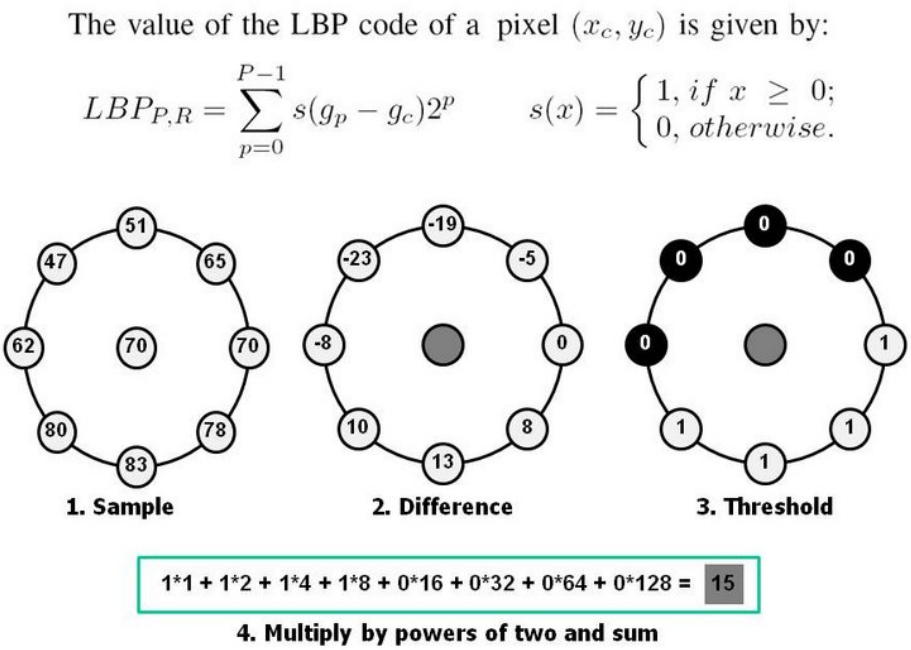

Figura 3.8: Computazione formale dell'LBP.

3.4 Matching tra le features

Arrivati a questo punto, i set di descrittori estratti da due immagini (dette rispettivamente *query* l'immagine per cui si effettua la ricerca e *train* quella in cui vogliamo ritrovare l'oggetto), possono essere comparati per verificare se vi è un'adeguata corrispondenza tra i due.

Quindi, i passi fondamentali per verificare se vi sia similarità tra un query e un train sono:

1. Localizzare i keypoint e generare i descrittori (nel nostro caso SIFT) in tutt'e due le immagini da confrontare.

2. Stabilire se vi sia corrispondenza (match) tra i descrittori delle due immagini.

Uno dei metodi più semplici, ma anche computazionalmente costosi, per il matching tra le features è il *Brute Force*. Esso consiste nel considerare ogni possibile coppia di descrittori (presi rispettivamente da query e train) per cercare, tramite la distanza euclidea, quali siano i descrittori più simili tra loro. Essendo la velocità uno dei requisiti richiesti per la ricerca di corrispondenze in video (il risultato sperato è che la computazione avvenga real-time con l'avanzamento del video), questa soluzione rallenta in modo significativo l'algoritmo. Per questo motivo, sono state utilizzate alcune tecniche per velocizzare l'implementazione del sistema.

3.4.1 Applicazione di una maschera nelle regioni di interesse

I loghi televisivi sono piccole immagini posizionate nella maggior parte dei casi (se non nella totalità) in uno dei quattro angoli dell'inquadratura. Essendo immagini permanenti nel tempo e fissate in un solo punto, l'idea di considerare solo alcune zone da analizzare risulta pratica ai fini di una ricerca più veloce.

Per ciascun frame esaminato del video, viene applicata una maschera che evidenzia, come regioni di interesse, i quattro angoli dell'immagine. Poichè ogni video può avere diversa dimensione spaziale, la maschera è calcolata in base ad essa: l'immagine viene divisa in una griglia 3x3 e viene ignorato tutto ciò che non riguarda i 4 angoli.

Figura 3.9: Esempio di un frame senza e con maschera. I pallini sono i keypoint trovati: si può notare come la ricerca venga dimezzata nel caso in cui viene applicata la maschera.

In questo modo, la ricerca di keypoints viene sostanzialmente dimezzata, contribuendo conseguentemente ad un raddoppiamento della velocità. Inoltre, eliminando parecchio rumore, migliora anche la qualità dei match.

3.4.2 K-d-tree

La struttura, utilizzata in questa tesi, attraverso la quale viene effettuato il match tra i set di descrittori presi da query e train, è k-d tree, ovvero un albero binario in cui ogni nodo indicizza un punto di dimensione k nello spazio k-dimensionale (nel nostro caso il descrittore SIFT di 128 elementi).

Ciascun elemento del nodo, che non sia una foglia, può essere pensato come generatore di un iperpiano di scissione che divide lo spazio in due parti, detti semispazi: i punti alla sinistra dell'iperpiano rappresentano il sottoalbero sinistro dell'elemento (minori), mentre i punti alla destra rappresentano il sottoalbero destro (maggiori). Per ciascun elemento k-esimo del punto, la direzione dell'iperpiano viene scelta in base alla direzione dell'asse corrispondente. In questo modo lo spazio k-esimo viene partizionato in *sub-regioni* i cui confini sono appunto occupati dai nodi.

Il confronto tra due insiemi di descrittori avviene in questo modo:

1. Viene creato un k-d tree per uno dei due set di keypoints.

2. Per ogni keypoint dell'altra immagine, viene effettuata la ricerca sul k-d tree in maniera ricorsiva, confrontando solo quei keypoint che vengono via via analizzati nell'albero, in base ai criteri di maggioranza o minoranza.

3. Quando un keypoint si ferma nella sua ricorsione (poichè i nodi figli non rispecchiano similarità), il nodo su cui si è fermato diventa il

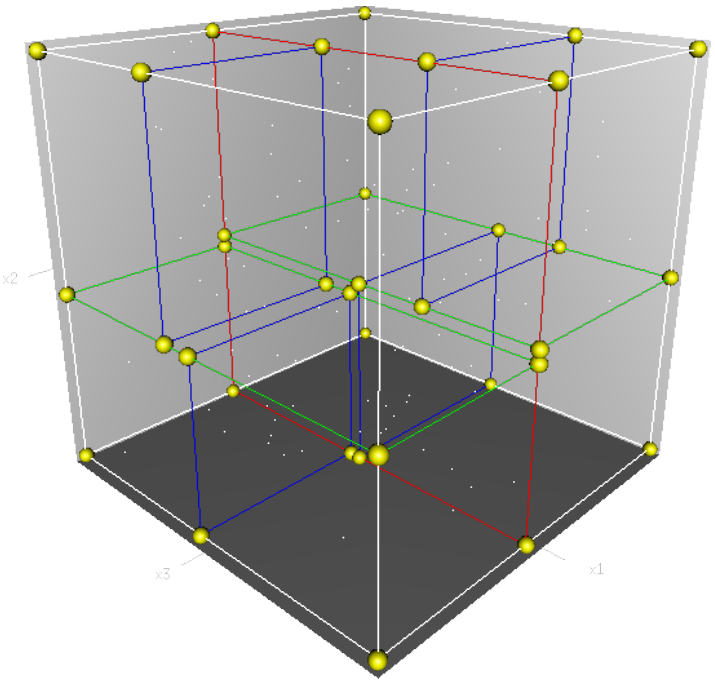

Figura 3.10: Esempio di 3-dimensional tree.

nearest neighbor del keypoint stesso.

In questo modo, è evidente che ogni punto dell'immagine train non viene confrontato con ciascuno dell'immagine query (come succede con il metodo Brute Force), ma vengono escluse regioni intere grazie alla binarizzazione ricorsiva dello spazio dei keypoint, andando ad influire enormemente sulla velocità dell'algoritmo.

Per i test che vengono effettuati successivamente riguardo la validità dei match, il matcher utilizzato provvede ad estrarre, tra il set di keypoints dell'immagine train, i due punti più vicini per ciascun keypoint dell'immagine query.

3.4.3 Ratio test e RANSAC test

Dopo aver trovato le corrispondenze tra le features di query e train, il sistema procede alla verifica di due test, per eliminare in parte le corrispondenze che potrebbero risultare errate.

Il primo di questi è il *Ratio Test*: esso consiste nell'effettuare il rapporto tra i due nearest neighbor trovati, per verificare la distanza tra i due. Se i due punti sono molto vicini tra loro (e il rapporto tende a 1), è molto probabile che sia stato effettuato un match errato: infatti, data l'alta dimensionalità spaziale dei frame, è possibile trovare falsi match molto simili tra loro. Più questo rapporto tende a 0, più i due punti più vicini trovati si discostano tra loro e in questo caso è molto probabile che il primo tra i due sia un match giusto. Sperimentalmente, è stato scelto, come soglia per l'esclusione di falsi match, che il rapporto sia inferiore a 0.8.

Il secondo test effettuato è il RANSAC test: esso è un metodo iterativo che, a partire da un insieme di dati contenente *outliers* (nel nostro caso falsi match), produce un risultato il cui scopo è scremare il dataset di punti iniziali per eliminare tali outliers.

L'algoritmo si basa sul calcolo della matrice fondamentale, ovvero una matrice 3x3 che rappresenta la trasformazione di coordinate dal set di keypoints della query a quello dei keypoints del train. Questo tipo di calcolo è utile soprattutto quando vengono riprese due immagini da diversi punti di vista: la matrice fondamentale permette di riprodurre la trasformazione che avviene da un punto di vista a un altro. Nel nostro caso, essendo il logo statico (ovvero riprendibile da un solo punto di vista), il calcolo della matrice è finalizzato semplicemente a rimuovere quei falsi match che

non trovano corrispondenza di coordinate (con un certo margine di errore), rispetto alla disposizione dei keypoints presi nel loro insieme.

3.4.4 Salvataggio dei valori in file CSV

Dopo aver superato i suddetti test, i valori dei match vengono salvati nel modo seguente. Per una generica query analizzata in un video (composto da n train frame) vengono salvati in un file di testo CSV i seguenti valori:

- Numero del frame analizzato
- Numero di match trovati
- Numero di match normalizzati rispetto al numero di keypoints della query (ovvero una misura che indica in termini di percentuale quanto il logo sia stato ritrovato nel frame)

254	20	0.465116
508	15	0.348837
762	19	0.44186
1016	22	0.511628
1270	7	0.162791
1524	9	0.209302
1778	21	0.488372
2032	19	0.44186
2286	16	0.372093

Figura 3.11: esempio di estratto da un file CSV: il video è analizzato frame per frame secondo una frequenza di campionamento prestabilita.

3.5 Classificatore SVM lineare

Dopo aver confrontato un dataset di loghi con un dataset di video, i cui risultati vengono trascritti, come già detto, in file di testo CSV, si passa all'addestramento supervisionato di un classificatore SVM lineare.

Il *Support Vector Machine* utilizzato è un classificatore binario che apprende il confine tra esempi appartenenti a due classi diverse. Formalmente, l'algoritmo determina un iperpiano, in uno spazio a dimensione finita, che separi in modo ottimo gli esempi appartenenti a due classi di dati; in uno spazio bidimensionale si cerca di trovare una *retta di separazione* ottima che divida due classi (Fig. 3.12). Tale retta viene trovata massimizzando la distanza tra gli esempi di training opposti più vicini.

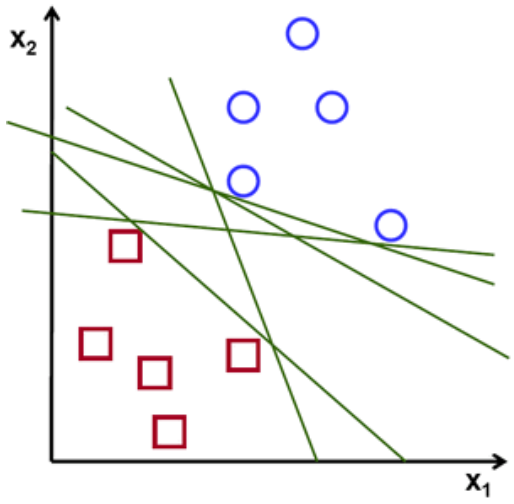

Figura 3.12: Addestramento di un classificatore: in questo caso gli esempi forniti appartenenti a due diverse classi vengono proiettati su un piano bidimensionale.

In questo modo, dopo la fase di *training*, qualsiasi altro input fornito al classificatore verrà automaticamente inserito in una delle due classi a seconda dei suoi parametri (*testing*).

Questo tipo di classificazione risulta utile per addestrare il sistema a ri-

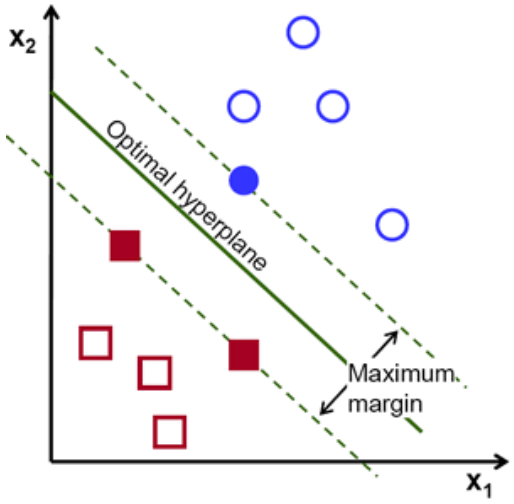

Figura 3.13: Calcolo da parte del classificatore dell'iperpiano ottimo, per la divisione delle regioni corrispondenti alle due classi.

conoscere in maniera automatica la presenza di un logo nel video. In questo caso, l'addestramento avviene fornendo file CSV ottenuti confrontando il generico logo con:

1. video in cui è presente il logo (match giusti)
2. video in cui non è presente il logo, o è presente un logo diverso (match errati)

Al classificatore vengono passati i valori di *media* e *deviazione standard* dei valori normalizzati di match nei file CSV, calcolati su finestre scorrevoli di dimensione e overlap variabili, dove per dimensione si intende un generico intervallo di frame, mentre per overlap si indica la sovrapposizione tra le finestre durante lo scorrimento dello stesso. Il passaggio di questi valori è motivato dal fatto che basarsi sul puro risultato di match (che a volte si trova isolato insieme a molti non match) renderebbe il classificatore instabile, col rischio di non riuscire a trovare una giusta soglia per la

separazione delle due classi. Inoltre, il loro calcolo arricchisce il sistema di dati, andando a beneficio della precisione.

3.5.1 Trainer

La prima fase, dunque, riguarda l'addestramento del classificatore.

L'algoritmo prende i seguenti parametri:

1. un file di testo contenente una lista di file CSV con esempi *positivi*
2. un file di testo contenente una lista di file CSV con esempi *negativi*
3. la dimensione della finestra da considerare per il calcolo di media e varianza
4. l'overlap della finestra

Figura 3.14: Finestra e overlap nello scorrimento di un file CSV.

Dopo aver calcolato media e deviazione standard dei valori appartenenti alle finestre che scorrono nel file CSV, il classificatore viene quindi addestrato in base a questi due parametri. In questo modo, vengono costruite le due regioni che itentificano i match giusti e quelli sbagliati, separati dal piano ottimo.

3.5.2 Tester

Dopo aver salvato i risulati del classificatore in un modello, l'algoritmo procede con la fase di testing. Questa è la fase finale e consiste nel prendere una lista di file CSV da video precedentemente analizzati (e che ovviamente non sono stati inseriti nel classificatore per addestrarlo) e far decidere, in base al modello estrapolato, se tali video contengono o meno il logo con cui sono stati confrontati.

Capitolo 4

Risultati sperimentali

4.1 Presentazione del dataset

In questo paragrafo si descrive il dataset utilizzato per l'analisi e la comparazione di loghi e video. I modelli ricercati provengono dai maggiori canali broadcast italiani e sono di 10 tipi differenti:

- Rai 1, Rai 2, Rai 3
- Rete 4, Canale 5, Italia 1
- La 7 (logo nuovo, logo vecchio, logo TG)
- Istituto Luce

Figura 4.1: Loghi per cui viene effettuata la ricerca: versione grafica.

I video analizzati in cui sono presenti tali loghi sono in tutto 122, provenienti principalmente da Youtube e la cui presenza in questa piattaforma comporta che vengano violati i diritti di proprietà intellettuale. Tali video sono così suddivisi:

- Rai (**48** video, di cui 19 di Rai 1, 10 di Rai 2, 19 di Rai 3)
- Mediaset (**54** video, di cui 15 di Rete 4, 19 di Canale 5, 20 di Italia 1)
- La7 (**16** video, comprendenti i tre tipi di loghi diversi)
- Istituto Luce (**4** video)

Figura 4.2: Loghi estratti dai video. In alcuni casi è evidente la differenza tra i modelli (fig. 4.1), dovuta soprattutto all'opacità o alla trasparenza.

Il formato è .mp4, mentre la dimensione dei frame varia di video in video dai 480 ai 640 pixel in larghezza e dai 270 ai 360 in altezza (ad eccezione dei 4 video di Istituto Luce che sono un po' più grandi rispetto agli altri).

In media ciascun video ha una durata di circa 4 minuti, per un totale complessivo di quasi 9 ore di filmati.

4.2 Estrazione delle features

La prima fase dell'algoritmo riguarda, come già detto, l'estrazione delle features. Il dataset di loghi è composto per una parte da loghi colora-

ti/opachi, per l'altra da loghi trasparenti. Per questo motivo, inizialmente, è stata fatta sui loghi trasparenti una prova di estrazione di features LBP, per verificare se il metodo fosse più robusto, in quanto rilevatore di texture. Scelto un logo da ricercare, l'analisi si è svolta in questo modo:

1. Sono stati estrapolati da alcuni frame, piccole sezioni della stessa dimensione del logo, alcune contenenti il logo nel video, altre no.

Figura 4.3: In alto, il logo da ricercare, seguito da sezioni con loghi estratte da frame del video. In basso, sezioni della stessa dimensione, senza logo.

2. Per ogni pixel di ciascuna sezione, viene calcolato l'LBP e successivamente un istogramma che mostra l'andamento delle features.

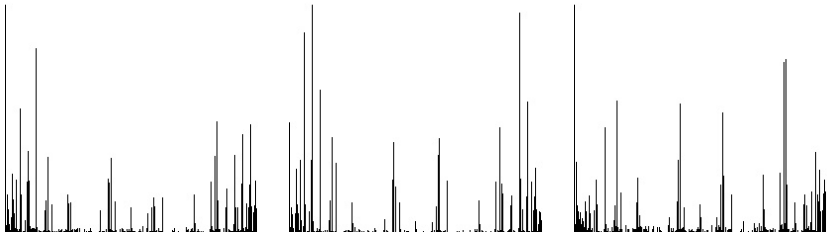

Figura 4.4: Tre istogrammi: a partire da sinistra, l'istogramma ricavato dagli LBP del logo da cercare, quello della sezione con logo e quello della sezione senza logo.

3. Infine, gli istogrammi ricavati, vengono confrontati e i valori salvati in un grafico. Più la distanza tra gli istogrammi è bassa, più probabilità c'è che le sezioni siano simili.

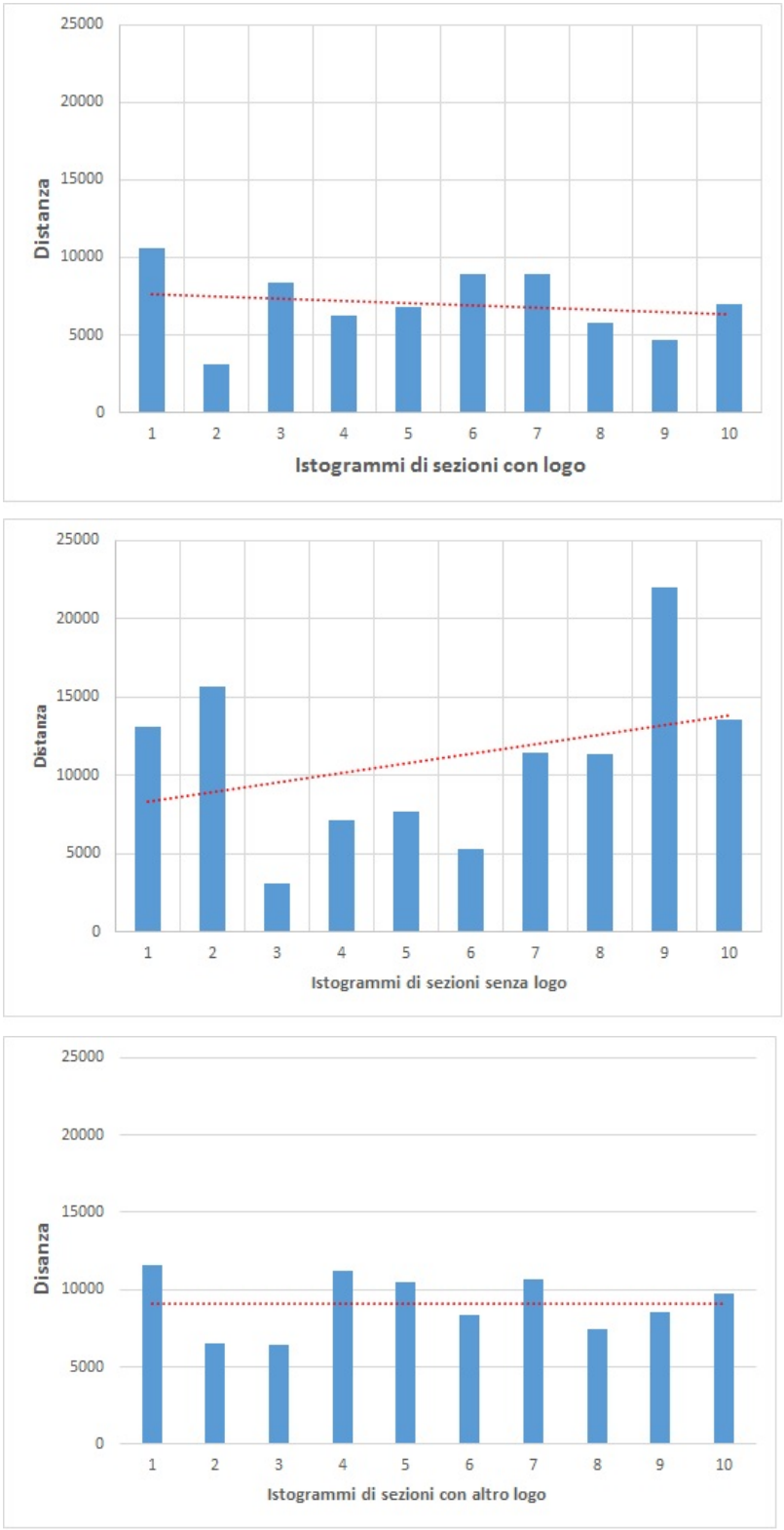

Figura 4.5: Esempio di confronto tra istogrammi. Il primo grafico mostra l'andamento del confronto tra l'istogramma del logo cercato con altri 10 istogrammi ricavati da sezioni con logo. Analogamente il secondo e il terzo mostrano gli andamenti nel caso di sezioni senza logo e sezioni con loghi diversi da quello cercato.

Come si nota dall'esempio proposto, nonostante gli istogrammi delle sezioni che contengono i loghi abbiano mediamente una distanza inferiore rispetto all'istogramma del logo ricercato, la separazione tra un caso e l'altro non è così netta. Questa ragione ci ha portato a rinunciare all'estrazione di features tramite LBP e propendere, nonostante la criticità del loghi trasparenti, per il metodo SIFT.

4.3 Analisi superivisonata dei match

4.3.1 Scelta del campionamento dei video

La prima parte del sistema, quindi, prevede che ognuno dei 10 loghi in questione venga ricercato in ciascuno dei 122 video del dataset. Riducendo al caso semplice dell'analisi di un logo e un video, ciò avviene confrontando le SIFT features estratte dal logo, con le SIFT features estratte da ciascun frame del video. Poichè ogni video contiene in media 6000 frame, analizzarli tutti significherebbe rallentare l'intero processo di rilevazione. Per questo motivo, si rende necessario campionare il set di frame per ridurre le tempistiche.

In tabella vengono mostrati i risultati delle detection fatte classe per classe (ovvero ricercando Rai1 in video con Rai1, Rai2 in video con Rai2, ecc.), dove:

- fps sono i frame per secondo analizzati
- la percentuale indica in quanti video è stato ritrovato almeno una volta (quindi in almeno un frame) il logo desiderato

fps	Rai1	Rai2	Rai3	Rete4	Canale5	Italia1	La7	Ist. Luce
0.1	100%	100%	95%	74%	84%	90%	100%	100%
0.3			100%	87%	100%	80%		
0.5				93%		90%		
1				93%		100%		

Figura 4.6: Tabella che mostra la percentuale di rilevamento di ciascun logo nei video di attinenza, in base alla scelta del campionamento dei frame.

In figura 4.6, è interessante notare come per i loghi *colorati/opachi* (Rai, La7 e Istituto Luce) basti un campionamento meno fitto di frame, mentre per i loghi *trasparenti* (Mediaset), sia necessario analizzare più frame. Ciò è dovuto al fatto che un logo trasparente è rilevato più facilmente se il suo sfondo è meno disturbato (il caso ottimo è quello in cui la superfice sia uniforme): aumentando i numero di frame analizzati, aumenta la probabilità che questo accada.

Per questo motivo, è stato scelto di trattare il dataset dei 10 loghi in maniera diversa, dividendolo in due categorie:

- Loghi opachi (Rai, La7, Istituto Luce)
- Loghi trasparenti (Mediaset)

4.3.2 Precision e Recall

Ciascun logo del dataset viene quindi confrontato con ciascuno dei 122 video. In ognuno di questi confronti possono capitare 4 tipi di situazioni:

- detection del logo, quando il logo è presente (*true-positive*)
- detection del logo, quando il logo non è presente (*false-positive*)
- mancata detection del logo, quando il logo è presente (*false-negative*)
- mancata detection del logo, quando il logo non è presente (*true-negative*)

Figura 4.7: Detection del logo, quando il logo è presente (*true-positive*).

Figura 4.8: Detection del logo, quando il logo non è presente (*false-positive*).

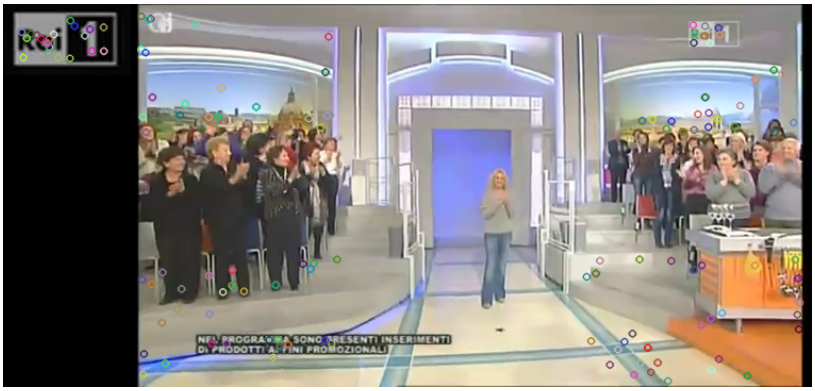

Figura 4.9: Mancata detection del logo, quando il logo è presente (*false-negative*).

Due parametri importanti nel calcolo delle performance delle detection sono *precision* e *recall*. Essi sono definiti come:

$$\text{precision} = \frac{\#\text{ true-positive}}{\#\text{ true-positive} + \#\text{false-positive}}$$

$$\text{recall} = \frac{\#\text{ true-positive}}{\#\text{ true-positive} + \#\text{false-negative}}$$

Figura 4.10: Mancata detection del logo, quando il logo non è presente (*true-negative*).

In altre parole, la precision è la probabilità che, dato un logo da ricercare, il video recuperato sia attinente (ovvero contenga il logo), mentre la recall è la probabilità che un video attinente venga recuperato.

Questi parametri sono stati calcolati sia a livello di classe di logo, sia a livello di categorie del dataset (opachi e trasparenti).

Loghi colorati/opachi

Della categoria loghi opachi, come già detto, fanno parte quelli di Rai, La7 e Istituto Luce, in tutto 7.

Nell'analisi delle detection i 3 loghi di Rai e i 3 loghi di La7 sono presi come appartenenti ad un unica classe, poichè è stato riscontrato che molto spesso tali loghi trovano corrispondenza tra loro, a causa dell'altissima similarità.

Il criterio di detection di un logo è che questo trovi corrispondenza (match) in almeno un frame del video analizzato. I risultati di true-positive (TP), false-positive (FP), false-negative (FN) e true-negative (TN) per ciascuna classe sono mostrati nelle tabelle seguenti, dove

- positive indica il numero di video di attinenza della classe di logo

- negative indica il numero di video in cui non è presente il logo

Rai	
positive(48)	negative(74)
TP=47	FP=4
FN=1	TN=70

Tabella 4.1: Detection di Rai.

La7	
positive(16)	negative(106)
TP=16	FP=26
FN=0	TN=80

Tabella 4.2: Detection di La7.

Istituto Luce	
positive(4)	negative(118)
TP=4	FP=11
FN=0	TN=107

Tabella 4.3: Detection di Istituto Luce.

Precision e recall sono quindi definite per ogni classe di logo nel modo seguente:

$$P_{Rai} = 0.92, \quad R_{Rai} = 0.98$$

$$P_{La7} = 0.38, \quad R_{La7} = 1$$

$$P_{Luce} = 0.27, \quad R_{Luce} = 1$$

Le prestazioni globali di tutte le classi di logo (pesate ciascuna rispetto al numero totale di video di attinenza), sono date da:

$$Precision_{LoghiOpachi} = \frac{0.92 * 48 + 0.38 * 16 + 0.27 * 4}{68} \simeq 0.75$$

,

$$Recall_{LoghiOpachi} = \frac{0.98 * 48 + 1 * 16 + 1 * 4}{68} \simeq 0.98$$

Loghi trasparenti

I loghi trasparenti ricercati sono Rete4, Canale5 e Italia1. Poichè, come già detto, la loro detection è in gran parte influenzata dallo loro sfondo (se troppo rumoroso, rende difficile l'estrazione di keypoints adeguati al confronto), si è deciso di confrontare ciascun video con un *metamodello* per ognuno di essi.

Figura 4.11: Tre modelli diversi diversi per il logo di Canale5. Anche se apparentemente sembrano uguali, l'estrazione delle features può cambiare da modello a modello, rendendo più probabile la detection nel video.

Un metamodello è l'unione dei match di più loghi dello stesso tipo: in poche parole, ciascuno dei 122 video, campionato con una frequenza di 1 fps, viene confrontato con più di modello per logo, per aumentare le probabilità che esso trovi corrispondenza.

Nella figura seguente si mostra l'andamento della detection dei tre modelli di logo di canale 5 con un generico video.

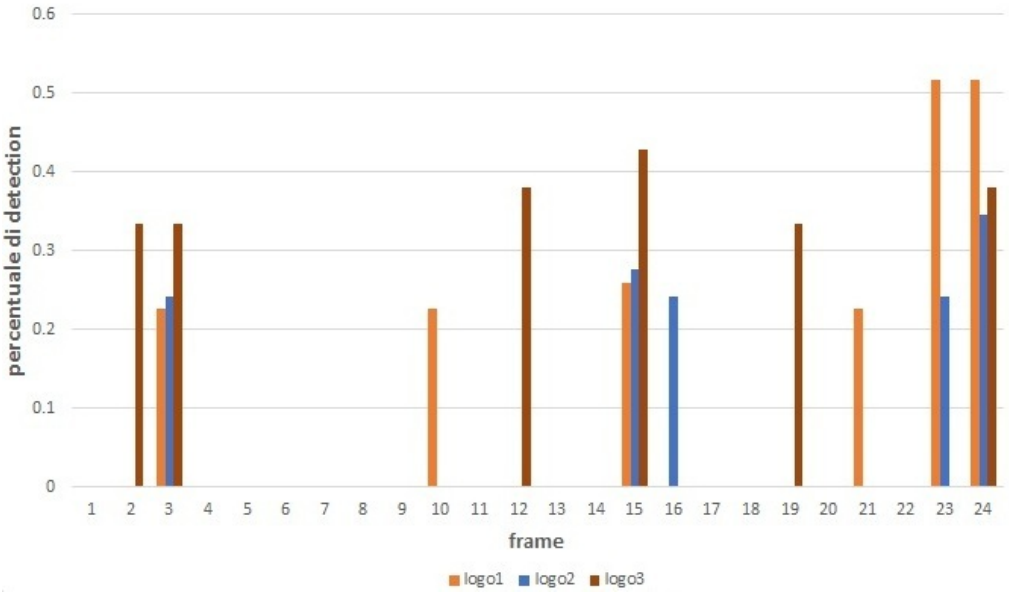

Figura 4.12: Andamento, per ciascun frame, della percentuale di detection di tre modelli diversi di logo.

E' interessante osservare come a volte succeda che un logo trovi corrispondenza, dove un'altro non è stato trovato. Questa compensazione assicura in parte che vi sia continuità nella detection del logo, evitando così che si riduca a picchi sporadici.

Per descrivere il metamodello in un unico insieme di match (come accade per i loghi opachi), vengono unite tutte detection con il criterio della massimizzazione: per ciascun frame, il logo che ha avuto una percentuale più alta è destinato a rimanere rispetto agli altri.

Analogamente alla categoria dei loghi opachi, i risultati di detection per i loghi trasparenti sono mostrati nelle tabelle seguenti.

Precision e recall sono quindi definite per ogni classe di logo nel modo seguente:

$$P_{Rete4} = 0.4, \quad R_{Rete4} = 0.93$$

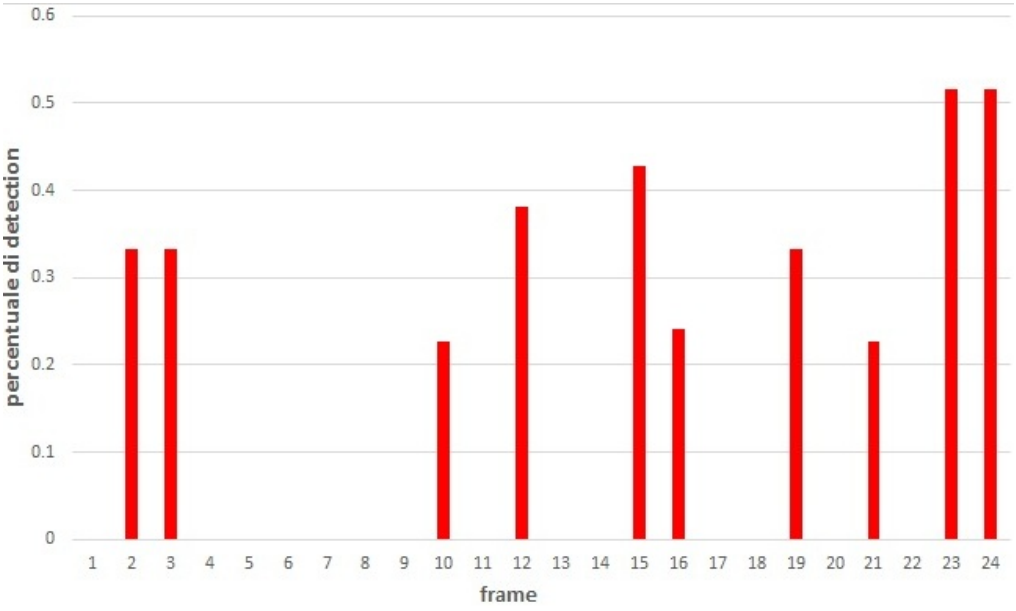

Figura 4.13: Andamento, per ciascun frame, della percentuale di detection dell'unione dei tre modelli diversi di logo, secondo il criterio del massimo.

Rete4	
positive(15)	negative(107)
TP=14	FP=21
FN=1	TN=86

Tabella 4.4: Detection di Rete4.

Canale5	
positive(19)	negative(103)
TP=19	FP=37
FN=0	TN=66

Tabella 4.5: Detection di Canale5.

$$P_{Canale5} = 0.34, \quad R_{Canale5} = 1$$

$$P_{Italia1} = 0.71, \quad R_{Italia1} = 1$$

Italia1	
positive(20)	negative(102)
TP=20	FP=8
FN=0	TN=94

Tabella 4.6: Detection di Italia1.

Le prestazioni globali di tutte le classi di logo sono date da:

$$Precision_{LoghiTrasparenti} = \frac{0.4 * 15 + 0.34 * 19 + 0.71 * 20}{54} \simeq 0.49$$

,

$$Recall_{LoghiTrasparenti} = \frac{0.93 * 15 + 1 * 19 + 1 * 20}{54} \simeq 0.98$$

4.3.3 Prestazioni globali

Le prestazioni globali complessive calcolate su tutto il dataset di loghi (opachi e trasparenti) sono quindi date da:

$$Precision = \frac{0.75 * 68 + 0.49 * 54}{122} \simeq 0.64$$

$$Recall = \frac{0.98 * 68 + 0.98 * 54}{122} = 0.98$$

4.4 L'addestramento del classificatore

Dopo l'analisi supervisionata dei match, si passa all'addestramento del classificatore, di modo che successivamente sia possibile analizzare un video senza sapere a priori se contiene o meno il logo d'interesse. Il dataset di 122 video, nella detection di tutti i loghi, ha riscontrato complessiva-

mente 120 *true-positive* (detection giuste) e 107 *false-positive* (detection sbagliate).

Tali risultati sono utilizzati, come mostrato nelle figure seguenti, rispettivamente come *positive examples* e *negative examples* per il classificatore.

Il dataset è stato diviso nelle due categorie di logo (opachi e trasparenti) per le fasi di *training* e *testing* (la scelta degli esempi nelle due fasi è totalmente casuale).

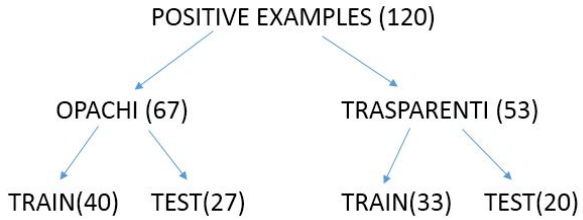

Figura 4.14: I 120 esempi positivi, estratti dall'analisi supervisionata dei match, sono divisi nelle due categorie di loghi: opachi (67 esempi) e trasparenti (53 esempi). Ciascuna delle due categorie è ulteriormente suddivisa per le fasi di training e testing del classificatore.

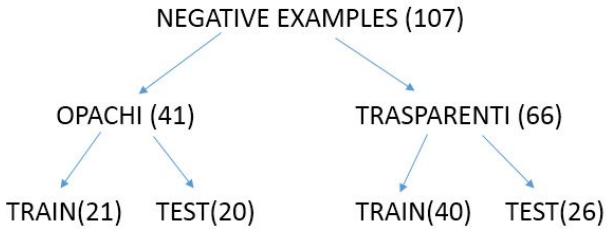

Figura 4.15: In modo analogo, vengono rappresentati i 107 esempi negativi, divisi tra categorie di logo e fasi del classificatore.

4.4.1 Scelta della finestra e overlap

Le due categorie di logo, per il modo in cui sono state diversamente trattate (diverso campionamento, utilizzo di uno o più modelli di logo), vengono differenziate anche nell'ultima fase di apprendimento del classificatore.

Per ciascuna categoria, nella fase di training, vengono creati 6 diversi modelli di classificazione, che variano in base alla scelta della finestra di frame da trattare e della sovrapposizione che avviene tra esse.

I modelli di classificazione sono:

1. con overlap (metà della finestra)
    - finestra= 4 frame, overlap=2 frame
    - finestra=10 frame, overlap= 5 frame
    - finestra=20 frame, overlap=10 frame
2. senza overlap
    - finestra= 4 frame
    - finestra=10 frame
    - finestra=20 frame

D'ora in poi i modelli si indicheranno semplicemente in base alla larghezza della finestra, differenziando i due casi con le diciture *senza overlap* e *con overlap*, essendo implicito che l'overlap è esattamente la metà della finestra considerata.

4.4.2 Testing dei loghi opachi

Dopo aver creato i 6 modelli di classificazione nella fase di training (che per la categoria dei loghi opachi ha acquisito 40 *positive examples* e 21 *negative examples*), si passa alla fase di testing. Esso è effettuato, a seconda del modello di classificazione scelto, con gli stessi parametri di finestra e overlap.

I risultati sono mostrati nelle tabelle seguenti, dove per true-positive e false-positive si intendono il numero di file CSV per i quali il classificatore decreta la presenza del logo in almeno una finestra tra quelle analizzate.

con overlap		
dimensione finestra	#truepositive	#falsepositive
4	27/27	14/20
10	26/27	1/20
20	27/27	17/20

Figura 4.16: Risultati del classificatore con overlap delle finestre.

senza overlap		
dimensione finestra	#truepositive	#falsepositive
4	27/27	8/20
10	27/27	3/20
20	27/27	4/20

Figura 4.17: Risultati del classificatore senza overlap delle finestre.

I modelli di classificazione migliori sono dati ottimizzando il numero di casi in cui avviene una detection corretta, che devono essere massimizzati, e quelli in cui avviene una falsa detection, analogamente minimizzati. Nel caso di loghi opachi, due scelte ottime sono date da:

1. con overlap, finestra=10 (true-positive=26/27, false-positive=1/20)
2. senza overlap, finestra=10 (true-positive=27/27, false-positive=3/20)

dove nel primo caso

$$precision = 0.96$$

$$recall = 0.96$$

mentre nel secondo

$$precision = 0.9$$

$$recall = 1$$

Come si può notare dalle tabelle, finestre più piccole vanno a incidere su un numero maggiore di false-negative. Questo dipende dal fatto che solitamente i file CSV estratti dall'analisi di video che presentano false detection, hanno tali detection molto diratate tra loro (picchi) e non continuative: più la finestra è piccola, più peso assumono i picchi nell'addestramento del classificatore, più probabilità ci saranno che in fase di testing tali picchi vengano inseriti tra le true detection.

4.4.3 Testing dei loghi trasparenti

In fase di training per i loghi trasparenti vengono inseriti nel classificatore 33 *positive examples* e 40 *negative examples*. La fase di testing prevede che ne siano analizzati rispettivamente 20 e 26. Analogamente alla categoria dei loghi opachi, i risultati vengono mostrati nelle figure successive.

Il caso ottimo è dato dalla scelta della finestra di 20 frame, senza overlap, dove

con overlap		
dimensione finestra	#truepositive	#falsepositive
4	20/20	26/26
10	19/20	17/26
20	19/20	5/26

Figura 4.18: Risultati del classificatore con overlap delle finestre.

senza overlap		
dimensione finestra	#truepositive	#falsepositive
4	20/20	26/26
10	19/20	17/26
20	19/20	4/26

Figura 4.19: Risultati del classificatore senza overlap delle finestre.

$$precision = 0.83$$

$$recall = 0.95$$

Il fatto che la finestra analizzata sia il doppio rispetto a quella dei loghi opachi è quasi prevedibile: avendo campionato i video con una frequenza di 1 fps (rispetto a 0.1 fps dei loghi opachi), il numero di frame dei file CSV è molto più alto e molto più alta è la distanza tra le false detection. Per minimizzarle, seguendo lo stesso ragionamento di prima, è adeguata una

finestra più ampia.

4.5 Esempi

Di seguito vengono forniti esempi di detection per ogni logo di interesse.

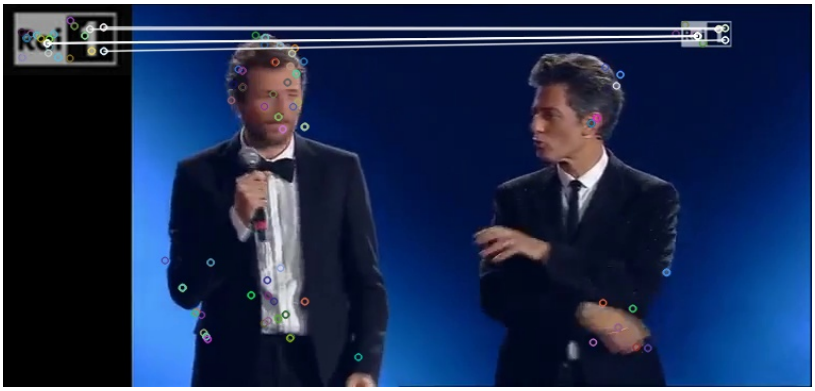

Figura 4.20: Rai1.

Figura 4.21: Rai2.

Figura 4.22: Rai3.

Figura 4.23: Rete4.

Figura 4.24: Canale5.

Figura 4.25: Italia1.

Figura 4.26: La7 new logo.

Figura 4.27: La7 old logo.

Figura 4.28: La7 tg.

Figura 4.29: Istituto Luce.

Capitolo 5

Conclusioni

L'obiettivo di questa tesi è stato quello di creare un sistema mediamente affidabile per il rilevamento di loghi televisivi in video presi dalla rete. Ricapitolando, le tre fasi del sistema sono state:

1. Confronto, attraverso estrazione e descrizione di punti salienti, del dataset di loghi con il dataset di video. Al termine di questi confronti vengono salvati file CSV che, frame per frame, indicano una percentuale di detection nel video (che può essere giusta o sbagliata).

2. Analisi supervisionata dei match e creazione di due distinte classi (positive examples e negative examples), che servono per istruire il classificatore nella fase di training.

3. Fase di testing per file CSV misti e relativi risultati sperimentali.

Il sistema dovrebbe poter funzionare con qualsiasi altro logo d'interesse e qualsiasi altro video preso dalla rete. Una miglioria futura di questo progetto potrebbe consistere nella parallelizzazione della prima e della terza fase sopraelencate per qualsiasi altro video in cui, dopo un adeguato adde-

stramento del classificatore, voglia essere ricercato uno specifico logo. In altre parole, l'applicazione potrebbe essere sviluppata *real time*, associando a ciascuna estrazione di features, il testing istantaneo del classificatore: in questo modo, nel caso in cui venga effettuata una detecion, l'analisi potrebbe fermarsi, senza dover scansionare i rimanenti frame del video, e dare un risultato immediato.

Bibliografia

[1] Jan Schietse, John P. Eakins, Remco C. Veltkamp, *Practice and Challenges in Trademark Image Retrieval*, 2007.

[2] Andrew D. Bagdanov, Lamberto Ballan, Marco Bertini, Alberto Del Bimbo, *Trademark Matching and Retrieval in Sports Video Databases*, 2007.

[3] Jinqiao Wang, Lingyu Duan, Zhenglong Li, Jing Liu, Hanqing Lu, Jesse S. Jin, *A Robust Method for TV Logo Tracking in Video Streams*, 2006.

[4] Alex Reis dos Santos, Hae Yong Kim, *Real-Time Opaque and Semi-Transparent TV Logos Detection*, 2006.

[5] Serge Belongie, Jitendra Malik, Jan Puzicha, *Shape Matching and Object Recognition Using Shape Context*, 2002.

[6] David G. Lowe, *Distinctive Image Features from Scale-Invariant Keypoints*, 2004.

[7] T. Ojala, M. Pietikäinen, D. Harwood, *A Comparative Study of Texture Measures with Classification Based on Feature Distributions*, 1996.