ALMA MATER STUDIORUM - UNIVERSITÀ DI BOLOGNA

FACOLTÀ DI INGEGNERIA

CORSO DI LAUREA IN INGEGNERIA INFORMATICA

DIPARTIMENTO DI ELETTRONICA, INFORMATICA E SISTEMISTICA

TESI DI LAUREA

in

SISTEMI INFORMATIVI-T

PROGETTAZIONE E REALIZZAZIONE DI UN TOOL PER LA SEGMENTAZIONE AUTOMATICA DI COLLEZIONI VIDEO

CANDIDATO
Alessandro Colace

RELATRICE
Prof.ssa Ilaria Bartolini

CORRELATORE
Prof. Marco Patella

Anno Accademico 2012/2013

Sessione I

SOMMARIO

Capitolo 1. INTRODUZIONE

Il costante aumento dell'utilizzo della banda larga, nel corso degli ultimi anni, ha prodotto un incremento esponenziale nella rete di contenuti multimediali e di servizi per la condivisione di questi, quali, per citare i più famosi, **YouTube** (per i video), **Flickr** o **Instagram** (per le immagini digitali), senza poter tralasciare **Facebook**, il social network per eccellenza, nonché il sito più cliccato del mondo, superando anche Google[1]. Da qui si può facilmente intuire come la fetta più grande del traffico dati globale di Internet sia, allo stato attuale delle cose, rappresentata dai dati multimediali.

Attualmente, la tecnologia utilizzata dai motori di ricerca principali opera sul solo contenuto testuale dei file; sfortunatamente, i documenti di tipo multimediale presentano una serie di problemi di visibilità per i motori tradizionali, nonostante lo sviluppo della tecnologia abbia portato notevoli progressi nell'efficacia di ricerca su determinati file binari, quali ad esempio pdf o animazioni flash, nonché la possibilità di confrontare rapidamente immagini o audio simili[2]. La complessità nell'analisi dei contenuti audiovisivi è data principalmente dalla struttura e dalle dimensioni elevate di quest'ultimi, infatti è particolarmente oneroso comparare fra loro questo tipo di file sulla base del loro contenuto specifico.

Al giorno d'oggi tutti i principali servizi online, citati nel primo capoverso di questo capitolo, utilizzano metodi di ricerca basati sulle

[1] Statistiche Alexa sul traffico internet globale: http://www.alexa.com/topsites

[2] Esempi di servizi esistenti: Google Goggles per le immagini, Shazam o Soundhound per la musica

parole chiave[3], le cosiddette **keywords**; questo è reso possibile dal fatto che alle immagini ed ai video vengano assegnate note testuali come titolo, descrizione, utente proprietario ed alcune etichette (**tag**). Questo metodo, seppur piuttosto performante, può portare a risultati inesatti a causa del modello non accurato che vi è alla base. In primo luogo, affidandosi esclusivamente ad una rappresentazione testuale, si ha una notevole perdita di informazione, in quanto non si considera realmente l'intero documento, ma solo i tag che lo rappresentano: due filmati descritti dagli stessi identici descrittori potrebbero presentare contenuti audio e video completamente discordanti. Per esempio, il problema delle omografie tra chiavi di ricerca può influire particolarmente sui risultati di una query: risulta evidente come ad esempio la parola "pesca" possa significare sia uno sport che un frutto. Un altro punto importante su cui ragionare è quello dell'internazionalizzazione: un video caratterizzato da keyword in lingua italiana non può evidentemente essere reperito se la query di ricerca di questo è stata impostata con parole in inglese. Da tutto questo si evince che occorre sorpassare il fallace modello tag-based per ottenere un avanzato e consistente strumento di data retrieval per file multimediali.

SHIATSU (*Semantic-Based Hierarchical Automatic Tagging of Videos by Segmentation using Cuts*) si prefigge proprio l'obiettivo di migliorare questo modello, affiancando alla normale ricerca testuale, quella per contenuti reali dei video. Il principio basilare è l'assunzione di un video come oggetto gerarchico formato da più scene (**shot**) a loro volta composte da una serie ordinata di immagini (**frame**).

In questo lavoro di tesi verranno dapprima introdotte la libreria Windsurf e i principi di funzionamento del framework SHIATSU, dopodiché la parte principale spiegherà cosa significa Cut Detection e di come sia stata implementata concretamente, ingegneristicamente parlando, all'interno del framework esistente. Sarà quindi strutturata in 6 capitoli:

[3] Youtube ha un servizio di rimozione automatica di video illegali che effettua comunque un'analisi approssimativa su traccia audio e video per verificare violazioni di copyright

Capitolo 2. LA LIBRERIA WINDSURF

2.1 STRUTTURA DI WINDSURF

La libreria **Windsurf** (*Wavelet-Based Indexing of Images Using Region Fragmentation*) (1) (2) è stata creata per operare unicamente con collezioni di immagini organizzate gerarchicamente, fornendo diversi algoritmi generali per la ricerca e il recupero di dati.

Possiamo descrivere il modello di ricerca di Windsurf con il seguente enunciato:
"Dato un database Db composto da N documenti, $Db = \{D_1,..., D_N\}$, dove ogni documento D è formato da n_D elementi, $D = \{R_1,..., R_{n_D}\}$. Ogni elemento R di D è descritto da un insieme di caratteristiche che rappresentano il proprio contenuto in modo appropriato. Posto un documento query $Q = \{Q_1,..., Q_n\}$, composto da n elementi, e una funzione di distanza δ che misura la dissimilarità fra elementi, usando le caratteristiche visuali, vogliamo determinare l'insieme dei documenti di Db che sono più simili a Q"

L'attuale implementazione di Windsurf scompone le immagini in **feature** di diverso genere, in base alla rappresentazione scelta, che può essere basata su regioni o su istogrammi HSV. Spiegheremo le differenze tra queste due modalità nei prossimi paragrafi.

2.2 REGION-BASED IMAGE RETRIEVAL (RBIR)

Ogni immagine nel database *Db* viene segmentata in regioni omogenee; ogni regione *R* è composta da insiemi di pixel che condividono le stesse caratteristiche visuali, di cui vengono salvati i descrittori che le identificano (**features**), come colore, trama e forma della regione. Risulta evidente quanto la corretta localizzazione di queste features sia di estrema importanza, in quanto un'erronea rilevazione dei descrittori non porterebbe ad un calcolo corretto della distanza.

Figura 2-1 Segmentazione immagini in regioni

È indispensabile avere una funzione che misuri inizialmente la distanza tra le regioni, quindi aggregare i risultati del calcolo delle distanze tra tutte le regioni delle due immagini, per ricavare la distanza complessiva

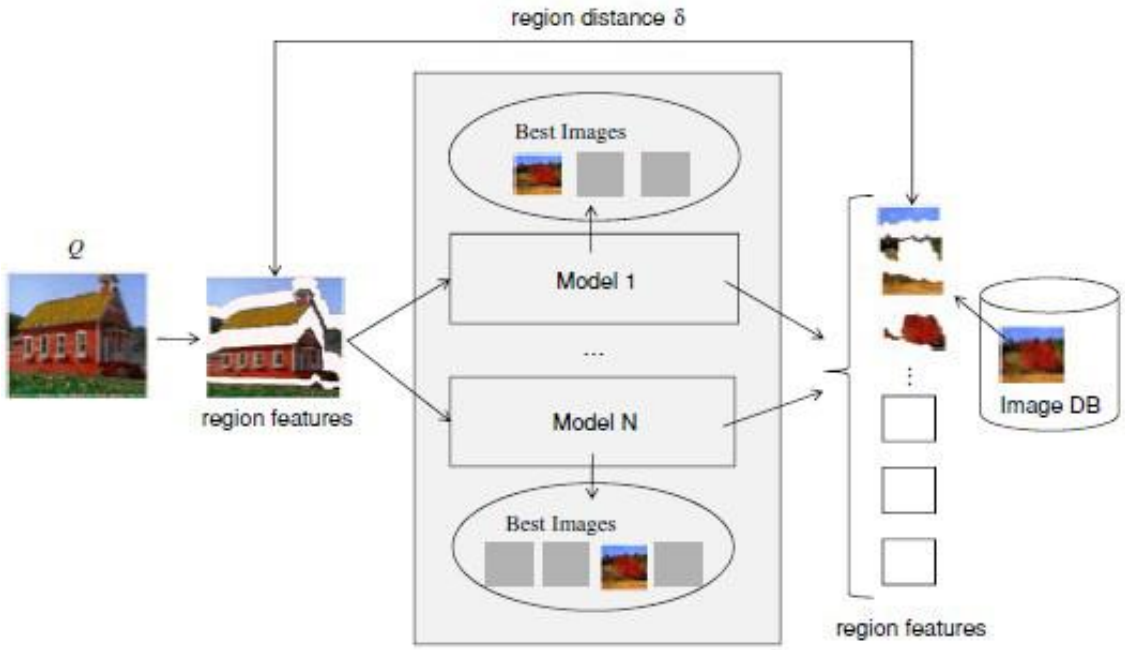


Figura 2-2 Modelli differenti di classificazione producono, per la stessa immagine, risultati differenti

La funzione già citata δ permette quindi di calcolare la distanza, o dissimilarità, tra due regione di un'immagine, sulla base delle features ottenute; vengono utilizzati quindi 2 generi di matching per il confronto tra le immagini:

1. **1-1**: associa ad ogni regione di un'immagine, al massimo una regione dell'altra e viceversa. Se le immagini non hanno lo stesso numero di regioni, allora le regioni escluse dal confronto restano senza corrispettivi.

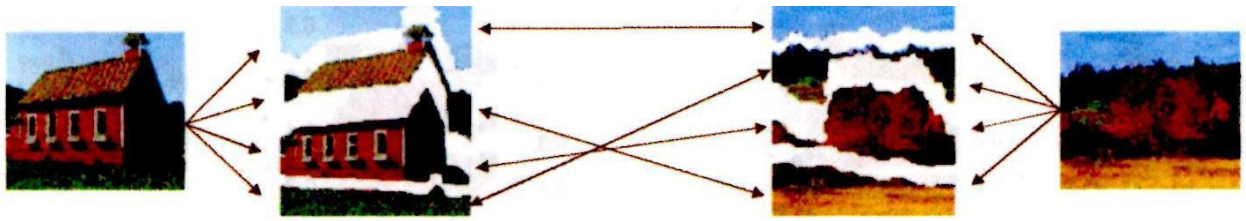


Figura 2-3 Matching 1-1

2. **N-M**: associa ad ogni regione di un'immagine tante regioni quante l'altra regione associa ad essa. Teoricamente non esistono restrizioni sui metodi di matching, mentre nell'implementazione reale ci sono limitazioni forzate che tengono conto della dimensione delle regioni, rispetto all'immagine integrale. La sommatoria delle distanze tra regioni deve essere uguale a 1.

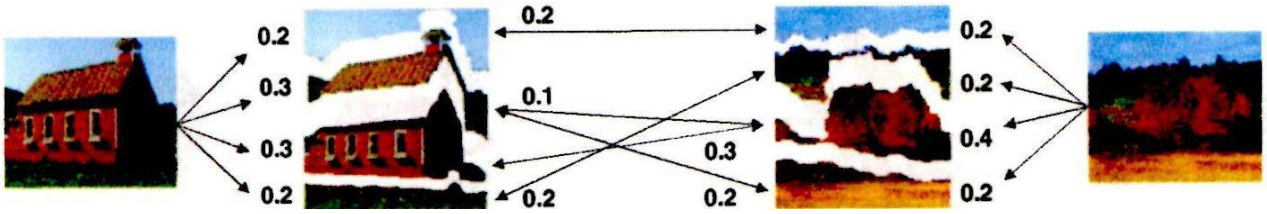


Figura 2-4 Matching N-M

La funzione δ è, di conseguenza, una matrice NxM contenente i valori delle distanze di tutti i confronti tra le varie regioni delle due immagini; occorre definire una distanza $d(Q,I)$ che rappresenti un singolo valore di distanza tra immagini. Le funzioni di matching principali di WindSurf sono tre:

1. **Windsurf Distance (WS):** matching 1-1, si inserisce un 1 nella matrice per ogni riga e colonna e si sceglie la miglior combinazione possibile di distanze, dopodiché si calcola la media tra le caselle a 1

2. **Earth's Mover Distance (EMD) (9)(10):** matching N-M, si ottiene la funzione di aggregazione dalla media tra i migliori matching (come già detto, la quantità di comparazioni di ogni immagine è limitata dalla dimensione dell'immagine)

3. **Integrated Region Matching (IRM):** si seleziona prima il valore di distanza più basso e gli si assegna il valore della grandezza minimo, tra le regioni cui fa riferimento, poi si procede in modo crescente per ogni elemento della matrice; alla fine si ottiene il totale sommando i prodotti tra i valori delle distanze della matrice iniziale, con i coefficienti della tabella ottenuta

I modelli illustrati sono tutti di tipo **k-Nearest Neighbour** (KNN), ovvero restituiscono le *k* immagini più vicine in termini di distanza.

In Windsurf, oltre a tutti questi modelli, troviamo infine il modello **Skyline**, non basato sul calcolo di un valore numerico preciso, ma sul concetto di dominanza tra documenti: "Un documento D_a viene considerato migliore di un documento D_b per la query Q se e solo se D_a non è mai peggiore di D_b per ciascun elemento che compone Q ed esiste almeno un elemento di Q in cui D_a è strettamente migliore di D_b".

Da qui, il seguente teorema (3): "Sia Q una query image, siano I_a e I_b due immagini del database; se la funzione di aggregazione g è monotona e la funzione di distanza δ è ottenuta mediante minimizzazione del valore di g, allora $\delta(Q, I_a) \leq \delta(Q, I_b)$". Si può capire intuitivamente che il risultato di una *query Skyline* non includerà mai alcuna immagine che sia sempre peggiore di un'altra.

2.3 HISTOGRAM-BASED IMAGE RETRIEVAL (HBIR)

Ad ogni immagine nel database Db viene associato un istogramma di colore che la rappresenta, ovvero una descrizione della distribuzione dei colori che la costituiscono. Gli istogrammi di colore più utilizzati sono quelli tridimensionali come HSV o RGB.

Un istogramma viene generato *discretizzando* i colori dell'immagine in un determinato numero di classi (**bin**), a seconda della qualità che si voglia analizzare; ad un maggior numero di bin corrisponde un grafico più

dettagliato, viceversa un basso numero di bin renderà l'istogramma creato di difficile comprensione e utilità.

Una stessa immagine ruotata su sé stessa in qualsiasi verso produrrà sempre lo stesso istogramma di colore.

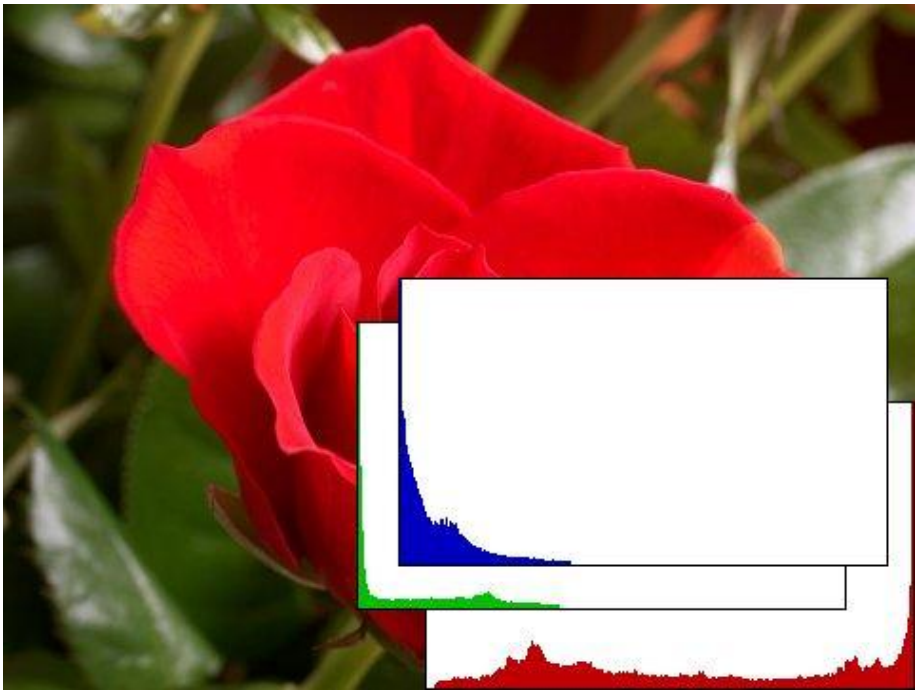


Figura 2-5 Esempio di istogrammi di colore, in questo caso abbiamo un grafico per ogni canale (RGB)

Il calcolo delle distanze tra immagini utilizzando questa tecnica è intuitivamente molto meno complesso dell'altro, quindi meno oneroso e più rapido, tuttavia questa modalità di ricerca risulta sensibilmente meno accurata.

Figura 2-6 Esempio di immagini considerate simili per istogrammi di colore

2.4 I PACKAGE PRINCIPALI

Il corpo della libreria, scritto in **JAVA**, è composto da 5 package:

- **Document**: contiene le classi relative alle regioni, alle immagini, alle features e le definizioni di distanza tra regioni δ e fra immagini d
- **FeatureExtractor**: definisce le classi relative alla segmentazione delle immagini in regioni e per il calcolo delle features relative
- **FeatureManager**: gestisce la memorizzazione e il recupero dei documenti sul DB usando un'astrazione dall'implementazione del DBMS sottostante (MySQL)
- **IndexManager**: definisce le classi per la gestione degli indici utilizzati per la creazione di query di tipo kNN (*vedi QueryProcessor*)
- **QueryProcessor**: definisce gli algoritmi che consentono la risoluzione delle ricerche sul database; questi possono sfruttare gli indici, ma anche lavorare in modo sequenziale nel caso in cui tali indici non siano disponibili

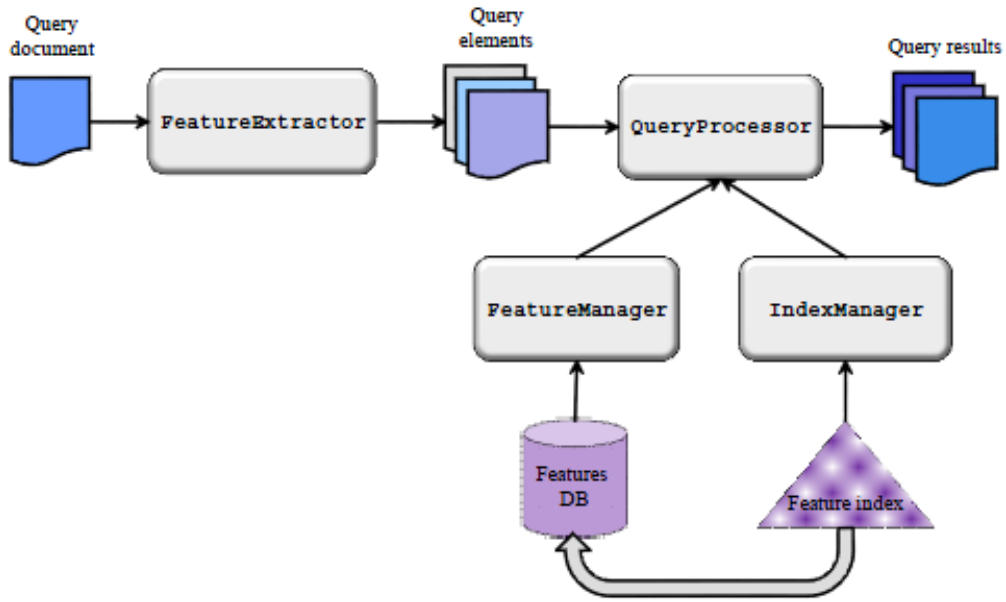


Figura 2-7 Ricerca con libreria WindSurf

2.5 INDICE M-TREE

Data l'indispensabilità di avere modalità di accesso efficienti per ricavare velocemente le risposte alle query per somiglianza, è stata introdotta l'indicizzazione mediante alberi **M-Tree** (4) che permette di organizzare dataset ad albero di grandi dimensioni con ottime performance in fase di ricerca.

Questo tipo di alberi usa la distanza relativa tra oggetti per organizzare e partizionare lo spazio di ricerca. Segue la definizione di **spazio metrico**: "Dato il dominio dei valori D rappresentanti le caratteristiche di un oggetto e data d come funzione di distanza fra gli oggetti, si considera spazio metrico la coppia $M = (D, d)$"

La funzione per il calcolo della distanza d deve rispettare le seguenti proprietà:

1. *Simmetria*: $d(O_x, O_y) = d(O_y, O_x)$
2. *Non negatività*: $d(O_x, O_y) > 0 (O_x \neq O_y)$ e $d(O_x, O_x) = 0$
3. *Disuguaglianza triangolare*: $d(O_x, O_y) \leq d(O_x, O_z) + d(O_z, O_y)$

L'indice M-Tree è, infine, completamente parametrico rispetto a d, di conseguenza, si può considerare l'implementazione della funzione come una *black-box* per l'indice.

Attualmente, la libreria che implementa M-Tree è basata su un'estensione del framework GiST[4] (*Generalized Search Tree*) ed è scritta in C++.

2.6 MISURAZIONI

Per la valutazione dell'efficienza degli algoritmi, ogni classe di Windsurf offre statistiche sulle operazioni principali:

- *Accesso ordinato*: calcola il numero di accessi agli indici sottostanti, che consentono di accedere al singolo dato in base al suo ordine
- *Distanza tra immagini*: calcola il numero di misurazioni di distanza tra immagini, considerabili solo per gli algoritmi di ricerca kNN
- *Distanza tra regioni*: calcola il numero di misurazioni di distanza tra regioni ed dipende dal numero di features e dalla funzione di distanza
- *Dominazione tra documenti*: calcola il numero di comparazioni tra documenti nelle query skyline
- *Tempo*: calcola il tempo totale per risolvere una singola richiesta, ovvero trovare una feature dal DB, l'accesso agli indizi, il calcolo di una distanza tra immagini o la comparazione di una dominanza (Skyline)

[4] Vedi: http://gist.cs.berkeley.edu/

Capitolo 3. SHIATSU

3.1 INTRODUZIONE

SHIATSU (5) (6) (7) (*Semantic-based Hierarchical Automatic Tagging of videos by Segmentation Using cuts*) è un framework per la gestione di video che si prefigge due scopi fondamentali:

- Annotazione semantica semiautomatica dei video
- Ricerca di video, o parti di essi, per similarità a partire da dati di input di tipo testuale (*tag*) o da altri oggetti multimediali quali video, immagini, audio[5]. Attualmente il programma accetta, come oggetto di query, singole immagini o porzioni di video (**shots**)

I moduli principali sono lo **Shot Detection Module**, ovvero il principale oggetto di questa tesi, e il **Tag Module**.

SHIATSU è una sorta di estensione di Windsurf: ne amplia la gerarchia dei dati multimediali, dovendo trattare tre tipologie di dati, ovvero video, shots (parti di video) e immagini, ordinate gerarchicamente in ordine decrescente.

Il problema focale di questa gerarchia è il modo in cui debba avvenire la **segmentazione automatica** di un video in singoli **shot**. Molto spesso infatti, un singolo video presenta situazioni considerevolmente differenti e, da qui, occorre pertanto considerare il video "padre", come un insieme di tanti piccoli mini filmati. Un esempio particolarmente calzante, è quello del notiziario televisivo, dove viene intuitivo valutare ogni singolo servizio come uno shot dell'intero TG.

Possiamo considerare quindi ciascuno shot come un *gruppo di frame contigui facenti parte della stessa scena e contenenti lo stesso evento / azione*.

[5] La parte che preveda la ricerca tramite audio non è implementata allo stato attuale delle cose, ma è possibile che in futuro se ne preveda uno sviluppo.

Dalla segmentazione in shot si passa poi a quella in immagini rappresentative, dette **keyframes**, scelte in numero direttamente proporzionale alla durata dello shot.

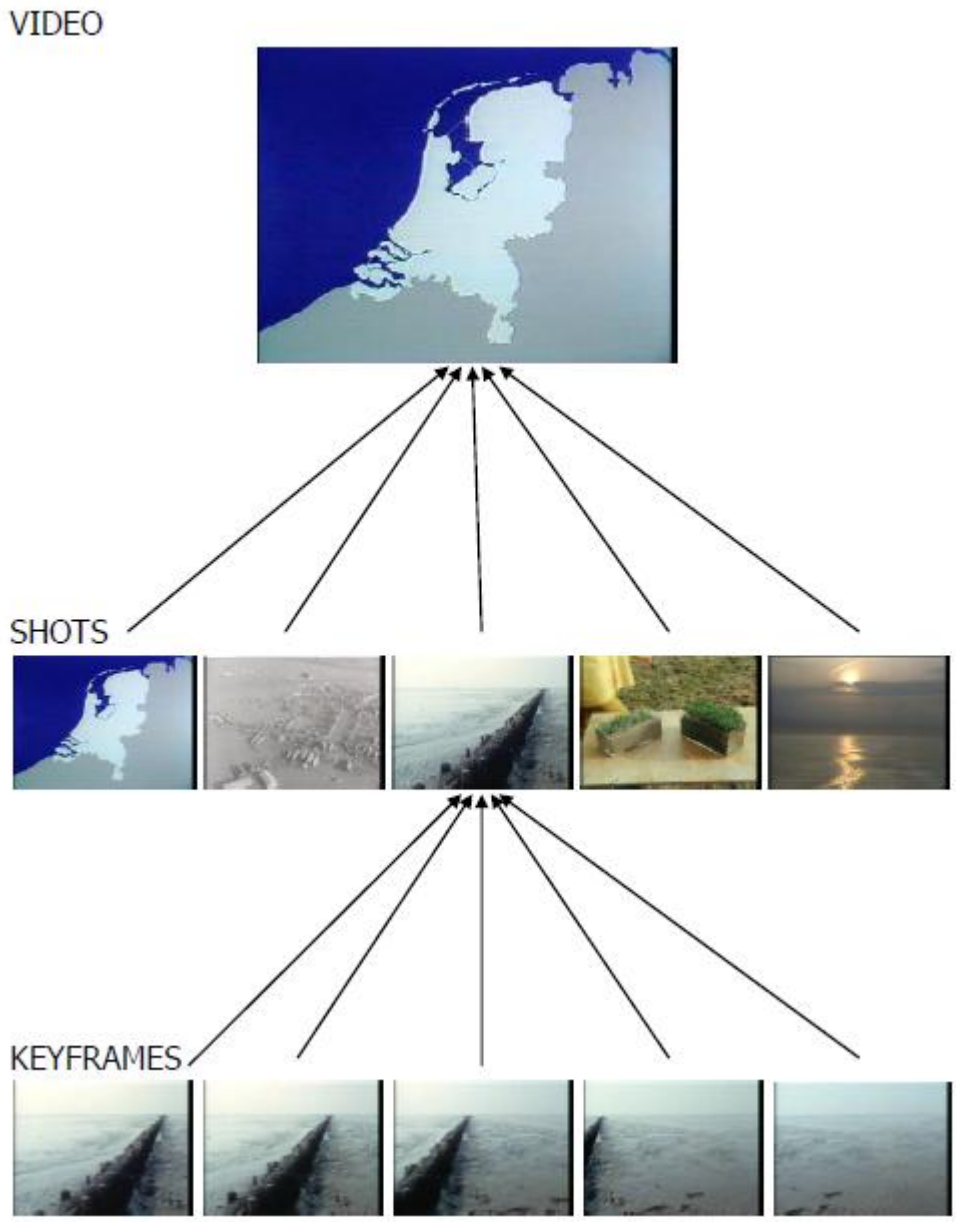


Figura 3-1Gerarchia dei dati di SHIATSU

3.2 FORMALIZZAZIONE

La formalizzazione del problema è identica a quella vista in Windsurf, ossia si considera l'immagine come documento e le relative regioni come gli elementi. Nel caso in cui l'utente voglia ottenere shots e video dalla sua ricerca, il framework associa semplicemente i risultati di ranking acquisiti dalla ricerca di immagini, ovvero restituisce gli esiti dopo aver calcolato la media dei keyframe di ogni shot e video.

3.3 STRUTTURA ARCHITETTURALE

La struttura di SHIATSU è composta da tre strati software

1. Strato superiore, la **GUI** (*graphical user interface)*, che si compone di due interfacce grafiche per fornire le funzionalità fondamentali del programma[6]. A questo livello troviamo quindi:

   a. *Annotation Tool*: per l'assegnazione semiautomatica di tag testuali a shot e video

   b. *Retrieval Tool*: per la ricerca di shot e video in base ai parametri provvisti dall'utente

2. Strato centrale, l'**Engine Layer**, costituito da tre componenti principali:

   a. *Visual Feature Extractor*: si occupa di ricavare le caratteristiche rappresentative di ciascun frame dei video

   b. *Annotation Processor*: implementa gli algoritmi per l'associazione automatica di tag testuali ai video e shot sulla base delle caratteristiche visive, utilizzando l'estrattore delle features appena spiegato

   c. *Query Processor*: implementa gli algoritmi per la risoluzione delle interrogazioni a partire dai tag o dalle caratteristiche visive

3. Strato inferiore, ovvero i tre **Database**:

   a. *Video DB*: contiene i video e i metatag (ovvero tutte quelle informazioni utili come, ad esempio, i timestamps dei tagli ottenuti dalla segmentazione automatica dei video in shots)

[6] Vedi paragrafo 3.1

b. *Feature DB*: contiene le caratteristiche rappresentative dei keyframes

c. *Tag DB*: contiene la tassonomia dei tag e le associazioni tra tag e video/shot

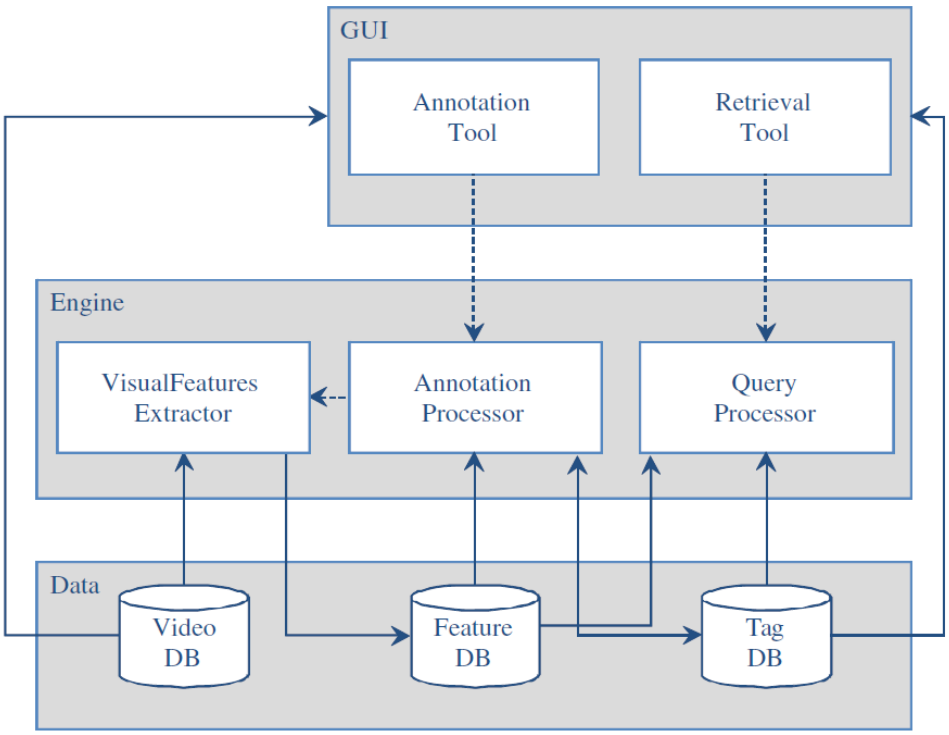


Figura 3-2 Architettura del framework SHIATSU

3.4 TAGGING

La soluzione utilizzata in SHIATSU è quella di non utilizzare tag monodimensionali, come accade normalmente nei servizi online, ma di suddividerli in gerarchie, dette **dimension**, che hanno tutte origini da una radice base indefinita, per poi svilupparsi ad albero: man mano che si scende nella gerarchia le foglie rappresentano via via concetti sempre più concreti, eliminando l'astrattezza dei primi livelli. In questo modo è possibile eliminare parole multi concettuali, ossia le parole omografe descritte nell'introduzione che, come già spiegato, possono portare a risultati di ricerca molto differenti. Per tornare all'esempio già fatto, in questo modo avremo il frutto pesca nella gerarchia "Piante/Frutti/Pesca", a

differenza della pesca sportiva, che potremmo trovare sotto “Sport/Pesca”. Ogni dimension è denominata dalla prima etichetta della propria gerarchia, quindi in questo caso le dimension sarebbero “Piante” e “Sport”.

Il software consente all’utente una grande autonomia per quanto riguarda il tagging, in quanto permette la modifica del punteggio di ogni tag associato al video, l’aggiunta o l’eliminazione manuale dei tag dei video, nonché la modifica completa delle gerarchie, consentendo di rimuovere, inserire e modificare le dimension.

SHIATSU offre inoltre anche il supporto alla compatibilità con applicazioni che usino strutture adimensionali, in quanto è in grado di lavorare con tassonomie a due livelli, cioè ponendo la radice generale al primo livello e tutte le altre foglie al medesimo livello sottostante.

Un’altra peculiarità particolarmente interessante offerta da SHIATSU è il tagging automatico dei video, basandosi sull’assunzione che immagini simili visivamente, lo siano anche semanticamente. L’associazione di certe caratteristiche, presenti in un database di partenza, a tag definiti permette di associare le feature ricavate da ogni immagine a tag, in modo automatico. Questo procedimento di tagging si articola in tre fasi:

1. Segmentazione fisica dei video in shots

2. Tagging semiautomatico degli shot ottenuti nel passo 1; vengono scelti i sei tag più frequenti dei keyframe dello shot, ordinati in ordine decrescente di frequenza

3. Annotazione dei video per propagazione, utilizzando i tag più rilevanti degli shot; vengono presi i primi dieci tag (al massimo) ordinati in ordine decrescente in base alla frequenza di uso negli shot e della lunghezza di questi

L’*Annotation Processor* è composto di conseguenza da tre moduli:

1. *Shot Detection Module*: per la suddivisione automatica dei video in shots, re-ingegnerizzato completamente durante questo lavoro di tesi

2. *Shot Tagging Module*: per l’annotazione degli shots

3. *Video Tagging Module*: per la propagazione delle annotazioni degli shots ai video

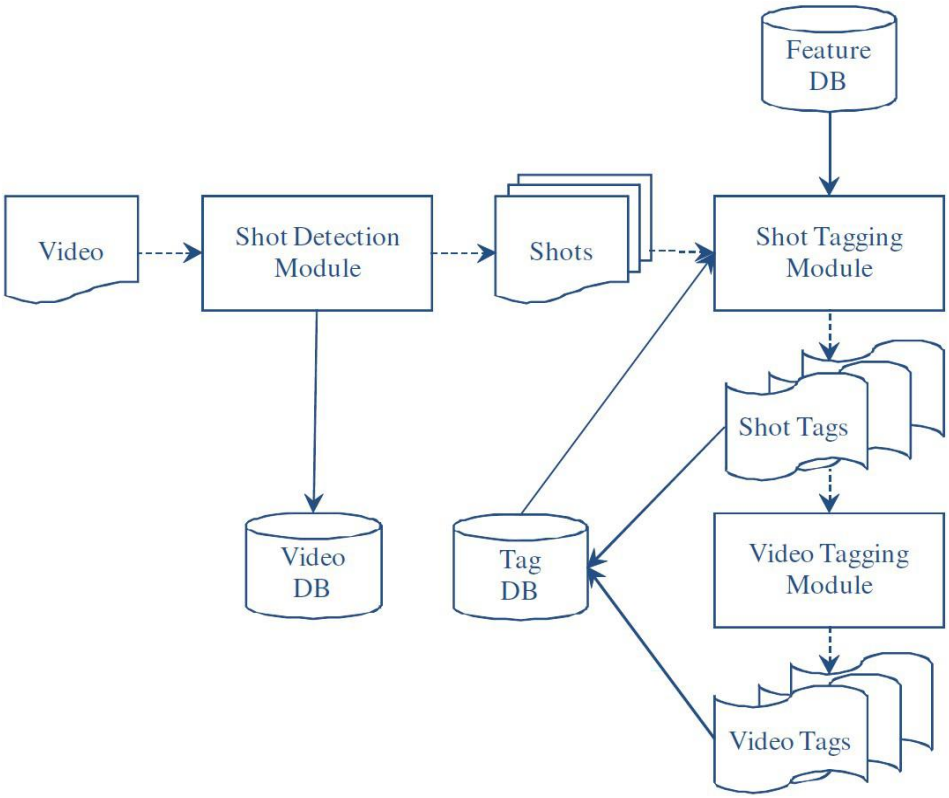


Figura 3-3 Processo di tagging semiautomatico, utilizzando SHIATSU

3.5 INTERFACCIA GRAFICA

Allo stato attuale delle cose, è possibile compiere ricerche in base a quattro. L'interfaccia grafica è basata sul pattern MVC leggero, dove il Model è distaccato da View e Controller, che invece sono molto legati in un'unica entità. È stata progetattata per effettuare ricerche e visualizzare gli shot e video trovati, tramite un media player.

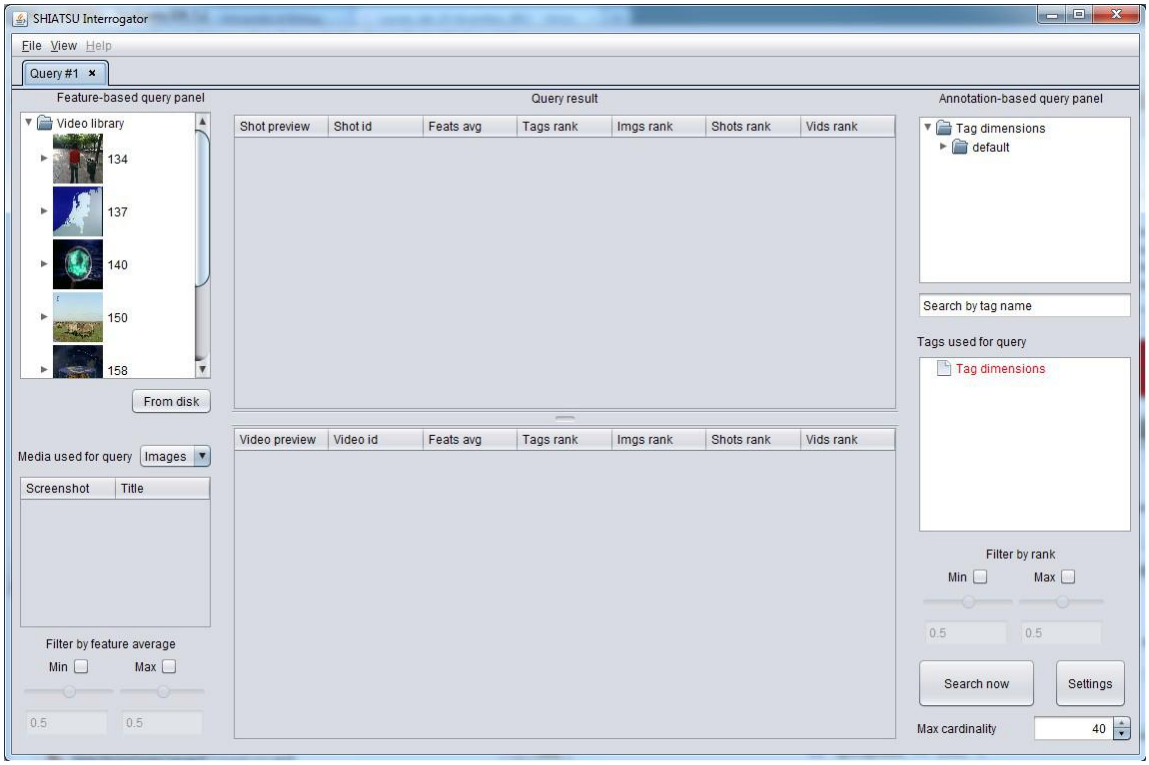


Figura 3-4 Schermata di ricerca

Allo stato attuale delle cose, è possibile compiere ricerche in base a quattro elementi differenti:

1. *Tag-Based*: permette all'utente di ricercare tramite la selezione, sulla parte destra dell'interfaccia, di uno o più tag, tra quelli presenti in database; i risultati verranno mostrati in output all'utente nella parte centrale, ordinati per punteggio di rilevanza

2. *Image-Based*: permette all'utente di effettuare la ricerca tramite il caricamento di una o più immagini residente su disco; il meccanismo di funzionamento di ricerca è paragonabile a ciò che avviene in Windsurf, cioè verranno ricercati i keyframe più simili all'immagine di partenza e, da questi, verranno ricavati gli shot che li contengono, per poi restituirli in output come risultato della query

3. *Shot-Based:* permette all'utente di selezionare uno degli shot già presenti, o di caricarne uno da disco, per restituire i video contententi immagini più simili ai keyframe che compongono lo shot passato in input

4. *Video-Based:* nonostante questo tipo di ricerca non sia stata ancora del tutto approfondita, in quanto si è scelto di raggiungere prima migliori risultati con la tecnica appena esplicata, permette all'utente di selezionare un video presente o direttamente da disco, per ottenere video e shot dal

contenuto simile. Importante ricordare che interi video di grosse dimensioni possono contenere shot con caratteristiche anche considerevolmente diverse tra loro

Occorre sottolineare che all'utente è permesso specificare ulteriori filtri sui risultati così ottenuti, in modo da affinare la ricerca considerando solo range specifici di punteggi di distanza.

In aggiunta a questa schermata troviamo la finestra di riproduzione di contenuti multimediali, per i video e shots presenti; qui troviamo il classico player multimediali, con i soliti tasti play/pause e la possibilità di scorrere tramite la barra di avanzamento del video.

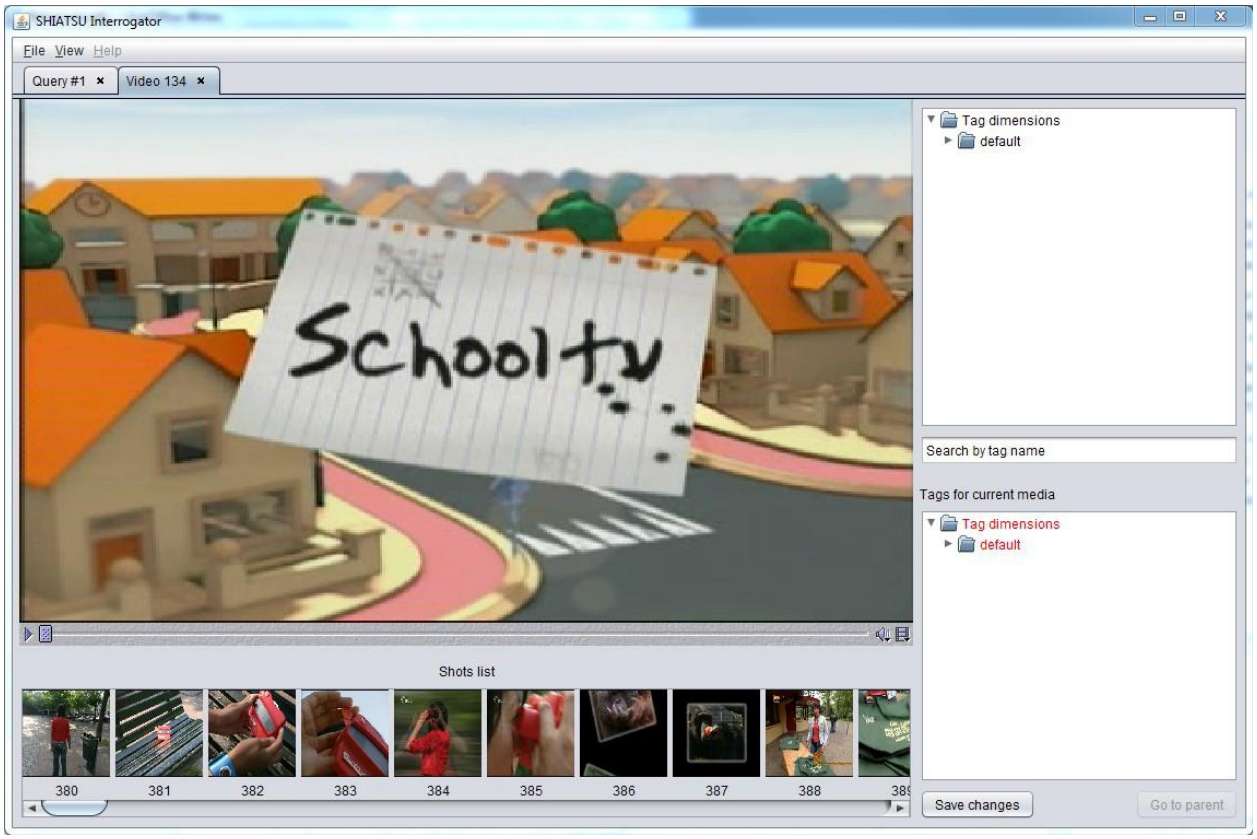


Figura 3-5 Finestra di visualizzazione di contenuti multimediali

Sono presenti in aggiunta al player, come si può vedere dallo screen soprastante, due pannelli laterali; quello superiore contiene la gerarchia completa dei tag, mentre in quello inferiore troviamo tutti i tag relativi al video, o shot, che si sta visualizzando. Tramite entrambi questi pannelli è possibile eseguire operazioni sui tag del video e di modificarne i relativi punteggi.

3.6 MODELLO

3.6.1 INTERFACCE

Il Model dell'applicativo SHIATSU è stato progettato mediante l'utilizzo di numerose interfacce, per migliorare l'estensibilità e la riusabilità del framework.

Possiamo identificare le principali nelle seguenti:

- **`IVideo`**: costituisce il video generico e contiene le signature per il setting di nome, path nel file system, tag associati e shot derivanti dalla segmentazione. Il metodo di maggiore importanza è *`getScreenShots()`*, il quale permette di ricavare la lista ordinata di tutti i keyframe estratti dal video. Un'altra caratteristica dell'interfaccia è l'implementazione di *`Iterable`*, infatti è necessario iterare i video per ottenere i suoi shots

- **`IShot`**: costituisce il generico shot ed eredita direttamente dall'interfaccia `IVideo`, perché è possibile eseguire una qualsiasi operazione per un video anche su un qualsiasi shot, essendo a sua volta un particolare tipo di video. A differenza di un normale video, uno shot possiede un padre e ha un path dove vengono memorizzati i propri keyframe

- **`IScreenShot`**: costituisce il generico keyframe. Espone informazioni sul genitore, sull'istante temporale corrispondente del video e offre la possibilità di ottenere una `ImageIcon` della risoluzione desiderata, utile da visualizzare nella GUI

- **`ITag`**, due semantiche distinte:
  - se punteggio ≤ 0, allora rappresenta un Tag generico di una certa ontologia
  - se punteggio > 0, allora rappresenta un Tag con cui è stato annotato un video o uno shot

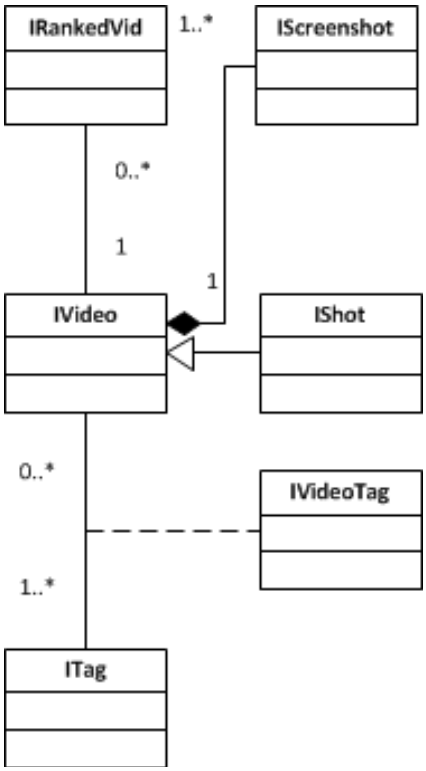


Figura 3-6 Diagramma UML delle interfacce principali del Model

Per ciascuna di queste interfacce, esiste un'opportuna *classe* ***Factory*** per permettere agevolmente il binding tra astrazioni e implementazioni degli oggetti. Ogni classe Factory implementa il pattern *Singleton*.

Infine abbiamo due *interfacce speciali*:

- **`IRankedVid`**: costituisce un video ottenuto dal risultato di una query, dotato quindi di differenti tipologie di punteggi (sui tag, sulle feature degli shot, etc…), ottenuti mediante un `IRankingAlgorithm`, e confronta i risultati sulla base di diversi criteri
- **`DefaultRankingAlgorithm`**: utilizzata per il calcolo dei i punteggi finali di un video o uno shot, che ricava semplicemente dal calcolo della media dei punteggi dei keyframe

3.6.2 MAPPER

Oltre alle già citate classi Factory, troviamo le *classi Mapper* per il recupero degli elementi direttamente dal framework SHIATSU. Queste classi rappresentano quindi l'intermediario di comunicazione tra applicazione e framework. I principali Mapper sono:

- **`GenericMapper`**: inizializza le comunicazioni, realizzando la connessione con il database MySQL tramite JDBC[7]

- **`IShotMapperNew`**: inizializza gli shots in vari modi, tramite SHIATSU, attraverso parametri come il nome del video padre, l'identificatore dello shot o un `ResultSet`. È inoltre in grado di interrogare il framework tramite i parametri passati dall'utente, per la restituzione di una lista di `IRankedVid`

- **`IVideoMapper`**: inizializza i video a partire dal nome del video o dalla riga attuale di un `ResultSet`

3.7 QUERY

Il *Query Processor*, responsabile della gestione delle richieste, sfrutta tre diversi paradigmi di query:

1. **Keyword-Based (KS)**: il più comune, utilizzato dai normali motori di ricerca; permette la ricerca di elementi che contengano, nella propria annotazione, le keyword inserite dall'utente; i risultati vengono ordinati per pertinenza e presenza

2. **Feature-Based (FS)**: è il tipo di ricerca, come suggerisce il nome, per somiglianza di feature tra i keyframe del video e l'immagine data in ingresso dall'utente

Keyword&Feature-Based (KFS): è la combinazione delle due ricerche appena analizzate, restituendo prima il risultato dell'intersezione delle due ricerche, seguiti da quelli solo Keyword-Based, infine solo i Feature-Based

3.7.1 QUERY SEQUENZIALI BASATE SU SHOT

La prima implementazione del framework prevedeva solamente le query sequenziali, avvalendosi dello stesso principio di Windsurf, ma innalzandosi di un livello nella gerarchia: si considerano gli shot come documenti e i keyframe come frammenti di questi.

`IScreenshotType`: permette l'astrazione dall'implementazione effettiva degli oggetti; offre il metodo *`GetDistance()`* per il calcolo della distanza tra due immagini

[7] Connettore per database che permette l'accesso alle basi di dati da qualsiasi software scritto in JAVA

Da questa interfaccia, abbiamo due implementazioni differenti:

- **`WindsurfImageScreenshot`**: istanzia i keyframe come WindsurfImage, estraendo le regioni che compongono l'immagine e calcolandone le features
- **`HistogramScreenshot`**: istanzia i keyframe come HSVImage, ottenendo i valori di Hue, Saturation e Value per ogni pixel e utilizzandoli poi per il calcolo della distanza tra immagini tramite istogrammi HSV a 32 bin

La classe `ScoredResultShot` rappresenta uno shot risultato di una query, contenente identificatore e distanza rispetto allo shot passato come argomento in input alla richiesta. Gli algoritmi di ricerca sono implementati tramite il metodo già descritto *kNN*.

La comparazione tra shots viene eseguita confrontando i keyframe che li compongono, attraverso il metodo *`executeShots()`* della classe `RandomAccessShiatsu`, che accede ordinatamente a tutti gli screenshots dei due shots che va a raffrontare.

Capitolo 4. VIDEO SEGMENTATION

Come già accennato nell'introduzione del capitolo 3, lo scopo della Video Segmentation è quello di suddividere una sequenza video in tanti shot, che rappresentino ognuno lo stesso evento o azione; possiamo individuare molto semplicemente ciascuno shot, considerando dei tagli nella sequenza nei momenti di accensione o spegnimento della telecamera, i cosiddetti "cambi di camera".

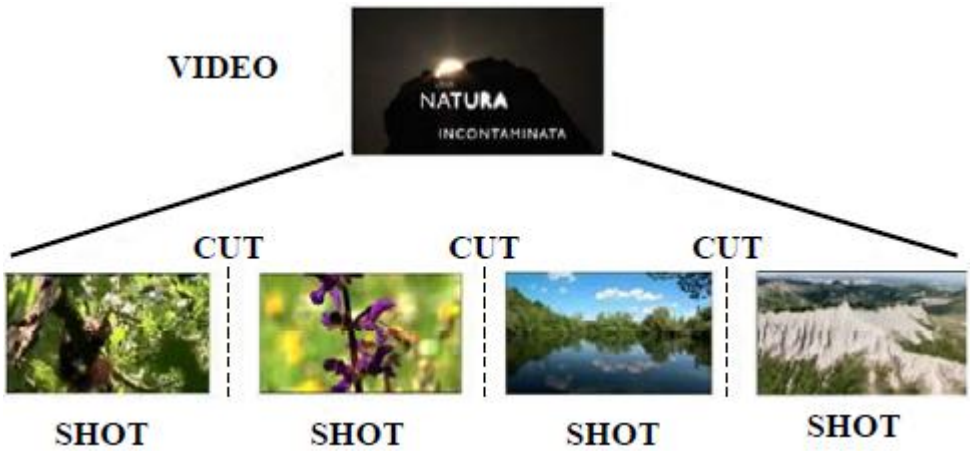


Figura 4-1 Esempio di suddivisione di un video in shot

È possibile classificare le transizioni tra shot contigui in due categorie: brusche o graduali. Nelle prime si ha un cambiamento netto tra un frame e il successivo; nelle seconde, la cui individuazione risulta intuitivamente più complessa, il passaggio graduale tra shot comprende una successione di effetti speciali (*fade in, fade out, wipe, etc…*). Altri tipi di transizioni possono essere attribuiti a fenomeni di *motion*, quali spostamenti della telecamera o di oggetti nella scena. Sul tema della detection dei video si è dibattuto per tutto l'ultimo ventennio (8).

Non essendo necessaria una precisione scrupolosa, che invece si deve avere in software quali ad esempio quelli di video-editing, in SHIATSU è stata privilegiata la velocità a discapito dell'accuratezza.

4.1 THRESHOLD DETECTION

La shot detection di default di SHIATSU sfrutta gli istogrammi di colore e le forme degli oggetti dei frame per rilevare un cambiamento di scena; questo perché, in una transizione, generalmente cambiano sia i colori che la struttura dei bordi degli oggetti dell'immagine.

L'algoritmo si serve quindi di una doppia soglia dinamica, perché valori fissi non sarebbero adatti a tante gamme di video differenti. Un cambio di scena è individuato, di conseguenza, prima usando le *color features*, quindi le *edge features*.

Per il procedimento di shot detection è stato scelto un approccio bilanciato per avere un buon rapporto tra efficienza e facilità di implementazione.

4.1.1 DIFFERENZA HSV

Le informazioni relative al colore di ogni singola immagine del video è rappresentata, come anticipato, utilizzando la distribuzione HSV dei pixel che la compongono. Ogni frame è caratterizzato usando tre istogrammi: 12 bin per la tonalità, 5 per la saturazione e 5 per il value.

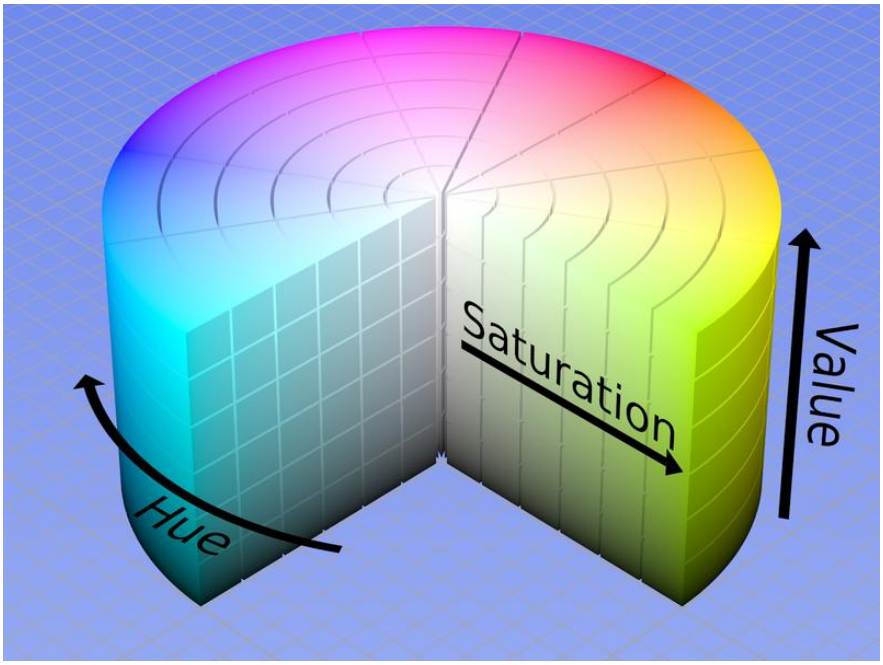


Figura 4-2 Descrizione grafica dello spazio colore HSV

La seguente formula è stata utilizzata per il calcolo della differenza tra due istogrammi di colore: $d_{b2b}(h,h') = \frac{1}{2N}\sum_i |\, h[i] - h'[i]\,|$ (23), dove N rappresenta il numero dei pixel del frame. Da qui, la distanza HSV tra due frame consecutivi *k* e *k*+1, $d_{HSV}(k,k+1)$, è definita come la somma delle differenze bin per bin dei tre istogrammi HSV sopraelencati, $d_{HSV}(k,k+1) = \frac{1}{6N}\sum_i |\, h_k[i] - h_{k+1}[i]\,|$. Rispetto a (31), questa formula permette una miglior discriminazione tra frame di taglio e non; questo perché i valori di distanza tra frame non di taglio, rimane nello stesso range per tutto il video.

4.1.2 ECR

Un cambio di scena presenta molto spesso una notevole modifica nelle forme degli oggetti presenti nell'immagine, ovvero nella disposizione del contenuto. Possiamo pertanto calcolare quante nuove forme entrano ed escono dall'immagine in due frame consecutivi, per determinare l'occorrenza di un taglio. Definiamo forme entranti tutte quelle che sono presenti nel frame attuale e non nel precedente e, di conseguenza, uscenti quelle presenti nel frame attuale e non nel successivo.

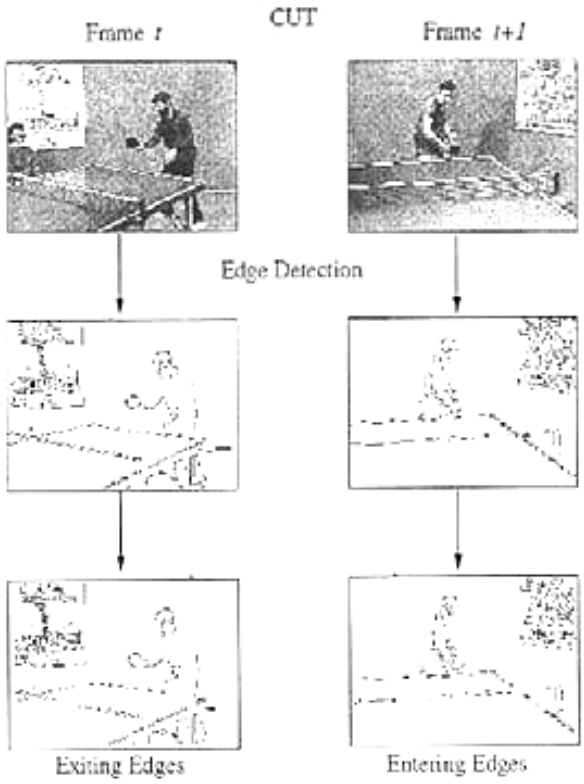


Figura 4-3 Esempio forme entranti e uscenti in una transizione

L'**Edge Change Ration (ECR)**, ovvero quanto cambiano le forme, entranti e uscenti, tra due frame consecutivi *k* e *k*+1 è calcolato dalla seguente formula:

$ECR(k,k+1) = \max\left(\frac{x_k^{out}}{\sigma_k}\frac{x_{k+1}^{\text{int}}}{\sigma_{k+1}}\right)$, dove σ_k è il numero di edge pixel nel frame *k* (analogamente σ_{k+1}), x_k^{out} e x_{k+1}^{in} rappresentano rispettivamente i pixel uscenti ed entranti nei frame *k* e *k*+1.

L'uso di ECR aiuta notevolmente la rilevazione di differenze tra frame e permette di evitare un gran numero di falsi positivi nei risultati.

4.1.3 THRESHOLD

Una volta ottenute le distanze tra frame consecutivi, occorre confrontarle con un valore di soglia: ogniqualvolta una distanza supera il valore di soglia, allora possiamo dichiarare che è presente una transizione di scena, quindi è necessario eseguire un taglio.

Di conseguenza la scelta di questo valore di soglia è di primaria importanza: un valore troppo basso produrrebbe intuitivamente una sovra-segmentazione, mentre un valore troppo alto porterebbe la non rilevazione di cambi di scena.

Attualmente esistono tre alternative in letteratura: valori di soglia fissi (21,28,31,35,43), clustering (2,26) e valori di soglia dinamici. L'uso di valori fissi non può essere impiegato nel nostro caso, in quanto abbiamo a che fare con una gamma di video troppo ampia, mentre il metodo di clustering non è molto efficiente quando le differenze tra frame non sono molto elevate.

Per SHIATSU, è stato modificato appropriatamente l'approccio illustrato in (31), che utilizza un sistema di doppia soglia dinamica, per migliorare l'accuratezza di rilevazione degli shot, mantenendo la complessità del processo a livelli ragionevoli.

Per determinare il valore di soglia HSV θ_{HSV}, SHIATSU calcola la media delle distanze HSV più elevate:

$$\theta_{HSV} = \frac{\beta_{HSV}}{M / f}\sum_{i=M-M/f+1}^{M} L_{HSV}(i),$$

dove β_{HSV} rappresenta il parametro di sensitività (default a 1), *M* è il totale di frame del video, *f* è il framerate del video e L_{HSV} è la lista ordinata in ordine crescente delle distanze HSV tra tutti gli shot consecutivi.

Per determinare il valore di soglia ECR θ_{ECR}, si procede quasi analogamente:

$$\theta_{ECR} = \frac{\beta_{ECR}}{2M / f} \sum_{i=M-2M/f+1}^{M} L_{ECR}(i),$$

dove β_{ECR} rappresenta il parametro di sensitività (default a 1) e L_{ECR} è la lista ordinata in ordine crescente dei valori ECR. Si considera una finestra di data più ampia (2M) per il calcolo di θ_{ECR} perché le distanze ECR sono solitamente più sparse rispetto a quelle HSV.

L'algoritmo sviluppato filtra quindi tutti i frame aventi distanza HSV $\leq \theta_{HSV}$ e considera i valori ECR dei candidati rimasti, a loro volta esclusi nel caso in cui il valore di ECR sia $\leq \theta_{ECR}$; i frame che superano entrambe le soglie sono quindi considerati come frame di taglio.

4.2 IMPLEMENTAZIONE

Prima di poter procedere all'utilizzo di SHIATSU è necessario configurare adeguatamente l'ambiente di sviluppo.

La configurazione utilizzata è stata la seguente:

- **Sistema**:
  - SO: Windows 7 Ultimate
  - CPU: Intel Pentium P6200 @2.13GHz
  - RAM: 2GB DDR3
- **Ambiente di sviluppo**:
  - SO: NetBeans IDE 7.3.1
  - JDK 7.2 (necessaria versione 32bit per utilizzare JMF)
  - MySQL Workbench 5.2
- **Ambiente di runtime**:
  - MySQL Server 5.5
  - Java Media Framework 2.1.1e con Performance Pack
  - JRE 7 (versione 32bit)

4.2.1 JAVA MEDIA FRAMEWORK

Il **Java Media Framework (JMF)** è una libreria che permette l'aggiunta e la manipolazione di contenuti audio, video e altri tipi di media all'interno di applicazioni e applet Java; questo pacchetto è opzionale ed estende la Java Platform Standard Edition (J2SE). Offre agli sviluppatori un insieme di strumenti per sviluppare applicazioni scalabili e indipendenti dalla piattaforma che si utilizza.

JMF ha un'architettura che permette l'accesso diretto ai dati multimediali e da la possibilità di essere personalizzato ed esteso; è stato infatti progettato principalmente per:

- Supportare la cattura di dati multimediali
- Sviluppare applicazioni Java per lo streaming multimediale
- Permettere l'implementazione di soluzioni personalizzate basate sulle API esistenti e di integrare facilmente nuove funzionalità con l'infrastruttura esistente
- Permettere l'accesso ai dati "grezzi"
- Permettere lo sviluppo di componenti personalizzabili, i cosiddetti *JMF plugins*, come ad esempio codecs, effects processors o renderers

I principali package incontrati nello sviluppo con JMF sono:

- **`javax.media`**: contiene le principali classi di JMF
- **`javax.media.control`**: permette la lettura e la modifica dei parametri quali bitrate, framerate e altri parametri del processo di encoding
- **`javax.media.format`**: per la descrizione dei formati supportati.

La struttura della libreria è risultata non di facile comprensione e, in aggiunta alla difficoltà di base, è necessario ricordare che il supporto e gli aggiornamenti della libreria, sono terminati nel 2004.[8]

[8] Maggiori informazioni su: http://www.oracle.com/technetwork/java/javase/faq-jmf-137005.html

4.2.2 MODELLO PREESISTENTE

Il modello preesistente constava principalmente di una macro classe difficilmente estendibile, nonché inservibile in quanto non completamente funzionante e indentata in maniera assolutamente casuale, denominata `FrameAccess` che, servendosi del codec creato ad hoc per l'accesso ai singoli frame, ossia la classe `FrameAccessCodec`, permetteva, o si premetteva di consentire, l'individuazione dei tagli.

È stato necessario re ingegnerizzare completamente il tutto, in quanto erano presenti innumerevoli dipendenze circolari e una buona parte del codice era commentata. Il mio lavoro è consistito principalmente in una *reverse engineering* del modello preesistente, ovvero è stato necessario analizzare dettagliatamente il funzionamento del codice preesistente, nonché lo studio della parte di JMF che lavorasse con i video, per poterlo re implementare da zero, sfruttando i pattern di programmazione, per rendere il modello del segmentatore estendibile e, soprattutto, usufruibile dalle classi preposte al taglio vero e proprio del video. Fino al mio intervento, non era ancora possibile automatizzare le operazioni di taglio, in quanto tutto il procedimento avveniva manualmente, utilizzando tool esterni al framework.

4.2.3 MODELLO

L'idea di base è stata quella di offrire ai livelli superiori e inferiori una singola classe. **`MediaVideo`** rappresenta un file video su cui si voglia eseguire la segmentazione, infatti il costruttore presenta come parametri il solo nome del file video da elaborare. I metodi principali offerti sono:

- `void setDetector(IDetector)`: permette di impostare il tipo di `IDetector`, il cui funzionamento illustrerò in seguito, da utilizzare per la rilevazione degli istanti di taglio
- `void processVideo()`: lancia il processo vero e proprio di detection degli istanti di taglio; nel caso in cui non sia stato impostato alcun Detector, di default utilizzerà il `ThresholdDetector`
- `Vector<Long> getCutTimeStamps()`: restituisce il vettore contenente tutti gli istanti di taglio, ordinati in modo ascendente
- `Vector<Long> getCutSequenceNumbers()`: restituisce il vettore contenente i frame in cui avviene una transizione, quindi un taglio, ordinati in modo ascendente

- `void setKeyFrameSelector(IKeyFrameSelector)`: permette l'impostazione del modo di scelta dei keyframe, ovvero un'implementazione dell'interfaccia IKeyFrameSelector, dati gli shot
- `Vector<Vector<Long>> SaveKeyFrames()`: salva i keyframe su disco e ne restituisce il vettore contenente un vettore di timestamp, indicanti gli istanti cui vengono salvati gli screenshots, per ogni shot individuato dal detector; nel caso in cui non sia stato impostato alcun selettore, di default utilizzerà un istanza di `IntervalSelector`, calibrato per selezionare un keyframe ogni 2 secondi

Oltre a questi principali, vengono ovviamente offerti tutti quei metodi per ottenere informazioni sul video, quali durata, framerate, dimensione della finestra, path assoluto e relativo, etc…

Per il processing del video, è stata mantenuta la scelta di avere un'implementazione dell'interfaccia `ControllerListener` fornita da JMF e dell'interfaccia `Codec`, anche questa offerta da JMF.

`VideoProcessor` implementa quindi `ControllerListener` e si occupa della gestione del processo vero e proprio di processing, sia nella fase di detection dei tagli, che nella parte del salvataggio dei keyframes.

Per una completa estendibilità, è stato previsto l'utilizzo del ***Pattern Strategy***, che è stato implementato due volte: per la strategia da associare al metodo di detection e per la strategia da associare al metodo di salvataggio dei keyframe. Il *pattern Strategy* permette l'intercambiabilità degli algoritmi in modo trasparente al Client; in questo modo, qualunque sia il metodo di rilevamento degli shot e qualunque sia il metodo di salvataggio dei keyframes su disco, dai risultati prodotti non è possibile fare alcuna assunzione su quale sia stata la strategia utilizzata.

`IDetector` è dunque il generico Detector, assimilabile all'*abstract strategy* del pattern appena illustrato; questo fa si che tutte le future implementazioni di un qualsivoglia detector siano costrette ad ereditare da questa interfaccia.

È stata necessaria l'introduzione di un'ulteriore interfaccia per i detector, per distinguere quelli che lavorano sui frame, come il Threshold, da quelli che lavorino a partire da altro, come ad esempio l'audio del video. Una possibile implementazione di un detector, che rilevi gli istanti di taglio sulla base del solo audio, è in fase di sviluppo da Giovanni Concas, attualmente tesista per la laurea specialistica.

Da qui, l'introduzione dell'interfaccia **`IFrameDetector`** (figlia di `IDetector`), per permettere al `VideoProcessor` di sapere se sia necessario lavorare o meno con l'implementazione di `Codec`, già anticipato, ovvero la classe **`MediaVideoCodec`**; questa espone all'esterno sostanzialmente un metodo per la pro cessazione di un frame e per il settaggio dello stato (sono al momento previsti solo i due stati di PROCESSING, in fase di detection, e SAVING, in fase di salvataggio dei keyframes). Questa classe viene in ogni caso utilizzata durante l'estrazione dei keyframes, in quanto è necessario scorrere il video per estrarre da esso un singolo frame e permetterne il salvataggio su disco.

Analogamente ad `IDetector`, l'interfaccia **`IKeyFrameSelector`** rappresenta il generico selettore di keyframe.

Le implementazioni, ovvero le strategy, create da questa interfaccia sono attualmente due:

- **`IntervalSelector`**: permette di selezionare, quindi salvare, dato in creazione un intervallo in secondi, tutti i keyframe trovati scorrendo lo shot dall'inizio fino alla fine, aggiungendo ogni volta il valore dell'intervallo; l'intervallo predefinito di default è ogni 2 secondi
- **`InitMiddleFinalSelector`**: permette di selezionare il salvataggio dei soli keyframe al primo, medio o istante finale, o di combinazioni di questi; potrei ad esempio decidere di selezionare solo il primo e il medio

La classe **`ThresholdDetector`** è stata di conseguenza la classe più grossa, nonché la più importante per il mio lavoro di reingegnerizzazione; questa implementa, come si può facilmente intuire dalle righe precedenti, l'interfaccia `IFrameDetector`, di conseguenza implementa sia i metodi di `IDetector`, ovvero sia quelli per ottenere i vettori degli istanti di taglio, che il metodo *`ProcessFrame(Buffer)`*, per il processing di un frame, utilizzato da `MediaVideoCodec`.

All'interno della classe sono stati implementati gli algoritmi per il calcolo delle soglie HSV ed ECR descritti in 4.1; di conseguenza i passi fondamentali che deve eseguire sono nell'ordine: il calcolo delle soglie HSV e ECR, e, una volta finito il processing di tutti i frame, il rilevamento degli shot di tipo abrupt, quindi di quelli di tipo graduale.

Per l'inizializzazione della classe, occorre passare nel costruttore i parametri β_{HSV} e β_{ECR}, il cui tuning consigliato per l'utente è stato definito grazie alla fase di testing.

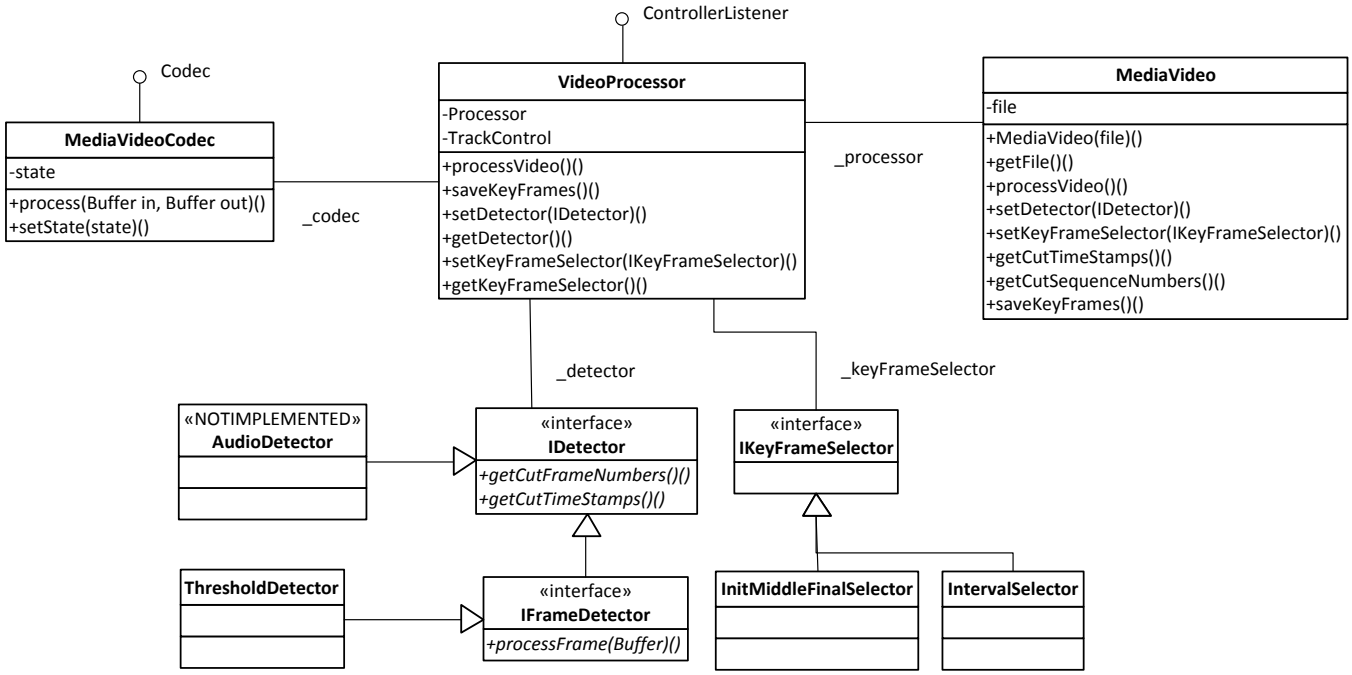


Figura 4-4 Schema UML delle classi

4.2.4 CLASSI DI UTILITY

La classe **`BufferUtility`** fornisce il metodo statico `BufferToArray(Buffer)`, per la conversione di un Buffer, oggetto del framework JMF presente nel package `javax.media`, in un array di interi; permette l'analisi del frame in quanto basata sul modello RGB a 32 bit.

La classe **`ImageConvertUtility`** fornisce il metodo statico `frameArrayToGrayImage(int buf, Dimension dim)` per convertire l'immagine passata, sotto forma di array (vedi capoverso precedente), nel tipo TYPE_BYTE_GRAY per permetterne il calcolo e l'analisi delle edges entranti e uscenti; si noti come `Dimension` sia un'altra classe appartenente al framework JMF.

Le ulteriori classi di utility contenute nel package da me prodotto contengono in larga parte adattamenti al mio codice, oppure sono state semplicemente

modificate e commentate in modo da essere leggibili e di facile comprensione a chi potrà prenderle in mano in futuro.

4.2.5 NOTE AGGIUNTIVE

Il primo modello da me ideato e realizzato era piuttosto differente da quello appena mostrato, in quanto prevedeva il processing di tutte le immagini del video a posteriori, dopo averle allocate in memoria; benché questo rendesse i tempi di elaborazione molto più rapidi, non era funzionale con filmati più grandi di 4-5 minuti, in quanto questa operazione saturava troppo in fretta la memoria; nonostante le singole immagini non avessero dimensioni eccessive, si fa presto a capire che video con framerate dell'ordine anche solo di 40fps, producono 2400 immagini al minuto; per questo tale modello è stato abbandonato dopo averlo testato con video più lunghi di un paio di minuti.

Il modello implementato processa tutti i frame, salvando per ogni frame praticamente solo le similarità e le edge uscenti ed entranti; questa soluzione permette di mantenere in memoria solo le informazioni strettamente necessarie al rilevamento dei punti di taglio.

Capitolo 5. RISULTATI SPERIMENTALI

5.1 PASSI PRELIMINARI

Per condurre esperimenti sulla precisione e l'efficacia del tool di segmentazione, è stato opportuno utilizzare un dataset di **Ground Truth** di video messo a disposizione da TRECVID[9] nel 2007, composto complessivamente da 40 video.

Si è quindi deciso di fare l'import di questo benchmark in SHIATSU, oltre che per i test sulla segmentazione e il taglio fisico dei video, anche per rendere i dati fruibili velocemente a future attività di testing sull'efficienza delle modalità di interrogazione esistenti, più approfondite di quanto non siano state fatte fino ad ora: il database precedente fornito con SHIATSU su cui tutti si erano adoperati per i test si è rivelato inadeguato a causa delle misere dimensioni (constava infatti di soli sette video).

TRECVID fornisce con i video anche una serie di informazioni relative ai filmati, prodotte da un nutrito gruppo persone, indicanti la presenza o meno di oggetti all'interno del video; questi oggetti sono stati categorizzati dalla TRECVID, per comodità, in una serie di 39 concept, ovvero delle categorie di appartenenza, quali ad esempio animali, sport, montagne, vegetazione, aeroplani o automobili, etc…

Per l'estrapolazione di queste informazioni dai file forniti con il dataset, è stata di conseguenza necessaria l'implementazione di una classe di parsing per ogni tipologia di file provvista; queste informazioni sono inserite

[9] http://trecvid.nist.gov/ Serie di conferenze sulla ricerca nell'ambito dell'information retrieval, sponsorizzata dal NIST (National Institute of Standards and Technology)

all'interno di documenti puramente testuali (file `.txt`), oppure in schemi XML[10].

Date queste due tipologie, si è pensato di creare due classi astratte dal quale ereditare ogni implementazione concreta di ciascun parser:

- **`GenericFileReader`**: generalizzazione di parser da cui erediteranno tutti indistintamente; offre i costruttori tramite File o String
- **`GenericXMLFileReader`**: astrazione che eredita dalla classe astratta appena descritta; rappresenta il generico parser per file XML e contiene, oltre ai costruttori del padre, una property di tipo `Document` che rappresenta il documento XML da parsare

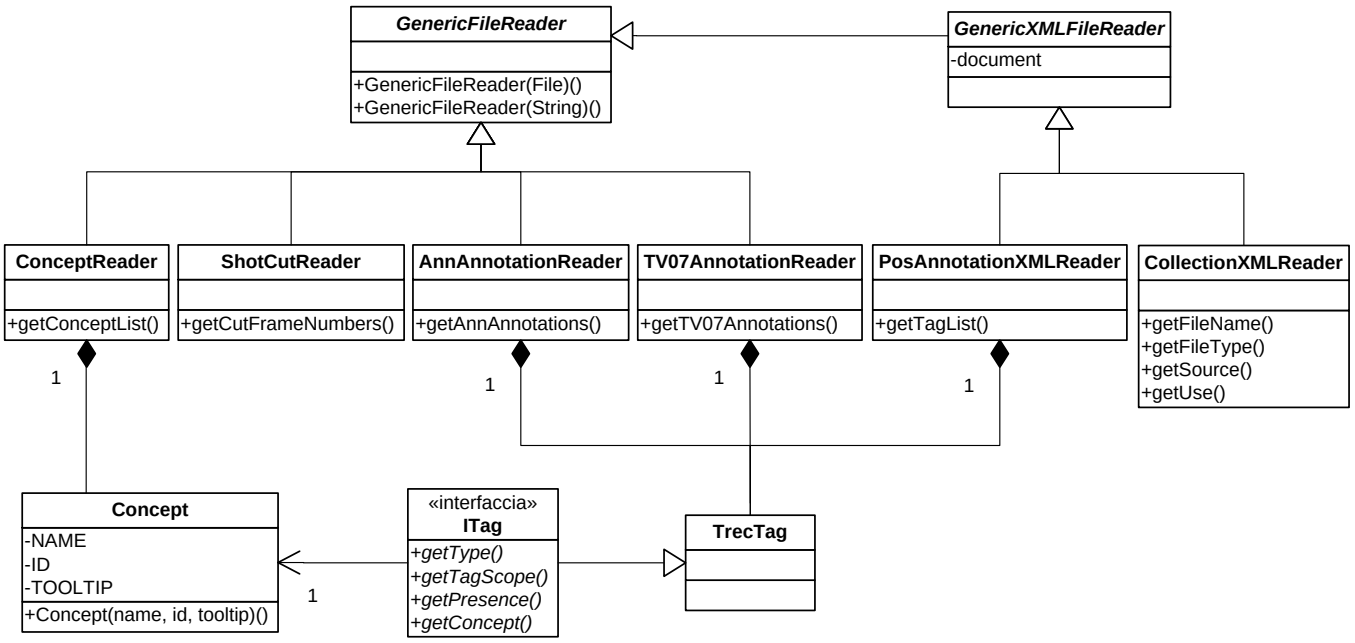


Figura 5-1 Schema UML delle classi necessarie per l'import

Come si può capire dallo schemo sovrastante, sono presenti tre tipi di file di annotazione, ognuno scritto in modo differente; da qui l'esigenza della creazione di un parser ad hoc per ciascuno (`AnnAnnotationReader`, `TV07AnnotationReader`, `PosAnnotationXMLReader`). Si può vedere dallo schema, che i parser producono ognuno una lista di Tag, contenenti il concetto che compare (spiegato precedentemente), la tipologia

[10] eXtensible Markup Language. Metalinguaggio utilizzato in ogni ambito, sia per la creazione di nuovi linguaggi di markup, da usare in ambito web, che per la descrizione di strutture, ovvero il caso che si è a noi presentato

dell'annotazione (pos, ann, tv), l'immagine cui riferisce, la presenza o meno di questo concetto nell'immagine.

`ConceptReader` popola la lista dei concetti esaminati da TRECVID, dal file contenente la descrizione dei 39 concept (feature_definition.txt).

`CollectionXMLReader` legge dal file .xml che contiene la descrizione della collezione completa di tutti i video del dataset; viene indicato l'id, il nome del file video, la fonte e la tipologia di file (il benchmark fa riferimento a soli MPEG-1[11]).

Oltre a tutti questi file, contenenti le annotazioni relative ai video, di primaria importanza per i miei test sulla segmentazione sono ovviamente i file contenenti gli il numero dei frame in cui iniziano e finiscono i vari shot (secondo la Ground Truth di TRECVID), ovvero i frame di taglio, per il confronto con quelli rilevati dalla detection automatizzata. Questi file rappresentano ciò che viene chiamata "Shot Reference".

Per il procedimento di import dei dati su DB è stato necessario creare una nuova implementazione di `IWorkHandler` (11), oltre che estensione di `SwingWorker` : `TrecShiatsuImporterWorkHandler`.

Questa classe permette l'inserimento in DB dei video di TRECVID (e quindi dei relativi shots, keyframes e features), dopo aver letto dal file di configurazione `Shiatsu.xml`, convertendoli in un formato utilizzabile dal cutter vero e proprio (che essendo basato su JMF, può lavorare su un ristretto numero di formati e bitrate); quindi avvia il cut dopo aver estratto i punti di taglio dai file di Shot Reference sopracitati.

Interessante notare la realizzazione di un nuovo tipo di `IDetector` per il passaggio degli istanti di taglio dallo `ShotCutReader`. La classe `TrecvidDetector` simula semplicemente il comportamento di un normale detector (offre infatti gli stessi metodi di `IDetector`), ma è preposto unicamente alla corretta istanziazione del parser e alla restituzione del vettore ricevuto da quest'ultimo a chi ne faccia richiesta; inoltre prevede una conversione da framenumbers a timestamps per la restituzione di quest'ultimi.

[11] Standard introdotto da MPEG (Moving Pictures Experts Group) ottimizzato per le applicazioni video a basso bitrate

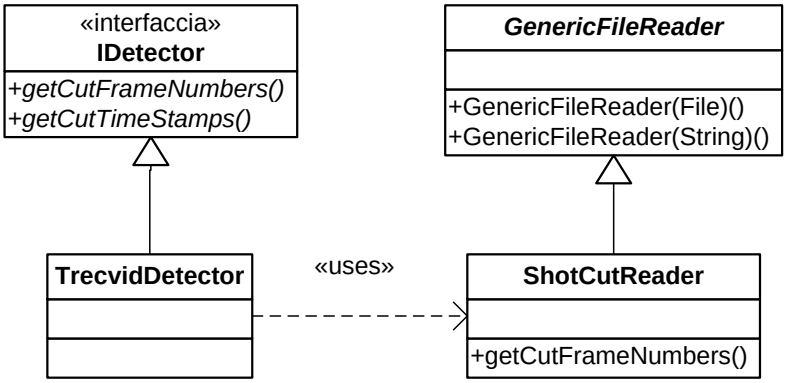


Figura 5-2 UML TrecvidDetector

5.1.1 PREMESSA SULLA CONVERSIONE

Come già detto, si è resa necessaria l'introduzione di un metodo di conversione in un formato standard dei video, per poter essere dapprima elaborati correttamente dal Detector, poi tagliati dalla classe adibita al taglio.

Per questo procedimento viene utilizzato uno strumento di elaborazione esterno (comunque invocato automaticamente tramite JAVA), ovvero FFMPEG, della omonima suite software[12].

È di conseguenza indispensabile la presenza fisica del file "`ffmpeg.exe`" sulla macchina che esegue SHIATSU (il path assoluto viene specificato in un apposito file di configurazione).

I video di TRECVID che finiscono su DB subiscono quindi questo processo di conversione, che li trasforma dal formato di compressione MPEG-1 al formato MOV, non compresso, con un framerate costante di 25fps; conseguenza di questo procedimento è l'enorme mole dei video. Questo è stato l'approccio migliore utilizzabile per rendere il tutto universale, avendo un discreto rapporto qualità/prezzo: altre prove effettuate hanno portato o a dimensioni di video ancora più grandi, o a qualità di visualizzazione davvero pietose.

Occorre far presente un notevole limite attuale, dovuto a JMF (nonché a Windows): se il video in formato `.mov` risultante da una conversione dovesse

[12] http://ffmpeg.org/ Suite software completa per la riproduzione, conversione e registrazione di contenuti multimediali

superare i 2GB, non sarebbe possibile per il framework JMF elaborarlo, per un limite di dimensioni imposto dalla Microsoft[13].

5.2 STRUTTURA DEL DATABASE DI TEST

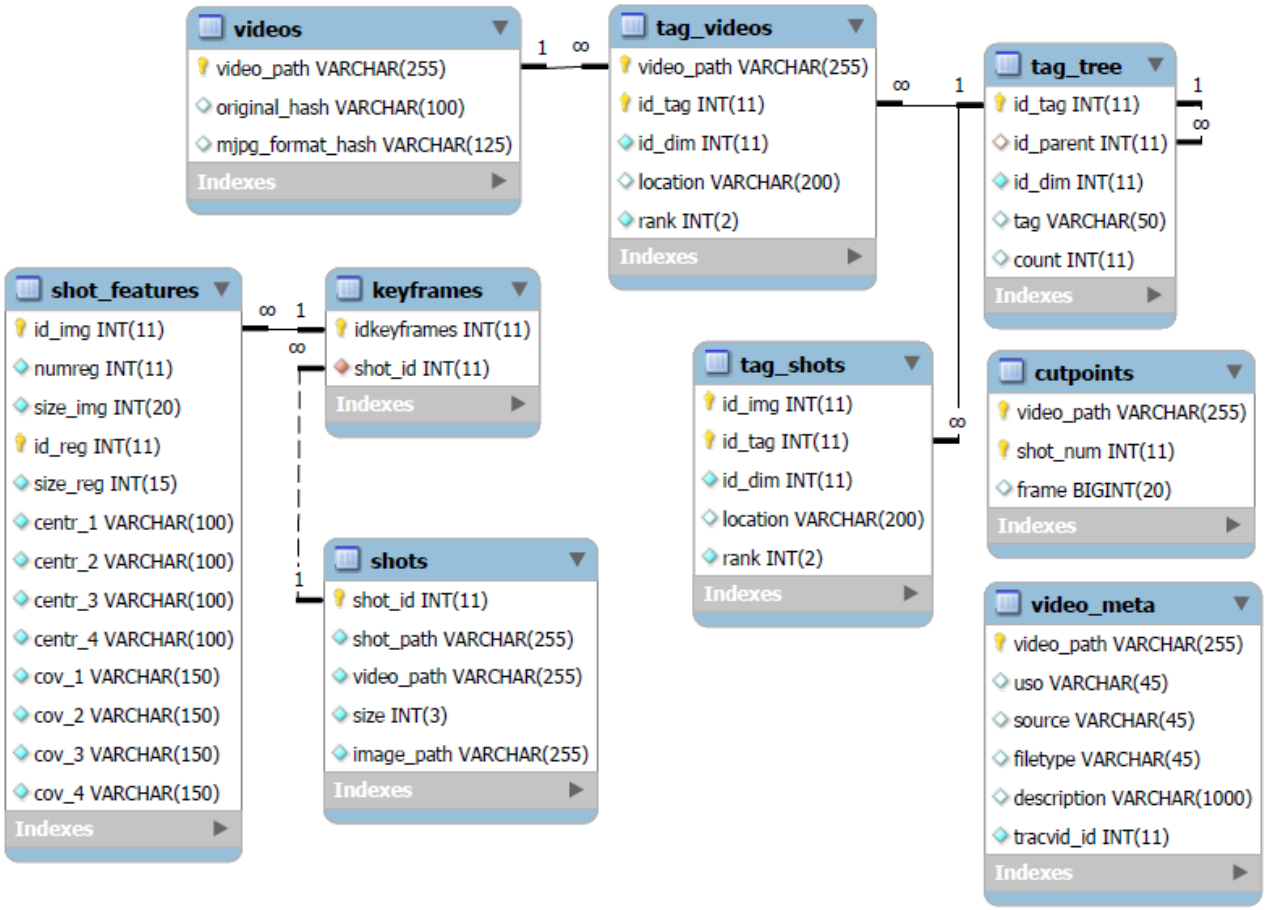


Figura 5-3 Struttura del database di testing

Per l'esecuzione dei test relativi all'accuratezza della detection, l'unica tabella dello schema sovrastante che è stato necessario interrogare è la tabella **`cutpoints`** (a destra nell'immagine).

La chiave primaria della tabella è data dall'unione dei due attributi **`video_path`**, rappresentante il path del video a partire dalla nostra cartella di riferimento, e **`shot_num`**, che indica il numero dello shot cui avviene il taglio; gli shot sono inseriti in ordine crescente partendo dallo shot 0 che

[13] Fonte: http://support.microsoft.com/kb/193656

rappresenta la parte di video che va dall'istante 0 all'istante del primo taglio. Il rimanente campo, **`frame`**, indica appunto il numero del frame cui avviene il taglio.

La classe adibita al testing del thresholding interroga la tabella appena illustrata tramite una semplice query JDBC; `TestingDao` implementa appunto il design pattern *DAO (Data Access Object)* che permette la restituzione dei framenumbers relativi alle transizioni tramite il comodo metodo `selectCutFrameNumbers(String videoname)`.

5.3 MISURA DELLA QUALITÀ

La qualità della shot detection è calcolata sulla base di due parametri:

- **Precision**, ovvero la probabilità che un taglio identificato dal detector sia un taglio reale

  $$P = \frac{CC}{CC + FC}$$

  CC (*correct cuts*) rappresenta il numero dei tagli correttamente rilevati; viene accettata una tolleranza di 250ms rispetto ai tagli forniti da TRECVID per indicare un taglio corretto
  FC (*false cuts*) rappresenta il numero dei tagli rilevati, ma erronei

- **Recall**, ovvero la probabilità che un taglio reale verrà rilevato dal nostro detector

  $$R = \frac{CC}{CC + MC}$$

  MC (*missed cuts*) rappresenta il numero dei tagli reali, ma non rilevati dall'algoritmo di ricerca

Il dominio di questi due parametri è compreso tra 0 e 1; quanto è più alto il valore, tanto è migliore l'algoritmo.

Per vedere come questi valori cambiassero al variare di βhsv e βecr, è bastato modificare man mano un singolo parametro di sensitività, mantenendo l'altro al valore di base; in questo modo si è potuto ottenere dei grafici abbastanza rappresentativi di come i parametri di sensitività incidano su *Precisione e Recupero*.

La finestra scelta per variare i parametri di sensibilità è stata presa sulla base dei risultati mostrati in (7), ovvero si è scelto di fare variare le due beta da 0.7 a 1.4 (con intervalli di 0.1) in quanto mostravano risultati più significativi.

5.4 CATEGORIZZAZIONE DEI VIDEO

Dal dataset sono stati, innanzitutto selezionati 9 video (costituenti complessivamente 842 shots, dai dati di Ground Truth di TRECVID) che avessero caratteristiche apparentemente differenti, in modo tale da avere risultati su video dalle proprietà eterogenee.

Sulla base dei grafici di Precision/Recall ottenuti dai test sulla detection, e osservando in maniera più approfondita i video, si è dedotto cosa li accomunasse, ed è stato così possibile suddividerli in 4 macrocategorie basate sulla tendenza primaria del contenuto.

5.4.1 VIDEO OMOGENEI

Questa tipologia di video è quella che più si avvicina ad un normale filmato all'aperto ripreso da una fotocamera digitale. In questa categoria sono rientrati infatti i documentari sulla natura e riprese di spazi aperti; a prescindere da questo l'elemento fondamentale che caratterizza i video di questa categoria è l'alternarsi omogeneo di colori, luci e ombre, senza parti più buie o meno colorate: la tonalità e la luce dell'immagine rimane abbastanza analoga per tutta la durata del video.

Ad esempio, come si può vedere nell'immagine sottostante, non ci sono colori predominanti e l'immagine è mediamente luminosa.

Figura 5-4 Esempio di immagini caratteristiche per tutta la durata del video

(i) Video 1

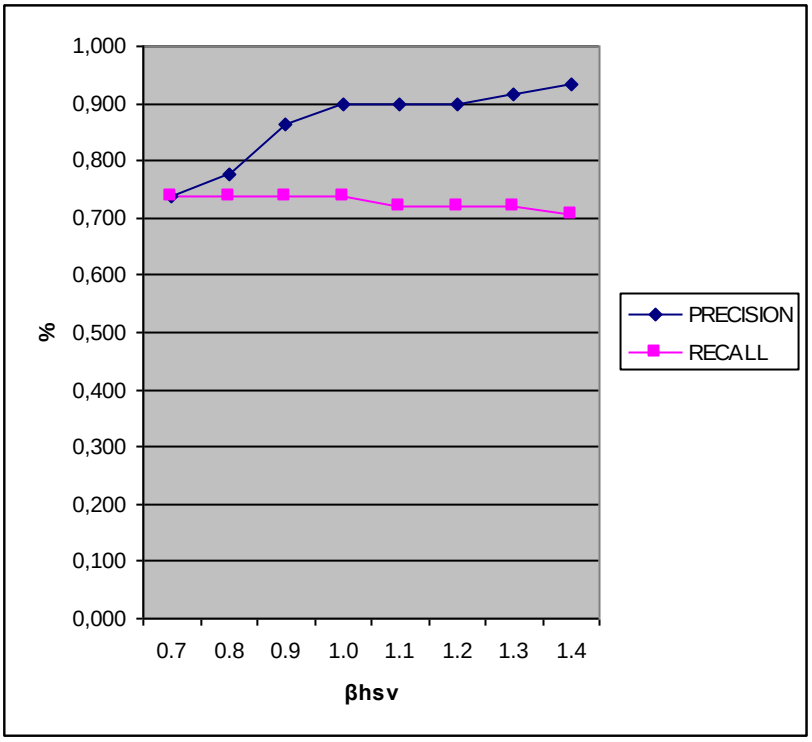


Figura 5-5 Precision e Recall al variare del solo βhsv (video 1)

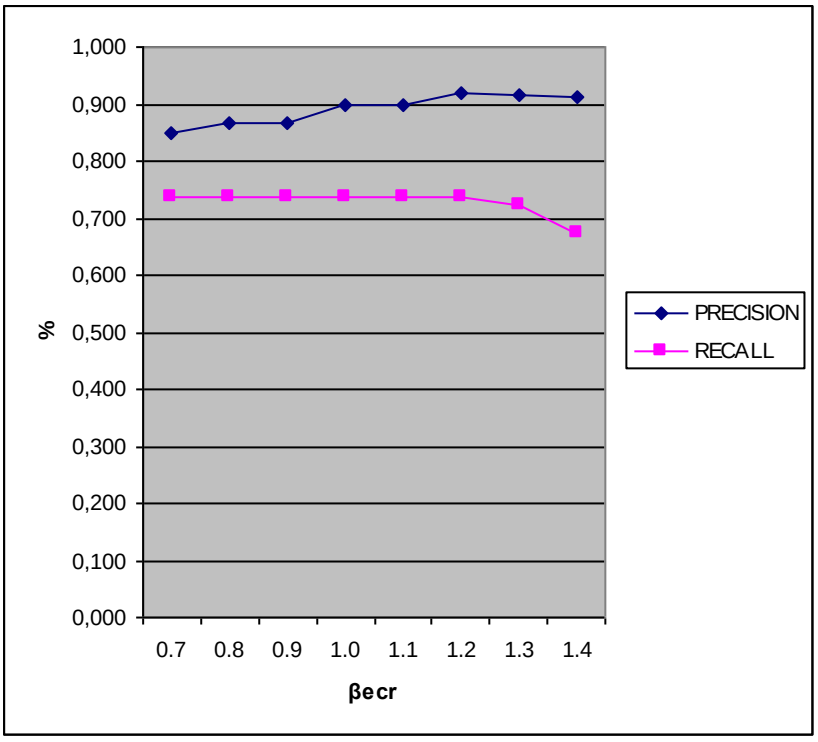


Figura 5-6 Precision e Recall al variare del solo βecr (video 1)

Ed ecco come variano CC, FC ed MC.

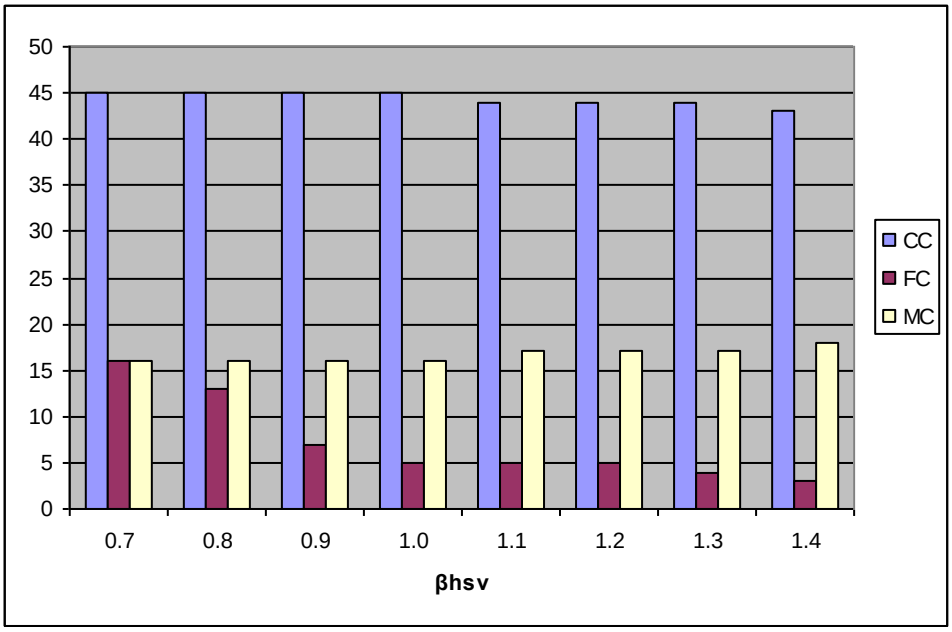


Figura 5-7 CC, FC e MC al variare del solo βhsv

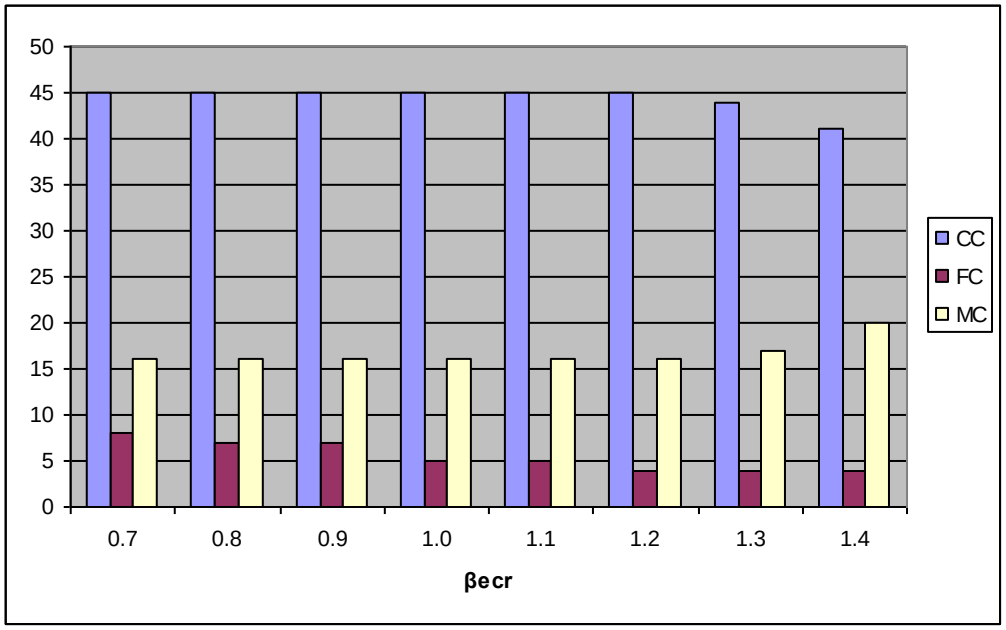


Figura 5-8 CC, FC e MC al variare del solo βecr

Saranno mostrati i grafici solo per questo primo video per dare un'idea di come questi valori cambino al crescere o al decrescere dei parametri di sensitività.

Al crescere della sensitività i cut rilevati correttamente tendono a diminuire, così come quelli erronei, mentre i cut non rilevati, viceversa aumentano.

(ii) Video 2

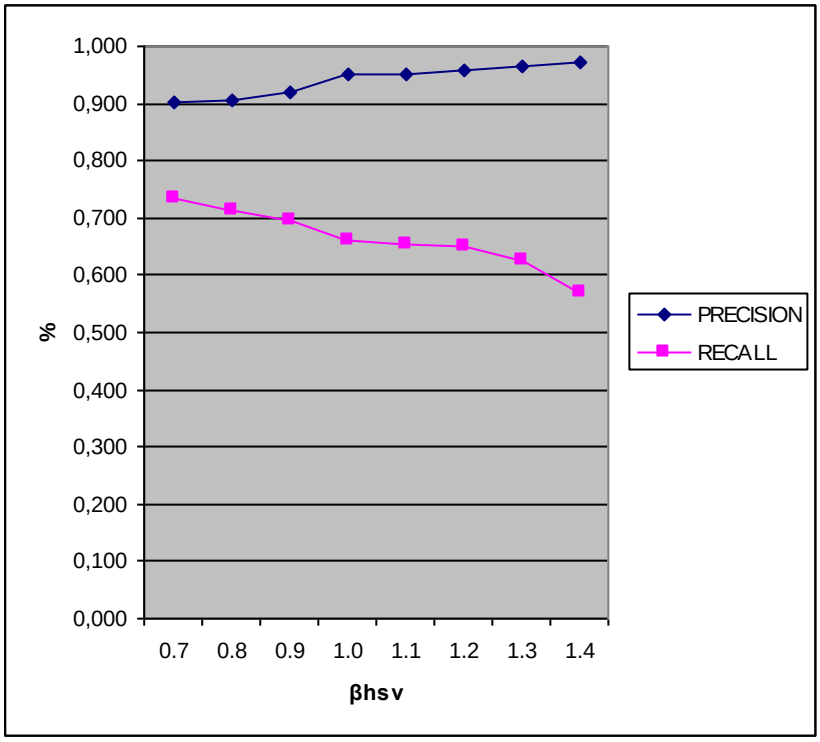


Figura 5-9 Precision e Recall al variare del solo βhsv (video 2)

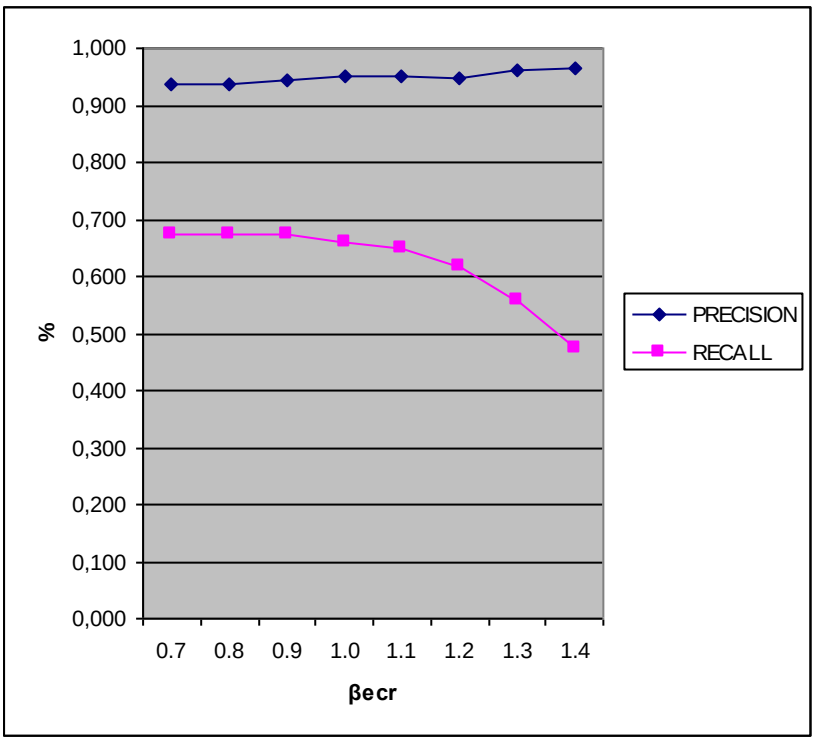


Figura 5-10 Precision e Recall al variare del solo βecr (video 2)

(iii) Video 3

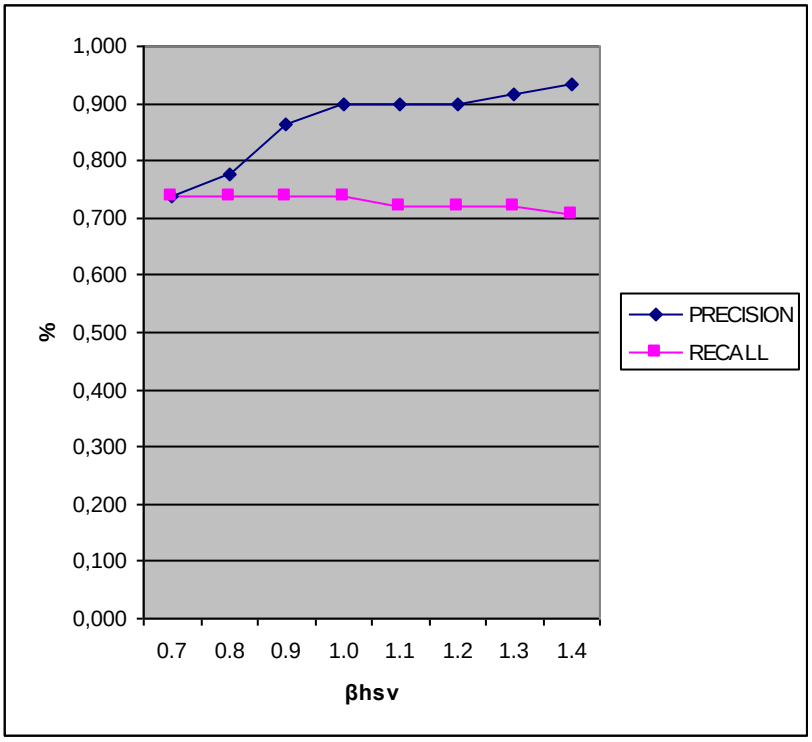


Figura 5-11 Precision e Recall al variare del solo βhsv (video 3)

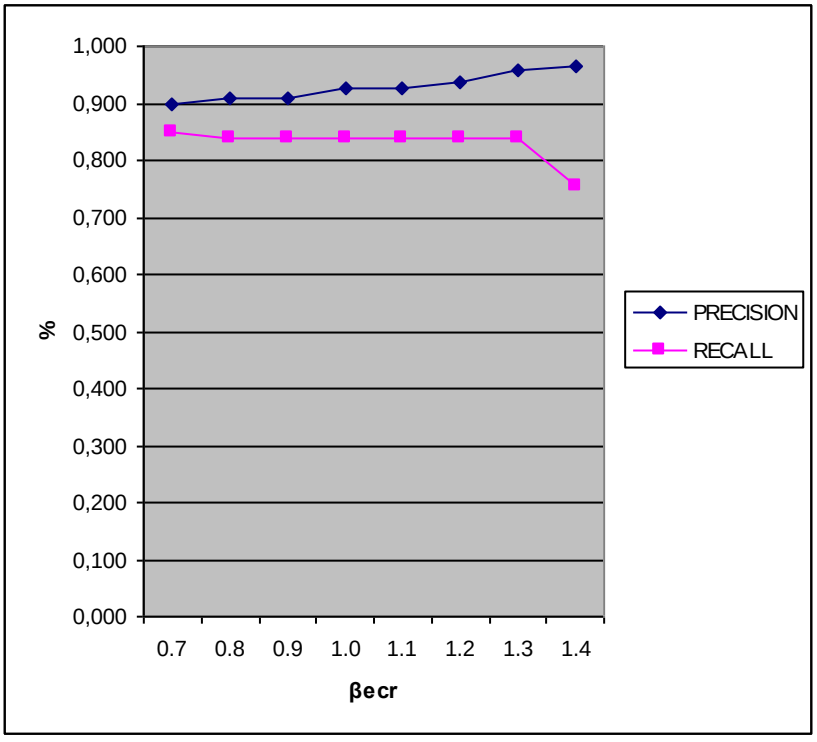


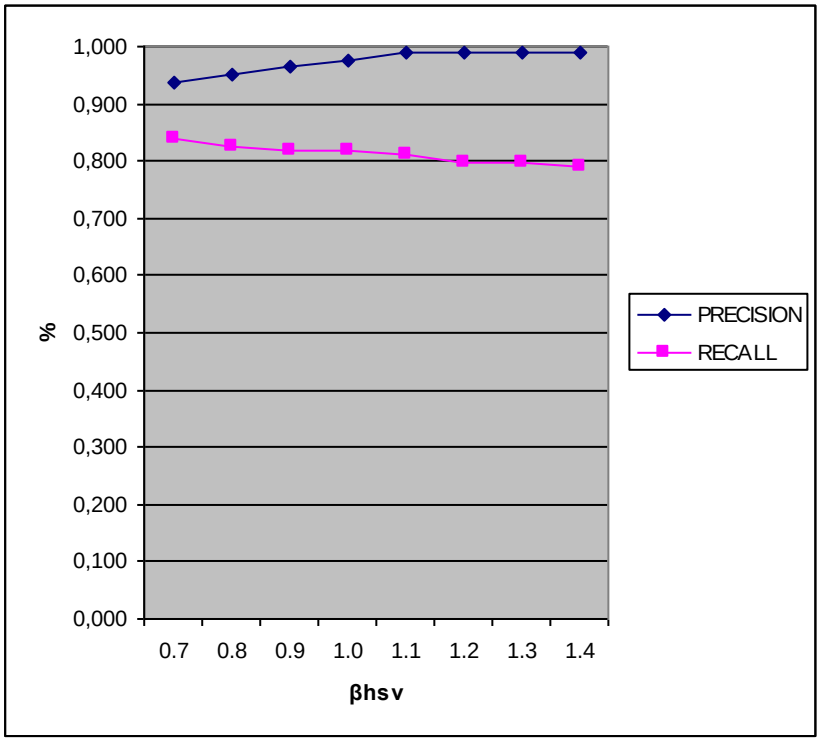
Figura 5-12 Precision e Recall al variare del solo βecr (video 3)

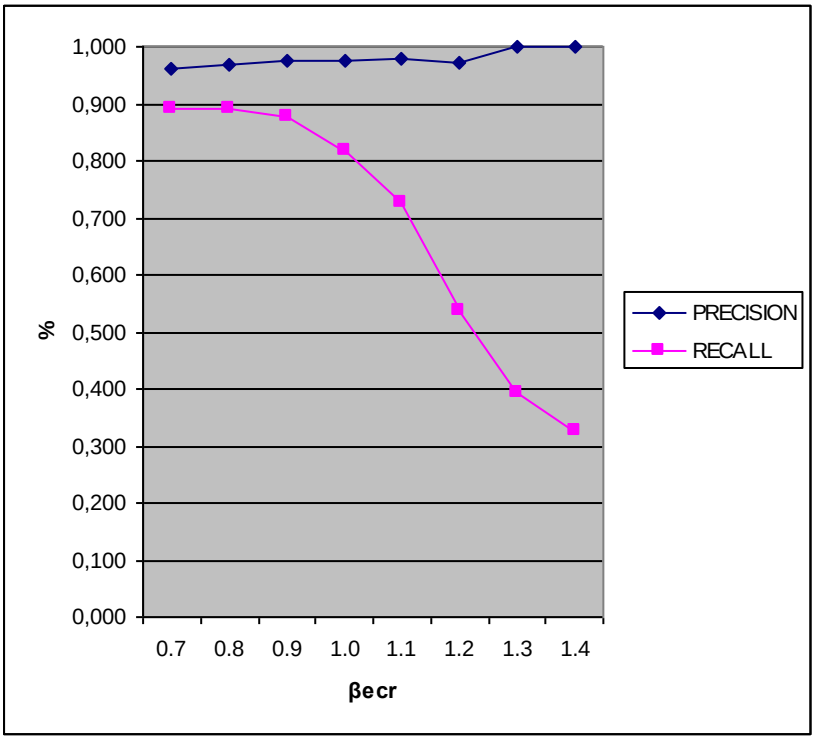
5.4.2 VIDEO BUI

In questa tipologia di video possiamo inserire tutti quei video che hanno un tema di fondo molto scuro. Sfondi neri, ombre forti e poche luci hanno un ruolo dominante per tutto il tempo.

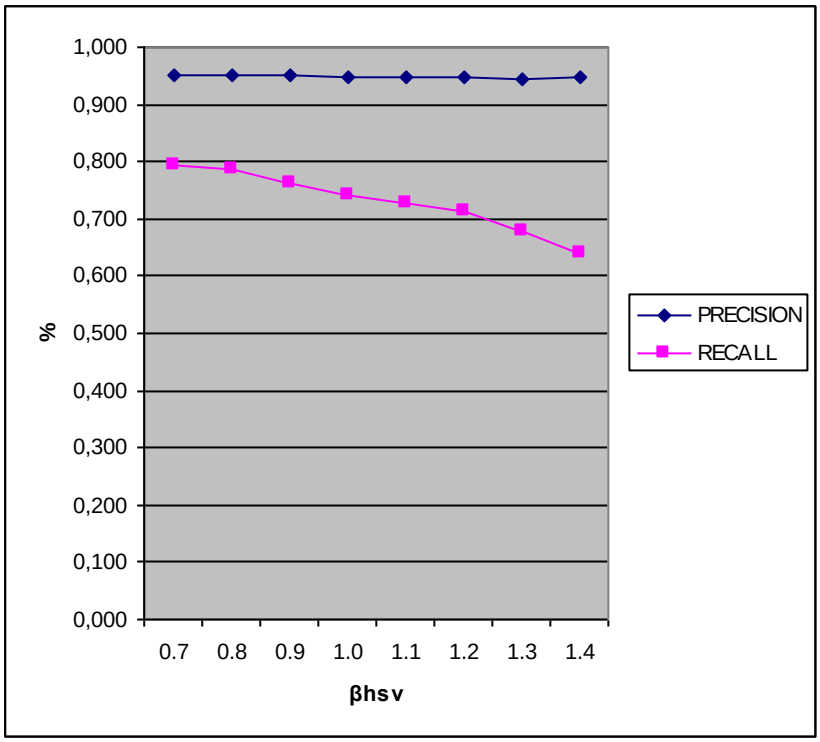
Figura 5-13 Esempio di toni principali dei video bui

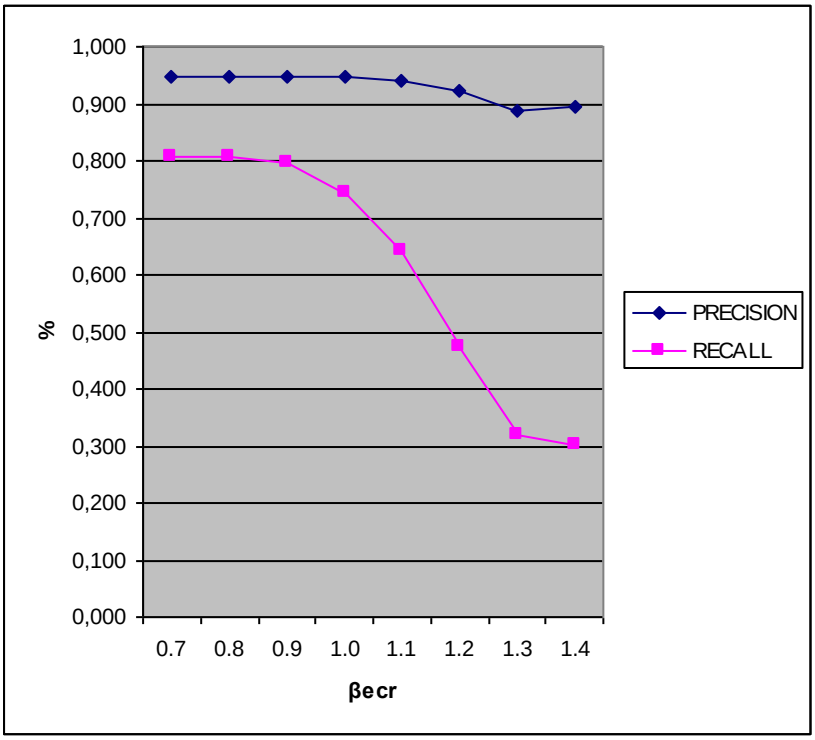
(i) Video 1




1,000
0,900
0,800
0,700
0,600
0,500
0,400
0,300
0,200
0,100
0,000
%
0.7 0.8 0.9 1.0 1.1 1.2 1.3 1.4
βecr
PRECISION
RECALL

(ii) Video 2


1,000
0,900
0,800
0,700
0,600
0,500
0,400
0,300
0,200
0,100
0,000
%
0.7 0.8 0.9 1.0 1.1 1.2 1.3 1.4
βhsv
PRECISION
RECALL



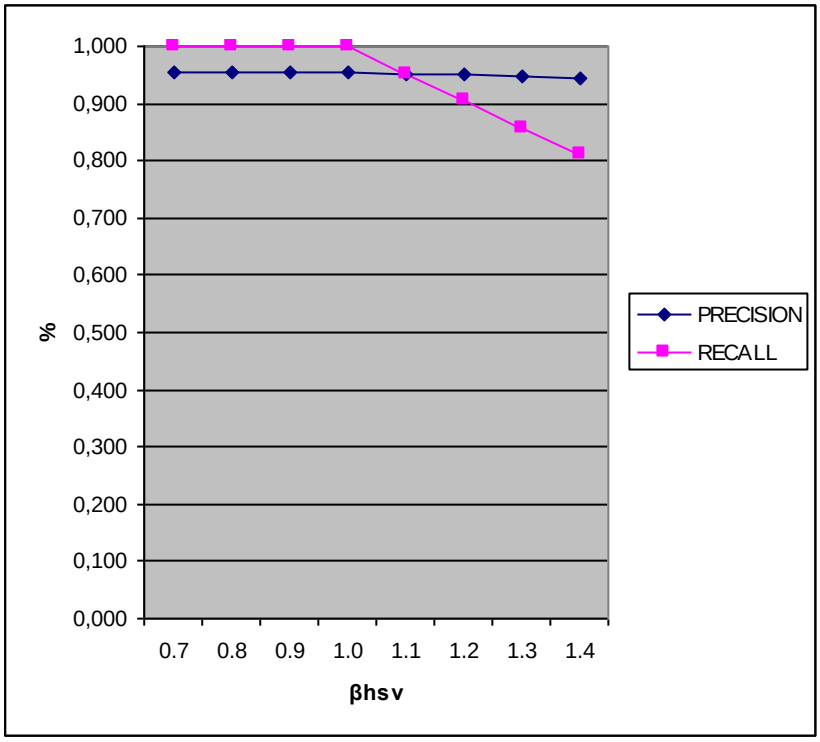
Dai grafici di questo tipo di video, si può facilmente vedere come alti valori di βecr influiscano pesantemente sulla Recall. Nel secondo grafico (vedi sopra) è opportuno far notare l'anomalo calo del valore di precision, in quanto al diminuire dei tagli correttamente rilevati non è conseguita anche una diminuzione di quelli erronei.

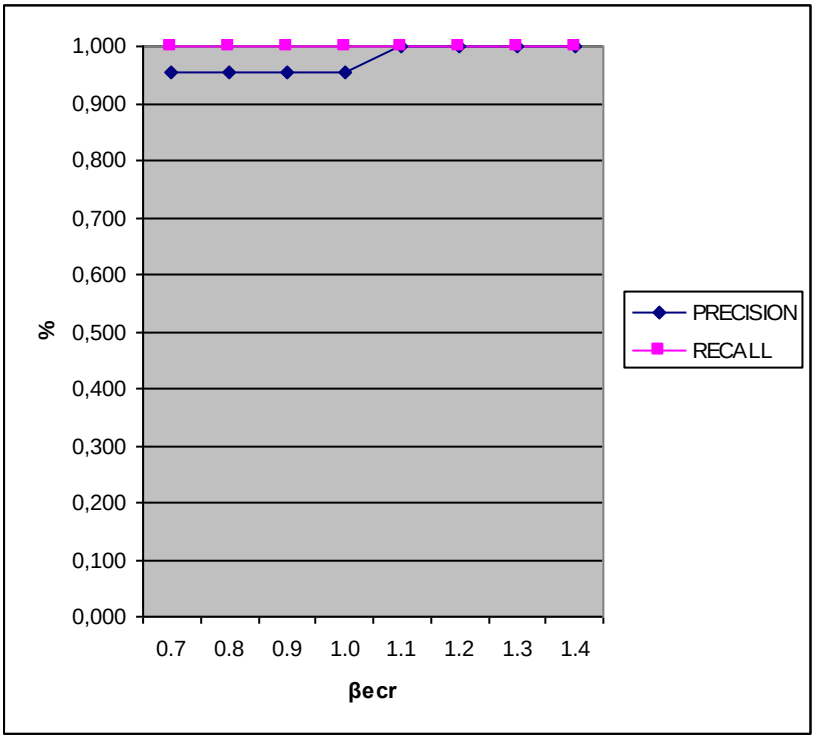
5.4.3 VIDEO IN BIANCO E NERO

Sono tutti quei video che dall'inizio alla fine non presentano colorazione; per intenderci, un classico esempio sono tutti quei video prodotti nella prima metà del XIX secolo.

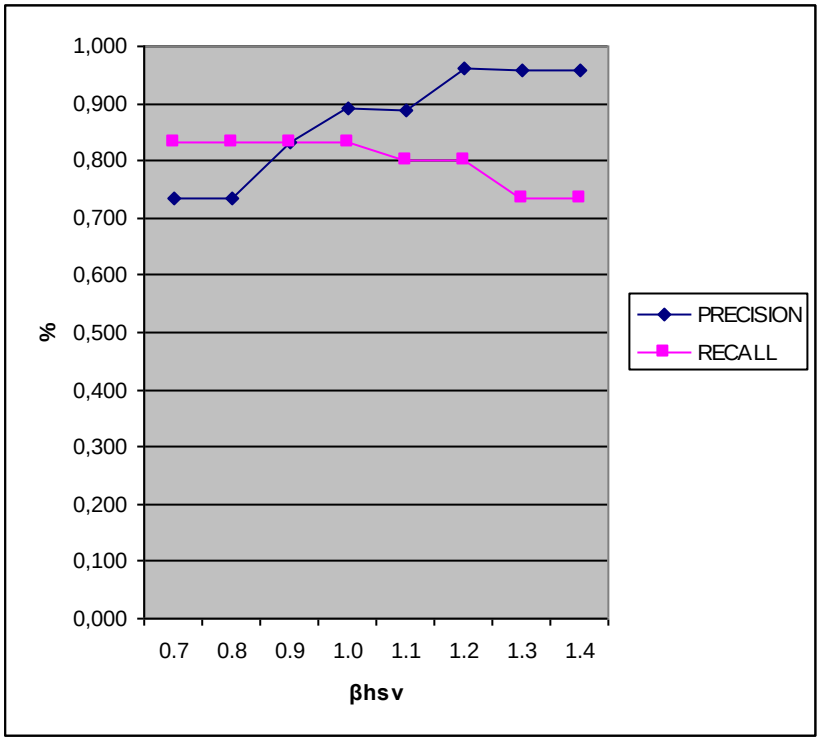
Figura 5-14 Esempio B/N

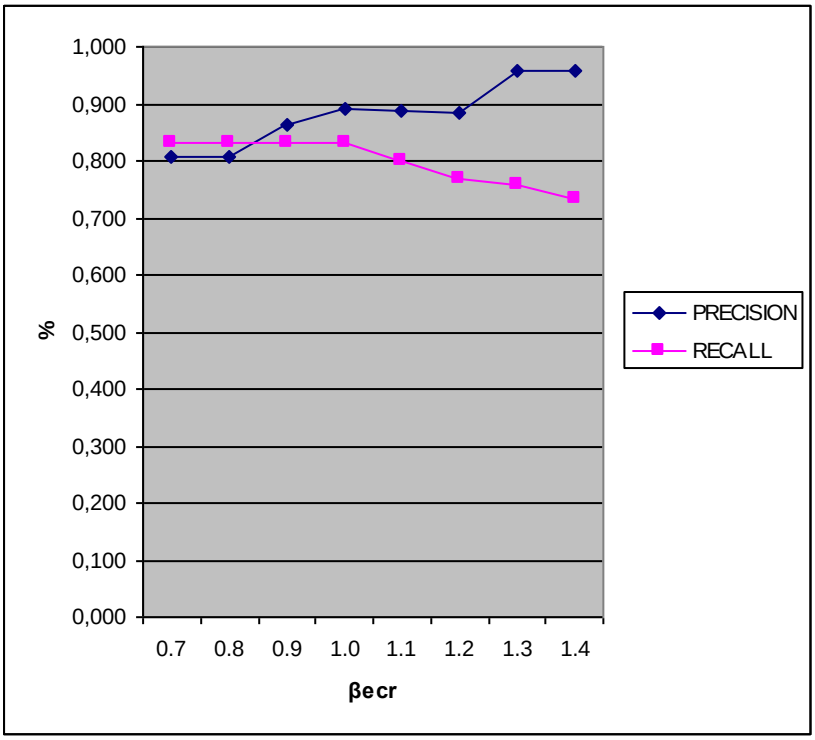
(i) Video 1


1,000
0,900
0,800
0,700
0,600
0,500
0,400
0,300
0,200
0,100
0,000
%
0.7 0.8 0.9 1.0 1.1 1.2 1.3 1.4
βhsv
PRECISION
RECALL


1,000
0,900
0,800
0,700
0,600
0,500
0,400
0,300
0,200
0,100
0,000
%
0.7 0.8 0.9 1.0 1.1 1.2 1.3 1.4
βecr
PRECISION
RECALL

(ii) Video 2





È opportuno fare notare che il primo video sia particolarmente breve e le transizioni, a differenza del secondo in cui sono presenti vari effetti graduali, sono tutte piuttosto nette.

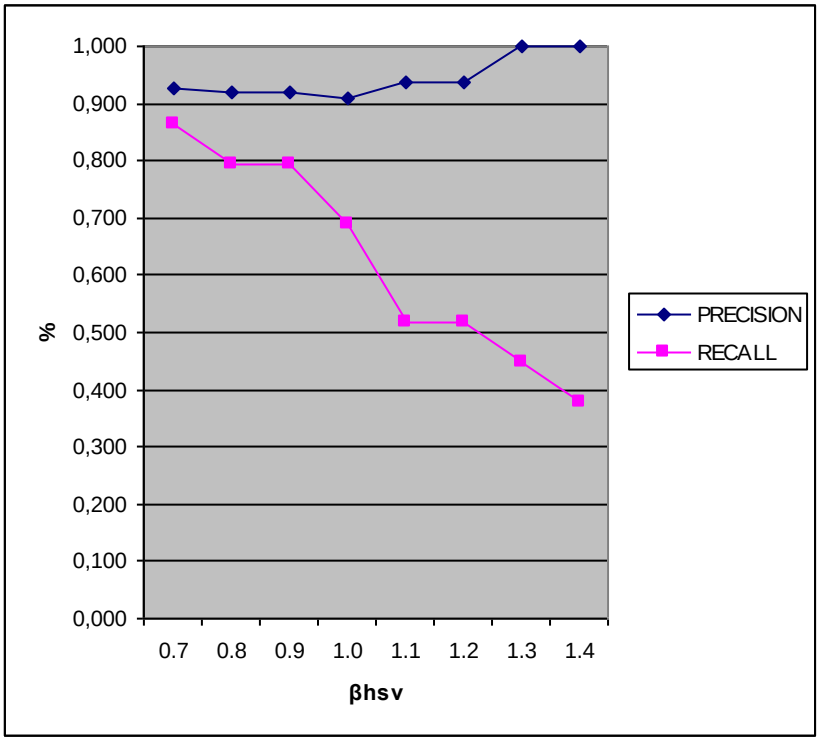
5.4.4 VIDEO FORTEMENTE ETEROGENEI

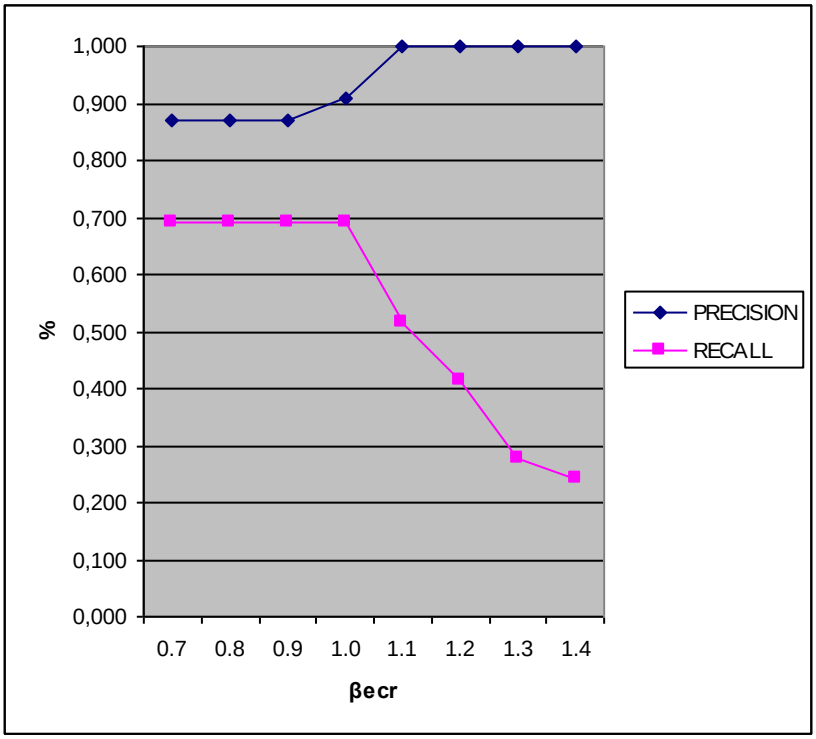
La definizione di questa tipologia nasce dal fatto che i video di questa categoria presentano forti differenze nelle varie parti. Per spiegare meglio, ad esempio, il primo video di cui saranno mostrati i grafici del testing, presenta nella prima metà un contenuto in bianco e nero e i soggetti sono tutti molto statici, mentre nella seconda metà le immagini cambiano completamente e troviamo movimenti veloci e tanti colori. Anche nel secondo video, se non con tutta questa demarcazione, troviamo sia tante parti in bianco e nero che a colori.

Figura 5-15 contenuto iniziale del primo video (staticità e assenza di colori)

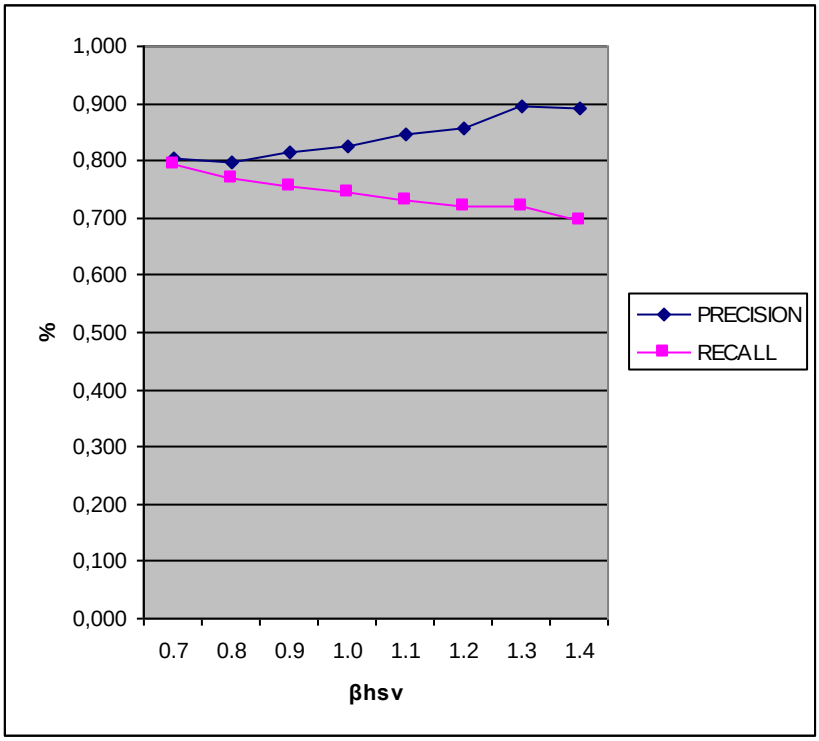
Figura 5-16 contenuto finale del primo video (forte dinamicità e colorazione)

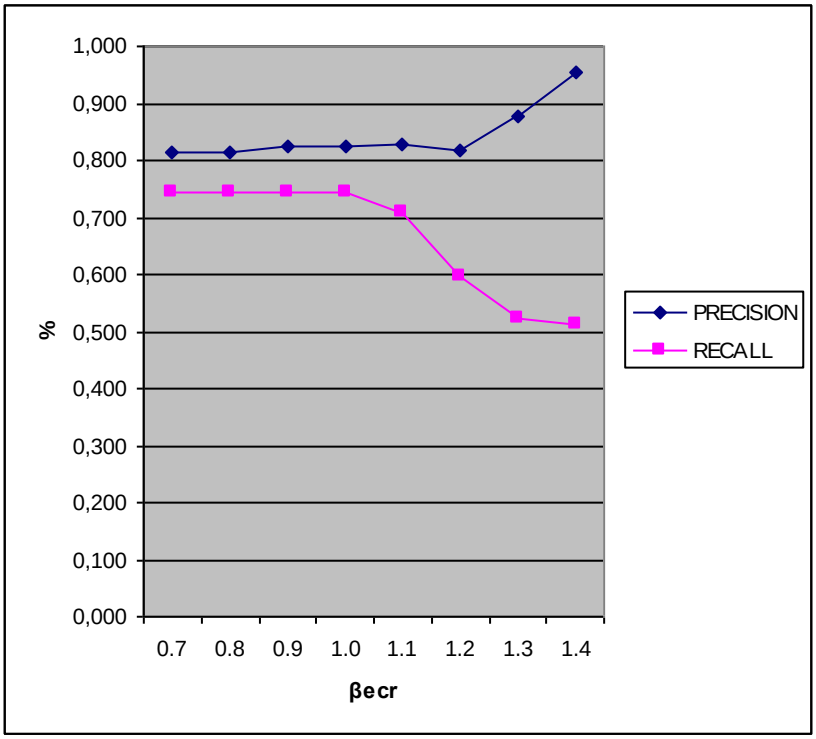
(i) Video 1


1,000
0,900
0,800
0,700
0,600
0,500
0,400
0,300
0,200
0,100
0,000
%
0.7 0.8 0.9 1.0 1.1 1.2 1.3 1.4
βhsv
PRECISION
RECALL


1,000
0,900
0,800
0,700
0,600
0,500
0,400
0,300
0,200
0,100
0,000
%
0.7 0.8 0.9 1.0 1.1 1.2 1.3 1.4
βecr
PRECISION
RECALL

(ii) Video 2


1,000
0,900
0,800
0,700
0,600
0,500
0,400
0,300
0,200
0,100
0,000
%
0.7 0.8 0.9 1.0 1.1 1.2 1.3 1.4
βhsv
PRECISION
RECALL


1,000
0,900
0,800
0,700
0,600
0,500
0,400
0,300
0,200
0,100
0,000
%
0.7 0.8 0.9 1.0 1.1 1.2 1.3 1.4
βecr
PRECISION
RECALL

5.5 TUNING DEI PARAMETRI

Sulla base dei risultati ottenuti, si è provveduto a trovare un settaggio bilanciato che funzioni nel miglior modo per ciascuna categoria.

In questo paragrafo saranno descritti e mostrati visivamente i risultati ottenuti utilizzando i migliori tuning, per ciascuna categoria proposta.

Alla fine del paragrafo sarà inserito un riepilogo in forma tabellare dei dati ottenuti dal testing.

5.5.1 VIDEO OMOGENEI

Per i "**Video omogenei**", dai grafici si può notare come aumentando il parametro β_{HSV} fino a 1.2, ci sia un buon aumento in termini di Precision, a discapito di una non sostanziale diminuzione del parametro di Recall (a differenza di quanto avviene andando ad aumentare l'altro parametro di sensitività); pertanto il settaggio più bilanciato viene ottenuto mantenendo il parametro β_{ECR} al valore di default, cioè 1.0, e andando ad aumentare l'altro (β_{HSV}) fino a 1.2.[14] Questo setting permette di raggiungere un valore medio di Precision di 95.4% e di Recall di 72.1%. Si potrebbero ottenere facilmente valori di Recall più alti, semplicemente diminuendo β_{HSV} anche sotto lo 0.7, ma questo guadagno non giustificherebbe la notevole perdita di Precision che si avrebbe.

[14] *È interessante notare come questo sia lo stesso miglior settaggio medio ottenuto in (7).*

Figura 5-17 Risultato di una segmentazione su un "video omogeneo" ottenuta mediante l'utilizzo dei migliori parametri di sensitività (β_{HSV} =1.2, β_{ECR} =1.0)

5.5.2 VIDEO BUI

Per i video considerati come "**Video bui**", la cosa migliore da fare è sembrata quella di mantenere entrambi i parametri di sensitività il più basso possibile (nella finestra considerata), di conseguenza il tuning migliore ottenuto è lavorando con $\beta_{HSV} = 0.7$ e $\beta_{ECR} = 0.8$.

Occorre fare presente che per testare la veridicità di questo, dal momento che era necessario modificare entrambi i parametri contemporaneamente, si è reso necessario fare un ulteriore test sui video per ottenere e visualizzare i risultati di questo tuning. Con questa configurazione si arriva ad ottenere un valore medio di Precision del 93.7% e di Recall dell'88.7%.

Figura 5-18 Risultato di una segmentazione su un "video buio" ottenuta mediante l'utilizzo dei migliori parametri di sensitività (β_{HSV} =0.7, β_{ECR} =0.8)

5.5.3 VIDEO IN BIANCO E NERO

I video in "**Bianco e Nero**" sembrano dare i migliori risultati con gli stessi parametri di default, ovvero $\beta_{HSV} = 1$ e $\beta_{ECR} = 1$. Questo "tweaking" consente di ottenere mediamente una Precision del 92.0% e una Recall del 90.2% (occorre fare presente che il campione di video, relativo a questa sezione, è dato da due soli video di lunghezza piuttosto inferiore rispetto agli altri video considerati per i vari test, quindi il risultato del tweaking migliore potrebbe non essere attendibile quanto gli altri settaggi presentati)

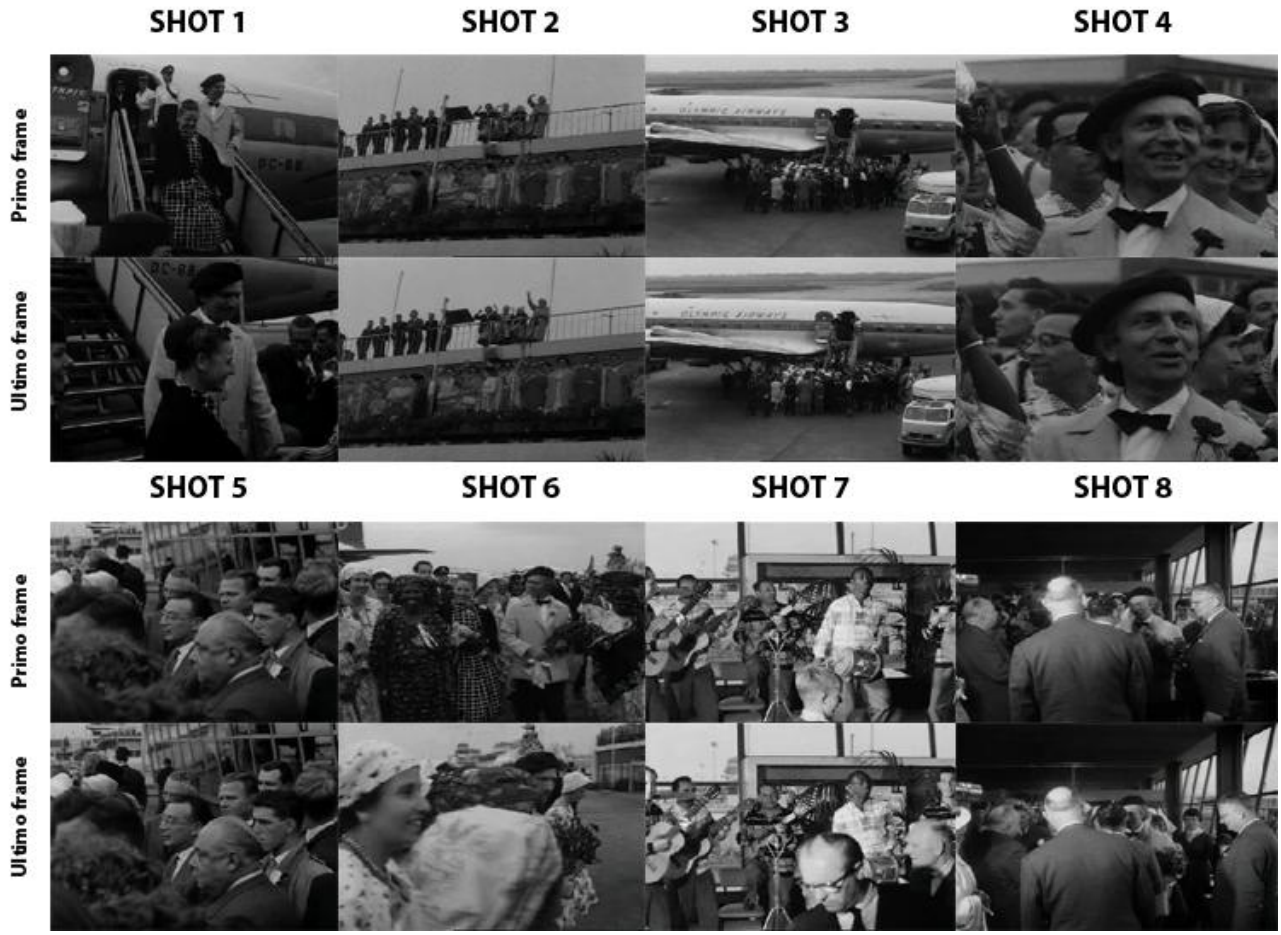


Figura 5-19 Risultato di una segmentazione su un "video in bianco e nero" ottenuta mediante l'utilizzo dei migliori parametri di sensitività (β_{HSV} =1.0, β_{ECR} =1.0)

5.5.4 VIDEO FORTEMENTE ETEROGENEI

Per quanto riguarda la categoria dei "**Video fortemente eterogenei**" dai test è risultato raccomandabile mantenere $\beta_{ECR} = 1$ e abbassare il parametro β_{HSV} fino a 0.7. Questo tuning porta al raggiungimento di un risultato medio di Precision di 83.3% e di Recall del 81.1%.

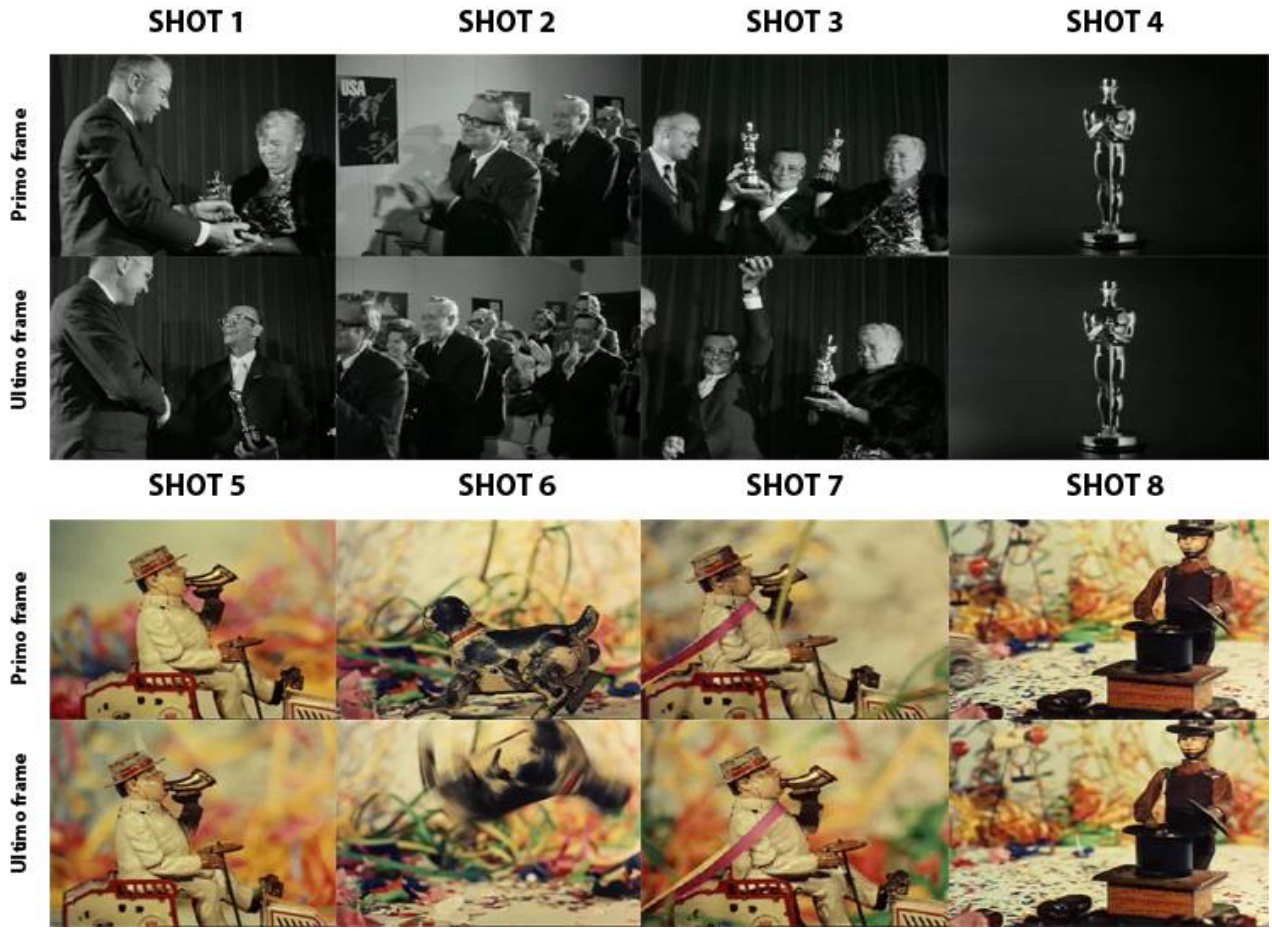


Figura 5-20 Risultato di una segmentazione su un "video fortemente eterogeneo" ottenuta mediante l'utilizzo dei migliori parametri di sensitività (β_{HSV} =0.7, β_{ECR} =1.0)

5.5.5 RIEPILOGO

La seguente tabella illustra i risultati finali del testing complessivo e i risultati ottenuti, mediamente, sui video mediante l'utilizzo dei migliori parametri di tuning.

Categoria	Best βhsv	Best βecr	Total CC	Total FC	Total MC	AVG Precision	AVG Recall
Omogenei	1.2	1.0	248	12	96	95,4%	72,1%
Bui	0.7	0.8	298	20	38	93,7%	88,7%
B/N	1.0	1.0	46	4	5	92,0%	90,2%
Eterogenei	0.7	1.0	90	18	21	83,3%	81,1%
TOTALE			682	54	160	92,7%	81,0%

Figura 5-21Tabella riepilogativa dei dati del testing

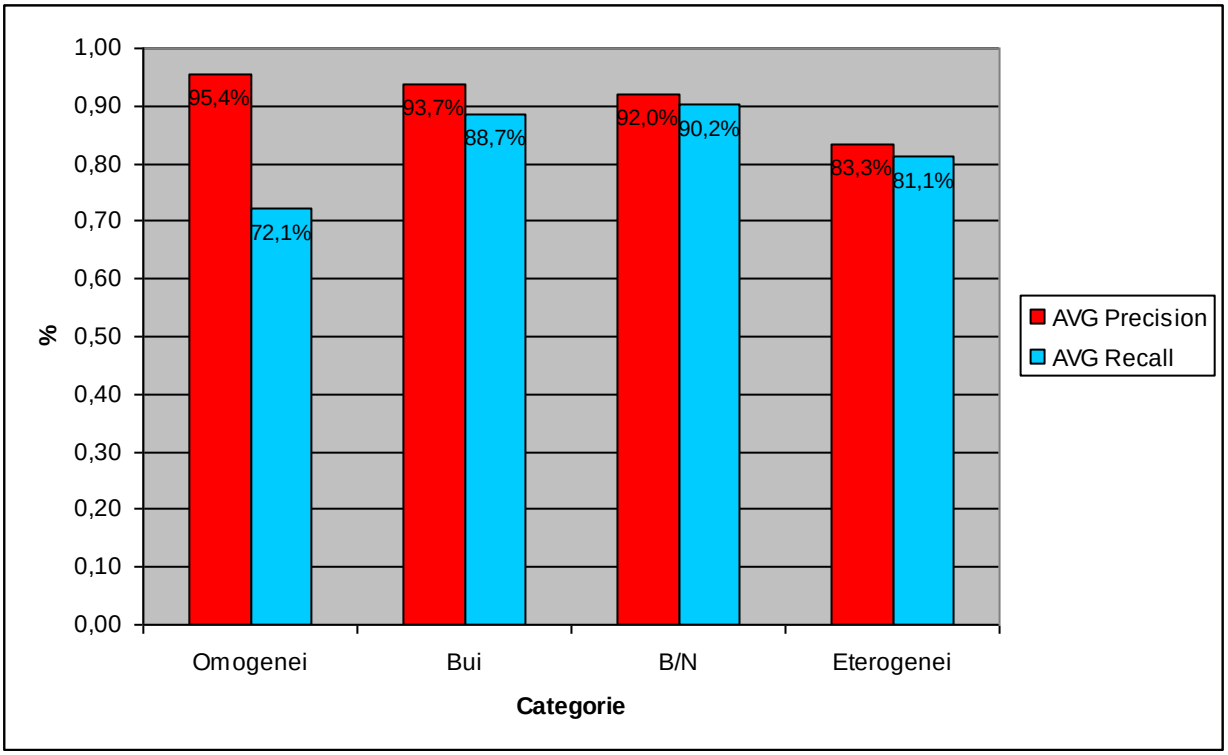


Figura 5-22 Media dei risultati qualitativi di Precision/Recall, suddivisi per categoria, con l'utilizzo dei migliori parametri di tuning

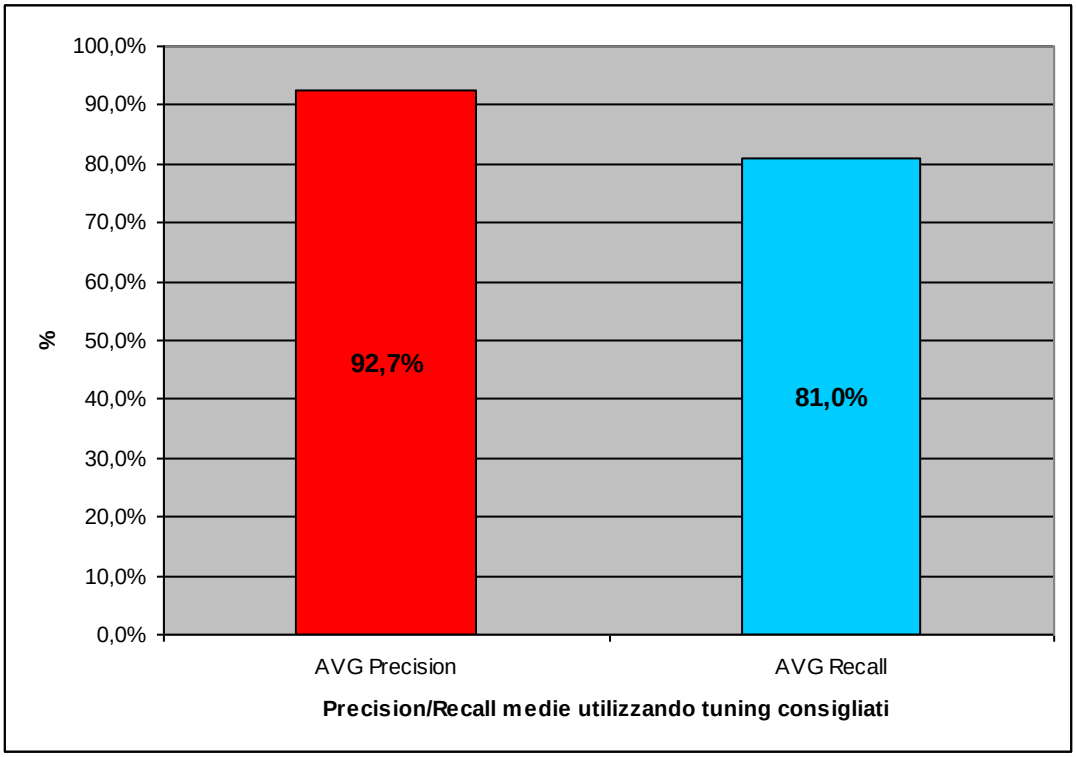


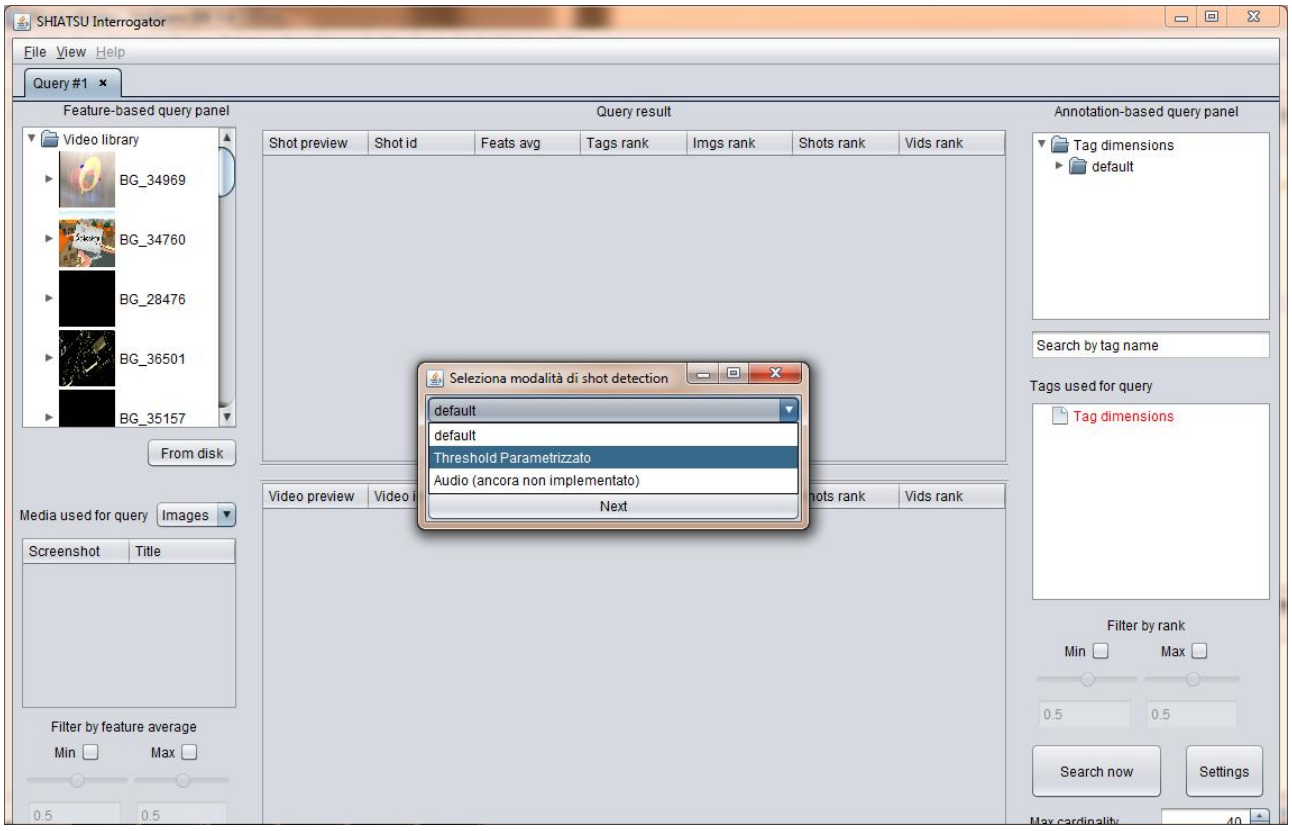
Figura 5-23 Risultati medi raggiungibili utilizzando sempre i parametri di tuning consigliati dal test

5.6 DEMO A SNAPSHOT

In questo paragrafo verrà illustrata visivamente, tramite l'utilizzo degli snapshot dell'interfaccia grafica, una sorta di demo di SHIATSU, per mostrare il procedimento di importazione di un video, da disco rigido, nel database, dando quindi prova di come possa, un possibile utilizzatore del software, configurare i parametri per il tuning della "Threshold Detection".

Per lanciare la fase di importazione di un video da disco, occorre cliccare sull'apposito bottone "From disk" nell'interfaccia principale; lo si trova nella parte sinistra della finestra, appena sotto la Video library.

Al click del bottone si aprirà una finestra che richiederà la modalità di shot detection con cui si vogliano reperire i tagli. Le opzioni proposte sono al momento quella di default (che lancia il Threshold Detector impostato ai valori di default, indicati in (7)) e l'impostazione manuale dei parametri del Threshold Detector. Nell'elenco è proposta anche la modalità Audio, ma questa non è ancora stata implementata definitivamente nel framework SHIATSU.



Una volta selezionato "*Threshold Parametrizzato*" e cliccato su "*Next*", all'utente verrà proposta una nuova finestra per la selezione dei parametri, dopo aver selezionato il video da importare da disco.

Verrà infatti dapprima proposta la classica "open dialog" per la selezione del video da importare

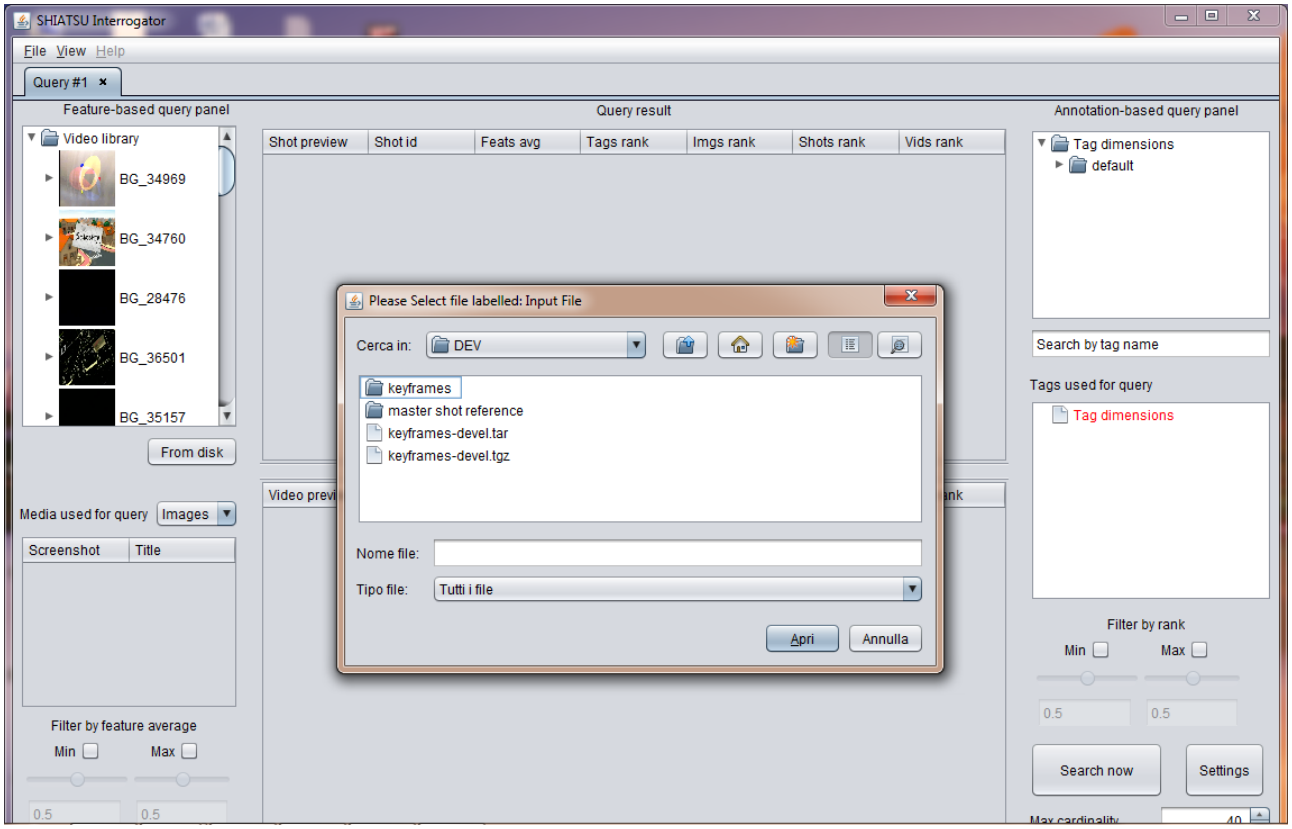


Figura 5-24 Selezione del file da disco rigido

Come si può vedere dalla foto sottostante, l'utente sarà in grado di selezionare la categoria in cui ritiene che il video rientri.

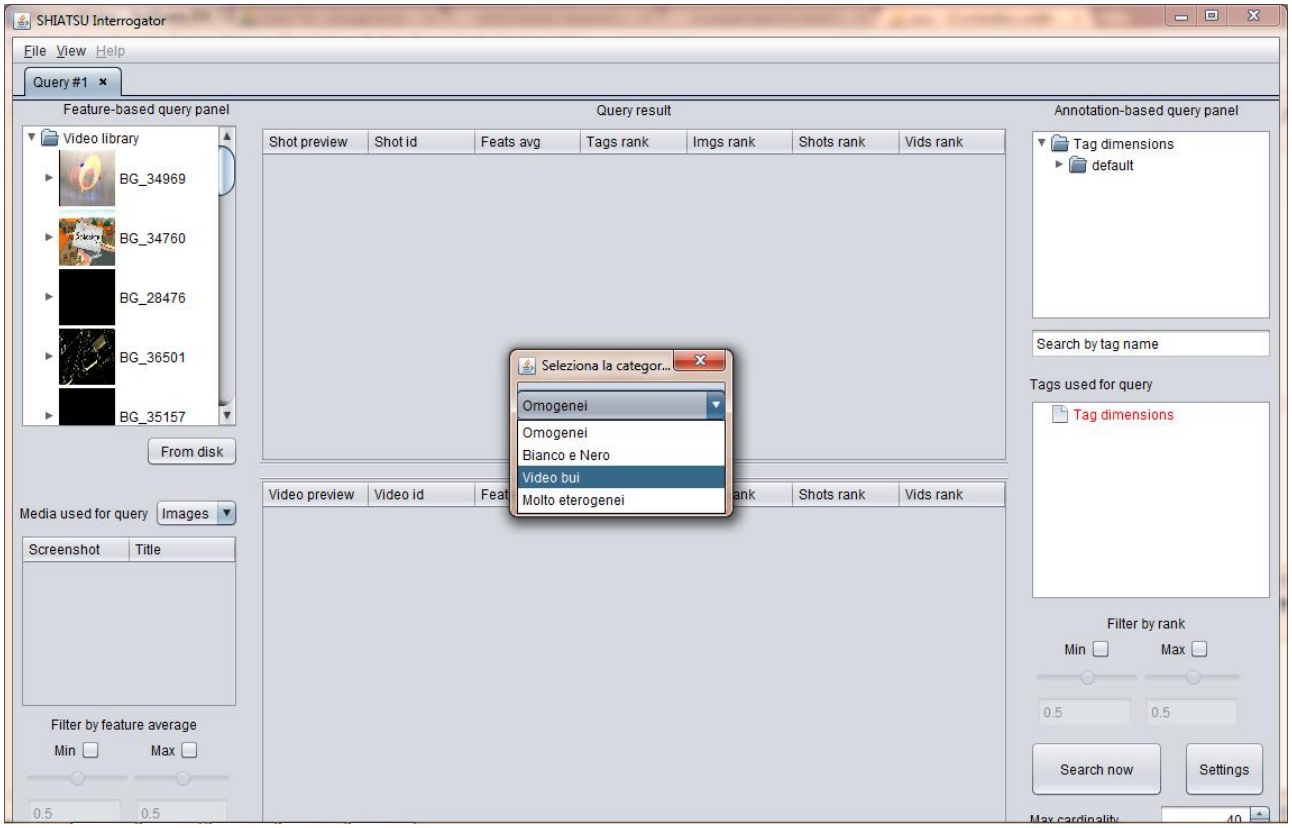


Figura 5-25 Interfaccia di selezione della categoria e di impostazione manuale dei parametri di sensitività

Nel momento in cui questa sarà stata cliccata, verranno mostrati, subito sotto la combobox di selezione della categoria, i parametri di sensitività per la categoria prescelta, su cui l'utente potrà comunque agire, modificandoli a proprio piacimento, o scegliendo un'altra categoria di video per la Threshold detection.

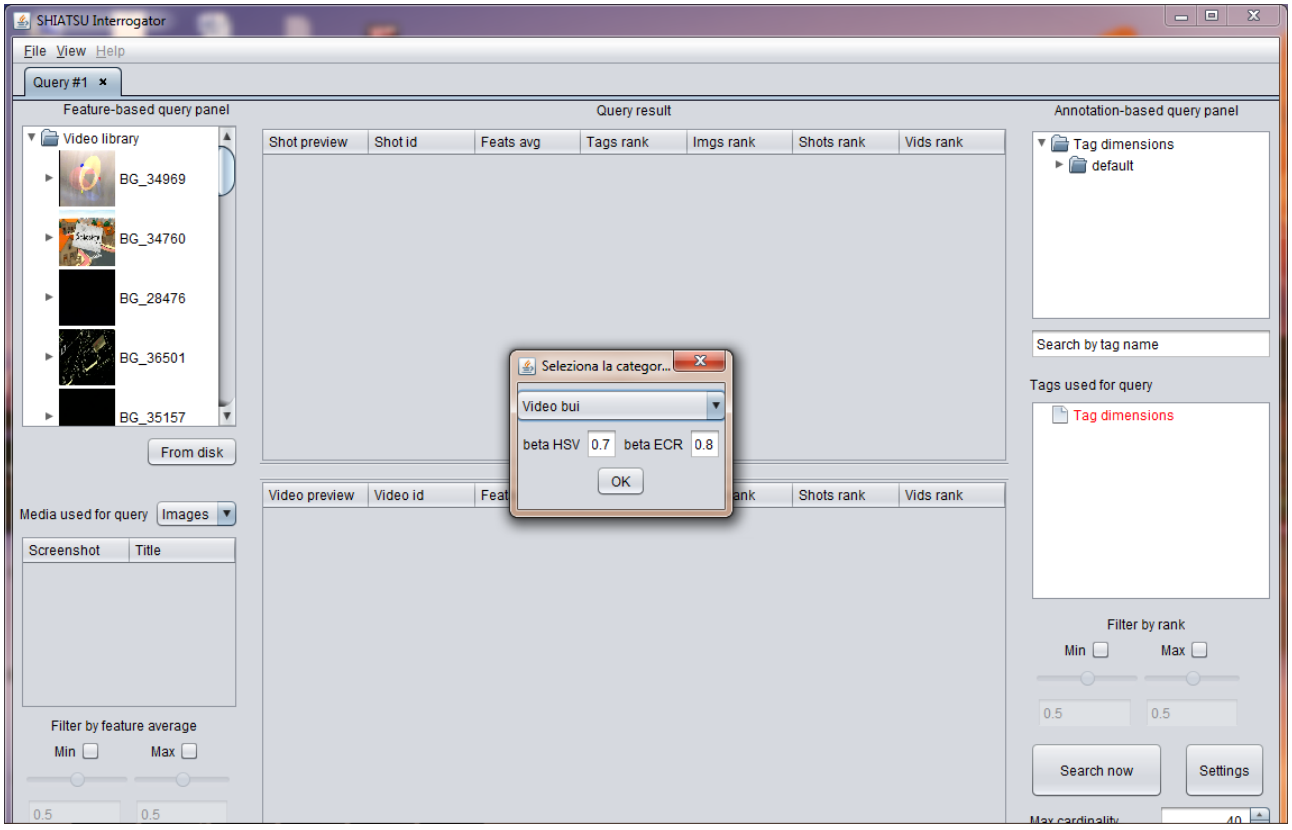
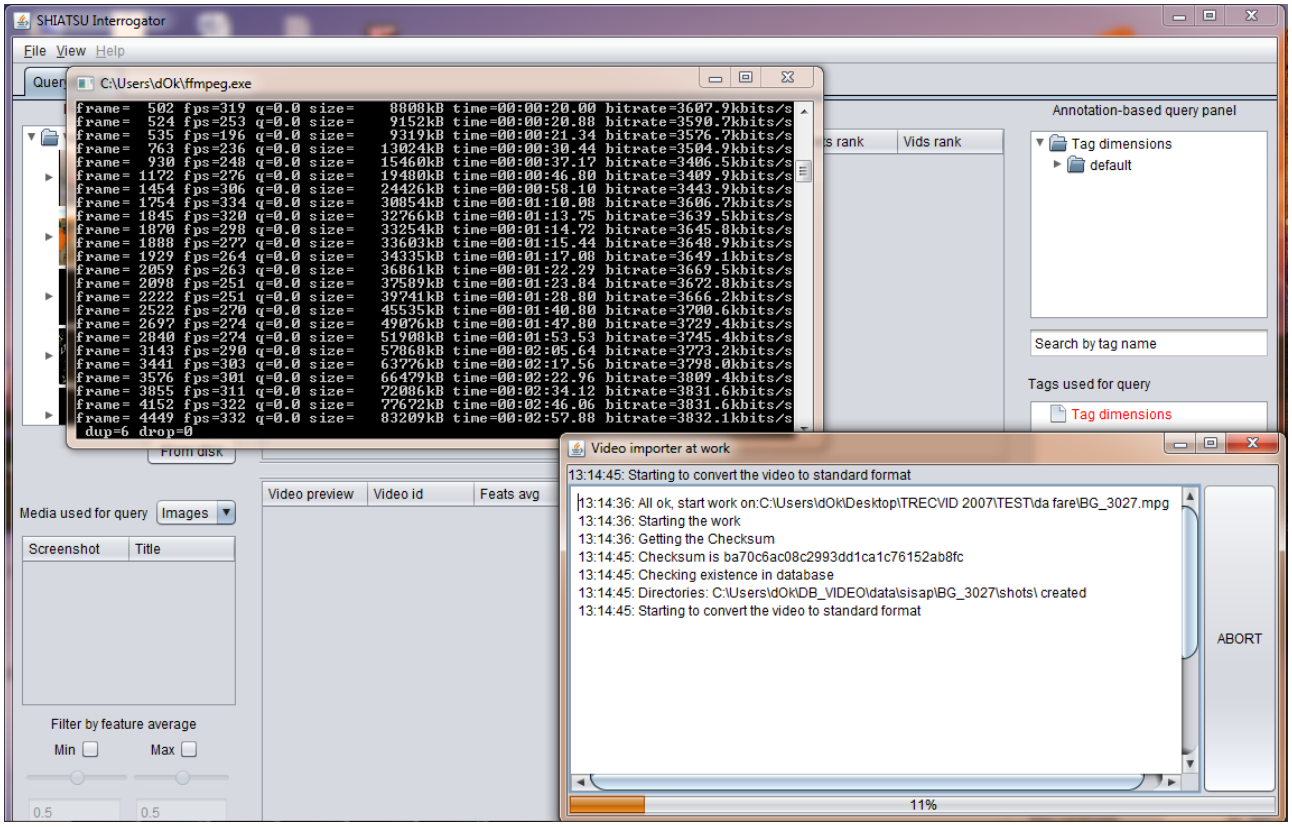


Figura 5-26 Selezionando la categoria verranno impostati automaticamente i parametri per la stessa, ma per l'utente è sempre possibile modificarli manualmente (nell'esempio dello snapshot sono stati generati automaticamente i parametri consigliati per la categoria selezionata, "Video bui")

Cliccando ancora una volta su ok, partirà quindi effettivamente il processo di import vero e proprio, al termine del quale, se tutto sarà andato come da aspettative, ci ritroveremo il nostro video pronto per essere utilizzato all'interno di SHIATSU, suddiviso in shots sulla base del tipo di segmentatore e dei parametri definiti nei passi precedenti.



Capitolo 6. CONCLUSIONI

I risultati sperimentali ottenuti sul parametro di Recall si sono rivelati leggermente al di sotto delle aspettative; questo forse è dovuto al fatto che per il testing si è preferito utilizzare i file di "shot reference" più completi, forniti da TRECVID come Ground Truth, quindi contenenti pressoché la totalità degli shots dei video del dataset, la cui rilevazione è stata probabilmente ottenuta attraverso un'annotazione manuale. In ogni caso, il testing prova le ottime capacità del detector nella rilevazione dei cut, ottenendo livelli di precisione molto alti per tutti i video esaminati.

Per quanto riguarda le finalità di estensione del framework, per l'introduzione di nuove classi di detection, si può dire che l'obiettivo di ingegnerizzazione è stato pienamente raggiunto e future implementazioni di nuovi tipi di detector potranno essere alquanto agevoli e veloci.

Relativamente agli sviluppi futuri, un attuale limite di SHIATSU si è rivelato proprio l'utilizzo di JMF per l'elaborazione dei video, che obbliga la conversione dei video da inserire in file `.mov` (necessari per avere la sicurezza che JMF non abbia problemi a manipolarli) di enormi dimensioni, come gli shots prodotti dalla segmentazione di questi; questo implica un notevole costo in termini di spazio, per la persistenza fisica del database. Una soluzione potrebbe essere la sostituzione delle librerie di JMF, ormai obsolete da quasi un decennio, con il suo corrispettivo opensource **FMJ**[15]; questo progetto si pone come valida alternativa a JMF, essendo API-Compatible con lo stesso. Per questo, teoricamente dovrebbe essere possibile eseguire il codice JMF attualmente implementato utilizzando FMJ, ma non è rara la necessità di trovare delle soluzioni nei casi in cui il codice non sia pienamente compatibile per le differenze tra i due framework. Un ulteriore incentivo all'utilizzo di questa alternativa è il sottoprogetto FFMPEG-Java, che non è niente di più che un java wrapper per FFMPEG; attualmente è necessaria la presenza fisica su disco dell'eseguibile "`ffmpeg.exe`" (vedi.

[15] http://fmj-sf.net/

5.1.1) per poter importare i nuovi video, di cui individuare i tagli, quindi estrarre gli shot in database.

Bibliografia

1. *Windsurf: Region-Based Image Retrieval Using Wavelets.* **Ardizzoni Stefania, Bartolini Ilaria e Patella Marco.** 1999. Proceedings of the 1st Workshop on Similarity Search (IWOSS 1999 - DEXA 1999). pp. 167-173.

2. *The Windsurf Library for the Efficient Retrieval of Multimedia Hierarchical Data.* **Bartolini Ilaria, Patella Marco e Stromei Guido.** 2011. Proceeding of the Proceedings of the International Conference on Signal Processing and Multimedia Applications (SIGMAP 2011).

3. *Query processing issues in region-based image databases.* **Bartolini Ilaria, Ciaccia Paolo e Patella Marco.** 2009

4. *M-tree: An efficient Access Method for Similarity Search in Metric Spaces.* **Ciaccia Paolo, Patella Marco e Zezula Pavel.** 1997. Proceedings of the 23rd VLDB Conference. pp. 426-435

5. *SHIATSU: Annotating Your Videos the Easy Way!* **Bartolini Ilaria e Romani Corrado.** 2010. Proceedings of the 3rd International Conference on Similarity Search and Applications (SISAP 2010). pp. 119-120

6. *SHIATSU: Semantic-Based Hierachical Automatic Tagging of Videos by Segmentation using Cuts.* **Bartolini Ilaria, Patella Marco e Romani Corrado.** 2010. Proceedings of the 3rd International ACM MM Workshop on Automated Information Extraction in Media Production (AIEMPro10). pp. 57-62

7. *SHIATSU: Tagging and Retrieving Videos without Worries.* **Bartolini Ilaria, Patella Marco e Romani Corrado.** 2011. Multimedia Tools and Applications Journal (MTAP).

8. *A formal study of shot boundary detection.* **Yuan J., Wang H., Xiao L., Zheng W., Li J. and Zhang B..** Feb 2007. IEEE Trans. on Circuits and Systems for Video Technology, 17 (2). pp. 168-186

9. *A Metric for Distributions with Applications to Image Databases.* **Rubner Yossi, Tomasi Carlo e Guibas Leonidas.** 1998. Proceedings of the 1998 IEEE International Conference on Computer Vision. pp. 59-66

10. *Code for the Earth Movers Distance (EMD).* **Rubner Yossi.** http://ai.stanford.edu/~*rubner/emd/default.htm*. [Online]

11. *Progettazione e Realizzazione di un benchmark di test per il sistema SHIATSU.* **Marconi Fabrizio.** 2013.