

Università degli Studi di Catania

Dipartimento di Matematica e Informatica

Corso di Laurea in Informatica | LM-18

Multimedia e Laboratorio

Analisi dell'algoritmo di classificazione di frame dell'encoder MPEG-4

Studente: Luigi Pacino
Matricola: 1000015484

9 gennaio 2026

Indice

Introduzione	2
Analisi dell'algoritmo	3
Riproduzione dell'algoritmo	4
Risultati	8
Conclusioni	10
Bibliografia	11

Introduzione

L'MPEG-4 è uno standard di compressione audio-video sviluppato dal gruppo MPEG (Moving Picture Experts Group) con l'obiettivo di offrire alta qualità multimediale a bitrate ridotti. Introdotto alla fine degli anni '90, l'MPEG-4 rappresenta un'evoluzione degli standard precedenti, integrando tecniche più avanzate per la codifica di video, audio e contenuti interattivi.

Ciò che lo rende particolarmente versatile è la capacità di adattarsi a numerosi scenari: dallo streaming su Internet ai dispositivi mobili, dalla videoconferenza alle applicazioni professionali. Lo standard supporta sia contenuti tradizionali, sia oggetti audiovisivi complessi, permettendo di trattare separatamente elementi come grafica, testo e immagini in movimento. Questa flessibilità, insieme a un'efficiente compressione, ha reso l'MPEG-4 una delle tecnologie più diffuse nella moderna distribuzione multimediale.

Uno dei punti chiave dell'algoritmo MPEG (qualsiasi sia la versione trattata) è riuscire a definire una GOP (Group Of Pictures) efficace, in grado di garantire un'ottima compressione sia dal punto di vista qualitativo, sia computazionale.

Per GOP s'intende la classificazione dei singoli frame del video rispettivamente in:

- I-frame, Intra Frame o frame chiave, poco compresso;
- P-frame, Predicted Frame, mediamente compresso;
- B-frame, Bidirectional Predicted Frame, molto compresso.

La scelta dei frame chiave è decisiva in quanto, a partire da quest'ultimi, è possibile ricostruire il resto dei fotogrammi. Nelle varie versioni dell'algoritmo MPEG sono state utilizzate, nel corso degli anni, tecniche differenti, passando da una versione statica a una più dinamica, che si adatti al contesto e che soprattutto esegua la classificazione dei frame in maniera più rapida.

L'obiettivo di questo progetto è quello di analizzare le nuove tecniche di classificazione real-time dei frame e ricreare un modello semplificato - ma fedele - dell'algoritmo responsabile di tale processo.

Analisi dell'algoritmo

Dall'analisi del codice dell'encoder x265 e di altre implementazioni dell'algoritmo MPEG, è stato possibile comprendere come viene effettuata la classificazione dei singoli fotogrammi in frame di tipo I, P o B.

Innanzitutto, in tutte le versioni dell'MPEG, a partire dalle prime implementazioni, viene definita una struttura statica per la GOP; l'algoritmo ha bisogno solo di due parametri iniziali: uno che definisca ogni quanto inserire un I-frame e l'altro che definisca il numero massimo di B-frame consecutivi prima di un frame di tipo I o P. Quindi, l'algoritmo assegna il tipo I, P o B staticamente ai fotogrammi in base alla loro posizione, senza controllare effettivamente lo stato del frame. Tuttavia, tale approccio non consente di ottenere risultati soddisfacenti: basti pensare, ad esempio, ai cambi di scena all'interno di un filmato; l'algoritmo, così definito, non è in grado di capire se un frame fa parte della scena precedente o di una nuova scena, dato che viene classificato come intra-frame o inter-frame solo in base alla struttura della GOP e senza fare controlli di alcun tipo tra un fotogramma e l'altro.

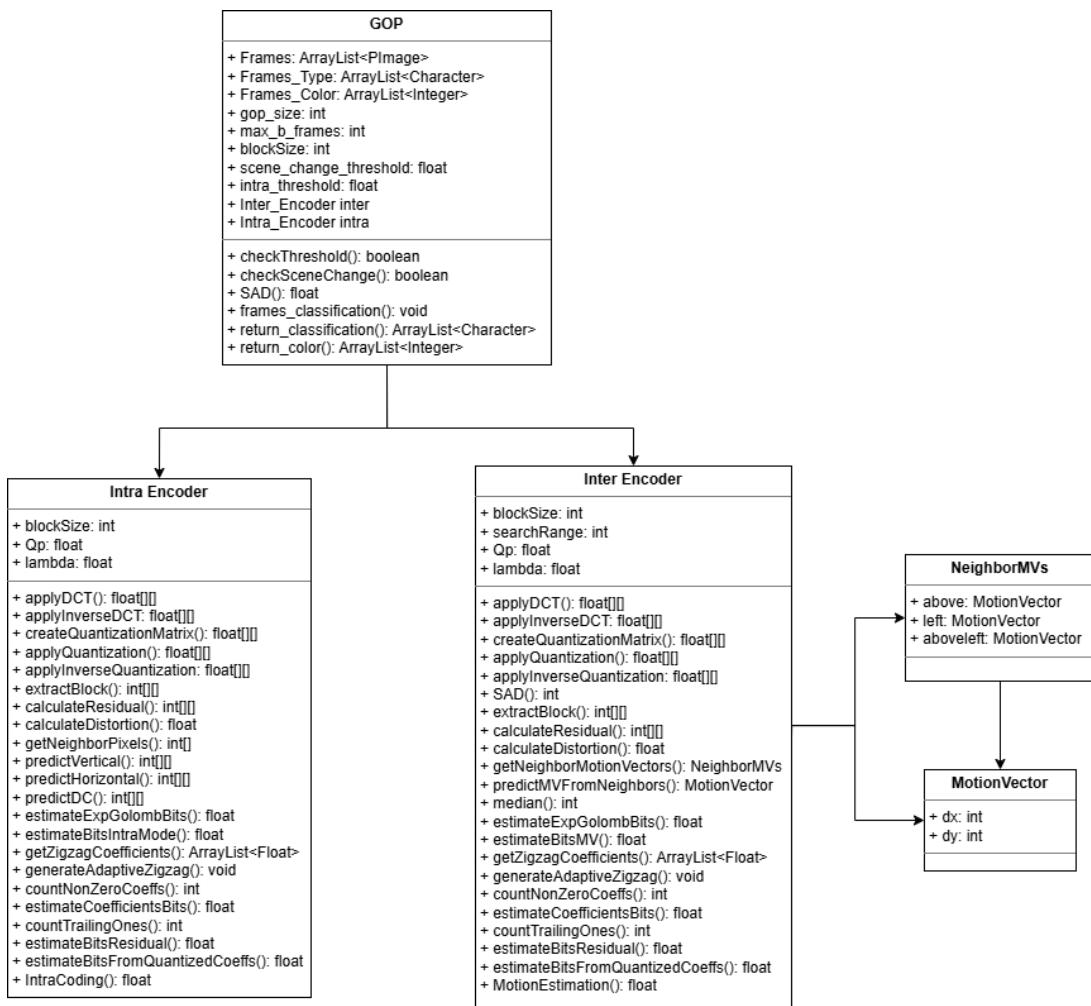
Quindi, risulta necessaria l'implementazione di un controllo che permetta di rilevare se in un determinato frame esiste un cambio di scena, così da poter forzare quel frame a key frame a prescindere dalla struttura iniziale della GOP. Questo è proprio quello che avviene all'interno dell'encoder MPEG: viene passato un ulteriore parametro di soglia che determini un limite sopra il quale viene rilevato il cambio di scena, forzando il fotogramma a I-frame. Nello specifico, l'algoritmo controlla per ogni frame quanto è simile al frame chiave precedente, calcolando una metrica di somiglianza, come ad esempio la SAD o l'SSE; se il risultato ha un valore più alto della soglia definita, allora, il fotogramma viene forzato a I-frame e da lì si ricomincia con la struttura iniziale della GOP.

Un altro aspetto importante per ottenere una buona compressione è la classificazione dei singoli macroblocchi dei fotogrammi in blocchi intra o inter. In breve, su un blocco intra viene eseguita una compressione senza grandi perdite di informazioni (proprio come se si trattasse di un'immagine), mentre per un blocco inter viene eseguita una stima del movimento per cui i valori salvati rappresentano dei veri e propri vettori di movimento, che servono a ricostruire la scena sulla base del frame chiave precedente. In un I-frame i macroblocchi sono tutti intra, mentre in un frame di tipo P o B la maggior parte saranno di tipo inter. Viene eseguito, nuovamente, un lavoro di ottimizzazione, controllando la percentuale di blocchi intra all'interno del fotogramma. Infatti, se questa percentuale è maggiore del 60-70% si sceglie di salvare il frame corrente come frame chiave per evitare errori durante la ricostruzione del movimento: disporre di pochi blocchi inter, significa disporre di pochi vettori di movimento per cui la ricostruzione del video potrebbe risultare inaccurata. Dunque, si inserisce un ulteriore parametro di soglia che rappresenti il limite di intra-blocchi consentiti, oltre il quale il fotogramma viene forzato a I-frame, ricominciando, anche in questo caso, con la struttura iniziale della GOP.

Riproduzione dell'algoritmo

L'algoritmo proposto è stato sviluppato in Java Processing, una libreria open-source di Java - linguaggio di programmazione ad alto livello - che permette di creare contenuti interattivi e, inoltre, di lavorare con immagini, con semplificazioni quali classi aggiuntive, funzioni e operazioni matematiche adatte allo scopo. Nonostante l'algoritmo MPEG sia implementato utilizzando linguaggi di basso livello (come, ad esempio, nel caso dell'x265 che utilizza C), Java Processing risulta essere una libreria adatta allo scopo del progetto, poiché permette di fornire una rappresentazione visiva dei risultati dati dal tipo di classificazione programmato, a discapito della velocità di esecuzione.

Di seguito viene fornito il diagramma UML delle classi implementate nel codice.



Per la realizzazione del progetto sono state sviluppate tre classi principali - GOP, Intra Encoder, Inter Encoder - e due ausiliari per facilitare la previsione dei motion vector.

La classe GOP è la classe primaria che si occupa di classificare i frame dati in input e di salvare i risultati in un ArrayList specifico nonché di differenziare per colori gli I-frame che vengono inseriti di default da quelli inseriti per via delle scene cut o dell'elevata presenza di intra-block. La funzione *frames_classification* è il metodo responsabile della classificazione. Tramite i due contatori, *i_counter* e *b_counter*, e il booleano, *i_frame*, viene gestito l'inserimento degli I-frame e dei B-frame sulla base dei due parametri, *gop_size* e *max_b_frames*, passati al costruttore.

```

if(counter_b < this.max_b_frames){
    Frames_Type.add('B');
    counter_b++;
    f.save("results/frame" + i + "_B.png");
}
else{
    Frames_Type.add('P');
    counter_b = 0;
    f.save("results/frame" + i + "_P.png");
}
if(counter_i >= this.gop_size - 1){
    counter_i = 0;
    counter_b = 0;
    i_frame = true;
}
else
    counter_i++;
Frames_Color.add(color(240));
}

```

Come si può notare dalle porzioni di codice sopraelencate, quando il *b_counter* supera il valore di *max_b_frames*, viene inserito un P-frame e il contatore viene riazzzerato, altrimenti si inserisce un B-frame. Se il valore dell'*i_counter* è maggiore della *gop_size*, vengono resettati entrambi i contatori e *i_frame* viene settato a *TRUE*, in modo tale da poter inserire un I-frame nella prossima esecuzione.

All'interno di questa funzione, vengono invocati altri due metodi:

- *checkSceneChange*, utilizzato per rilevare i cambi di scena;
- *checkThreshold*, impiegato per rilevare i frame con un'elevata presenza di intra-block.

La prima funzione si limita a calcolare la SAD (Sum of Absolute Difference) fra il fotogramma in esame e l'ultimo frame chiave; in seguito, il frame viene forzato a I-frame, qualora il risultato della SAD risulti maggiore rispetto alla soglia definita dal parametro *scene_change_threshold*, fornito in input al costruttore della classe.

```

boolean checkSceneChange(PImage frame1, PImage frame2){
    if(SAD(frame1, frame2) > this.scene_change_threshold)
        return true;
    return false;
}

```

Il metodo *checkThreshold*, invece, sfrutta le due classi Encoder, trattate precedentemente. Facendo un passo indietro, il costruttore della classe GOP prende in input nove parametri:

- l'array di frame;
- *gop_size*, che indica ogni quanti frame inserire un frame chiave;
- *max_b_frames*, che indica il numero massimo di B-frames consecutivi;
- *scene_change_threshold*, che indica la soglia sopra la quale viene rilevato un cambio di scena;

- *intra_threshold*, ovvero la percentuale massima di blocchi intra ammissibili prima di forzare il fotogramma a key frame;
- *blockSize*, che rappresenta le dimensioni dei singoli blocchi NxN dell'immagine;
- *searchRange*, ovvero il range di ricerca di blocchi (necessario per la motion estimation)
- *Qp* (Quantization Parameter), che rappresenta il parametro di quantizzazione;
- *lambda*, ovvero il parametro lambda necessario per il calcolo dei costi.

```
GOP(ArrayList<PIImage> f, int g, int b, int bs, int sR, float Qp, float lmb, float st, float it){
    this.Frames = f;
    this.gop_size = g;
    this.max_b_frames = b;
    this.blockSize = bs;
    this.scene_change_threshold = st;
    this.intra_threshold = it;
    Frames_Type = new ArrayList<Character>();
    Frames_Color = new ArrayList<Integer>();
    inter = new Inter_Encoder(bs, sR, Qp, lmb);
    intra = new Intra_Encoder(bs, Qp, lmb);
}
```

Come si può notare dall'immagine riportata, il costruttore utilizza gli ultimi quattro parametri citati per instanziare due oggetti, inter e intra, responsabili della codifica dei blocchi. Quest'ultimi vengono utilizzati dalla funzione *checkThreshold* per calcolare i rispettivi costi, per ogni blocco NxN. Infatti, vengono richiamati i metodi *MotionEstimation* e *IntraCoding*, rispettivamente della classe *Inter_Encoder* e *Intra_Encoder*, tramite i quali vengono calcolati i due costi da confrontare: il costo minimo determina il tipo di codifica da utilizzare. Per ogni codifica intra scelta viene incrementato un contatore tramite il quale è possibile calcolare la percentuale di intra-block presenti nel singolo frame in modo tale da poter verificare se supera i valori di soglia indicati da *intra_threshold*.

```
boolean checkThreshold(PIImage frame, PImage key_frame){
    float j_intra, j_inter;
    int intraCounter = 0;
    int totalBlocks = (frame.width * frame.height) / (blockSize * blockSize);

    for(int i = 0; i < frame.width; i+=blockSize){
        for(int j = 0; j < frame.height; j+=blockSize){
            j_intra = intra.IntraCoding(i, j, frame);
            j_inter = inter.MotionEstimation(i, j, frame, key_frame);
            if(j_intra < j_inter)
                intraCounter++;
        }
    }

    float intra_percentage = (float(intraCounter) / float(totalBlocks)) * 100.0;

    if(intra_percentage >= intra_threshold)
        return true;
    return false;
}
```

Dal codice del costruttore, si può notare, inoltre, che vengono creati due *ArrayList*: *Frame_Type* e *Frame_Color*.

Il primo salva i dati della classificazione, il secondo, invece, è stato implementato per una visualizzazione più chiara dei risultati.

L'algoritmo differenzia i frame chiave dal resto dei frame colorando tutti i P o B frame di bianco, mentre tutti gli I-frame vengono colorati nel seguente modo:

- azzurro, per gli I-frame inseriti seguendo lo schema della GOP;
- giallo, per i cambi di scena;
- rosso, per i frame con una percentuale alta di intra-block.

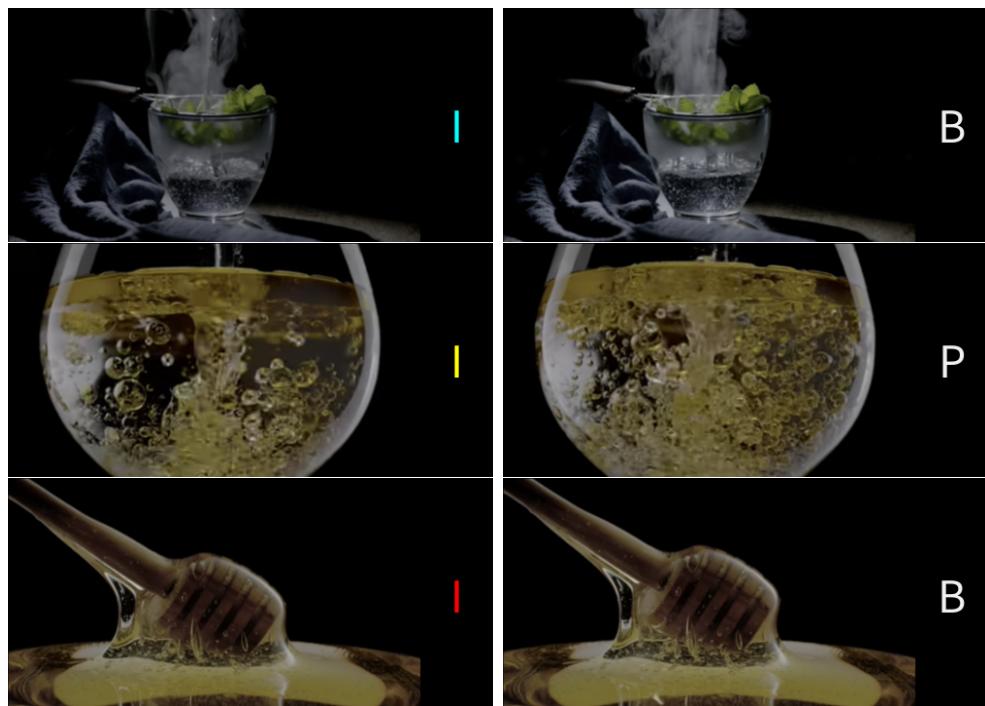
```

if(i_frame == true){
    i_frame = false;
    counter_i++;
    last_key_frame = f;
    Frames_Type.add('I');
    Frames_Color.add(color(0,255,255));
    f.save("results/frame" + i + "_I_b.png");
    continue;
}
if(checkSceneChange(f, last_key_frame) == true){
    i_frame = false;
    counter_i = 1;
    counter_b = 0;
    last_key_frame = f;
    Frames_Type.add('I');
    Frames_Color.add(color(255,255,0));
    f.save("results/frame" + i + "_I_y.png");
    continue;
}
if(checkThreshold(f, last_key_frame) == true){
    i_frame = false;
    counter_i = 1;
    counter_b = 0;
    last_key_frame = f;
    Frames_Type.add('I');
    Frames_Color.add(color(255,0,0));
    f.save("results/frame" + i + "_I_r.png");
    continue;
}

```

Risultati

Tramite la canvas di Processing, è possibile osservare i risultati della classificazione eseguita dal codice. Ogni frame viene mostrato accanto alla sua classificazione, com'è possibile notare dalle seguenti immagini.



Inoltre, i singoli fotogrammi vengono salvati nella cartella results, ridenominati per numero di frame e tipo.

Il video utilizzato per testare il programma è una sequenza di tre clip concatenate a 25 fps, per un totale di 251 frame, con una risoluzione di 240p.

Sono stati effettuati tre test cambiando solo i parametri della threshold dei blocchi intra e del range di ricerca.

Il primo test è stato effettuato impostando *intra_threshold* a 65 e *searchRange* a 8.

Nel secondo test, è stato aumentato solo il valore di *intra_threshold* da 65 a 70.

Nell'ultimo test, *intra_threshold* è stato nuovamente impostato a 70, mentre è stato aumentato il valore di *searchRange* da 8 a 16.

Il resto dei parametri, invece, è stato fissato seguendo le norme dell'algoritmo MPEG, come segue:

- $gop_size = 50$;
- $max_b_frame = 3$;
- $scene_change_threshold = 35$;
- $blockSize = 8$;
- $Qp = 26$;
- $lambda = 0.85$.

A seguito dei test effettuati, risulta evidente la presenza di due singoli I-frame gialli, dovuti ai due cambi di scena, e la presenza di I-frame rossi concentrati solo nelle due clip successive, i cui movimenti risultano più complessi da prevedere rispetto a quelli della prima clip: ciò rappresenta il motivo per il quale si è scelto di testare l'algoritmo, cambiando il limite di blocchi intra concessi prima di forzare il fotogramma a frame chiave.

Infatti, si osserva che aumentando la soglia del 5%, a parità di $searchRange$, il numero di I-frame rossi diminuisce di circa il 60%, passando da 279 a 114.

Il test successivo, invece, è stato effettuato aumentando anche il valore del range di ricerca da 8 a 16. Con questa configurazione, si è passati da 114 a 121 I-frame rossi totali; tuttavia, è bene ricordare che un range più ampio garantisce una motion estimation più accurata, seppur computazionalmente più lenta.

I risultati sono stati salvati e sono consultabili direttamente dalla cartella del progetto.

Conclusioni

In questo progetto è stata presentata un'analisi dettagliata dell'algoritmo di classificazione usato per le compressioni video MPEG.

È stato effettuato uno studio per comprendere il reale funzionamento dell'algoritmo, al fine di riprodurre fedelmente lo stesso modello di classificazione.

Una volta compreso il meccanismo alla base dell'MPEG, si è passati all'implementazione del codice, con l'obiettivo di fornire risultati visivi per una maggiore comprensione.

Ultimato il codice, sono stati eseguiti test su un filmato di 251 frame, con l'obiettivo di studiare e valutare come la variazione dei valori dei parametri utilizzati per la classificazione possa influenzare gli esiti dell'algoritmo.

Dalla visione dei risultati è stato possibile osservare quanto sia rilevante il valore dei parametri scelti per ottenere una classificazione soddisfacente e, di conseguenza, una buona compressione video.

Bibliografia

- Y. Wang, J. Osternmann and Ya-Qin Zhang, *Video Processing and Communications*, Prentice Hall, Pearson Education, 2002.
- I. E. Richardson, *The H.264 Advanced Videocompression Standard*, A John Wiley and Sons, Ltd., Publication, 2nd Edition, 2010.
- *x265*, Documentation. [Online]. Available at: <https://x265.readthedocs.io/en/master/>.
- R. S. Prasad and K. Ramkishor, "Efficient Implementation of MPEG-4 Encoded on RISC Core," in *IEEE International Conference on Consumer Electronics Dig. Tech. Papers*, Los Angeles (CA), USA, 2002, pp. 278-279.