FIC: Fractal Image Compression

L. De Sano, A. Donizetti

Settembre 2015

Indice

1 Introduzione

Con il termine "Fractal image Compression" si va ad indicare una famiglia di tecniche di compressione di immagini (o video) basate sulle proprietà matematiche dei frattali. Tali metodi di compressione si rivelano sopra ogni altra cosa adatti a comprimere textures e immagini naturali, o, più in generale, immagini che sono caratterizzate da un elevato livello di *self-similarity* (ovvero aventi delle parti che, al netto di rotazioni e ingrandimenti/riduzioni, somigliano ad altre parti dell'immagine).

La compressione di immagini tramite frattali (così come altre, più diffuse, ad esempio *JPEG*) appartiene a quel gruppo di tecniche di compressione *lossy*, ovvero in cui la compressione dell'immagine avviene al costo di una perdita di informazione. Tuttavia, a differenza di quanto accade quando si utilizza uno dei metodi di compressione basati sui pixel (come *JPEG*, *GIF* o *MPEG*), nella compressione frattale nessuna parte dell'immagine viene effettivamente memorizzata. Ciò che viene memorizzato è invece la *struttura interna* dell'immagine (ad esempio un indice di quali parti, effettuate le dovute trasformazioni, sono simili ad altre parti).

Poiché nessun pixel dell'immagine originale viene memorizzato, la decompressione parte da un singolo pixel, di colore qualsiasi, e procede alla ricostruzione dell'immagine originale applicando iterativamente una mappa ricavata dalla struttura interna dell'immagine originale.

In questo documento tratteremo delle tecniche di compressione di immagini basate su frattali, iniziando con una panoramica teorica del loro funzionamento, proseguendo con la discussione di alcuni aspetti pratici e presentando una implementazione giocattolo realizzata in MATLAB, e concludendo infine con alcuni test che consentiranno di valutare praticità e *performances* di una libreria di compressione basata su frattali, anche in confronto con altre tecniche di compressione *lossy* maggiormente utilizzate.

2 Aspetti teorici

2.1 Introduzione ai frattali

Un frattale è un oggetto geometrico che si ripete nella sua forma, allo stesso modo, su diverse scale. Questo comportamento fa si che ingrandendo una sua qualsiasi componente, ciò che si ottiene è una figura simile all'originale. In linea di massima, si può dire che perché l'insieme F sia considerato un frattale, F dovrebbe avere almeno le seguenti proprietà:

- F ha dettagli ad ogni scala d'ingrandimento;
- F gode di autosimilitudine (a qualunque scala si osservi, presenta sempre le stesse caratteristiche globali);
- la dimensione frattale di F è maggiore della sua dimensione topologica;
- esiste un algoritmo relativamente semplice per costruire F.

TODO: definire dimensione frattale e topologica

`http://www.vanderbilt.edu/AnS/psychology/cogsci/chaos/workshop/Fractals.html`

Un esempio di oggetto geometrico secondo i principi di costruzione di un frattale e che rispetta le proprietà elencate è la curva di *Koch*. La costruzione comincia con una linea di lunghezza 1 chiamata *initiatior*. Da questa linea si rimuove il terzo centrale e lo si sostituisce con due linee della stessa lunghezza della parte rimossa. Questa nuova forma viene chiamata *generator*. La prima parte della costruzione è mostrata in figura 2.1.

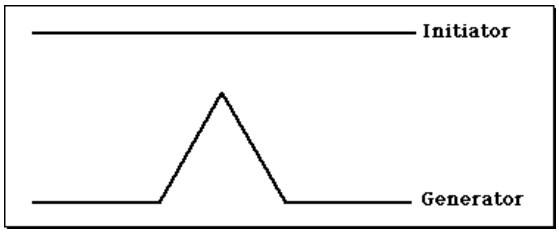

Figura 2.1: Initiator e generator per la curva di Koch

La regola può essere nuovamente applicata su ogni linea, così da andare a sostituirla ogni volta come fanno nel passaggio da *initiator* a *generator*. Il secondo livello è visibile in figura 2.2.

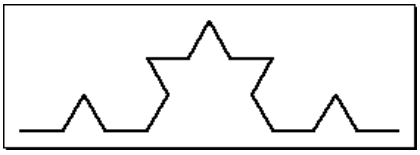

Figura 2.2: Livello 2 per la curva di Koch

Una volta che la procedura è avviata può proseguire a piacimento. Il terzo e il quarto livello sono visibili nelle figure 2.3 e 2.4 rispettivamente.

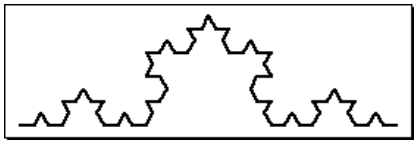

Figura 2.3: Livello 3 per la curva di Koch

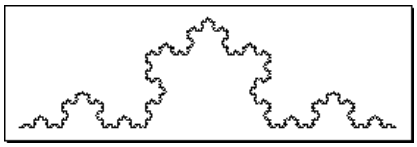

Figura 2.4: Livello 4 per la curva di Koch

2.2 Iterated Function Systems

Uno strumento matematico che si rivelerà fondamentale nella fase di decompressione delle immagini trattate con un metodo di compressione frattale è quello degli *Iterated Function Systems*, che andiamo per questo motivo a descrivere qui.

Senza eccedere in formalità, definiamo un *Iterated Function System* (IFS) come una collezione di trasformazioni contrattive $\{w_i : \mathbb{R}^2 \to \mathbb{R}^2\}_{i=1,\dots,n}$ che mappa il piano su se stesso. Questa collezione di trasformazioni definisce una mappa

$$W(\cdot) = \bigcup_{i=1}^{n} w_i(\cdot)$$

Tale mappa è applicata ad insiemi di punti, ed il risultato è definito in questa maniera: dato un insieme di punti $S \in \mathbb{R}^2$, calcoliamo $w_i(S)$ per ogni i (ovvero produciamo i copie ridotte di S), e poi calcoliamo l'unione dei risultati (ovvero, mettiamo insieme tutte le copie ridotte), ottenendo così un nuovo insieme $S' = W(S)$.

W così definito è una mappa tra sottoinsiemi di $\mathbb{R}^2$ (che d'ora in avanti per noi saranno "immagini"), e gode di due importanti (e rilevanti per la compressione frattale) proprietà, che andiamo ad enunciare senza dimostrazione.

Proprietà 1 *Se tutte le w_i sono contrattive, allora W è contrattiva in uno spazio di sottoinsiemi del piano.*

Proprietà 2 *Fissata una mappa W, esiste una immagine x_W, chiamata* attrattore per *W, tale che*

- $W(x_W) = x_W$
- *data una qualunque immagine di partenza S_0, vale che*

$$x_W \equiv \lim_{n\to\infty} W^n(S_0)$$

  ovvero l'applicazione ripetuta per n volte di W porta ad ottenere x_W, per n sufficientemente grande, ed indipendentemente dalla scelta dell'immagine di partenza.

- *x_W è unica. Se una qualsiasi immagine S soddisfa $W(S) = S$, allora S è l'attrattore per W.*

La seconda di queste proprietà è nota come *Contractive Mapping Fixed-Point Theorem.*

2.3 Self-similarity nelle immagini

Immagini come oggetti matematici

Per poter applicare la teoria degli *IFS* alle immagini come comunemente le intendiamo (ovvero, in sostanza, matrici di pixels rappresentati da 1 byte – *greyscale* – oppure più di uno – es. RBG –), è necessario formalizzare in qualche modo il concetto di "immagine digitale", cercando di inserirlo in un contesto matematico che ci consenta di manipolarla utilizzando di strumenti messi a disposizione dall'algebra. Per semplicità, ci limiteremo qui a considerare le immagini in *grayscale* (un singolo canale di colore, l'intensità indicata con un numero da 0 a 255).

Consideriamo un'immagine in scala di grigi, memorizzata come matrice di pixel, 1 byte per ogni pixel. Non è difficile immaginare di poter rappresentare tale immagine utilizzando una funzione $f : \mathbb{R}^2 \to \{1, 2, \ldots, 255\}$, in modo tale che ad ogni pixel alle coordinate (x, y) sia fatto corrispondere un punto (x, y) nel piano, al quale a sua volta viene associata un'altezza z, che corrisponde al valore di grigio del pixel in questione.

Ad esempio, ad un'immagine con il primo pixel in basso a sinistra avente livello di grigio 177, viene fatta corrispondere una funzione f che, per quanto riguarda quel preciso punto, avrà $f(0, 0) = 177$. Effettuando questa operazione su tutti i pixel dell'immagine originale, è possibile definire completamente f su tutta la parte del piano di nostro interesse (e, in genere, si riduce per convenzione l'immagine ad avere dominio $[0, 1]^2$). Nella

figura qui sotto riportata si può vedere un esempio del risultato finale della procedura, applicata ad una immagine grayscale 128×128 pixels.

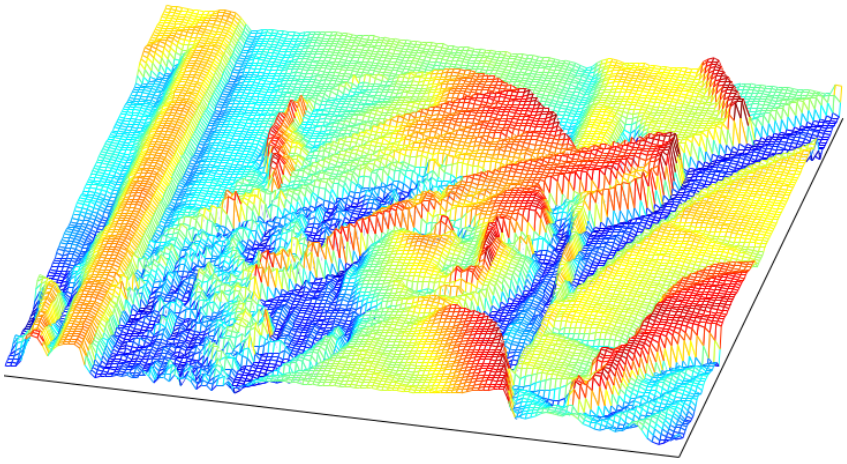

Figura 2.5: lena grayscale, 128×128 pixels, trasformata in funzione $[0,1]^2 \to \{0, \ldots, 255\}$

La trasformazione descritta ci permette di ignorare l'aspetto "visivo" della modalità in cui una immagine è memorizzata, e ci consente di identificare una immagine con la funzione f associata, su cui si andrà ad operare durante il procedimento di compressione.

Distanza tra due immagini

Un secondo concetto che è necessario introdurre per poter operare sulle immagini durante il procedimento di compressione e decompressione è quello di "distanza" tra due immagini. La contrattività di una mappa W (ovvero, intuitivamente, il fatto che essa rende le immagini più "piccole") può essere verificata solo se si ha a disposizione una qualche metrica sulla base della quale è possibile valutare la distanza tra due sottoinsiemi del piano (ovvero, nel nostro caso, due immagini).

Date due immagini f, g, definiamo la loro distanza $d(f, g)$, utilizzando la media quadratica, come

$$d(f,s) = \sqrt{\int_{[0,1]^2} f(x,y) - g(x,y) \quad \mathrm{d}x\mathrm{d}y}$$

La metrica ci fornisce una maniera di valutare la distanza tra due immagini pesando in maniera uniforme tutti i punti in esse contenuti.

Self-similarity debole

A differenza di quanto accade quando si osserva un frattale vero e proprio, all'interno di immagini "naturali" non troviamo una *self-similarity* forte: solitamente non abbiamo parti dell'immagine che sono simili all'immagine complessiva, bensì parti dell'immagine che sono simili ad altre parti dell'immagine. Un altro aspetto da considerare è quello della colorazione: due parti dell'immagine potrebbero essere simili nella composizione dei pixel, ma differenti per quanto riguarda la gradazione di grigio dei pixels che le compongono.

I basilari IFS che abbiamo descritto in precedenza vanno leggermente complicati, nel contesto della compressione frattale delle immagini, in modo da accomodare i due aspetti qui sopra descritti. Come prima cosa, notiamo che al momento di definire le trasformazioni w_i, dovremo fare in modo che ad ogni trasformazione sia consentito di andare a toccare solo un particolare sottoinsieme del piano, in maniera da accomodare il requisito di poter rappresentare la *self-similarity* debole (similarità tra parti e altre parti, e non con l'intera immagine). Come seconda cosa, alle mappe vanno aggiunti due parametri, *contrasto* e *luminosità*, che ci consentiranno di definire non solo trasformazioni *spaziali*, ma anche trasformazioni che vanno a coinvolgere la differenza di *colorazione* (sempre su scala di grigi) tra due sottoinsiemi di piano.

Invece della classica trasformazione su $\mathbb{R}^2$ che ci aspetteremmo considerato il fatto che stiamo operando su funzioni $\mathbb{R}^2 \to \mathbb{R}$, definiamo le w_i come trasformazioni su $\mathbb{R}^3$:

$$w_i \begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} a_i & b_i & 0 \\ c_i & d_i & 0 \\ 0 & 0 & s_i \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ z \end{bmatrix} + \begin{bmatrix} e_i \\ f_i \\ o_i \end{bmatrix}$$

La trasformazione è formata da una componente *spaziale*, ovvero

$$v_i \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} a_i & b_i \\ c_i & d_i \end{bmatrix} \cdot \begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} e_i \\ f_i \end{bmatrix}$$

e di due parametri aggiuntivi s_i ed o_i, che agiscono rispettivamente come moltiplicatore e scala tramite somma sulla componente z della funzione. Questi due parametri, che chiamiamo rispettivamente *contrasto* e *luminosità*, non influenzano in alcun modo le componenti spaziali della trasformazione, e ci consentono di definire trasformazioni sullo spazio di *colore* dell'immagine (che, ricordiamo, è rappresentato nella funzione associata all'immagine come la componente z).

Ricordando che l'immagine è rappresentata da una funzione $z = f(x, y)$, dove x ed y sono pixels e z il grado di grigio dell'immagine *grayscale*, una singola trasformazione w_i viene applicata calcolando $w_i(f) = w_i(x, y, f(x, y))$. La parte spaziale di w_i determina come una sotto-parte dell'immagine è mappata su altre sottoparti, mentre s_i ed o_i determinano contrasto e luminosità della trasformazione.

Il fatto che il mapping sia da effettuarsi da sottoinsiemi a sottoinsiemi dell'immagine è permesso dal fatto che le varie w_i sono implicitamente definite in modo da operare su un sottoinsieme di $[0,1]^2$, che chiamiamo D_i, dando come risultato un sottoinsieme di $[0,1]^2$, che chiameremo R_i. In simboli, scriviamo che

$$v_i(D_i) = R_i$$

ovvero la componente spaziale della trasformazione ha dominio in sottoinsiemi del piano che chiamiamo D_i e codominio in sottoinsiemi del piano che chiamiamo R_i.

Poiché richiediamo che $W(f)$, l'unione delle w_i, sia a sua volta un'immagine compatta, le trasformazioni devono soddisfare due proprietà basilari: prima di tutto gli R_i devono partizionare completamente $[0,1]^2$, ovvero deve valere $\bigcup_i R_i = [0,1]^2$. Inoltre gli R_i non devono sovrapporsi, ovvero deve valere $R_i \neq R_j$ se $i \neq j$. Queste due proprietà garantiscono che l'applicazione della mappa W su una immagine S risulti in una immagine S', possibilmente differente, ma completa ed univoca.

Poiché la mappa W è contrattiva solo se lo sono le trasformazioni w_i, è necessario scegliere accuratamente i parametri di queste ultime. Dato che la metrica di distanza d che abbiamo definito sopra non prende in considerazione le direzioni x ed y, ma soltanto la direzione z, per assicurare la contrattività delle w_i è sufficiente fare attenzione al parametro di scala s_i. In particolare:

Proprietà 3 *Le trasformazioni w_i sono contrattive per $s_i < 1$*

Decoding

A questo punto abbiamo tutti gli strumenti che servono per effettuare il *decoding* di una immagine compressa utilizzando frattali: partendo da una immagine qualunque, si applica ripetutamente la mappa W fino a quando non si raggiunge il suo punto fisso x_W, che rappresenta il risultato dell'operazione di decodifica.

L'aspetto interessante (e complicato) della tecnica di compressione che stiamo descrivendo è però un altro: come fare, data una immagine f da comprimere, a trovare un insieme di mappe $\{w_i\}_i$ tale che f sia il punto fisso della loro unione (W)? Questa operazione, che rappresenta il nucleo delle tecniche di compressione tramite frattali, è l'argomento della prossima sezione.

2.4 Encoding

Come anticipato nella precedente sezione, per comprimere un'immagine f è necessario trovare una mappa W tale che

$$f = W(f)$$

ovvero una mappa W avente f come attrattore. Ricordando la definizione di W, la proprietà richiesta diventa

$$f = w_1(f) \cup w_2(f) \cup \cdots \cup w_N(f)$$

Per ottenere questo, dovremmo procedere con il partizionare l'immagine f in tanti pezzi tali per cui, applicando le trasformazioni $\{w_i\}_i$, si otterrebbe l'immagine di partenza. Chiaramente, in generale, questa è una condizione troppo restrittiva: ottenere *esattamente* l'immagine di partenza non è qualcosa che sia possibile sperare (a parte nei casi in cui si stia cercando di comprimere un'immagine frattale!).

Quello che facciamo in pratica è cercare delle trasformazioni che, applicate, ci restituiscano un'immagine *differente*, diciamo f', tale per cui la distanza con l'immagine di partenza, ovvero $d(f, f')$, è piccola. Operativamente, procediamo calcolando le w_i con l'obbiettivo di minimizzare la distanza tra il pezzo di immagine che stiamo attualmente considerando e il pezzo di immagine ottenuto dopo l'applicazione della trasformazione. In formula, minimizziamo

$$d\big(f \cap (R_i \times I), w_i(f)\big)$$

per tutti gli i. In termini di R_i ed D_i (i concreti pezzi di immagini su cui andremo ad operare), dobbiamo partizionare l'immagine f in una collezione di pezzi R_i, e cercare da una seconda collezione D_i dei pezzi tali per cui la mappa da D_i ad R_i comporta un errore (calcolato come distanza d) che sia piccolo.

Partizionare le immagini

Una questione che è necessario innanzitutto risolvere è quella relativa alla scelta delle collezioni R_i e D_i.

La maniera più ovvia di partizionare f è quella di dividerla in quadrati di una dimensione predeterminata, ma questa scelta presenta dei problemi. Nel caso all'interno dell'immagine da trattare non vi sia una uniforme distribuzione della "complessità dei dettagli" (succede, ad esempio, per un ritratto di persona, che avrà un alto livello di complessità nella zona dove si trova il viso del soggetto e un basso livello di complessità ai bordi, dove si vede un muro di colore uniforme che fa da sfondo alla fotografia) una divisione uniforme andrà a penalizzare le zone con maggior presenza di dettagli, e a sprecare inutilmente blocchi (che saranno tutti uguali) per lo sfondo.

Un metodo di partizionamento migliore è quello basato sui *quadtrees*. Si divide inizialmente l'immagine in 4 parti, e si controlla il livello di fedeltà dei sotto-quadrati rispetto

all'immagine orignale. Se il livello di fedeltà è entro un limite e_c prefissato, il blocco entra a far parte dell'insieme degli R_i. Altrimenti lo si partiziona in 4 parti e si ri-applica nuovamente, ricorsivamente, il procedimento appena descritto.

In genere è anche conveniente, nel contesto della compressione di immagini, scegliere due parametri $Q_{\max}$ e $Q_{\min}$ che definiscono rispettivamente la grandezza massima e minima dei blocchi R_i. Nel caso un blocco abbia dimensioni superiori a $Q_{\max}$, lo si partiziona a prescindere dalla complessità interna; nel caso un blocco avente dimensioni $Q_{\min}$ si cessa di partizionarlo, a prescindere dal livello di uniformità interna. Questi due parametri consentono di avere un maggior controllo sulle dimensioni dei blocchi R_i, e soprattutto sulla cardinalità dell'insieme degli R_i, che influenza pesantemente il tempo di esecuzione dell'operazione di codifica.

L'immagine sottostante fornisce un esempio del tipo di partizione che si può ottenere utilizzando un *quadtree*. Si nota che nei punti in cui l'immagine è uniforme i blocchi sono di dimensioni maggiori, mentre nei punti ricchi di dettagli (il viso, la piuma del cappello) la suddivisione è tanto fine quanto lo consente il parametro $Q_{\min}$.

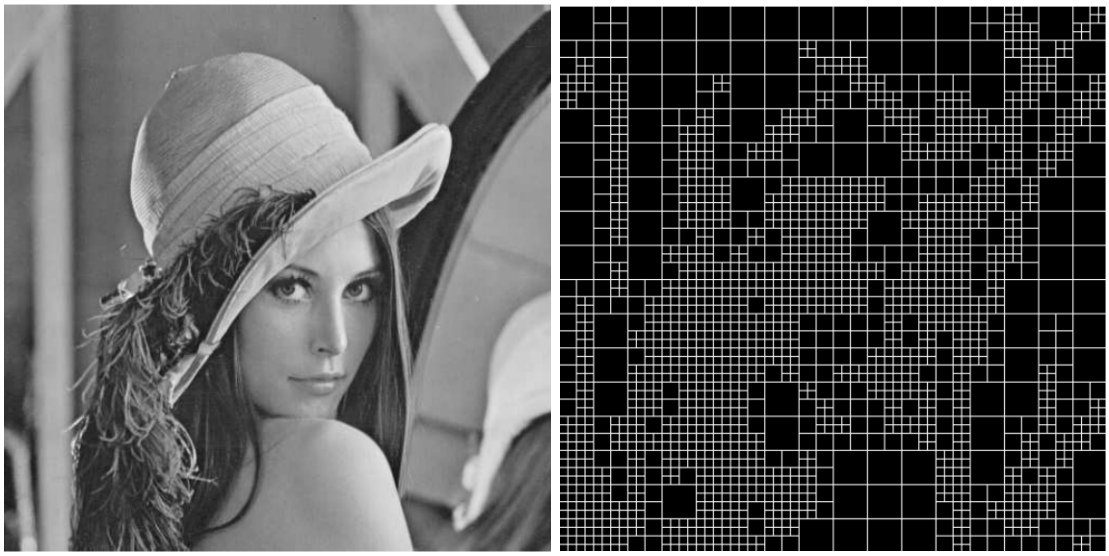

Figura 2.6: Quadtree su Lena, $Q_{\min} = 8$, $Q_{\max} = 64$, $e_c = 0.4$

Oltre al partizionamento con *quadtree*, esistono altri metodi applicabili in questo contesto, ad esempio il metodo di partizionamento HV ed il metodo di partizionamento triangolare, che va a generare sottoimmagini non quadrate e può essere quindi vantaggioso nei casi in cui l'immagine di partenza contenga un gran numero di linee oblique (che non sono facilmente coperte da metodi di partizionamento basati su quadrati).

L'algoritmo

Abbiamo ora a disposizione tutti gli elementi che ci servono per descrivere ad alto livello il funzionamento dell'algoritmo di codifica frattale. Lo riportiamo di seguito:

procedure ENCODE(e_c, $r_{\min}$)
  $R_1 \leftarrow [0,1]^2$
  $R \leftarrow R \cup R_1$
  Segna R_1 come non coperto
  while $\exists R_i \in R$ non coperto **do**
    $D_i, w_i \leftarrow$ BESTCOVER(R_i)
    if $d\big(f \cap (R_i \times I), w_i(f)\big) < e_c$ OR size(R_i) $\leq r_{\min}$ **then**
      Segna R_i come coperto
      $W \leftarrow W \cup w_i$
    else
      Partiziona R_i in 4 parti $R_a, R_b R_c, R_d$
      $R \leftarrow R \setminus R_i$
      $R \leftarrow R \cup \{R_a, R_b, R_c, R_d\}$
      Segna $R_{\{a,b,c,d\}}$ come non coperti
    end if
  end while
end procedure

La procededura BESTCOVER prende come input una sottimmagine R_i, e ritorna la sottoimmagine D_i e la rispettiva trasformazione w_i che coprono meglio R_i.

procedure BESTCOVER(R_i)
  $score \leftarrow \infty$
  $d, w \leftarrow$, NULL, NULL
  for all $D_i \in D$ **do**
    $w_{\text{temp}}, s \leftarrow$ SCORE(D_i, R_i)
    if $s < score$ **then**
      $w \leftarrow w_{\text{temp}}$
      $score \leftarrow s$
      $d \leftarrow D_i$
    end if
  end for
  return d, w
end procedure

La procedura SCORE restituisce una trasformazione w_i ed uno *score* che indica il livello di somiglianza tra la sottoimmagine di partenza R_i e la sottoimmagine calcolata D_i.

3 Aspetti Pratici e implementazione

4 Test, benchmarks