Appunti del corso di *“Matematica Computazionale”*

prof.ssa *Biancamaria della Vecchia*

Trasformata di Fourier & Trasformata Wavelet Discreta

Antonio Petrella 1159178 pausatep@gmail.com

Indice

Introduzione

Fotografia Analogica e Digitale

Nella macchina fotografica analogica il segnale, considerato continuo, passa attraverso l'otturatore e impressiona la pellicola.

Per quanto riguarda la macchia fotografica digitale, nell'obbiettivo sono disposti dei sensori in grado di associare un valore all'intensità di luce che li colpisce.

Di conseguenza nel momento in cui viene scattata una foto verrà scritta una matrice con i valori rilevati dai sensori (1 byte per sensore), c'è un problema: la dimensione della matrice potrebbe essere molto grande.

Supponiamo di avere una macchina fotografica da 1 megapixel, quindi 1.000.000 di pixel, equivalenti a 3.000.000 di sensori nell'obbiettivo, abbiamo 3 sensori per ogni pixel (uno per ogni colore della terna RGB[1]), ciò signifca scrivere una matrice da 3.000.000 byte (circa 3 Mb) ogni qualvolta si scatta una foto.

Immaginiamo di voler inviare la foto appena scattata ad un amico usando la nostra connessione a 56k[2] (trasferimento a cira 6.000 byte/s):

$$\frac{3.000.000}{6.000} = 500\ secondi = 8\ minuti$$

un po' troppo per mandare una singola foto, anche perché il caso preso in considerazione utilizza una fotocamera da appena 1 megapixel (immaginiamo di voler inviare una foto da 10 megapixel).

L'immagine è troppo grande, è necessario ridurre la sua dimensione e per farlo si utilizzano degli appositi algoritmi di compressione.

[1] in realtà nelle macchine fotografiche è presente un numero inferiore di sensori.

[2] al momento vengono comunemente usate connessioni nettamente superiori al 56k, ma il problema si pone comuqnue, ad esempio: nel momento in cui si decide di inviare un album da centinaia di foto.

Capitolo 1

Algoritmi di Compressione

Nella compressione delle immagini vengono usati algoritmi di tipo lossy[1], sfruttando il fatto che la perdita d'informazione non è sempre visibile.

1.1 Metodo dell'accetta

Si tratta di un algoritmo molto semplice e con un rapporto di compressione molto alto, l'unico difetto è che viene persa troppa informazione, infatti esso opera eliminando tutte le righe e tutte le colonne pari (o dispari) dalla matrice di riferimento, riducendo la risoluzione dell'immagine.

1.2 Rappresentazione Funzionale

Osserviamo una riga della matrice, se la rappresentiamo su un grafico in cui l'ascissa è la posizione nella riga e l'ordinata è l'intensità associata a quella posizione (fig. 1.1), possiamo cercare di trovare una funzione che passi per i punti del grafico. Per immagini molto semplici (un raggio di luce obliquo in un ambiente buio) si può usare la funzione della retta (con $f(x)$ si intende l'intensità nel punto x):

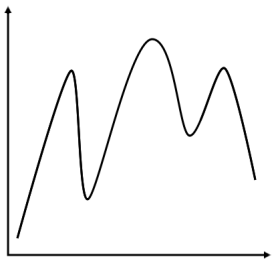

Figura 1.1: Grafico della riga

$$f(x) = ax + b$$

Questo non è un caso realistico, infatti la maggiorparte delle immagini sono molto più complesse di un raggio di luce.

[1] esistono due tipi di compressione: lossy (con perdita d'informazione) e lossless (senza perdita d'informazione).

Potremmo provare un approccio polinomiale con grado n non maggiore del numero di punti della riga della matrice:

$$f(x) = a_0 + a_1 x^1 + a_2 x^2 + a_3 x^3 + ... + a_{n-1} x^{n-1}$$

Purtroppo i polinomi non vanno bene, un grado molto alto porta a funzioni molto oscillanti e a problemi di stabilità numerica.

Un altra soluzione potrebbe essere la combinazione lineare di funzioni note moltiplicate per alcuni parametri:

$$f(x_i) = a_0\varphi_0(x_i) + a_1\varphi_1(x_i) + a_2\varphi_2(x_i) + ... + a_{n-1}\varphi_{n-1}(x_i)$$

Una volta scelte le φ_i, essendo le $f(x_i)$ note, è necessario calcolare i coefficienti, ad esempio, nel primo pixel l'intensità è:

$$f_1 = a_0\varphi_0(x_1) + a_1\varphi_1(x_1) + ... + a_{n-1}\varphi_{n-1}(x_1)$$

nel secondo:

$$f_2 = a_0\varphi_0(x_2) + a_1\varphi_1(x_2) + ... + a_{n-1}\varphi_{n-1}(x_2)$$

$$\vdots$$

$$\vdots$$

$$\vdots$$

nell'n-esimo:

$$f_n = a_0\varphi_0(x_n) + a_1\varphi_1(x_n) + ... + a_{n-1}\varphi_{n-1}(x_n)$$

Si ottiene un sistema di equazioni lineari a n incognite che risolto fornisce la soluzione.

Un problema, da non sottovalutare, di questa rappresentazione è la scelta delle φ_i, infatti una scelta sbagliata potrebbe portare ad una complessità dell'algoritmo molto elevata oppure ad una rappresentazione non corretta dell'informazione di partenza.

Capitolo 2

Trasformata di Fourier

L'idea è di utilizzare le funzioni sin e cos per le φ_i, ottenendo:

$T(x) = a_0 \cos(0\pi x) + a_1 \cos(2\pi x) + a_2 \cos(4\pi x) + \ldots + b_0 \sin(0\pi x) + b_1 \sin(2\pi x) + b_2 \sin(4\pi x) + \ldots$

Possiamo scrivere la formula di sopra in modo compatto:

$$T(x) = \sum a_j \cos(2j\pi x) + b_i \sin(2j\pi x) \tag{2.1}$$

Osservazioni: Proviamo a risolvere la 2.1 per $T(0)$ e per $T(1)$.
Ricordando che $\sin(2\pi) = 0$ e che $\cos(2\pi) = 1$, è facile verificare che:

- $\sin(2j\pi 0) = \sin(2j\pi 1) = 0$;
- $\cos(2j\pi 0) = \cos(2j\pi 1) = 1$;

da cui otteniamo:

- $T(0) = a_0 + a_1 + a_2 + a_3 + \ldots + 0$;
- $T(1) = a_0 + a_1 + a_2 + a_3 + \ldots + 0$;

di conseguenza agli estremi dell'intervallo $[0, 1]$ abbiamo gli stessi risultati.
Proviamo a risolverla ora per $T(x + 1)$:

$$T(x+1) = \sum a_j \cos(2j\pi(x+1)) + b_j \sin(2j\pi(x+1)) =$$

$$= \sum a_j \cos(2j\pi x + 2j\pi) + b_j \sin(2j\pi x + 2j\pi)$$

sapendo che:

- $\cos(h + 2k\pi) = \cos(h)$;
- $\sin(h + 2k\pi) = \sin(h)$;

otteniamo:

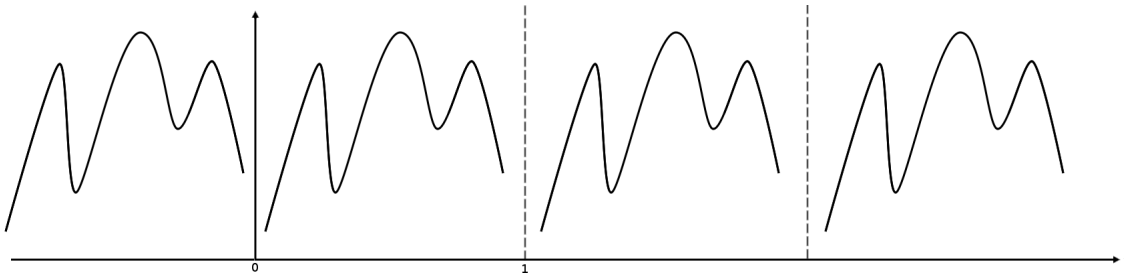

Figura 2.1: Periodo della trasformata di Fourier

$$T(x+1) = \sum a_j \cos(2j\pi x + 2j\pi) + b_j \sin(2j\pi x + 2j\pi) =$$

$$= \sum a_j \cos(2j\pi x) + b_j \sin(2j\pi x) = T(x)$$

Da queste osservazioni notiamo che la formula 2.1 è di periodo 1, ciò significa che la trasformata si ripete a destra e a sinistra dell'intervallo $[0, 1)$ (fig. 2.1).

2.1 Sintesi e Analisi di Fourier

La formula che esprime l'analisi di Fourier è la seguente:

$$T_N(x_k) = \sum_{j=0}^{N} a_j \cos(2j\pi x_k) + b_j \sin(2j\pi x_k) \tag{2.2}$$

dove:

- $2N$ è il numero di pixel dell'immagine;
- x_k è il pixel, essi sono equidistanti nell'intervallo $[0, 1]$, quindi:

$$x_k = \frac{k}{2N} \qquad 0 \leq k \leq 2N$$

per rappresentare un'immagine in questo modo il nostro obbiettivo è calcolare gli a_j e b_j.

Osservazioni: Il numero totale di pixel viene rappresentato con $2N$, mentre la sommatoria va da $j = 0$ a N, questa scelta è necessaria per poter calcolare i parametri (a_j e b_j), infatti abbiamo due parametri per ogni j, quindi $2N + 2$ parametri.

In base a quanto detto sopra, per poter risolvere il sistema e trovare i $2N+2$ parametri, dovremmo avere $2N + 2$ equazioni, ricordando che il polinomio è periodico in $[0, 1)$, contiamole:

$$T_N(x_0) = f_0 = \sum_{j=0}^{N} a_j \cos(2j\pi \frac{0}{2N}) + b_j \sin(2j\pi \frac{0}{2N}) \qquad x_0 = \frac{0}{2N}$$

$$f_1 = \sum_{j=0}^{N} a_j \cos(2j\pi \frac{1}{2N}) + b_j \sin(2j\pi \frac{1}{2N}) \qquad x_1 = \frac{1}{2N}$$

$$\vdots$$

$$\vdots$$

$$\vdots$$

$$f_{2N-1} = \sum_{j=0}^{N} a_j \cos(2j\pi \frac{2N-1}{2N}) + b_j \sin(2j\pi \frac{2N-1}{2N}) \qquad x_{2N-1} = \frac{2N-1}{2N}$$

quindi abbiamo $2N$ equazioni.

C'è un problema: abbiamo $2N+2$ parametri da calcolare e solo $2N$ equazioni, ma osserviamo meglio il sistema:

- per b_0 abbiamo:
$$b_0 \sin \frac{2 \cdot 0\pi k}{2N} = b_0 \sin 0;$$
- per b_N abbiamo:
$$b_N \sin \frac{2N\pi k}{2N} = b_N \sin \pi k;$$

Ricordando che $\sin 0 = 0$ e $\sin \pi k = 0$, si nota che nei casi contenenti b_0 e b_N abbiamo dei risultati indefiniti, dato che vengono moltiplicati per 0, per cui possiamo escluderli dalla formula.

Ricapitolando, abbiamo $2N + 2$ incognite, ma solo $2N$ equazioni, abbiamo anche visto però che non è necessario calcolare b_0 e b_N, di conseguenza il numero di incognite è sceso a $2N$, rendendo il sistema risolvibile.

2.1.1 Risoluzione del sistema

Il sistema è ortogonale:

- $\sum \cos 2j\pi x_k \cdot \cos 2l\pi x_k = \begin{cases} 0 & se\ j \neq l \\ \neq 0 & se\ j = l \end{cases};$

- $\sum \sin 2j\pi x_k \cdot \sin 2l\pi x_k = \begin{cases} 0 & se\ j \neq l \\ \neq 0 & se\ j = l \end{cases}$;
- $\sum \sin 2j\pi x_k \cdot \cos 2l\pi x_k = 0$;

ciò ci consente di risolverlo analiticamente, senza usare costosi metodi numerici:

$$\begin{cases} a_j = \frac{1}{2N} \cdot \sum_{k=0}^{2N-1} f_k \cos 2j\pi x_k \\ b_j = \frac{1}{2N} \cdot \sum_{k=0}^{2N-1} f_k \sin 2j\pi x_k \end{cases} \tag{2.3}$$

dove $f_k = T_N(x_k)$.
La 2.3 è l'analisi di Fourier.

Osservazioni: Assumendo che i sin e i cos siano già stati calcolati, analizziamo il costo computazionale della 2.3:

- dobbiamo fare prima $2N$ moltiplicazioni ($f_k \cdot \cos 2j\pi x_k$);
- in seguito abbiamo $2N$ somme;
- 1 moltiplicazione ($\frac{1}{2N}$);
- tutto ciò per $2N$ volte (le operazioni vanno ripetute per tutti gli a_j e b_j);
- per un totale di $2N(2N + 2N + 2) = 8N^2 + 4N = O(N^2)$.

2.2 Trasformata Veloce di Fourier (FFT)

Riepilogo sui Numeri Complessi

Un numero complesso è rappresentato da una parte reale e da una parte immaginaria:

$$(a + \imath b)$$

Proprietà importanti

$\imath$ è l'unità immaginaria:

$$\imath = \sqrt{-1} \qquad\qquad \imath^2 = -1$$

La somma di due numeri complessi è così definita:

$$(a + \imath b) + (c + \imath d) = a + \imath b + c + \imath d = (a + c) + \imath(b + d)$$

La moltiplicazione:

$$(a + \imath b) \cdot (c + \imath d) = ac + \imath ad + \imath bc + \imath^2 bd =$$

$$= ac + \imath ad + \imath bc - bd =$$

$$= (ac - bd) + \imath(ad + bc)$$

Esponenziale:

$$e^{\imath x} = \cos x + \imath \sin x \qquad e^{-\imath x} = \cos(-x) + \imath \sin(-x) = \cos x - \imath \sin x$$

Formule d'Eulero:

$$e^{\imath x} + e^{-\imath x} = 2\cos x \qquad \cos x = \frac{e^{\imath x} + e^{-\imath x}}{2} \tag{2.4}$$

$$e^{\imath x} - e^{-\imath x} = 2\imath \sin x \qquad \sin x = \frac{e^{\imath x} - e^{-\imath x}}{2\imath} \tag{2.5}$$

Infine:

$$\frac{1}{\imath} = -\imath$$

2.2.1 Aumentare la compattezza dell'analisi di Fourier (2.2)

Partendo dalla 2.2:

$$T_N(x_k) = \sum_{j=0}^{N} a_j \cos(2j\pi x_k) + b_j \sin(2j\pi x_k)$$

e utilizzando le forumule di Eulero (2.4, 2.5), otteniamo:

$$T_N(x_k) = \sum_{j=0}^{N} a_j \cdot \frac{e^{\imath 2j\pi x_k} + e^{-\imath 2j\pi x_k}}{2} + b_j \cdot \frac{e^{\imath 2j\pi x_k} - e^{-\imath 2j\pi x_k}}{2i} =$$

$$= \sum_{j=0}^{N} a_j \cdot \frac{e^{\imath 2j\pi x_k} + e^{-\imath 2j\pi x_k}}{2} - \imath b_j \cdot \frac{e^{\imath 2j\pi x_k} - e^{-\imath 2j\pi x_k}}{2} =$$

$$= \sum_{j=0}^{N} \frac{a_j e^{\imath 2j\pi x_k} + a_j e^{-\imath 2j\pi x_k} - \imath b_j e^{\imath 2j\pi x_k} + \imath b_j e^{-\imath 2j\pi x_k}}{2}$$

mettendo in evidenza rispetto a $e^{\imath}$ e $e^{-\imath}$:

$$T_N(x_k) = \sum_{j=0}^{N} e^{\imath 2j\pi x_k} \frac{(a_j - \imath b_j)}{2} \cdot e^{-\imath 2j\pi x_k} \frac{(a_j + \imath b_j)}{2} =$$

$$= \sum_{j=-N}^{N} c_j \cdot e^{\imath 2 j \pi x_k} \tag{2.6}$$

con:

$$c_j = \begin{cases} \frac{a_j - \imath b_j}{2} & 0 < j < N \\ \frac{a_j + \imath b_j}{2} & -N < j < 0 \\ \frac{a_N}{2} & j = N \ o \ j = -N \\ a_0 & j = 0 \end{cases}$$

La sintesi di Fourier è data da:

$$C_j = \frac{1}{N} \cdot \sum_{k=0}^{N-1} f_k \cdot e^{\frac{-\imath 2 j \pi k}{N}} \tag{2.7}$$

2.2.2 Suddividere la trasformata

Partendo dalla 2.7, vogliamo scrivere la somma come coppie di termini:

$$C_j = \frac{1}{N} \cdot \sum_{k=0}^{\frac{N}{2}-1} f_k \cdot e^{\frac{-\imath 2 j \pi k}{N}} + f_{k+\frac{N}{2}} \cdot e^{\frac{-\imath 2 j \pi (k+\frac{N}{2})}{N}} =$$

$$= \frac{1}{N} \cdot \sum_{k=0}^{\frac{N}{2}-1} f_k \cdot e^{\frac{-\imath 2 j \pi k}{N}} + f_{k+\frac{N}{2}} \cdot e^{\frac{-\imath 2 j \pi k}{N}} \cdot e^{\frac{-\imath 2 j \pi \frac{N}{2}}{N}}$$

ponendo $w = e^{\frac{-\imath 2 \pi}{N}}$:

$$C_j = \frac{1}{N} \cdot \sum_{k=0}^{\frac{N}{2}-1} f_k \cdot w^{jk} + f_{k+\frac{N}{2}} \cdot w^{jk} \cdot w^{j\frac{N}{2}} \tag{2.8}$$

analizziamo il termine $w^{j\frac{N}{2}}$:

$$w^{j\frac{N}{2}} = (e^{\frac{-\imath 2 \pi}{N}})^{j\frac{N}{2}} = e^{\frac{-\imath 2 \pi j \frac{N}{2}}{N}} =$$

$$e^{\frac{-\imath 2 \pi j}{2}} = e^{-\imath \pi j}$$

ricordando che $\sin \pi k = 0$ e che $e^{\imath x} = \cos x + \imath \sin x$, abbiamo:

$$e^{-\imath \pi j} = \cos(-\pi j) + \imath \sin(-\pi j) = \cos \pi j - \imath \sin \pi j = \cos(\pi j)$$

$$\cos \pi j = \begin{cases} 1 & per \ j \ pari \\ -1 & per \ j \ dispari \end{cases}$$

quindi:

$$\cos \pi j = (-1)^j$$

Possiamo riscrivere la 2.8 così:

$$C_j = \frac{1}{N} \cdot \sum_{k=0}^{\frac{N}{2}-1} f_k \cdot w^{jk} + f_{k+\frac{N}{2}} \cdot w^{jk} \cdot (-1)^j$$

A questo punto possiamo suddividere due casi, a seconda che j sia pari o dispari:

$$\begin{cases} C_{2m_1} = \frac{1}{N} \cdot \sum_{k=0}^{\frac{N}{2}-1} (f_k + f_{k+\frac{N}{2}}) w^{2m_1 k} \\ C_{2m_1+1} = \frac{1}{N} \cdot \sum_{k=0}^{\frac{N}{2}-1} (f_k - f_{k+\frac{N}{2}}) w^{(2m_1+1)k} \end{cases} =$$

$$= \begin{cases} C_{2m_1} = \frac{1}{N} \cdot \sum_{k=0}^{\frac{N}{2}-1} (f_k + f_{k+\frac{N}{2}}) w^{2m_1 k} \\ C_{2m_1+1} = \frac{1}{N} \cdot \sum_{k=0}^{\frac{N}{2}-1} (f_k - f_{k+\frac{N}{2}}) w^k \cdot w^{2m_1 k} \end{cases}$$

ponendo ora:

$$w^2 = w_1; \qquad (f_k + f_{k+\frac{N}{2}}) = f_k^{(1)}; \qquad (f_k - f_{k+\frac{N}{2}}) w^k = f_{k+\frac{N}{2}}^{(1)}; \qquad \frac{1}{N} = \frac{1}{2} \cdot \frac{1}{\frac{N}{2}}$$

otteniamo:

$$\begin{cases} C_{2m_1} = \frac{1}{2} [\frac{1}{\frac{N}{2}} \cdot \sum_{k=0}^{\frac{N}{2}-1} f_k^{(1)} \cdot w_1^{m_1 k}] \\ C_{2m_1+1} = \frac{1}{2} [\frac{1}{\frac{N}{2}} \cdot \sum_{k=0}^{\frac{N}{2}-1} f_{k+\frac{N}{2}}^{(1)} \cdot w_1^{m_1 k}] \end{cases} \tag{2.9}$$

Osservazioni: Assumendo di aver già calcolato gli esponenziali, analizziamo la complessità della 2.9:

- per i numeri pari:
  - occorre fare $\frac{N}{2}$ somme per calcolare gli $f_k^{(1)}$;
  - in seguito vanno fatte $\frac{N}{2}$ moltiplicazioni;
  - in più altre $\frac{N}{2}$ somme per risolvere la sommatoria;
  - 2 moltiplicazioni;
  - il tutto ripetuto per $\frac{N}{2}$ volte;
  - per un totale di $\frac{N}{2}(\frac{N}{2} + \frac{N}{2} + \frac{N}{2} + 2) = \frac{3}{4} \cdot N^2 + N$;

- per i numeri dispari:
  - occorre fare $\frac{N}{2} + \frac{N}{2}$ per calcolare gli $f^{(1)}_{k+\frac{N}{2}}$;
  - le restanti operazioni sono identiche a quelle per i numeri pari;
  - per un totale di $\frac{N}{2}(\frac{N}{2} + \frac{N}{2} + \frac{N}{2} + \frac{N}{2} + 2) = N^2 + N$;

Il numero totale di operazioni necessarie è: $\frac{3}{4} \cdot N^2 + N + N^2 + N = O(N^2)$, di conseguenza non abbiamo guadagnato molto rispetto alla 2.3.

2.2.3 Velocizzare la trasformata di Fourier (solo se $N = 2^L$)

Confrontando la 2.7 con la 2.9, si nota che sono molto simili tra loro, ambedue sono delle sommatorie tra valori (f_k, $f^{(1)}_k$) moltiplicati per degli esponenziali, di conseguenza si può ulteriormente suddividere la 2.9, ottenendo un'altra formula della stessa forma a sua volta suddividibile.

Si può continuare questo procedimento fino ad eliminare completamente la sommatoria, infatti all'inizio si avranno N somme da N termini, in seguito $\frac{N}{2}$ somme da $\frac{N}{2}$ termini e così via, graficamente:

$$\left\{ N \textit{ somme da } N \textit{ termini} \quad \Rightarrow \begin{cases} \frac{N}{2} \textit{ somme da } \frac{N}{2} \textit{ termini} & \Rightarrow \begin{cases} \frac{N}{4} \textit{ somme da } \frac{N}{4} \textit{ termini} & \cdots \\ \frac{N}{4} \textit{ somme da } \frac{N}{4} \textit{ termini} & \cdots \end{cases} \\ \frac{N}{2} \textit{ somme da } \frac{N}{2} \textit{ termini} & \Rightarrow \begin{cases} \frac{N}{4} \textit{ somme da } \frac{N}{4} \textit{ termini} & \cdots \\ \frac{N}{4} \textit{ somme da } \frac{N}{4} \textit{ termini} & \cdots \end{cases} \end{cases} \right.$$

esempio con 8 punti:

$$\begin{pmatrix} f_0 \\ f_1 \\ f_2 \\ f_3 \\ f_4 \\ f_5 \\ f_6 \\ f_7 \end{pmatrix} \Rightarrow \begin{pmatrix} f_0 + f_4 & = f^{(1)}_0 \\ f_1 + f_5 & = f^{(1)}_1 \\ f_2 + f_6 & = f^{(1)}_2 \\ f_3 + f_7 & = f^{(1)}_3 \\ \cdots\cdots\cdots & \cdots \\ f_0 - f_4 & = f^{(1)}_4 \\ (f_1 - f_5) \cdot w & = f^{(1)}_5 \\ (f_2 - f_6) \cdot w^2 & = f^{(1)}_6 \\ (f_3 - f_7) \cdot w^3 & = f^{(1)}_7 \end{pmatrix} \Rightarrow \begin{pmatrix} f^{(1)}_0 + f^{(1)}_2 & = f^{(2)}_0 \\ f^{(1)}_1 + f^{(1)}_3 & = f^{(2)}_1 \\ \cdots\cdots\cdots & \cdots \\ (f^{(1)}_0 - f^{(1)}_2) & = f^{(2)}_2 \\ (f^{(1)}_1 - f^{(1)}_3) \cdot w_1 & = f^{(2)}_3 \\ \cdots\cdots\cdots & \cdots \\ f^{(1)}_4 + f^{(1)}_6 & = f^{(2)}_4 \\ f^{(1)}_5 + f^{(1)}_7 & = f^{(2)}_5 \\ \cdots\cdots\cdots & \cdots \\ (f^{(1)}_4 - f^{(1)}_6) & = f^{(2)}_6 \\ (f^{(1)}_5 - f^{(1)}_7) \cdot w_1 & = f^{(2)}_7 \end{pmatrix} \Rightarrow \begin{pmatrix} f^{(2)}_0 + f^{(2)}_1 = C'_0 \\ \cdots\cdots\cdots \quad \cdots \\ f^{(2)}_0 - f^{(2)}_1 = C'_4 \\ \cdots\cdots\cdots \quad \cdots \\ f^{(2)}_2 + f^{(2)}_3 = C'_2 \\ \cdots\cdots\cdots \quad \cdots \\ f^{(2)}_2 - f^{(2)}_3 = C'_6 \\ \cdots\cdots\cdots \quad \cdots \\ f^{(2)}_4 + f^{(2)}_5 = C'_1 \\ \cdots\cdots\cdots \quad \cdots \\ f^{(2)}_4 - f^{(2)}_5 = C'_5 \\ \cdots\cdots\cdots \quad \cdots \\ f^{(2)}_6 + f^{(2)}_7 = C'_3 \\ \cdots\cdots\cdots \quad \cdots \\ f^{(2)}_6 - f^{(2)}_7 = C'_7 \end{pmatrix}$$

Osservazioni: Analizziamo ora il numero di operazioni necessarie per calcolare tutti i C'_k:

- è necessario fare N somme per ogni passaggio;
- per un totale di $\log_2 N$ passaggi;

di conseguenza il costo complessivo dell'algoritmo è $O(N \cdot \log_2 N)$, un enorme guadagno rispetto agli $O(N^2)$ precedenti.

Si nota che con questo procedimento viene perso l'ordine dei coefficienti (es: il quinto coefficiente che calcoliamo è C'_1 invece di C'_4), per ricostruire tale ordine: si prende la rappresentazione in binario fino a k e la si riflette, come se fosse allo specchio.

Con $k = 8$ (come nell'esempio precedente) sarà così:

$$\begin{pmatrix} 0 & 000 \\ 1 & 001 \\ 2 & 010 \\ 3 & 011 \\ 4 & 100 \\ 5 & 101 \\ 6 & 110 \\ 7 & 111 \end{pmatrix} \iff \begin{pmatrix} 000 & 0 \\ 100 & 4 \\ 010 & 2 \\ 110 & 6 \\ 001 & 1 \\ 101 & 5 \\ 011 & 3 \\ 111 & 7 \end{pmatrix}$$

Attenzione: Nell'esempio precedente, i C'_k **non sono** i coefficienti della trasformata di Fourier, questo perché la suddivisione della 2.9 viene fatta solo sulla parte tra parentesi quadre, quindi:

- è necessario moltiplicare $\frac{1}{2}$ per ogni passaggio;
- avendo un totale di $\log_2 N$ passaggi, otteniamo che bisogna moltiplicare il totale per $\frac{1}{2^{\log_2 N}} = \frac{1}{N}$;

di conseguenza:

$$C_k = \frac{C'_k}{N}$$

stavolta i C_k sono esattamente i coefficienti della trasformata di Fourier.

Da notare che l'aggiunta della moltiplicazione per $\frac{1}{N}$ non cambia la complessità asintotica dell'algoritmo, questo perché:

- sono necessari $O(N \cdot log_2 N)$ passaggi per calcolare i C'_k;
- vengono fatti altri N passaggi per i C_k;
- per un totale di $O(N \cdot \log_2 N) + N = O(N \cdot \log_2 N)$.

È necessario tener presente che questo procedimento può essere applicato solo se N è una potenza di 2.

2.3 Considerazioni

Grazie alla sua bassissima complessità, la trasformata di Fourier viene implementata a livello hardware, consentendo di fare la compressione di immagini in tempo reale.

2.3.1 Ridurre la dimensione dell'immagine

Una volta applicata la Trasformata di Fourier abbiamo un elevato numero di coefficienti (uno per ogni punto), però non tutti i coefficienti sono necessari per ottenere una rappresentazione accettabile dell'immagine, infatti ce ne saranno molti con valori vicini allo 0 (es: $0,001$), la cui eliminazione non porta ad un visibile degrado dell'immagine.

In altre parole, vengono eliminati i coefficienti "inutili", riducendo drasticamente il numero di coefficienti che vengono salvati.

Il motivo per cui possiamo eliminare dei coefficienti essendo sicuri di non variare troppo il risultato della trasformata è dato dal teorema di Parseval:

$$\int T^2(x) = \sum a_k^2 + b_k^2 \tag{2.10}$$

che dice che l'energia del segnale è pari alla somma dei quadrati dei coefficienti.

Osservando la 2.10, notiamo anche che i coefficienti con un valore maggiore di 1 contribuiscono all'energia molto di più di quelli compresi tra 0 e 1, ad esempio: se abbiamo un coefficiente pari a $\frac{1}{1.000}$ il suo quadrato è $\frac{1}{1.000.000}$, quindi $\frac{1}{1.000} < \frac{1}{1.000.000} \cong 0$.

2.3.2 Trasformata di Fourier con numero di punti diverso da 2^L

In letteratura esistono trasformate di Fourier basate su potenze diverse da 2, però tali algoritmi non possono essere usati a livello hardware, quindi vengono usati solo a livello software.

Esistono diverse soluzioni al problema, implementabili via hardware, cioè:

- Aumentare la dimensione dell'immagine fino a raggiungere la potenza di 2 più vicina (ad es: aggiungendo degli 0 a fine immagine).
- Suddividere l'immagine in sezioni di dimensione pari ad una potenza di 2 e applicare la trasformata su ogni singola sezione (questo approccio è utilizzato dall'algoritmo JPEG).

2.3.3 Trasformata di Fourier nell'algoritmo JPEG

Abbiamo visto che per aumentare la velocità della trasformata di Fourier si utilizzano le formule di Eulero (2.4, 2.5), che si basano sulle proprietà dei numeri complessi, in particolare:

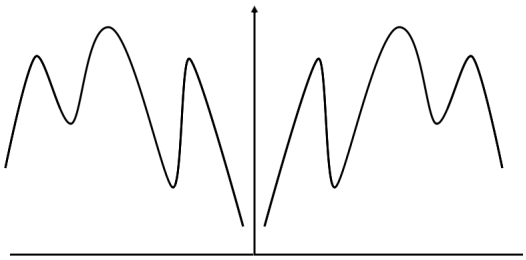

Figura 2.3: Segnale Riflesso

$$e^{ix} = \cos x + \imath \sin x$$

tuttavia, lavorare con la parte immaginaria di un numero complesso può rappresentare dei problemi, questa situazione sconveniente viene risolta dall'algoritmo JPEG utilizzando la trasformata coseno, ovvero una variante della trasformata di Fourier che utilizza solo funzioni coseno.

L'eliminazione delle funzioni seno dalla trasformata è effettuata basandosi sulle caratteristiche delle funzioni trigonometriche.

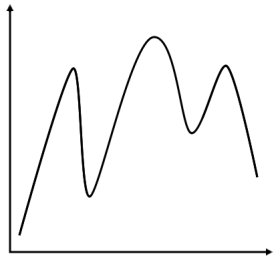

Figura 2.2: Segnale Base

Sappiamo che la funzione coseno è pari[1] ($\cos x = \cos(-x)$), mentre la funzione seno è dispari[2] ($\sin(-x) = -\sin x$), quindi il "trucco" per eliminare le funzioni seno è rendere il segnale dell'immagine simmetrico rispetto all'asse delle ordinate, ciò è ottenuto facilmente riflettendo i punti del segnale prima di eseguire la trasformata, ricapitolando:

1. ottengo il segnale (fig. 2.2);
2. rifletto a sinistra il segnale (fig. 2.3);
3. applico la trasformata di Fourier.

A questo punto siamo sicuri che la trasformata utilizzerà solo funzioni coseno dato che deve rappresentare un segnale derivante da una funzione pari.

Per quanto riguarda la dimensione dell'immagine, l'algoritmo JPEG suddivide l'immagine in blocchi da 8×8 e applica la trasformata su di essi (dopo averli riflessi), di conseguenza la dimensione di un immagine deve essere un multiplo di 8.

[1] una funzione è pari se $f(x) = f(-x)$, queste funzioni sono simmetriche rispetto all'asse delle ordinate.

[2] una funzione è dispari se $f(-x) = -f(x)$, queste funzioni sono simmetriche rispetto all'origine.

2.3.4 Problemi della Trasformata di Fourier

La trasformata è ottima per individuare le frequenze all'interno di un segnale, ma non ci dice nulla riguardo alla loro posizione, ciò la rende utilizzabile solo per segnali stazionari (segnali con frequenze periodiche che non variano con il tempo).

Il motivo è che la funzione coseno è definita su tutto l'intervallo del segnale, $[0, 1)$, quindi non può fornire nessuna informazione di localizzazione (come varia nel tempo) su di esso.

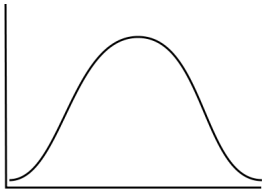

Figura 2.4: Funzione di Gauss

Una possibile soluzione a questo problema è l'utilizzo della STFT (Trasformata di Fourier Short-Time), che consiste nel considerare il segnale localmente e in seguito applicare la trasformata.

La localizzazione viene fatta moltiplicando il segnale per alcune funzioni, che possono essere (ad esempio):

- la funzione box (vale 1 in un intervallo 0 altrimenti), il problema di questa funzione è che porta ad una rappresentazione con discontinuità (fig. 2.5);
- la funzione "campana" (funzione di Gauss), fig. 2.4, in questo caso il problema è la scelta della dimensione della campana, una campana troppo stretta porta ad un'individuazione poco precisa delle frequenze, mentre una troppo larga non risolve il problema della localizzazione del segnale.

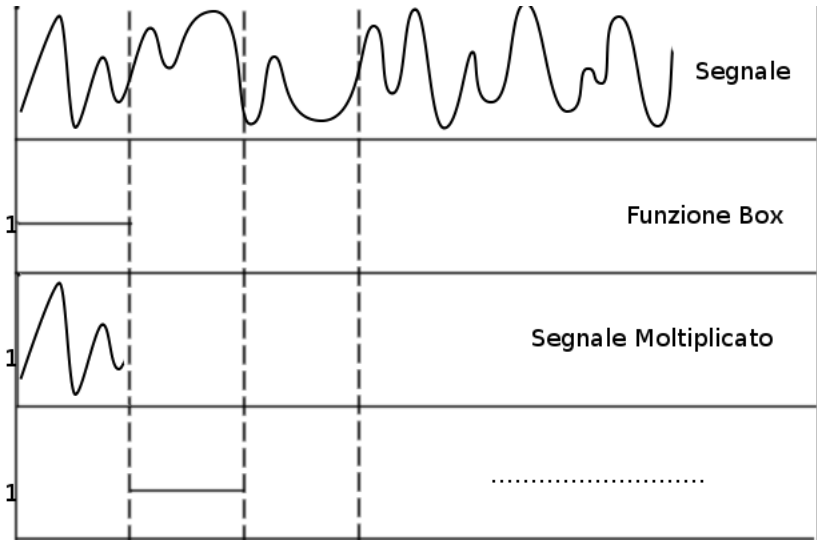

Figura 2.5: Applicazione di una funzione box

Capitolo 3

Trasformata Wavelet Discreta

La Trasformata di Fourier presenta dei problemi di localizzazione, ciò significa che per alcune immagini si ottengono dei risultati poco gradevoli, un'altra trasformata applicabile alle immagini è la Trasformata Wavelet.

La Trasformata di Fourier è una combinazione di funzioni lineari note:

$$f(x_i) = a_0\varphi_0(x_i) + a_1\varphi_1(x_i) + a_2\varphi_2(x_i) + ... + a_{n-1}\varphi_{n-1}(x_i)$$

e come funzioni base, φ_i, si utilizzano le funzioni sin e cos, ma queste non sono le uniche funzioni utilizzabili in questo campo, nel caso della Trasformata Wavelet si utilizzano le funzioni wavelet[1], queste sono delle funzioni molto interessanti che hanno la caratteristica di partire da 0, oscillare e ritornare a 0, esse possono essere sia molto localizzate, sia molto regolari.

Le funzioni wavelet vengono utilizzate tramite l'applicazione di due filtri wavelet:

- uno definito da L coefficienti h_k, $0 \leq k < L$, detto filtro passabasso, che si riferisce a una funzione di scaling;
- un'altro definito da L coefficienti g_k, $0 \leq k < L$, detto filtro passaalto, che si riferisce alla funzione wavelet e i cui elementi sono determinati a partire dagli h_k secondo la seguente relazione:

$$g_k = (-1)^k \cdot h_{L-k-1}, \qquad 0 \leq k < L;$$

3.1 Esempio Pratico

Come per la Trasformata di Fourier, rappresentiamo il segnale dell'immagine su un grafico, supponiamo di avere le seguenti coppie di punti (x, y):

$$(0,1); (1,3); (2,3); (3,2); (4,4); (5,5); (6,4); (7,3).$$

[1] In particolare, nella Trasformata Wavelet Discreta si utilizzano soprattutto le wavelet di Haar e quelle di Daubechies.

Fissato $L = 2$ possiamo scegliere di usare una funzione box, come filtro passabasso, e una wavelet di Haar (fig: 3.1.1), come filtro passaalto, con i seguenti filtri:

$$(h_0 = \frac{1}{2},\ h_1 = \frac{1}{2}) \qquad (g_0 = \frac{1}{2},\ g_1 = -\frac{1}{2})$$

3.1.1 Primo passo della trasformata

Dobbiamo applicare alle coppie di punti il filtro passabasso:

- data una generica coppia di punti $[(x_0, y_0), (x_1, y_1)]$, che rappresenta un intervallo $[x_0, x_1+1)$, l'applicazione consiste nel generare una nuova coppia di punti dove:
  $x = \frac{x_0+x_1+1}{2}$ e $y = y_0 \cdot h_0 + y_1 \cdot h_1$;
- avremo quindi:

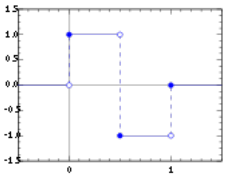

Figura 3.1: Wavelet di Haar

$(0,1)\ (1,3)$	$(2,3)\ (3,2)$	$(4,4)\ (5,5)$	$(6,4)\ (7,3)$	*valori del segnale*
$\searrow\ \swarrow$	$\searrow\ \swarrow$	$\searrow\ \swarrow$	$\searrow\ \swarrow$	
(h_0, h_1)	(h_0, h_1)	(h_0, h_1)	(h_0, h_1)	*filtro passabasso*
$\downarrow$	$\downarrow$	$\downarrow$	$\downarrow$	
$(1,2)$	$(3,\frac{5}{2})$	$(5,\frac{9}{2})$	$(7,\frac{7}{2})$	(x, c_i)

Per il filtro passaalto avremo:

- analogamente all'applicazione del filtro passabasso dobbiamo generare una nuova coppia di punti dove: $x = \frac{x_0+x_1+1}{2}$ e $y = y_0 \cdot g_0 + y_1 \cdot g_1$;
- quindi:

$(0,1)\ (1,3)$	$(2,3)\ (3,2)$	$(4,4)\ (5,5)$	$(6,4)\ (7,3)$	*valori del segnale*
$\searrow\ \swarrow$	$\searrow\ \swarrow$	$\searrow\ \swarrow$	$\searrow\ \swarrow$	
(g_0, g_1)	(g_0, g_1)	(g_0, g_1)	(g_0, g_1)	*filtro passaalto*
$\downarrow$	$\downarrow$	$\downarrow$	$\downarrow$	
$(1,-1)$	$(3,\frac{1}{2})$	$(5,-\frac{1}{2})$	$(7,\frac{1}{2})$	(x, d_i)

Osservazioni: I filtri vengono applicati solo alla y delle coppie di punti.

Il filtro passabasso genera un nuovo insieme di punti chiamati coefficienti di approssimazione o di scaling (c_i).

L'applicazione del filtro passabasso porta ad un'approssimazione del segnale di partenza, però non consente di fare il procedimento inverso, per questo motivo viene applicato anche il filtro passaalto ottenendo i coefficienti di dettaglio (d_i), che consentono di ricostruire il segnale di partenza partendo dai coefficienti di approssimazione.

In pratica, dopo questo primo passo, otteniamo:

- il segnale di partenza approssimato (c_i);
- il dettaglio perso (d_i).

Si nota che sono necessarie circa N operazioni per ogni filtro, di conseguenza il costo per il primo passo è circa: $2N = O(N)$.

L'applicazione dei filtri è detta convoluzione.

Rappresentazione Matriciale: È possibile scrivere l'applicazione dei filtri wavelet sotto forma di prodotto tra matrici, ad esempio, se si hanno 8 punti:

$$\begin{pmatrix} c_0 \\ d_0 \\ c_1 \\ d_1 \\ c_2 \\ d_2 \\ c_3 \\ d_3 \end{pmatrix} = \begin{pmatrix} h_0 & h_1 & 0 & 0 & 0 & 0 & 0 & 0 \\ g_0 & g_1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & h_0 & h_1 & 0 & 0 & 0 & 0 \\ 0 & 0 & g_0 & g_1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & h_0 & h_1 & 0 & 0 \\ 0 & 0 & 0 & 0 & g_0 & g_1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & h_0 & h_1 \\ 0 & 0 & 0 & 0 & 0 & 0 & g_0 & g_1 \end{pmatrix} \cdot \begin{pmatrix} y_0 \\ y_1 \\ y_2 \\ y_3 \\ y_4 \\ y_5 \\ y_6 \\ y_7 \end{pmatrix}$$

chiameremo la matrice che consente la convoluzione W.

La rappresentazione matriciale non è utile dal punto di vista computazionale[2], ma ci consente di notare che la trasformata inversa può essere fatta semplicemente applicando l'inversa della matrice W al vettore dei coefficienti di approssimazione e dettaglio.

Inoltre si nota che, se la matrice W è ortogonale[3], la sua inversa è uguale alla sua trasposta[4], $W^{-1} = W^T$, quindi è possibile fare il passo inverso applicando W^T ai coefficienti di scaling e di dettaglio:

$$\begin{pmatrix} y_0 \\ y_1 \\ y_2 \\ y_3 \\ y_4 \\ y_5 \\ y_6 \\ y_7 \end{pmatrix} = \begin{pmatrix} h_0 & g_0 & 0 & 0 & 0 & 0 & 0 & 0 \\ h_1 & g_1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & h_0 & g_0 & 0 & 0 & 0 & 0 \\ 0 & 0 & h_1 & g_1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & h_0 & g_0 & 0 & 0 \\ 0 & 0 & 0 & 0 & h_1 & g_1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & h_0 & g_0 \\ 0 & 0 & 0 & 0 & 0 & 0 & h_1 & g_1 \end{pmatrix} \cdot \begin{pmatrix} c_0 \\ d_0 \\ c_1 \\ d_1 \\ c_2 \\ d_2 \\ c_3 \\ d_3 \end{pmatrix}$$

Attenzione: Perché la matrice W sia ortogonale i filtri della funzione box dovrebbero essere:

$$(h_0 = \frac{1}{\sqrt{2}},\ h_1 = \frac{1}{\sqrt{2}}) \qquad (g_0 = \frac{1}{\sqrt{2}},\ g_1 = -\frac{1}{\sqrt{2}})$$

[2]Il costo di un prodotto tra matrici è $O(N^2)$, mentre abbiamo visto che è possibile fare il primo passo spendendo $O(N)$.

[3]Una matrice A è ortogonale se $A \cdot A^T = I$, quindi se e solo se le sue righe e le sue colonne formano tra loro una base ortonormale.

[4]La trasposta di una matrice A, A^T, è la matrice stessa, in cui le colonne diventano le righe e le righe diventano le colonne.

per semplificare i calcoli non sono stati usati in questo esempio.

3.1.2 Passi successivi della trasformata

Ricapitolando, al primo passo della trasformata abbiamo la seguente situazione:

$$\begin{pmatrix} c_0 & = & 2 \\ d_0 & = & -1 \\ c_1 & = & \frac{5}{2} \\ d_1 & = & -\frac{1}{2} \\ c_2 & = & \frac{9}{2} \\ d_2 & = & -\frac{1}{2} \\ c_3 & = & \frac{7}{2} \\ d_3 & = & \frac{1}{2} \end{pmatrix} \Rightarrow \; riordinando \; \Rightarrow \begin{pmatrix} c_0 & = & 2 \\ c_1 & = & \frac{5}{2} \\ c_2 & = & \frac{9}{2} \\ c_3 & = & \frac{7}{2} \\ \cdots & \cdots & \cdots \\ d_0 & = & -1 \\ d_1 & = & -\frac{1}{2} \\ d_2 & = & -\frac{1}{2} \\ d_3 & = & \frac{1}{2} \end{pmatrix}$$

Sappiamo inoltre che i coefficienti di approssimazione forniscono il segnale approssimato a un numero inferiore di punti, di conseguenza è possibile applicare nuovamente la matrice di convoluzione sui soli c_i:

$$\begin{pmatrix} c_0^{(1)} \\ d_0^{(1)} \\ c_1^{(1)} \\ d_1^{(1)} \end{pmatrix} = \begin{pmatrix} h_0 & h_1 & 0 & 0 \\ g_0 & g_1 & 0 & 0 \\ 0 & 0 & h_0 & h_1 \\ 0 & 0 & g_0 & g_1 \end{pmatrix} \cdot \begin{pmatrix} c_0 \\ c_1 \\ c_2 \\ c_3 \end{pmatrix}$$

avremo quindi:

$$\left(\begin{array}{ccc} c_0^{(1)} & = & \frac{9}{4} \\ d_0^{(1)} & = & -\frac{1}{4} \\ c_1^{(1)} & = & 4 \\ d_1^{(1)} & = & \frac{1}{2} \\ \cdots & \cdots & \cdots \\ d_0 & = & -1 \\ d_1 & = & -\frac{1}{2} \\ d_2 & = & -\frac{1}{2} \\ d_3 & = & \frac{1}{2} \end{array}\right) \Rightarrow \; riordinando \; \Rightarrow \left(\begin{array}{ccc} c_0^{(1)} & = & \frac{9}{4} \\ c_1^{(1)} & = & 4 \\ \cdots & \cdots & \cdots \\ d_0^{(1)} & = & -\frac{1}{4} \\ d_1^{(1)} & = & \frac{1}{2} \\ \cdots & \cdots & \cdots \\ d_0 & = & -1 \\ d_1 & = & -\frac{1}{2} \\ d_2 & = & -\frac{1}{2} \\ d_3 & = & \frac{1}{2} \end{array}\right)$$

Abbiamo approssimato ulteriormente il segnale, quindi possiamo applicare di nuovo la matrice di convoluzione, stavolta su due soli punti:

$$\left(\begin{array}{c} c_0^{(2)} \\ d_0^{(2)} \end{array}\right) = \left(\begin{array}{cc} h_0 & h_1 \\ g_0 & g_1 \end{array}\right) \cdot \left(\begin{array}{c} c_0^{(1)} \\ c_1^{(1)} \end{array}\right)$$

A questo punto abbiamo un unico coefficiente d'approssimazione:

$$\left(\begin{array}{ccc} c_0^{(2)} & = & \frac{25}{8} \\ \cdots & \cdots & \cdots \\ d_0^{(2)} & = & -\frac{7}{8} \\ \cdots & \cdots & \cdots \\ d_0^{(1)} & = & -\frac{1}{4} \\ d_1^{(1)} & = & \frac{1}{2} \\ \cdots & \cdots & \cdots \\ d_0 & = & -1 \\ d_1 & = & -\frac{1}{2} \\ d_2 & = & -\frac{1}{2} \\ d_3 & = & \frac{1}{2} \end{array}\right)$$

quindi possiamo fermarci.

Osservazioni: Nel complessivo i passaggi della Trasformata Wavelet sono stati:

$$\begin{pmatrix} y_0 &=& 1 \\ y_1 &=& 3 \\ y_2 &=& 3 \\ y_3 &=& 2 \\ y_4 &=& 4 \\ y_5 &=& 5 \\ y_6 &=& 4 \\ y_7 &=& 3 \end{pmatrix} \Rightarrow \begin{pmatrix} c_0 &=& 2 \\ c_1 &=& \frac{5}{2} \\ c_2 &=& \frac{9}{2} \\ c_3 &=& \frac{7}{2} \\ \cdots & \cdots & \cdots \\ d_0 &=& -1 \\ d_1 &=& -\frac{1}{2} \\ d_2 &=& -\frac{1}{2} \\ d_3 &=& \frac{1}{2} \end{pmatrix} \Rightarrow \begin{pmatrix} c_0^{(1)} &=& \frac{9}{4} \\ c_1^{(1)} &=& 4 \\ \cdots & \cdots & \cdots \\ d_0^{(1)} &=& -\frac{1}{4} \\ d_1^{(1)} &=& \frac{1}{2} \\ \cdots & \cdots & \cdots \\ d_0 &=& -1 \\ d_1 &=& -\frac{1}{2} \\ d_2 &=& -\frac{1}{2} \\ d_3 &=& \frac{1}{2} \end{pmatrix} \Rightarrow \begin{pmatrix} c_0^{(2)} &=& \frac{25}{8} \\ \cdots & \cdots & \cdots \\ d_0^{(2)} &=& -\frac{7}{8} \\ \cdots & \cdots & \cdots \\ d_0^{(1)} &=& -\frac{1}{4} \\ d_1^{(1)} &=& \frac{1}{2} \\ \cdots & \cdots & \cdots \\ d_0 &=& -1 \\ d_1 &=& -\frac{1}{2} \\ d_2 &=& -\frac{1}{2} \\ d_3 &=& \frac{1}{2} \end{pmatrix}$$

analizziamone la complessità per un numero arbitrario di punti:

- al primo passo i filtri vengono applicati su N punti;
- al secondo passo su $\frac{N}{2}$ punti;
- al terzo su $\frac{N}{4}$ e così via;
- quindi i filtri vengono applicati su $\frac{N}{2^i}$ punti, dove i indica l'i-esimo passo.

Inoltre, ad ogni passo, il numero di operazioni necessario è il doppio del numero di punti usati e vengono fatti un totale di $\log N$ passi.

Il numero di operazioni effettivo sarà:

$$\sum_{i=0}^{\log N} \frac{2N}{2^i} = 2N \cdot \sum_{i=0}^{\log N} \frac{1}{2^i} = 2N \cdot \frac{1 - \frac{1}{2^{\log N + 1}}}{1 - \frac{1}{2}} = 2N \cdot \frac{1 - (\frac{1}{2^{\log N}} \cdot \frac{1}{2})}{\frac{1}{2}} =$$

$$= 2N \cdot (1 - \frac{1}{2N}) \cdot 2 = 2N \cdot (\frac{2N-1}{2N}) \cdot 2 = 2 \cdot (2N - 1) =$$

$$= 4N - 2 = O(N)$$

quindi il calcolo della Trasformata Wavelet può essere eseguito in tempo lineare $O(N)$, che è inferiore a $O(N \cdot \log N)$.

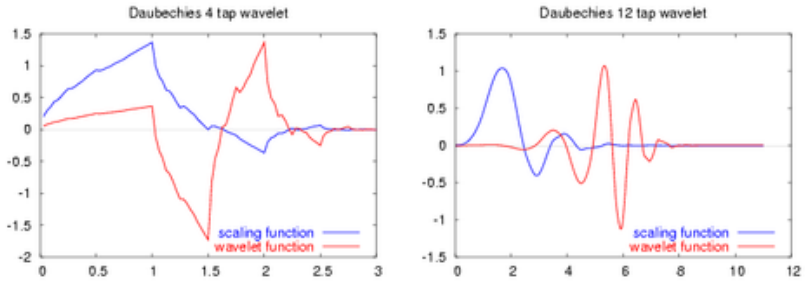

Figura 3.2: *Wavelet di Daubechies da 4 e 12 punti*

3.2 Wavelet di Daubechies

Nella pratica hanno riscosso un enorme successo, per la Trasformata Wavelet Discreta, le funzioni di Daubechies (fig: 3.2).

Esse sono definite per un numero arbitrario di punti e vengono costruite in modo che i filtri rispettino le seguenti proprietà:

$$\begin{aligned} &\sum_{k=0}^{L-1} h_k = \sqrt{2}; && \sum_{k=0}^{L-1} h_k^2 = 1; \\ &\sum_{k=0}^{L-1} g_k = 0; && \sum_{k=0}^{L-1} k \cdot g_k = 0. \end{aligned} \tag{3.1}$$

che implicano:

$$\sum_{k=0}^{L-1} h_k \cdot g_k = 0$$

garantendo un sistema ortogonale, quindi risolvibile analiticamente, senza costosi metodi numerici.

Tra le proprietà indicate nella 3.1, quelle riguardanti le g_k servono a garantire che i filtri approssimino un polinomio di grado $p = \frac{L}{2}$ e sono dette condizioni di approssimazione.

3.2.1 Convoluzione con Wavelet da 4 punti

Consideriamo un segnale da 8 punti, indichiamo con gli apici il livello di trasformata a cui i coefficienti si riferiscono:

- Livello 3:

$$\begin{pmatrix} c_0^{(2)} \\ d_0^{(2)} \\ c_1^{(2)} \\ d_1^{(2)} \\ c_2^{(2)} \\ d_2^{(2)} \\ c_3^{(2)} \\ d_3^{(2)} \end{pmatrix} = \begin{pmatrix} h_0 & h_1 & h_2 & h_3 & 0 & 0 & 0 & 0 \\ g_0 & g_1 & g_2 & g_3 & 0 & 0 & 0 & 0 \\ 0 & 0 & h_0 & h_1 & h_2 & h_3 & 0 & 0 \\ 0 & 0 & g_0 & g_1 & g_2 & g_3 & 0 & 0 \\ 0 & 0 & 0 & 0 & h_0 & h_1 & h_2 & h_3 \\ 0 & 0 & 0 & 0 & g_0 & g_1 & g_2 & g_3 \\ h_2 & h_3 & 0 & 0 & 0 & 0 & h_0 & h_1 \\ g_2 & g_3 & 0 & 0 & 0 & 0 & g_0 & g_1 \end{pmatrix} \cdot \begin{pmatrix} y_0 \\ y_1 \\ y_2 \\ y_3 \\ y_4 \\ y_5 \\ y_6 \\ y_7 \end{pmatrix} \tag{3.2}$$

$$\begin{pmatrix} c_0^{(2)} & c_1^{(2)} & c_2^{(2)} & c_3^{(2)} & \vdots & d_0^{(2)} & d_1^{(2)} & d_2^{(2)} & d_3^{(2)} \end{pmatrix}$$

- Livello 2:

$$\begin{pmatrix} c_0^{(1)} \\ d_0^{(1)} \\ c_1^{(1)} \\ d_1^{(1)} \end{pmatrix} = \begin{pmatrix} h_0 & h_1 & h_2 & h_3 \\ g_0 & g_1 & g_2 & g_3 \\ h_2 & h_3 & h_0 & h_1 \\ g_2 & g_3 & g_0 & g_1 \end{pmatrix} \cdot \begin{pmatrix} c_0^{(2)} \\ c_1^{(2)} \\ c_2^{(2)} \\ c_3^{(2)} \end{pmatrix} \tag{3.3}$$

$$\begin{pmatrix} c_0^{(1)} & c_1^{(1)} & \vdots & d_0^{(1)} & d_1^{(1)} & \vdots & d_0^{(2)} & d_1^{(2)} & d_2^{(2)} & d_3^{(2)} \end{pmatrix}$$

- Livello 1:

$$\begin{pmatrix} c_0^{(0)} \\ d_0^{(0)} \end{pmatrix} = \begin{pmatrix} 2h_0 & 2h_1 \\ 2g_0 & 2g_1 \end{pmatrix} \cdot \begin{pmatrix} c_0^{(1)} \\ c_1^{(1)} \end{pmatrix} \tag{3.4}$$

$$\begin{pmatrix} c_0^{(0)} & \vdots & d_0^{(0)} & \vdots & d_0^{(1)} & d_1^{(1)} & \vdots & d_0^{(2)} & d_1^{(2)} & d_2^{(2)} & d_3^{(2)} \end{pmatrix}$$

Osservazioni: Nel nostro esempio abbiamo supposto che il segnale sia periodico, infatti la 3.2 equivale a:

$$\begin{pmatrix} c_0^{(2)} \\ d_0^{(2)} \\ c_1^{(2)} \\ d_1^{(2)} \\ c_2^{(2)} \\ d_2^{(2)} \\ c_3^{(2)} \\ d_3^{(2)} \end{pmatrix} = \begin{pmatrix} h_0 & h_1 & h_2 & h_3 & 0 & 0 & 0 & 0 & 0 & 0 \\ g_0 & g_1 & g_2 & g_3 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & h_0 & h_1 & h_2 & h_3 & 0 & 0 & 0 & 0 \\ 0 & 0 & g_0 & g_1 & g_2 & g_3 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & h_0 & h_1 & h_2 & h_3 & 0 & 0 \\ 0 & 0 & 0 & 0 & g_0 & g_1 & g_2 & g_3 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & h_0 & h_1 & h_2 & h_3 \\ 0 & 0 & 0 & 0 & 0 & 0 & g_0 & g_1 & g_2 & g_3 \end{pmatrix} \cdot \begin{pmatrix} y_0 \\ y_1 \\ y_2 \\ y_3 \\ y_4 \\ y_5 \\ y_6 \\ y_7 \\ \cdots \\ y_0 \\ y_1 \end{pmatrix}$$

lo stesso ragionamento vale per la 3.3.

Per quanto riguarda la 3.4 sono stati utlizzati dei filtri ad-hoc, in quanto il numero di punti rimanenti è inferiore alla dimensione dei filtri.

L'aumento della dimensione dei filtri non cambia la complessità dell'algoritmo, che rimane $O(N)$.

3.3 Confronto con la Trasformata di Fourier

3.3.1 Localizzazione

Abbiamo visto che la Trasformata di Fourier non offre una localizzazione del segnale, cosa che nella Trasformata Wavelet Discreta non avviene perché:

- il segnale viene considerato localmente (come nella STFT);
- le fuzioni Wavelet hanno la caratteristica di partire da 0 e tornare a 0;
- le funzioni Wavelet sono continue.

Dato che il segnale viene considerato localmente, può esserci un problema di continuità ai bordi, ciò può essere risolto in vari modi, ad esempio:

- considerando il segnale periodico (es: 3.2);
- utilizzando dei filtri differenti solo per i bordi (es: 3.4).

3.3.2 Complessità

Come la Trasformata di Fourier, anche la Trasformata Wavelet Discreta, per la sua semplicità, può essere implementata su un hardware, consentendo di calcolarla in tempo reale.

Ricordiamo che la Trasformata di Fourier Veloce ha complessità $O(N \cdot \log N)$, mentre la trasformata Wavelet ha complessità $O(N)$, quindi è più performante.

Un'altra cosa importante da notare è che nella Trasformata Wavelet è possibile fermarsi ad ogni passo ed essere comunque in grado di ricostruire il segnale, nella Trasformata di Fourier Veloce, invece, è necessario completare tutti i passi per avere dei coefficienti validi.

3.3.3 Compressione

Analogamente alla Trasformata di Fourier, nella Trasformata Wavelet per comprimere basta eliminare i coefficienti di dettaglio "piccoli".

Capitolo 4

Trasformata Bidimensionale

Finora abbiamo visto come applicare una trasformata su una riga della matrice che rappresenta l'immagine, vediamo ora come applicarla su tutta la matrice:

- si applica la trasformata su tutte le righe della matrice, ottenendo una nuova matrice composta dai coefficienti prodotti;
- a questo punto, sulla nuova matrice, si applica la trasformata su tutte le colonne.

Questo procedimento si può estendere anche a più di due dimensioni.

In modo del tutto analogo si applica la trasformata inversa (prima sulle righe e poi sulle colonne).

Osservazioni: Supponendo una matrice con N righe e M colonne si scelgono delle basi per le x e per le y:

$$\left\{ \begin{array}{cccccc} \varphi_0^x(x_0) & \varphi_1^x(x_1) & \varphi_2^x(x_2) & \cdots & \varphi_{M-1}^x(x_{M-1}) \end{array} \right\}$$

$$\left\{ \begin{array}{cccccc} \varphi_0^y(y_0) & \varphi_1^y(y_1) & \varphi_2^y(y_2) & \cdots & \varphi_{N-1}^y(y_{N-1}) \end{array} \right\}$$

L'applicazione della trasformata prima sulle righe e poi sulle colonne produce un prodotto incrociato tra le basi:

$$\varphi_i^x(x_i) \cdot \varphi_j^y(y_j) \Rightarrow \varphi_{i,j}(x_i, y_j) \qquad i = 0, 1, \ldots, M-1;\ j = 0, 1, \ldots, N-1$$

tale prodotto è separabile, ciò significa che non è importante l'ordine con cui vengono fatte le operazioni (prima sulle righe e poi sulle colonne o viceversa), garantendo di poter fare la trasformata inversa senza preoccuparsi dell'ordine con cui sono stati eseguiti i calcoli al momento della compressione.