UNIVERSITÀ
DEGLI STUDI DI TRIESTE

Corso di Elaborazione Elettronica di Immagini II

CODIFICA DI IMMAGINI SENZA PERDITA

Gabriele Guarnieri

Sommario

Motivazione

Gli algoritmi per la codifica di immagini utilizzano tipicamente diversi metodi per ridurre la correlazione tra campioni e ridurre l'entropia dei dati

- Conversioni di spazio colore (es. RGB $\rightarrow$ YUV)
- Trasformate (es. DCT)
- Codifica differenziale ...

Alcune di queste operazioni trasformano numeri *interi* in numeri *reali*, che richiedono una *quantizzazione*
$\Rightarrow$ Codifica con perdita

Per ottenere una codifica senza perdita, è necessario operare soltanto con numeri interi o rappresentabili in virgola fissa

Esempio

Reversible Color Transform (RCT), usata nel JPEG2000

$$Y_r = \left\lfloor \frac{R + 2G + B}{4} \right\rfloor \qquad C_b = B - G \qquad C_r = R - G$$

$$G = Y - \left\lfloor \frac{C_b + C_r}{4} \right\rfloor \qquad R = C_r + G \qquad B = C_b + G$$

Codifica mediante predittori (*predictive coding*)

Metodo per ridurre la ridondanza inter-pixel. Un esempio è già stato visto: codifica differenziale

Una funzione matematica, detta *predittore*, tenta di stimare il valore di un nuovo campione a partire da alcuni di quelli già elaborati

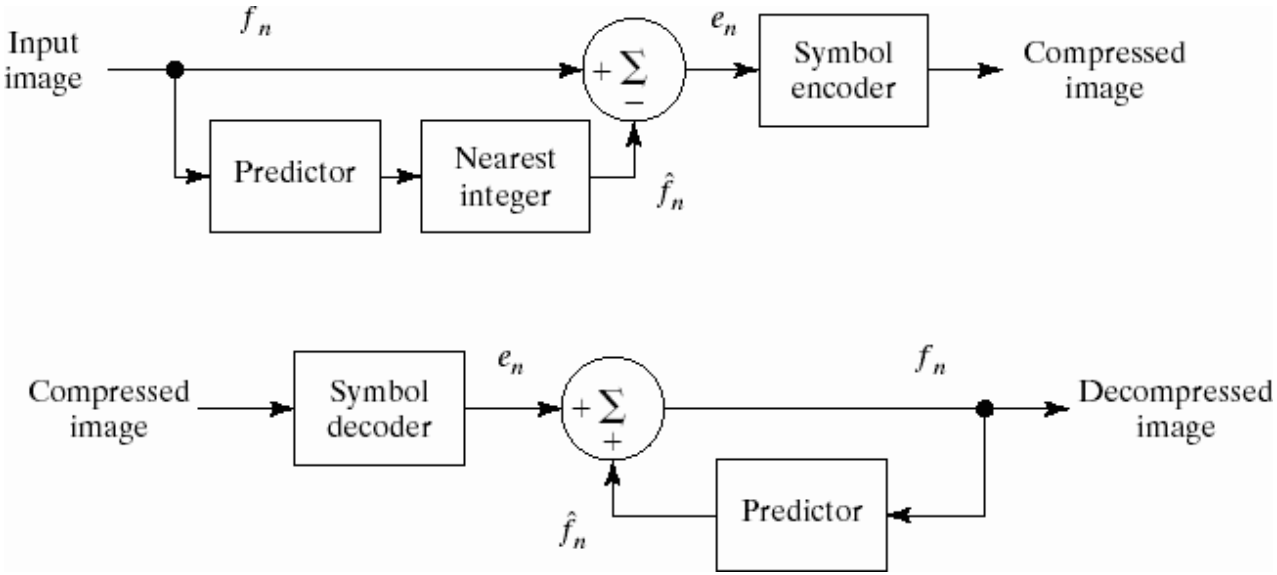
La differenza tra il valore reale del campione e il valore stimato (*prediction error*) viene codificata e trasmessa

Rispetto alla codifica mediante trasformate, è in generale

- ☹ Meno efficace
- ☺ Facilmente realizzabile in aritmetica intera o finita

Struttura

Schema a blocchi di un codificatore e decodificatore

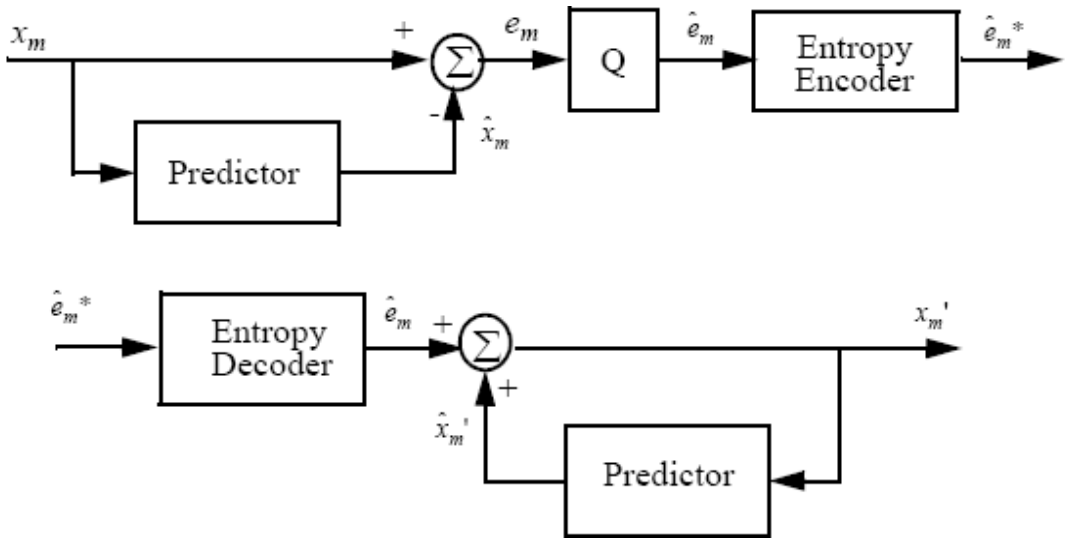


Il *symbol encoder* effettua una codifica entropica, usando ad esempio i metodi descritti in precedenza

Lossy predictive coding

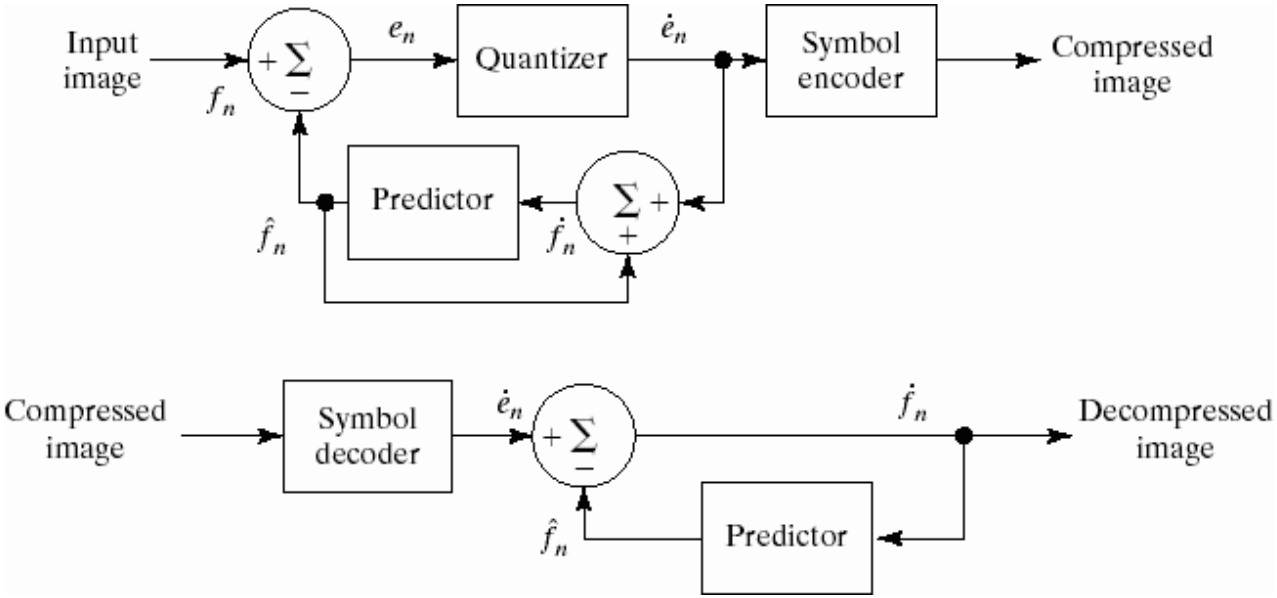
Quantizzando i valori di *prediction error*, si può ottenere una codifica con perdita ma più efficiente. Attenzione: il predittore deve usare i valori quantizzati

Metodo errato:



Lossy predictive coding

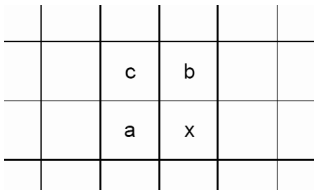
Metodo corretto:



Metodo utilizzato nella codifica video (vedremo)

Esempio: Lossless JPEG

Il *lossless mode* del JPEG può utilizzare 7 tipi di predittori *lineari* (uno per ogni *scan* nell'immagine)



Selection-value	Prediction
0	No prediction (See Annex J)
1	Px = Ra
2	Px = Rb
3	Px = Rc
4	Px = Ra + Rb – Rc
5	Px = Ra + ((Rb – Rc)/2)[a)]
6	Px = Rb + ((Ra – Rc)/2)[a)]
7	Px = (Ra + Rb)/2

[a)] Shift right arithmetic operation

Regole apposite vengono usate per i pixel ai bordi dell'immagine: R_a per la prima riga, R_b per la prima colonna, 2^{P-1} per il primo pixel

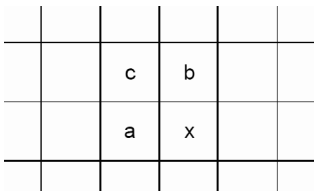
Esempio: Lossless JPEG

I valori di *prediction error* vengono codificati in un modo analogo a quello usato per i coefficienti DIFF DC

SSSS	Difference values
0	0
1	–1,1
2	–3,–2,2,3
3	–7..–4,4..7
4	–15..–8,8..15
5	–31..–16,16..31
6	–63..–32,32..63
7	–127..–64,64..127
8	–255..–128,128..255
9	–511..–256,256..511
10	–1 023..–512,512..1 023
11	–2 047..–1 024,1 024..2 047
12	–4 095..–2 048,2 048..4 095
13	–8 191..–4 096,4 096..8 191
14	–16 383..–8 192,8 192..16 383
15	–32 767..–16 384,16 384..32 767
16	32 768

Esempio: PNG

Il formato PNG può utilizzare 3 predittori lineari e un predittore non lineare (uno per ogni riga)



Filter name	**Predicted value**
None	0
Sub	A
Up	B
Average	Mean of A and B, rounded down
Paeth	Closest to $A + B - C$

I predittori sono calcolati sui byte, non sul valore dei pixel

I valori di *prediction error* vengono codificati con l'algoritmo DEFLATE (zlib, zip, gzip, . . .)

Lempel-Ziv coding e varianti

Il codice di Huffman visto prima sfrutta soltanto la diversa probabilità dei simboli e non le loro relazioni

- Ogni simbolo viene codificato separatamente dagli altri
- Il codice è costruito a partire dalle sole probabilità dei simboli

Esistono codici che sfruttano le relazioni tra i simboli, e in particolare tentano di individuare e codificare in modo efficiente le sequenze di simboli ripetute

Uno dei primi codici di questo tipo (tuttora usato con alcune varianti) è stato proposto da Lempel e Ziv nel 1977 $\rightarrow$ "LZ77"

Jacob Ziv and Abraham Lempel "A Universal Algorithm for Sequential Data Compression", IEEE Transactions on Information Theory, 23(3), pp. 337–343, May 1977.

Codifica LZ77

Il codificatore riceve in ingresso una sequenza di caratteri, che vengono elaborati uno dopo l'altro. Non è necessario conoscere la loro probabilità

Ad ogni passo, il codificatore verifica se i caratteri successivi della sequenza erano già apparsi in precedenza

Ad ogni passo, il codificatore produce 3 simboli

- Lunghezza (eventualmente nulla) della sequenza ripetuta
- Distanza della sequenza ripetuta (può sovrapporsi all'ingresso)
- Carattere seguente

Tipicamente si limita la distanza, per ridurre il numero di bit necessari per codificarla $\rightarrow$ "Sliding window"

Codifica DEFLATE/ZLIB

Esistono diverse varianti dell'algoritmo LZ77, che usano diversi metodi per codificare la terna di simboli prodotta ad ogni passo

L'algoritmo DEFLATE/ZLIB utilizza un codice di Huffman (+ extra bits), con due dizionari distinti

- Caratteri (*literal*), lunghezze, simbolo EOB
- Distanze (possono apparire solo dopo una lunghezza)

I dati vengono divisi in blocchi, di lunghezza variabile (< 65 kB), che possono essere trattati in modi diversi

- Senza compressione
- Codice di Huffman predefinito (non trasmesso)
- Codice di Huffman "su misura" (trasmesso)

Lempel-Ziv-Welch coding

Metodo di codifica basato su dizionario, usato ad esempio nei formati GIF, TIFF e PDF, coperto da brevetto (scaduto nel 2003). Miglioramento di un metodo sviluppato da Lempel e Ziv nel 1978

Il codificatore e il decodificatore costruiscono, seguendo regole comuni, un dizionario con diversi simboli

- Caratteri singoli (*literal*)
- Clear-table marker
- End-of-data marker
- Sequenze di più caratteri già incontrate in precedenza

Il dizionario viene inizializzato con i *literal* e i *marker*, le sequenze vengono aggiunte in seguito durante la codifica e decodifica

---

Terry Welch, "A Technique for High-Performance Data Compression", IEEE Computer, June 1984, p. 8–19.

Lempel-Ziv-Welch coding

Ad ogni passo il codificatore

- Cerca nel dizionario la sequenza più lunga uguale ai caratteri successivi dell'ingresso
- Emette il codice corrispondente
- Aggiunge al dizionario una nuova sequenza ottenuta concatenando la sequenza trovata prima e il carattere successivo (che non è necessario trasmettere!)

Quando il dizionario è "pieno", il codificatore emette un *clear-table marker* e ricomincia

Lempel-Ziv-Welch coding

Sono state proposte diverse tecniche per codificare i simboli del dizionario

- Lunghezza fissa (*clear-table* quando il dizionario è pieno)
- Lunghezza variabile (bit ingresso + 1, aumentati quando il dizionario è pieno, *clear-table* quando si raggiunge una lunghezza massima) $\rightarrow$ Metodo usato nel PDF

Talvolta conviene azzerare il dizionario prima che sia pieno, ad esempio se le proprietà statistiche dei dati cambiano

Run-length encoding

Rappresenta in modo compatto sequenze ("run") di valori uguali, che appaiono ad esempio in immagini sintetiche (grafici, disegni) con aree di colore uniforme

Abbiamo visto una forma particolare di RLE nello standard JPEG (codifica dei coefficienti DCT-AC). Quel metodo però ha alcune limitazioni

- Il simbolo ripetuto è fisso (0)
- Ogni coefficiente non nullo è preceduto da un simbolo RRRR

Il metodo si può generalizzare per renderlo adatto alla codifica di immagini generiche

Run-length encoding

Un codificatore RLE divide i dati in ingresso in blocchi di 2 categorie

- Sequenze di m simboli diversi ("non-run")
- Sequenze di n simboli uguali ("run")

Una sequenza *non-run* viene rappresentata con un (unico) simbolo N_m che ne indica la lunghezza, seguito dagli m simboli della sequenza

Una sequenza *run* viene rappresentata da un simbolo R_m che ne indica la lunghezza, seguito dal simbolo ripetuto

I simboli N_m e R_m si trovano in posizioni note, e quindi non è necessario delimitarli

Esempio: codifica RLE usata nel formato PDF

I dati in ingresso sono sequenze di byte. I blocchi *non-run* e *run* possono avere una lunghezza ≤ 128 byte (blocchi più lunghi vengono spezzati)

Ogni blocco è preceduto da un byte che identifica il tipo di blocco e la lunghezza

Valore (k)	Significato
0 – 127	*Non-run* lungo $k+1$ byte
129 – 255	*Run* lungo $257-k$ byte
128	*End of data*

Prestazioni limite

- *Run* di 128 byte $\Rightarrow$ 2 byte trasmessi
- *Non-run* di 128 byte $\Rightarrow$ 129 byte trasmessi