UNIVERSITÀ
DEGLI STUDI
DI PADOVA

DIPARTIMENTO DI INGEGNERIA DELL'INFORMAZIONE

CORSO DI LAUREA IN INGEGNERIA DELL'INFORMAZIONE

"COMPRESSIONE DI IMMAGINI SENZA PERDITE: JPEG-LS"

Relatore: Prof. Giancarlo Calvagno

Laureando: Sevastian Guzun

ANNO ACCADEMICO 2022 – 2023

Data di laurea 13/03/2023

Sommario

Nel processo di trasmissione delle immagini un ruolo importante è svolto dalla compressione delle immagini. Un tipo di compressione è quella senza perdite caratterizzata dal fatto che, una volta ottenuta l'immagine compressa, sia possibile ottenere l'originale senza perderne la qualità.

La compressione si articola in diversi passaggi quali il calcolo di una predizione e la successiva codifica del residuo.

Il lavoro svolto si concentra su tre tipi di codifiche: la codifica di Huffman, la codifica di Golomb e la codifica a lunghezza fissa.

Capitolo 1

Introduzione

1.1 Compressione

Lo sviluppo della tecnologia e il grande impatto della nascita di internet hanno portato alla necessità e allo sviluppo della scienza della compressione di dati. Dalla semplice chiamata telefonica alla video lezione su zoom, queste azioni che da parte nostra richiedono un click, sono tutte possibili grazie a una buona compressione di dati.

Usando il computer ci si può imbattere in suffissi come .jpeg o .mpeg, tali sono chiamati gli standard che permettono la compressione di immagini, video e audio. Algoritmi di compressione di dati sono utilizzati in questi standard per ridurre il numero di bit necessari per rappresentare le immagini, i video o gli audio.

La compressione è un modo di rappresentazione di un file in maniera compatta. La ragione per cui è indispensabile è motivata dal fatto che ogni informazione generata per via digitale richiede un enorme ammontare di dati. Per rappresentare un file audio di 2 minuti su un CD, ovvero a una frequenza di 44.1 kHz e 16 bit a campionamento, servirebbero 84.6 Mbit o 10,6 MB. I telefoni attuali vantano spesso la quantità di mega pixel della propria fotocamera. Una foto scattata da una fotocamera di 48 Mpixel, se non compressa avrebbe la necessità di 144 MB, e per condividerla sui social richiederebbe un tempo non indifferente, diversamente da quella compressa di una decina di MB.

In base alle necessità dell'applicazione che si vuole utilizzare, si possono realizzare due tipi di compressione.

La prima compressione è quella senza perdite, che come da nome, permette di avere una rappresentazione compatta senza perdere informazione. Se i file sono stati compressi in questo modo, è sempre possibile fare una ricostruzione dei file originali a partire da quelli compressi. Questo tipo di compressione è usato nelle applicazioni che non tollerano una differenza tra file originali e ricostruiti. Un'area in cui viene utilizzata tale compressione è quella della compressione del testo dove è importante avere una perfetta ricostruzione del testo originale poiché piccole differenze possono portare a significati diversi. Si consideri ad esempio la frase " Dopo chiamo *Mario*" e "Dopo chiamo *Maria*". Nel caso di immagini, un esempio importante può essere il campo medico, dove vorremmo che un esame radiologico avesse la massima qualità disponibile e non perdesse di informazioni in modo da poter evidenziare tutti i dettagli. In questo caso si può definire il *compression ratio*, ovvero un rapporto che indica il grado di compressione. Un'immagine di 256×256 pixel ha bisogno di 65.5 kB. Se la sua versione compressa è di 16.4 kB, il rapporto di compressione sarà di $4 : 1$. La percentuale di compressione invece sarà di 75% .

Il secondo tipo di compressione è quella con perdite e, al contrario di quella precedente, comporta una perdita di informazioni che non può più essere recuperata o rico-

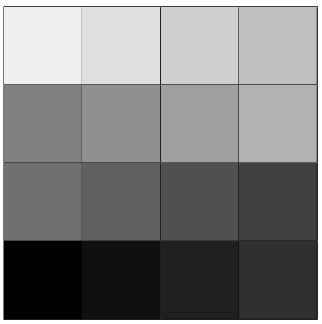


Figura 1.1: Matrice di bit

struita. D'altra parte l'introduzione di questa distorsione permette un maggiore rapporto di compressione a discapito della qualità. Questo tipo di compressione viene ad esempio usato sui social, dove le foto fatte oltre ad essere riscalate, perdono di dettagli e diventano sfocate.

La struttura di un immagine è così definita: si ha una matrice di pixel, ciascuno rappresentato da un numero binario di solito di 8 bit che identifica la luminosità di quel pixel. Un pixel di valore 255 è bianco, quelli compresi fino allo 0 saranno grigi e quello pari a 0 sarà totalmente nero. Questa immagine sarà in scala di grigi. Ad esempio la Figura 1.1. Le immagini a colori sono formate dalla sovrapposizione di tre matrici che rappresentano i colori quale il rosso, il verde e il blu.

La compressione di un'immagine si articola in diversi passi:

1. Si manipola l'immagine originale per ottenerne una sua versione alterata. Questa è una prima fase preparatoria utile per evidenziare lo spaziamento o i bordi dell'immagine. Si cerca quello che viene chiamato predizione, ovvero un modello che rappresenti l'immagine originale con meno bit.

2. Si calcola il residuo. Se la prima fase cercava un modello che assomigli ai dati dell'immagine, questa seconda fase ne evidenzia la differenza da quella originale.

3. Il residuo trovato deve essere codificato. Il tipo di codifica è il nucleo della compressione e rappresenta l'efficacia di tale algoritmo.

4. Valutare l'efficienza di una codifica. Nel caso di una senza perdite, si può calcolare il fattore di compressione e confrontare con altre tecniche.

5. Si fa un confronto delle diverse predizioni e l'uso di diverse codifiche usate su più immagini per capire il rapporto di compressione di ciascuna e sotto quali aspetti le diverse codifiche sono preferibili.

La Figura 1.2 mostra lo schema della compressione di un immagine.

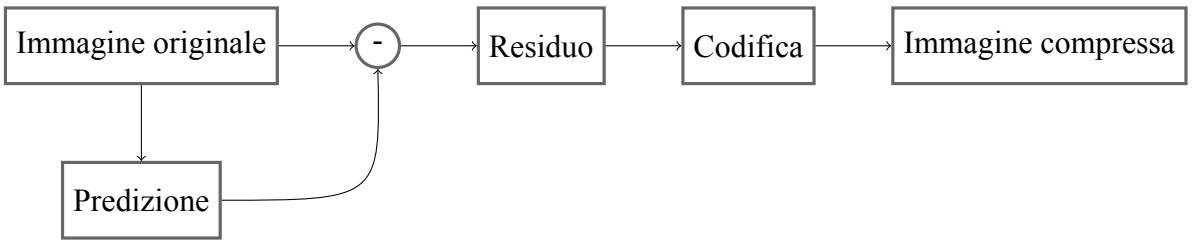


Figura 1.2: Schema della compressione di un immagine

Capitolo 2

Predizione e residuo

Il primo passo della codifica è l'analisi dei dati, ovvero la ricerca di un modello che rappresenti il più fedelmente possibile i valori dei pixel da utilizzare. La ricerca del modello serve a facilitare la compressione, dato che un'adeguata codifica risulta essere più efficiente poiché sfrutta meglio certi tipi di probabilità.

Per trovare un modello è necessario per prima cosa ridurre di dimensione la varietà del file iniziale. Quel che si fa è manipolare il file in entrata in modo da ottenere meno valori e poco dispersivi. Si calcola una predizione, in particolare il valore attuale si trova come funzione di quelli precedenti che sono già stati calcolati.

Ad esempio data la sequenza:

1	2	5	6	2	-2	0	-5	-3	-1	1	-2	3

una sua possibile predizione può essere ottenuta come l'aggiunta di un +2 al valore precedente.

0	3	4	7	8	4	0	2	-3	-1	1	3	0

La differenza tra i dati e il modello scelto (predizione) si chiama residuo. La scelta del modello permette di variare le probabilità dei valori in modo da poter utilizzare una codifica rispetto a un'altra.

Nel caso dell'esempio precedente il residuo risulta il seguente:

1	-1	1	-1	6	-6	0	-7	0	0	0	-5	3

La sequenza originale è formata da 13 valori di interi ciascuno equiprobabile, mentre il residuo trovato contiene molti più 0, 1 e -1. La loro probabilità quindi sarà maggiore, mentre l'entropia della sequenza sarà minore. Ciò permetterà di usare meno bit durante la codifica. È importante che la regola usata per trovare il residuo sia nota al decodificatore, poiché questo garantirà la perfetta ricostruzione dell'immagine.

2.1 Il primo standard JPEG sena perdite

Il Joint Photographic Experts Group (JPEG) è un gruppo appartenente alla International Organization for Standardization/ International Telecommunication Union (ISO/ITU),

comitato responsabile dello sviluppo di standard di codifica. Lo standard più conosciuto è quello con perdite, ma al momento della sua creazione il gruppo creò anche uno standard per la compressione esenza perdite.

Anche se ormai in disuso, il vecchio standard è molto utile per capire la codifica tramite predizione. La prima versione di JPEG forniva otto diverse tecniche di predizione che sono riportate di seguito. Ricordando la struttura dell'immagine come matrice, si indica l'(i, j)-esimo pixel come valore della i-esima riga e j-esima colonna. Il pixel dell'immagine originale è quindi $I(i, j)$, mentre quello predetto è $\hat{I}(i, j)$. La prima tecnica non rappresenta una vera predizione poiché il valore predetto in realtà è nullo. Da questa immagine non è possibile ottenere una predizione per cui si fa la codifica direttamente. Le successive tre tecniche sono unidimensionali mentre le restanti quattro sono bidimensionali.

$$
\begin{aligned}
&\mathbf{0.}\hat{I}(i,j) = 0\\
&\mathbf{1.}\hat{I}(i,j) = I(i-1,j)\\
&\mathbf{2.}\hat{I}(i,j) = I(i,j-1)\\
&\mathbf{3.}\hat{I}(i,j) = I(i-1,j-1)\\
&\mathbf{4.}\hat{I}(i,j) = I(i,j-1) + I(i-1,j) - I(i-1,j-1)\\
&\mathbf{5.}\hat{I}(i,j) = I(i,j-1) + (I(i-1,j) - I(i-1,j-1))/2\\
&\mathbf{6.}\hat{I}(i,j) = I(i,j-1) + (I(i,j-1) - I(i-1,j-1))/2\\
&\mathbf{7.}\hat{I}(i,j) = (I(i,j-1) + I(i-1,j))/2
\end{aligned}
$$

Differenti immagini possono avere differenti strutture che possono sfruttare meglio una di queste tecniche. Nelle applicazioni che lo permettono, si può calcolarle tutte e otto e in seguito scegliere la migliore. La tecnica usata viene poi salvata in un header di 3 bit assieme all'immagine compressa.

2.2 JPEG-LS

Al contrario della prima versione proposta da JPEG, la nuova versione di predizione per immagini è una sola di tipo bidimensionale. In un'immagine, un dato pixel generalmente contiene un valore simile a uno dei suoi vicini. Quale dei pixel vicini è simile a quello che vogliamo predire dipende dalla struttura dell'immagine. La presenza di bordi verticali o orizzontali nelle vicinanze influenza la predizione di tale pixel. La predizione può essere quindi il valore del pixel superiore o adiacente oppure una media pesata dei pixel vicini a quello di cui vogliamo calcolare la predizione. Il valore trovato sarà tanto vicino all'originale quanto i pixel adiacenti sono uniformi.

Il calcolo della predizione si fa eseguendo l'Algoritmo 1 che produrrà come uscita una matrice delle stesse dimensione dell'immagine originale. Prima di eseguire l'algoritmo, visto che questo deve essere eseguito su ogni pixel dell'immagine ovvero ogni valore della matrice a partire dalla posizione $(1, 1)$, si è creata una matrice di supporto. Se l'immagine originale ha dimensione $n \times n$ pixel, quella di supporto ha dimensione $n + 1 \times n + 1$ con la prima colonna e prima riga di valore 128 e il restate è l'immagine originale. Questo passo preliminare è necessario per garantire un corretto calcolo della prima riga e colonna dato che l'algoritmo verrà poi usato a partire dal pixel $(2, 2)$. I

NW	N
W	X

Figura 2.1: Pixel usati durante la predizione.

pixel usati durante la predizione sono riportati in Figura 2.1. In questa figura il pixel di cui volgiamo trovare la predizione è nominato X. Il pixel superiore sarà N, quello adiacente sarà W, quello sulla diagonale sarà NW. Si noti che al momento del calcolo della predizione del pixel X, tutti gli altri pixel (W, NW, W) saranno già stati predetti.

Prendiamo ora in considerazione la presenza di bordi orizzontali o verticali nella struttura dell'immagine. In presenza di bordi l'algoritmo si comporterà nel seguente modo: se c'è un bordo verticale a sinistra di X il valore più probabile scelto come predizione $\hat{X}$ sarà il pixel W mentre se c'è un bordo orizzontale al di sopra di X si sceglierà come predizione N.

if $NW \geq max(W, N)$ **then**
  $\hat{X} = max(W, N)$
else
  if $NW \leq min(W, N)$ **then**
    $\hat{X} = min(W, N)$
  else
    $\hat{X} = W + N - NW$
  end
end

Algorithm 1: Algoritmo usato per calcolare la predizione $\hat{X}$

2.3 Residuo

Dopo aver trovato la predizione, si calcola il residuo, che rappresenta lo scostamento del modello scelto dai dati originali. Il residuo E di un'immagine si calcola come:

$$E = I - \hat{I}$$

dove I è l'immagine originale e $\hat{I}$ è l'immagine trovata eseguendo la predizione. Supponiamo ora che ogni pixel sia rappresentato usando 8 bit. I pixel sia dell'immagine originale che della sua predizione avranno valori che spaziano da 0 a 255. Il residuo invece potrà avere un intervallo molto più ampio come $[-255, 255]$.

Arrivati a questo punto, dopo aver calcolato il residuo, si può valutare la probabilità di ciascun valore che sarà poi necessaria alla scelta di una codifica adeguata del residuo. I valori del residuo si possono tipicamente modellare con una variabile aleatoria geometrica bilaterale, grazie alla presenza di valori negativi. Tuttavia prima di codificare il residuo con la codifica di Golomb è necessario mappare tutti i valori i negativi in positivi per poter usare tale codifica. La variabile X che si trova è quindi detta geometrica di parametro $p \in (0, 1]$, $\mathcal{X} \sim$ G(p), con alfabeto $\mathcal{X} = \{0, 1, ...\}$ e densità discreta

$$p_X(k) = (1 - p)^k p.$$

Capitolo 3

Codifica del residuo

La codifica μ è quel meccanismo che associa un elemento di un dato insieme a un altro elemento di un altro insieme. Il codice Morse ad esempio è una associazione di ciascuna lettera dell'alfabeto a una sequenza di punti e linee. Il principio che sta alla base della codifica è quello dell'alfabeto. Nel caso delle immagini, ogni valore dei pixel presente nell'immagine viene raccolto in un insieme, alfabeto A_x, affiancato alla sua probabilità. Per ogni valore viene poi assegnato un numero in codice binario che lo rappresenta nella sua forma codificata. Ogni valore è detto parola di informazione x e ogni valore codificato è detto parola di codice y. L'insieme di tutte le parole di codice è detto alfabeto A_y .

Una volta creato l'alfabeto A_y, è possibile codificare il residuo, ovvero per ogni pixel del residuo si usa la sua codifica, e trasmetterlo come sequenza di bit. Al ricevitore ci sarà un decodificatore che riceverà le varie sequenze e avrà il compito di ricostruire il residuo codificato in precedenza e per fare ciò è necessario che l'alfabeto A_y e le regole di codifica, quale μ, siano note al codificatore e al decodificatore. Un codice è decodificabile se è possibile ricostruire la sequenza dei valori originali dalla sequenza codificata. Una condizione necessaria ma non sufficiente per garantire ciò, è che la mappa che associa ogni parola di informazione a una parola di codice sia iniettiva. Tale condizione viene violata se il decodificatore non è in grado di analizzare la sequenza di bit e di delimitare le parole di codice. Nel caso di codice a lunghezza fissa, la condizione viene garantita grazie alla mappa μ, ma nel caso di codifica a lunghezza variabile il decodificatore non conosce la lunghezza della parola di codice corrente. Una soluzione a questo problema è la cosiddetta *codifica a prefisso*, dove nessuna parola di codice $\boldsymbol{b}$ è prefisso di un'altra parola di codice $\boldsymbol{b'}$.

3.1 Codifica a lunghezza fissa

La codifica a lunghezza fissa è la più semplice e intuitiva dove ogni parola di codice ha la stessa lunghezza. Questo tipo di codifica converte tutte le parole di informazione in numero binario, la cui lunghezza delle parole sarà funzione della dimensione dell'alfabeto della sorgente. Ogni numero sarà codificato con $n = \lceil log_2(A) \rceil$ bit, dove A è dimensione dell'alfabeto della sorgente.

Un problema di questa rudimentale codifica è l'aumento della lunghezza delle parole di codice nel caso di parole di informazione con segno. Infatti per rappresentare un numero negativo in codice binario è necessario un bit che tiene conto del segno (di solito è il primo). Nel nostro caso il residuo calcolato ha anche valori negativi che possono arrivare fino a -255, il che ci obbliga ad utilizzare 9 bit invece di 8. La Tabella 3.1

riporta un esempio di codifica a lunghezza fissa usando 8 bit per rappresentare solo numeri positivi e 9 bit per rappresentare anche quelli negativi.

Parola di informazione a 8 bit	Parola di codice
0	00000000
1	00000001
2	00000010
⋮	⋮
255	11111111
Parola di informazione a 9 bit	**Parola di codice**
-255	100000001
⋮	⋮
-2	111111110
-1	111111111
0	000000000
1	000000001
2	000000010
⋮	⋮
255	111111111

Tabella 3.1: Esempio di codifica a lunghezza fissa.

Questo tipo di codifica non rappresenta una vera e propria compressione, poiché i bit necessari per rappresentare un'immagine saranno sempre gli stessi di quella originale, ovvero $256 \times 256 = 65.538$ kB.

Durante la creazione dell'alfabeto si è preferito fare una mappatura di tutti i valori in modo tale da avere solo valori positivi e poi fare la conversione in binario. La mappatura usata è stata la seguente:

- Se x è positivo, allora lo sostituisco con $2x - 1$.
- Se x è negativo, allora lo sostituisco con $2x$.

Facendo questo tipo di mappatura si ha sempre un aumento di 1 bit.

Questa codifica non ha bisogno della probabilità di ciascuna parola di informazione per costruire un codice e tutte le parole di informazione possono essere considerate come punti di una variabile aleatoria uniforme. Le prossime due codifiche sono migliori perché sfruttano la probabilità di ciascuna parola di informazione per creare un alfabeto più efficiente, infatti posso avere molti valori che usano pochi bit e pochi valori che usano molti bit.

3.2 Codifica di Huffman

Una codifica più efficiente è quella di Huffman. Questo tipo di codifica sfrutta la conoscenza di un modello, come la probabilità, per costruire un codice a prefisso che è *ottimo*. Per essere *ottimo* un codice deve soddisfare le seguenti richieste:

1. In un codice ottimo i simboli che compaiono più frequentemente (hanno maggiore probabilità di occorrenza) avranno parole di codice di lunghezza inferiore rispetto a simboli che compaiono con frequenza minore.

2. In un codice ottimo i due simboli che compaiono di meno, hanno la stessa lunghezza di parole di codice.

Per verificare il primo punto basti pensare al fatto che, se i simboli che compaiono più spesso avessero parole di codice di lunghezza maggiore rispetto a quelle parole di codice che compaiono meno ma con frequenza minore, si avrebbe una lunghezza media per parola molto maggiore che se si invertisse tale condizione. Il secondo punto viene verificato per contraddizione. Supponiamo di avere un codice ottimo. Se le due parole di minore probabilità avessero una lunghezza di codice differente, e una fosse più lunga di k bit, essendo questo un codice a prefisso, la parola di lunghezza inferiore non potrebbe essere il prefisso di quella di lunghezza maggiore e perciò le due parole sarebbero sempre differenti. Se si eliminassero gli ultimi k bit dalla parola più lunga si otterrebbe un nuovo codice con lunghezza media minore della precedente, ma ciò implica la contraddizione nella scelta iniziale di un codice ottimo.

Il codice di Huffman aggiunge anche un altro requisito. Le due parole meno frequenti, che avranno lunghezze pari, differiscono solo nell'ultimo bit. La parola di codice di probabilità inferiore avrà come ultimo bit uno **0**, quella di probabilità maggiore un **1**.

La procedura che permette di costruire un codice di Huffman è la seguente:

1. Si ordina ogni parola di informazione in base alle loro probabilità in modo decrescente. Questi sono tutti i nodi foglia.

2. Le due parole meno probabili saranno fuse in una nuova parola di probabilità la somma delle due precedenti. Questo è il nodo padre delle due foglie. Il figlio sinistro è quello di probabilità minore, quello destro di probabilità maggiore.

3. Si assegna come bit **0** al figlio sinistro e **1** al figlio destro.

Questa procedura è ricorsiva e bisognerà ripeterla finché non si arriverà ad avere un solo nodo padre e non si potrà continuare ulteriormente. Al termine della procedura si farà una visita dell'albero e si scriverà il codice di ogni parola come concatenazione di zeri e uni letti a partire dal nodo radice fino a ciascuna foglia.

Abbiamo detto che la codifica di Huffman è ottimale, ma non abbiamo detto cos'è la lunghezza media delle parole di un codice ottimale. La lunghezza di codice dipende dalla grandezza dell'alfabeto e dalla probabilità di ciascuna parola di codice. Se definiamo l_j la lunghezza della parola j-esima, possiamo trovare la lunghezza media di una parola di codice come:

$$L = \sum_j l_j \times p(x_j)$$

Vogliamo ora definire un'altra quantità chiamata *informazione*. Supponiamo di avere un evento A che rappresenta l'esito di un dato esperimento. Se $P(A)$ rappresenta la probabilità che tale evento A accada, l'informazione associata a tale evento A è data da

$$i(A) = log_b \frac{1}{P(A)} = -log_b P(A)$$

L'informazione che porta un dato evento A sarà tanto maggiore quanto è bassa la sua probabilità, questo è conseguenza della presenza del $-log$. Nel nostro caso $P(A)$ rappresenta l'occorrenza di un dato pixel e b vale 2. L'unità di misura dell'informazione perciò è il bit.

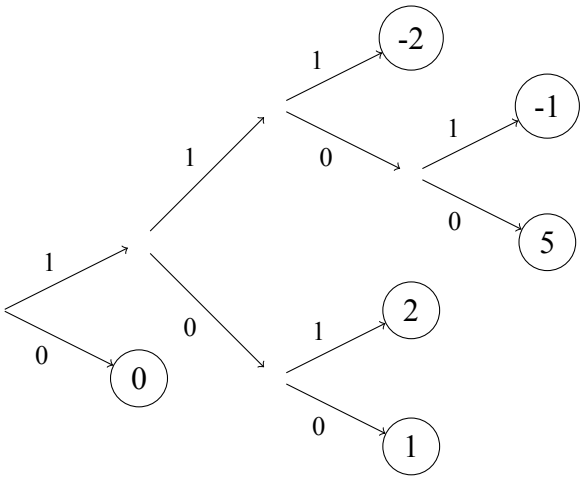
A questo punto possiamo definire un'altra grandezza che rappresenta l'Informazione media:

$$H = \sum_j P(A_j) i(A_j) = -\sum_j P(A_j) log_b P(A_j)$$

Tale grandezza viene chiamata $entropia$ associata a un esperimento e rappresenta l'informazione media associata. Anche in questo caso se b vale 2, l'entropia si misura in bit. L'entropia è una misura dei bit necessari in media per fare una codifica. Un valore di entropia alto vuol dire una informazione associata alta e per fare una codifica saranno necessari molti bit. Sarà necessario scegliere un modello che, una volta codificato, mantenga l'entropia bassa. Nel caso della codifica di Huffman la lunghezza media rispetterà la seguente disuguaglianza:

$$H \leq L < H + 1$$

Un esempio con 5 valori è riportato in Figura 3.1 affiancato alla tabella che contiene le parole di informazione, le probabilità e la loro codifica. Questo esempio mostra la situazione finale dopo aver eseguito la procedura per creare una codifica di Huffman. La lunghezza media che si ottiene per tale codifica è $L = 2.24$ bit, mentre l'entropia è $H = 2.219$ bit. La lunghezza media conferma la disuguaglianza precedente infatti $2.219 \leq 2.24 < 2.219 + 1$.



x	$p(x)$	$Codifica$
-2	0.16	111
-1	0.09	1101
5	0.05	1100
2	0.13	101
1	0.12	100
0	0.45	0

Figura 3.1: Esempio codifica di Huffman su un alfabeto di 5 parole.

3.3 Codifica di Golomb

La codifica di Golomb appartiene alla famiglia di codifiche progettata per codificare numeri interi assumendo che più grande sia un intero, minore sia la sua probabilità di occorrenza. La più semplice codifica che rispetta tale assunzione è quella del codice *unario*. Il codice unario di un intero positivo n, è la concatenazione di n uni (1) seguito da uno zero (0). Il codice di 5 è 111110, mentre il codice di un 10 è 11111111110. Il codice unario è lo stesso di quello di Huffman definito su un alfabeto semi-infinito positivo {1, 2, 3, ...} con probabilità

$$P[x] = \frac{1}{2^x}$$

Poiché il codice di Huffman è un codice ottimo, lo è anche il codice unario per un dato modello. Tuttavia il codice unario è ottimo solo sotto determinate condizioni. Un

passo successivo nella complessità di codifica sono le famiglie di codice che suddividono l'intero da codificare in due parti, rappresentando una parte con il codice unario e la seconda con un altro codice. Un esempio di tale codice è la codifica di Golomb.

La codifica di Golomb è una famiglia di codici parametrizzati da un intero $m > 0$. In questa codifica di parametro m noi rappresentiamo un intero $n > 0$ usando due numeri *q* e *r*, dove

$$q = \left\lfloor \frac{n}{m} \right\rfloor,$$

$$r = n - qm,$$

che sono rispettivamente la divisione intera di *m* su *n* e il resto di tale divisone. Il quoziente *q* può avere i valori 0, 1, 2, ... e viene rappresentato in codifica unaria. Il resto *r* può avere i valori 0, 1, 2, ..., $m - 1$. Nel nostro caso *m* è potenza di 2 e perciò usiamo $log_2 m$ bit per rappresentare il resto r.

Anche questo tipo di codifica ha un'efficienza maggiore rispetto a quella a lunghezza fissa, ma l'aspetto che la distingue è la probabilità delle parole di informazione. Questa codifica funziona bene per valori che seguono la variabile aleatoria geometrica G(p), dove

$$p_X(k) = (1 - p)^k p.$$

Anche in questo caso, prima di eseguire la codifica è necessario fare una mappatura di tutti i valori in quanto questa può essere utilizzata solo per interi positivi. La mappatura usata per la codifica di Golomb è la stessa di quella usata nella codifica a lunghezza fissa ovvero:

- Se x è positivo, allora lo sostituisco con $2x - 1$.
- Se x è negativo, allora lo sostituisco con $2x$.

Un esempio di come funziona la codifica di Golomb con parametro $m = 4$ è riportato in Tabella 3.2.

n	parola di codice	n	parola di codice
0	000	9	11001
1	001	10	11010
2	010	11	11011
3	011	12	111000
4	1000	13	111001
5	1001	14	111010
6	1010	15	111011
7	1011	16	1111000
8	11000	⋮	⋮

Tabella 3.2: Esempio di codifica di Golomb su un alfabeto di 8 valori con m = 4.

Capitolo 4

Risultati e conclusione

4.1 Primo modello JPEG

Il primo modello JPEG forniva 8 tecniche di predizione. Sono tutte tecniche basate sulla posizione di un pixel piuttosto che il suo valore. La prima tecnica JPEG-0 non è una predizione, ma usa l'immagine originale. Tutte le immagini dopo aver calcolato il residuo sono state codificate con la codifica di Huffman, poiché tale codifica non necessita di particolari probabilità per valori. Le immagini usate sono tutte di grandezza 256×256 pixel e in scala di grigi, senza codifica necessiterebbero perciò di 65.536 kB. Le immagini usate sono quelle delle Figure 4.1 e 4.2.

Tutte le immagini hanno una struttura differente cosicché non c'è una predizione migliore delle altre. Se la compressione fosse usata in applicazioni non in tempo reale, ad esempio per l'archiviazione, si potrebbe provare ciascuna predizione e sceglierne la migliore. I risultati sono riportati in Tabella 4.1.

L'immagine 4.1a è meglio compressa usando la tecnica di predizione JPEG 4. Tale predizione permette di comprimere l'immagine originale fino a 32.9 kB, con un rapporto di compressione di $1.99 : 1$ ovvero quasi del 50%. La seconda immagine 4.1b viene compressa maggiormente fino a 32.862 kB usando la tecnica di predizione JPEG 2. Anche qui il rapporto di compressione è quasi pari a $1.99 : 1$. Per la terza immagine 4.2 invece, la tecnica migliore è JPEG 1, che permette di comprimere l'immagine solo fino a 49.089 kB. Il suo rapporto di compressione è di $1.33 : 1$ e del 25%. Le prime due immagini hanno un rapporto di compressione quasi identico, tuttavia a causa della loro struttura le tecniche migliori per la perdizione sono diverse. Entrambe hanno un passaggio di pixel chiari-scuri uniforme e non ci sono bordi estremamente definiti, ma nell'immagine 4.1b i pixel scuri sono molti di più e la predizione JPEG 4 funziona molto meglio della predizione JPEG 2 monodimensionale. La terza immagine 4.2 ha molti bordi e la differenza di pixel chiari-scuri è molto più marcata.

(a) Sensin. (b) Earth.

Figura 4.1: Immagini usate con il primo modello di predizione JPEG.

Figura 4.2: Omaha.

Immagine	JPEG 0	JPEG 1	JPEG 2	JPEG 3	JPEG 4	JPEG 5	JPEG 6	JPEG 7
Sensin	60.147	41.840	37.492	44.024	**32.900**	33.806	36.250	37.040
Earth	39.252	32.992	**32.862**	35.023	34.235	33.892	33.919	33.569
Omaha	57.092	**49.089**	51.662	54.749	53.822	53.795	52.900	52.428

Tabella 4.1: kB necessari per comprimere le immagini usando le diverse tecniche di predizione e codificare il residuo.

4.2 JPEG-LS

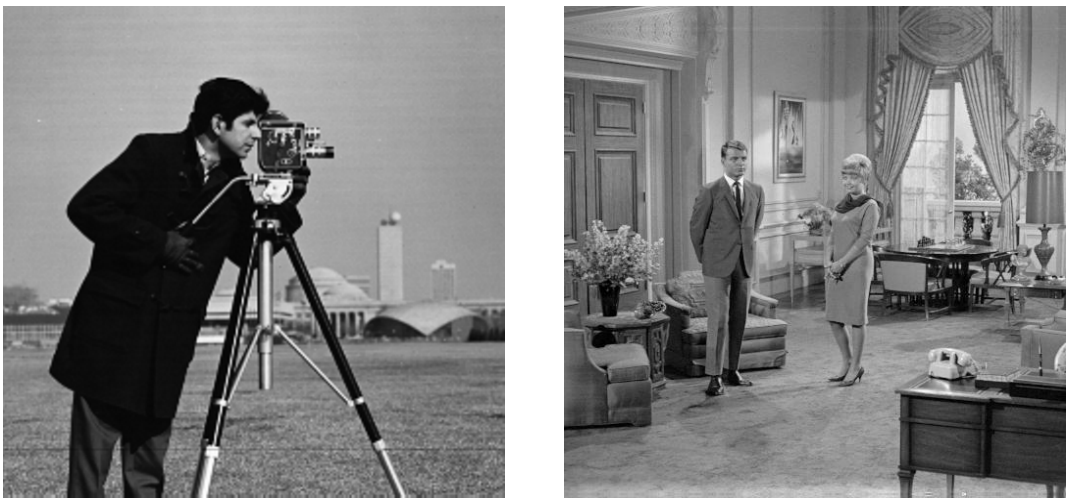
Per calcolare la predizione di JPEG-LS si è utilizzato l'Algoritmo 1 e in seguito si è calcolato il residuo. Le immagini di prova utilizzate sono quelle delle Figure 4.3 e 4.4. Queste tre immagini sono in scala di grigi a 8 bit con grandezza rispettivamente di 512×512 pixel le prime due e la terza di 1024×1024 pixel. Le prime due senza essere compresse necessiterebbero di 262.144 kB, mentre la terza di 1048.576 kB. In Tabella 4.2 sono riportati i byte necessari per ogni immagine compressa e non. In un caso si ha calcolato la predizione JPEG e successivamente il residuo che si poi è codificato con le tre codifiche. In un altro caso si è provato a codificare l'immagine originale per vedere quanto si sarebbe guadagnato in termini di efficienza.

I risultati evidenziano il fatto che la codifica di Huffman sia la più efficiente e la codifica di Golomb sia paragonabile. La codifica di Huffman può essere utilizzata anche nel caso in cui non viene fatta nessun tipo di previsione, ma in questo caso la sua efficacia sarà minore. Al contrario, la codifica di Golomb non può essere utilizzata senza nessuna predizione poiché i pixel del residuo non seguiranno la variabile aleatoria geometrica. Di questa codifica si è fatto un fit dei dati per trovare la probabilità e vedere quanto i dati si discostano dal modello. Infine, la codifica a lunghezza fissa non riesce ad eseguire una compressione, anzi in alcuni casi i valori del residuo necessitano di più bit per essere codificati. I risultati evidenziano anche la necessità di usare la predizione JPEG prima di fare la codifica del residuo per ottenere una compressione maggiore.

La prima immagine 4.3a è sfocata ma ha una buona distinzione di luminosità di pixel chiari-scuri. I bordi presenti sono poco definiti perciò il residuo non ha valori elevati. La codifica di Huffman di tale immagine permette di avere un fattore di compressione di $1.9 : 1$ ovvero del 47%. Usando la codifica di Golomb la compressione è del 33%. Facendo un fit del residuo come nella Figura 4.5 si trova la probabilità $p = 0.2215$. Senza nessuna predizione invece la codifica di Huffman permette di ottenere una compressione solo del 11%.

La seconda immagine 4.3b non ha bruschi cambi di luminosità dei pixel, tuttavia ha molti dettagli e molti bordi. Facendo la codifica del residuo con la codifica a lunghezza fissa si nota quindi che i bit necessari sono molti di più. La codifica di Huffman permette di avere una compressione del 32% oppure un fattore di compressione di $1.48 : 1$. Usando invece la codifica di Golomb si ottiene un rapporto di compressione simile, $1.48 : 1$ con una probabilità $p = 0.0757$. Il fit è riportato in Figura 4.6. Infine, senza usare nessuna predizione in questo caso si ottiene una compressione anche minore della precedente immagine, solo del 8%.

La terza immagine 4.4 è molto sfocata ed è rappresentata da quasi tutti pixel scuri. Non ci sono molti dettagli e i bordi non sono ben definiti. Si nota quindi che in questo caso la codifica di Huffman funziona anche senza nessuna predizione. Il rapporto di compressione è di $1.4 : 1$ ovvero del 29%. Usando invece la predizione e la codifica di Huffman si ottiene una compressione di $1.67 : 1$ del 40%, superiore al caso precedente. Infine, la codifica di Golomb permette una compressione intermedia tra i due casi precedenti con un rapporto di compressione del $1.52 : 1$ o del 34%. La sua probabilità è del $p = 0.0867$. Il fit è riportato in Figura 4.7.



(a) Cameraman. (b) Livingroom.

Figura 4.3: Immagini usate con la tecnica di predizione JPEG-LS.

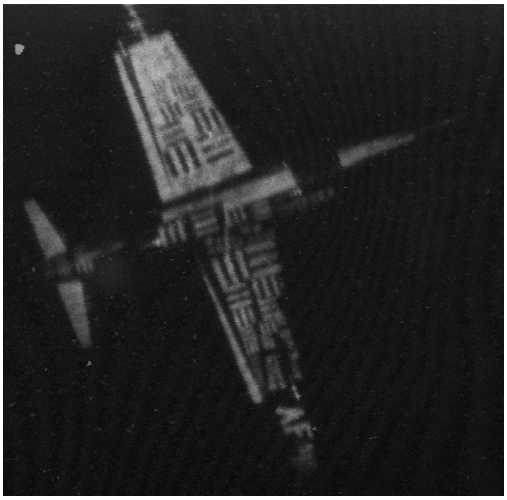


Figura 4.4: Airplane.

Immagine	Predizione	Codifica	kB
Cameraman	JPEG	Huffman	137.630
	JPEG	Lunghezza fissa	262.144
	JPEG	Golomb	174.42
	Nessuna	Huffman	232.06
	Nessuna	Lunghezza fissa	262.144
Livingroom	JPEG	Huffman	176.620
	JPEG	Lunghezza fissa	294.912
	JPEG	Golomb	183.47
	Nessuna	Huffman	240.05
	Nessuna	Lunghezza fissa	262.144
Airplane	JPEG	Huffman	624.580
	JPEG	Lunghezza fissa	1048.576
	JPEG	Golomb	687.230
	Nessuna	Huffman	744.06
	Nessuna	Lunghezza fissa	1048.576

Tabella 4.2: Byte necessari per la compressione delle tre immagini in diversi casi.

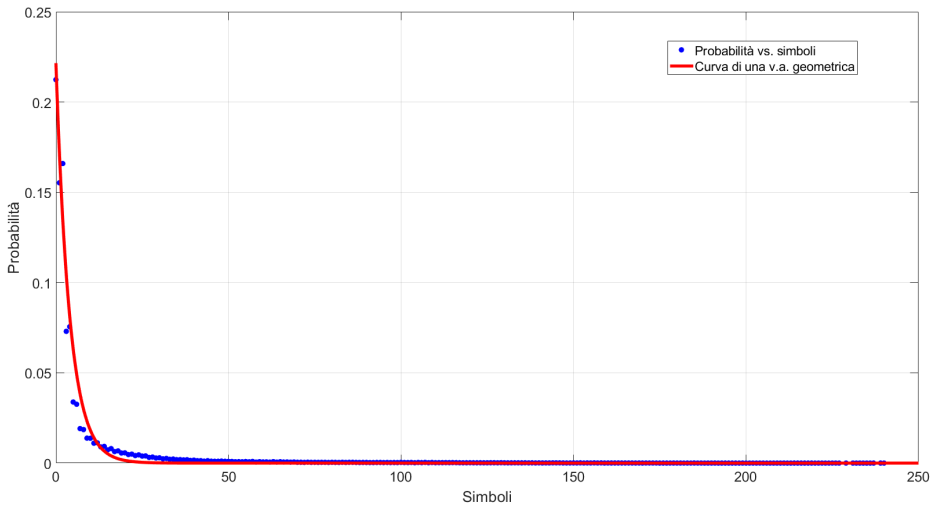


Figura 4.5: Fit eseguito sul residuo della Figura 4.3a

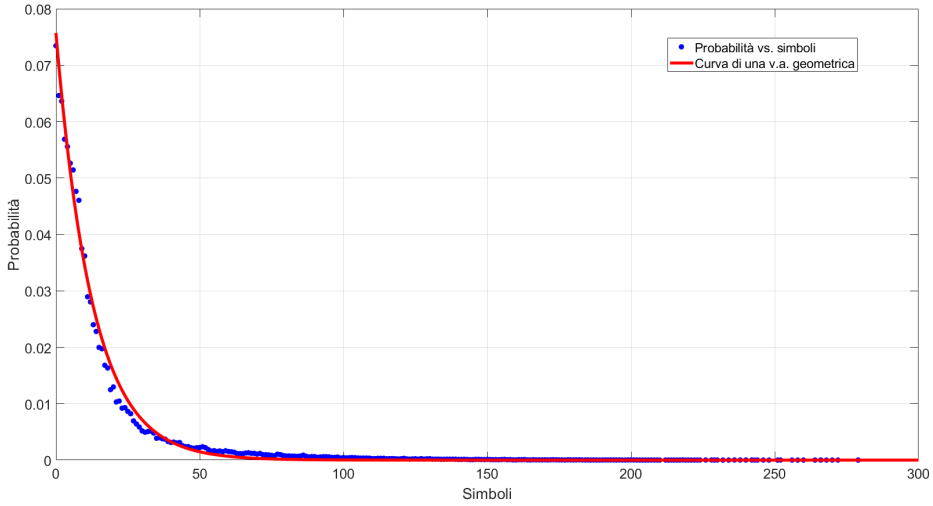


Figura 4.6: Fit eseguito sul residuo della Figura 4.3b

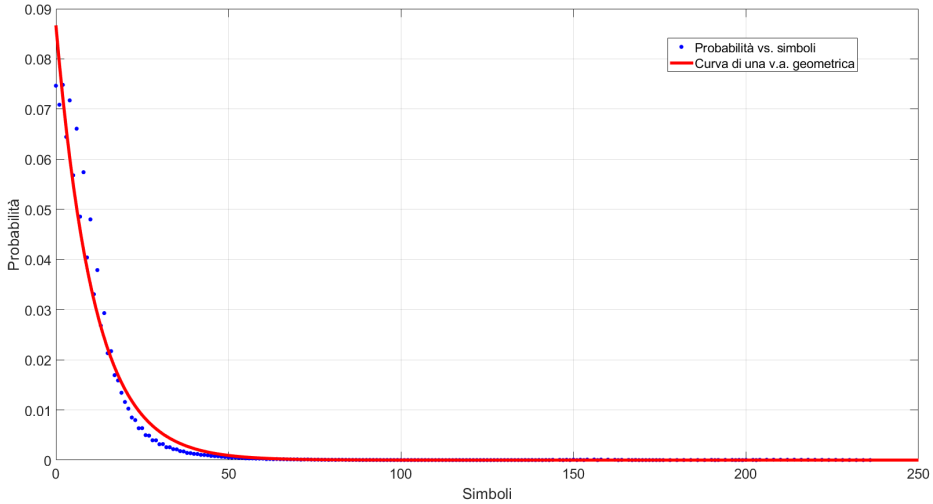


Figura 4.7: Fit eseguito sul residuo della Figura 4.4

4.3 Immagini a colori

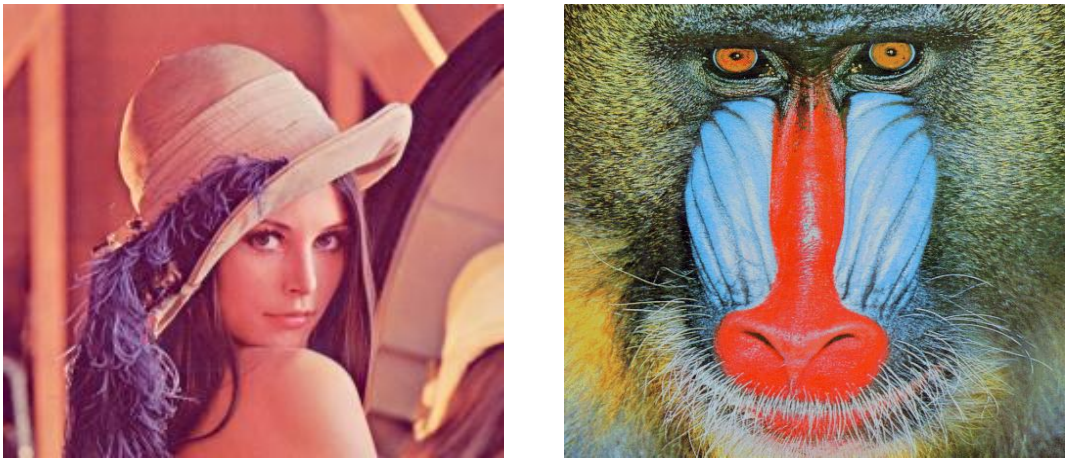
Per la compressione di immagini a colori si sono usate le immagini in Figura 4.8. Si è diviso l'immagine originale in tre piani colori e dopo aver fatto la predizione JPEG per ciascuna, le si ha codificate con la codifica di Huffman. In Tabella 4.3 sono riportati tutti i risultati con i byte necessari per codificare ciascun piano colore di ogni immagine.

La prima immagine 4.8a ha grandezza di 256×256 pixel e i byte necessari sono 196.608 kB. Invece se si esegue la compressione di Huffman del residuo su ciascun piano colore, i byte totali necessari sono 126.247 kB con un rapporto di compressione di $1.55 : 1$ e del 35%.

La seconda immagine 4.8b è grande 512×512 pixel e i byte necessari sono 786.432 kB. In questa immagine ci sono molti più dettagli o bordi pixel-pixel che influenzano la codifica. Senza codifica i byte necessari sono 786,432. Mentre usando la codifica si ottiene una compressione di $1.22 : 1$ o del 18%.

Immagine	Predizione	kB
lena-color-1	JPEG	40.963
lena-color-2	JPEG	43.776
lena-color-3	JPEG	41.508
lena-color	nessuna	196.608
mandril-color-1	JPEG	212.00
mandril-color	JPEG	208.76
mandril-color-3	JPEG	220.42
mandril-color	nessuna	786.432

Tabella 4.3: Byte usati per la compressione di ciascun piano colore per l'immagine originale.



(a) Lena (b) Mandril

Figura 4.8: Immagini a colori usate per la compressione.

4.4 Conclusione

La compressione di immagini senza perdite JPEG è utile quando si ha la necessità di diminuire i byte necessari ma non si è disposti a perderne di qualità. Tale tipo di compressione può essere utilizzato in applicazioni che non richiedo l'uso di immagini in tempo reale quali ad esempio l'archiviazione. Come si vede dai risultati, è sempre consigliabile usare la predizione JPEG anche se la compressione che si ottiene non è elevata. Infatti, nel caso di immagini piene di dettagli, il rapporto di compressione resta basso, ma questo permette di ricostruire esattamente l'immagine originale. Prendendo in esame le tre codifiche analizzate si evidenzia la codifica di Huffman come più versatile e a seguire la codifica di Golomb utilizzabile solo sotto certe condizioni. In conclusione quando si vuole comprimere immagini mediche, immagini di documenti, immagine dettagliate oppure non si vuole perde informazioni di immagini, è utile utilizzare la compressione JPEG.

Indice

Bibliografia

[1] S. Golomb. “Run-length encodings (Corresp.)” In: *IEEE Transactions on Information Theory* 12.3 (1966), pp. 399–401. DOI: 10.1109/TIT.1966.1053907.

[2] G.K. Wallace. “The JPEG still picture compression standard”. In: *IEEE Transactions on Consumer Electronics* 38.1 (1992), pp. xviii–xxxiv. DOI: 10.1109/30.125072.

[3] M.J. Weinberger, G. Seroussi e G. Sapiro. “The LOCO-I lossless image compression algorithm: principles and standardization into JPEG-LS”. In: *IEEE Transactions on Image Processing* 9.8 (2000), pp. 1309–1324. DOI: 10.1109/83.855427.

[4] Nevio Benvenuto. *Principles of communications networks and systems*. eng. 1st edition. Chichester, West Sussex, U.K. ; Wiley, 2008, 87, Geometric r.v. ISBN: 1-283-25828-5.

[5] Nevio Benvenuto. *Principles of communications networks and systems*. eng. 1st edition. Chichester, West Sussex, U.K. ; Wiley, 2008. Cap. 3.3.4, Optimal Source Coding. ISBN: 1-283-25828-5.

[6] Lorenzo Finesso. *Lezioni di probabilità / Lorenzo Finesso*. ita. 3. ed. Padova: Libreria Progetto, 2019. Cap. 4.8.1, V.a. geometriche. ISBN: 9788831901185.

[7] Khalid Sayood. *Introduction to data compression / Khalid Sayood*. eng. 4. ed. «The »Morgan Kaufmann series in multimedia information and systems. Waltham (MA): Morgan Kaufmann, ©2012. Cap. 1, Introduction. ISBN: 9780124157965.

[8] Khalid Sayood. *Introduction to data compression / Khalid Sayood*. eng. 4. ed. «The »Morgan Kaufmann series in multimedia information and systems. Waltham (MA): Morgan Kaufmann, ©2012. Cap. 3,Huffman Coding. ISBN: 9780124157965.

[9] Khalid Sayood. *Introduction to data compression / Khalid Sayood*. eng. 4. ed. «The »Morgan Kaufmann series in multimedia information and systems. Waltham (MA): Morgan Kaufmann, ©2012. Cap. 7.1-7.5, Lossless Image Compression. ISBN: 9780124157965.