

Multimedia

Homework 1

Alessandro Trigolo

30 Aprile 2024

Indice

1	Obiettivo	4
2	Codice sorgente	4
2.1	Entropia dell'immagine	4
2.2	Codifica con dizionario	6
2.3	Discussione risultati parziali	7
2.4	Codifica semplice	7
2.4.1	Analisi	8
2.5	Codifica avanzata	8
2.5.1	Analisi	10
3	Conclusioni	10

Elenco delle figure

1	Estrazione della luminanza da una immagine a colori.	5
2	Rappresentazione del modulo dell'errore di predizione nella codifica semplice.	8
3	Rappresentazione del modulo dell'errore di predizione nella codifica avanzata.	10
4	Rappresentazione della differenza tra l'errore della codifica semplice e della codifica avanzata.	11

Todo list

Descrivi decentemente l'obiettivo	4
Rifai introduzione	4
migliora risposta task 3	7
valuta se farla subsection	8
capisci cosa significa il valore EG-bpp	8
valuta se farla subsection	10

1 Obiettivo

Descrivi decentemente l'obiettivo

2 Codice sorgente

Rifai introduzione

Il linguaggio scelto per completare le richieste dell'homework è Python; all'interno del documento saranno presenti solo i punti salienti dello script, che comunque può essere ispezionato al seguente link.

2.1 Entropia dell'immagine

La prima richiesta dell'homework è divisa in due macro parti, la prima richiede di selezionare e mostrare un'immagine mentre la seconda richiesta chiede di calcolare l'entropia dell'immagine.

L'immagine scelta è un'immagine di *Spider-Man* a colori, di conseguenza è necessario estrarne la luminanza per farla diventare in bianco e nero. Dopo aver caricato l'immagine a colori con l'opportuna funzione `imread` del pacchetto `matplotlib.image` è necessario usare la funzione `cvtColor` del pacchetto `cv2` di `opencv`. Il seguente script, dopo aver eseguito le suddette operazioni si occupa di mostrare le due immagini a schermo attraverso una `subplot`.

```
1 # Prepare to load the image
2 img_file_name = "spiderman"
3 img_extension = ".jpg"
4 current_dir = os.getcwd()
5
6 # path to reach the img
7 path_to_img = os.path.join(current_dir, "multimedia", "hw-1",
8     "script", "imgs") + "/"
9
10 # loads the colored image
11 img = mpimg.imread(path_to_img + img_file_name + img_extension)
12
13 # extracts the luminance
14 gray_img = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
15
16 # creates a figure with two subplots
17 fig, axs = plt.subplots(1, 2, figsize=(12, 6))
18
19 # displays the colored image in the first subplot
20 axs[0].imshow(img, cmap='gray')
21 axs[0].set_title('Colored image')
22 axs[0].axis('off')
23
24 # displays the grayscale image in the second subplot
```

```

24 | axs[1].imshow(gray_img, cmap='gray')
25 | axs[1].set_title('Grayscale image')
26 | axs[1].axis('off')

```

Dopo aver eseguito lo script soprastante si può notare che la luminanza dell'immagine è stata estratta con successo (figura 1).



Figura 1: Estrazione della luminanza da una immagine a colori.

In secondo luogo è necessario calcolare l'entropia dell'immagine in bianco e nero. L'entropia di una variabile aleatoria X (in questo caso l'immagine) è definita come l'**informazione media** degli eventi della sorgente; l'informazione di un evento è descritta dalla funzione seguente:

$$I(X) = \log_2 \left(\frac{1}{p_i} \right)$$

Che è una variabile che diminuisce all'aumentare della probabilità dell'evento p_i . Questo è ragionevole in quanto più un evento è improbabile (quindi $p_i \rightarrow 0$) e più la sua informazione è alta ($I(X) \rightarrow +\infty$). Assumendo che gli eventi della sorgente siano indipendenti, l'informazione media si traduce nella seguente formula:

$$H(X) = E[I(X)] = \sum_{i=1}^M p_i \log_2 \left(\frac{1}{p_i} \right) = - \sum_{i=1}^M p_i \log_2 p_i$$

Dove con M si indica il numero di elementi nell'insieme X . Questa formula si riassume nel seguente script, dove la variabile contenente l'immagine in bianco e nero viene trasposta e convertita in un vettore di pixel monodimensionale. In secondo luogo attraverso la funzione `numpy.histogram` vengono contate il numero di occorrenze per ogni valore di pixel. Successivamente, per calcolare la probabilità, si divide il numero di occorrenze per il numero totale di occorrenze, escludendo eventuali valori diversi da zero. Una volta calcolare la probabilità, attraverso le funzioni `numpy.sum` e `numpy.log2` si ottiene il valore dell'entropia $H(X)$.

```

1 # flatten the transposed matrix to read pixels row by row
2 raster_scan = np.transpose(gray_img).flatten()
3
4 # count the occurrences of each pixel value
5 occurrences = np.histogram(raster_scan, bins=range(256))[0]
6
7 # calculate the relative frequencies
8 rel_freq = occurrences / np.sum(occurrences)
9
10 # remove zero-values of probability
11 p = rel_freq[rel_freq > 0]
12
13 # compute and display the entropy
14 HX = - np.sum(p * np.log2(p))
15 print(f"\nThe entropy of {img_file_name}{img_extension} is
      {HX:.3f} bpp")

```

Dopo aver eseguito lo script, l'entropia dell'immagine scelta è di **7.530 bpp**.

2.2 Codifica con dizionario

La seconda task chiede di utilizzare una compressione a dizionario, come `zip` nel caso di Windows per poi calcolare il *bitrate* risultante. Lo script necessario per soddisfare la richiesta è presentato nel frammento di codice sottostante. In particolare le prime due righe si occupano di "zippare" il file mentre le istruzioni seguenti estraggono la dimensione dell'immagine compressa (in bytes). Infine nelle ultime linee del frammento di codice viene computato l'effettivo *bitrate* dividendo la dimensione del file compresso con la dimensione dell'immagine originale, ottenuta tramite i valori di ritorno dell'attributo `img.shape`.

```

1 # zip the image
2 cmd = f"zip {path_to_img}{img_file_name}.zip
      {path_to_img}{img_file_name}.jpg"
3 os.system(cmd)
4
5 # get the zip bytes
6 img_stats = os.stat(f"{path_to_img}{img_file_name}.jpg")
7 zip_bytes = img_stats.st_size
8
9 # get img size

```

```

10 height, width = gray_img.shape
11 img_size = width * height
12
13 # get the birate
14 zip_bitrate = zip_bytes * 8 / img_size
15 print(f"\nThe bitrate of {img_file_name}.zip is {zip_bitrate:.3f}
    bpp\n")

```

Dunque, dopo aver eseguito lo script si ottiene il valore del bitrate, corrispondente a **1.340 bpp**.

migliora risposta task
3

2.3 Discussione risultati parziali

Il risultato trovato calcolando il bitrate della codifica con dizionario (1.340 bpp) è molto più basso rispetto al valore dell'entropia $H(X)$, calcolato nel primo punto (7.530 bpp), suggerendo che quindi la codifica con dizionario risulta efficace nel ridurre la quantità di informazione necessaria per rappresentare i dati, sfruttando le ridondanze presenti nel segnale.

2.4 Codifica semplice

La quarta richiesta dell'homework è quella di effettuare una codifica predittiva *semplificata*. In particolare, data l'immagine (che chiameremo $x(n)$), la codifica predittiva $y(n)$ è definita come segue:

$$y(n) = \begin{cases} x(n) - 128 & \text{se } n = 0 \\ x(n) - x(n - 1) & \text{altrimenti} \end{cases}$$

La suddetta codifica predittiva si traduce nel seguente Python script il quale, dopo aver calcolato la predizione, mostra l'immagine codificata.

```

1 simple_coding_error = raster_scan[0] - 128
2 simple_coding_error = np.append(simple_coding_error,
3                                 np.diff(raster_scan))
4
5 # plot error graph
6 plt.figure()
7 plt.imshow(np.transpose(np.reshape(np.abs(simple_coding_error),
8                                (width, height))), cmap = 'seismic')
9 plt.axis('image')
10 plt.axis('off')
11 plt.colorbar()
12 plt.title('Simple coding prediction error')

```

Dopo aver eseguito lo script sopra citato si ottiene l'immagine mostrata nella figura 2. Dall'immagine si può notare che le zone più rosse, ovvero le zone in cui il predittore ha fatto più errori, sono le zone dei contorni, come la skyline della città oppure lo

stacco tra il personaggio raffigurato e il cielo. Questo perchè la differenza dei colori tra una sezione e l'altra è particolarmente accentuata. D'altro canto, le zone colorate di blu sono le zone dove i colori sono più uniformi: ecco quindi che i palazzi e il cielo sono per lo più dello stesso colore, suggerendo una zona dove la variazione di colore - e quindi di informazione - è molto bassa.

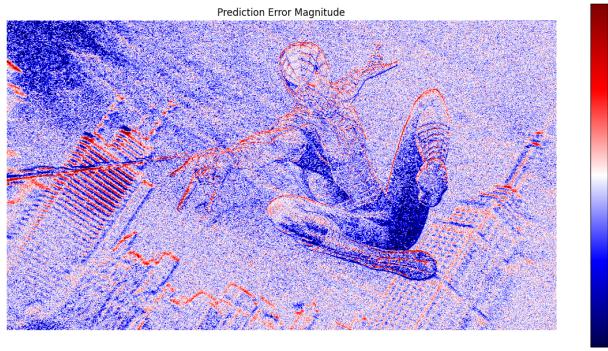


Figura 2: Rappresentazione del modulo dell'errore di predizione nella codifica semplice.

2.4.1 Analisi

Seguedo le direttive della task numero 5 dell'homework, calcoliamo

```

1 bit_count = 0
2 for symbol in simple_coding_error:
3     codeword = exp_golomb_signed(symbol)
4     bit_count += len(codeword)
5
6 exp_golomb_bpp = bit_count / img_size
7 print(f"The S-EG coding rate on prediction error is
     {exp_golomb_bpp:.4f}\n")

```

valuta se farla subsection

capisci cosa significa il valore EG-bpp

2.5 Codifica avanzata

La penultima richiesta, esige di creare un nuovo tipo di codifica, detta *avanzata* definita come segue:

$$y(n, m) = \begin{cases} x(n, m) - 128 & \text{se } n = m = 0 \\ x(n, m - 1) & \text{se } n = 0 \\ x(n - 1, m) & \text{se } m = 0 \\ \text{med}[x(n, m - 1), x(n - 1, m), x(n - 1, m - 1)] & \text{se } m = \text{MAX} \\ \text{med}[x(n, m - 1), x(n - 1, m), x(n - 1, m + 1)] & \text{altrimenti} \end{cases}$$

Dove in questo caso l'immagine anziche essere vista come un vettore x è vista come una matrice di dimensione $n \times m$. Le richieste della codifica sono tradotte in uno script contenente due cicli annidati - il primo che itera lungo le righe della matrice e il secondo che itera lungo le colonne - dove allintero sono presenti una serie di if statement che soddisfano la definizione di codifica avanzata precedentemente citata. Inoltre si osserva la mediana dei tre valori è calcolata tramite una funzione definita dall'utente, descritta anch'essa all'interno dello script.

```

1 # median function for advanced coding
2 def median(a, b, c):
3     vector = [a, b, c]
4     vector.remove(min(vector))
5     vector.remove(max(vector))
6
7     return vector[0]
8
9
10
11 # blank image
12 predicted_img = np.zeros_like(gray_img)
13
14 # iterates through the rows (height)
15 for row in range(height):
16
17     # iterates through the cols (width)
18     for col in range(width - 1):
19
20         if row == 0 and col == 0: # first pixel
21             predicted_img[row][col] = gray_img[row][col] - 128
22
23         elif row == 1:           # first row
24             predicted_img[row][col] = gray_img[row][col - 1]
25
26         elif col == 1:          # first col
27             predicted_img[row][col] = gray_img[row - 1][col]
28
29         elif col == (width - 1): # last col
30             predicted_img[row][col] = median(gray_img[row - 1][col], gray_img[row][col - 1], gray_img[row - 1][col - 1])
31

```

```

32     else:                      # other cases
33         predicted_img[row][col] = median(gray_img[row -
34                                         1][col], gray_img[row][col - 1], gray_img[row -
35                                         1][col + 1])

```

Dopo aver eseguito lo script è possibile mostrare l'errore di predizione sottraendo l'immagine predetta `predicted_img` con l'immagine iniziale `img` e "pottando" il suo valore assoluto come segue mediante uno script simile a quello mostrato per la codifica semplice (2.4). L'immagine mostrata in figura 3 è ciò che risulta degli errori commessi durante la predizione.

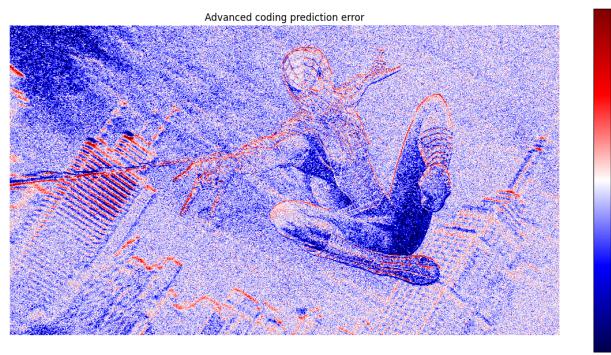


Figura 3: Rappresentazione del modulo dell'errore di predizione nella codifica avanzata.

Si osserva che per quanto l'immagine 3 sia simile alla figura 2, è presente una differenza: è sufficiente mostrare un'immagine che contenga la differenza in modulo tra le variabili `adv_coding_error` e `simple_coding_error`. Dal risultato, mostrato nella figura 4, si può notare che non è di colore uniforme ma presenta diverse sfumature di blu, suggerendo quindi una bassa differenza tra le due che quindi le rende non identiche.

2.5.1 Analisi

valuta se farla subsection

3 Conclusioni

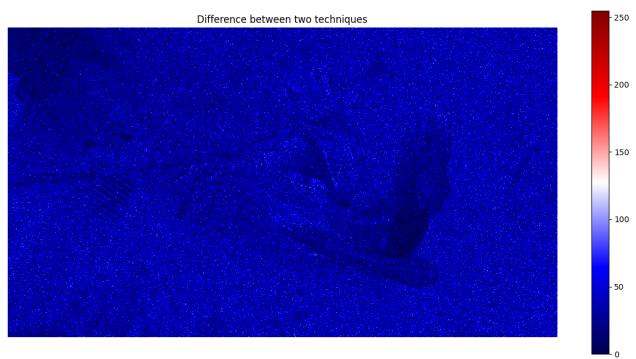


Figura 4: Rappresentazione della differenza tra l'errore della codifica semplice e della codifica avazata.