

# Multimedia

## Homework 1

Alessandro Trigo

30 Aprile 2024

## 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](#).

### Task 1

La prima richiesta dell'homework era divisa in due macro parti, la prima era quella di selezionare e mostrare un'immagine mentre la seconda richiesta chiedeva 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 binario 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
25 axs[1].imshow(gray_img, cmap='gray')
26 axs[1].set_title('Grayscale image')
27 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 di un'immagine, ma in genere di qualsiasi tipo di sorgente (o meglio, variabile aleatoria  $X$ ), è un numero che indica l'**informazione media** ed è definita come segue:

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

Dove con  $M$  si indica il numero di elementi nell'insieme  $X$  e con  $p_i$  si indica la probabilità di accadere dell'elemento  $i$ -esimo della variabile aleatoria  $X$ . Questa formula si riassume nel seguente script, dove l'immagine viene trasportata e convertita in un vettore di pixel monodimensionale. In secondo luogo attraverso la funzione `numpy.histogram` si contano il numero di occorrenze per ogni valore di pixel. Infine, per calcolare la probabilità, si divide il numero di occorrenze per il numero totale di occorrenze, escludendo eventuali valori diversi da zero.

```
1 # flatten the transposed matrix to read pixels row by row
2 rasterScan = np.transpose(gray_img).flatten()
3
4 # count the occurrences of each pixel value
5 occurrences = np.histogram(rasterScan, 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} is {HX:.3f} bpp\n")

```

Dopo aver eseguito lo script si ottiene che l'entropia dell'immagine scelta è di circa **7.530 bpp**.

## Task 2

La seconda task richiedeva 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 seguenti si occupano di ottenere la dimensione dell'immagine compressa. Infine nelle ultime righe si calcola l'effettivo *bitrate* dividendo la dimensione del file compresso con la dimensione dell'immagine originale, ottenuta mediante l'attributo **size**.

```

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

```

Dopo aver eseguito lo script si ottiene quindi il valore del bitrate che corrisponde a **1.340 bpp**, che è molto più basso al valore dell'entropia  $H(X)$  osservato nel punto precedente.

## Task 3

completa in  
base al risul-  
tato della 2

## Task 4

La quarta richiesta dell'homework è quella di effettuare una codifica predittiva *semplice*. In altre parole,

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

Dopo aver eseguito lo script soprastante si ottiene un'immagine simile alla 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 perché il distacco dei colori tra una sezione e l'altra è estremamente differente. 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 poca entropia.

sistema paragrafo

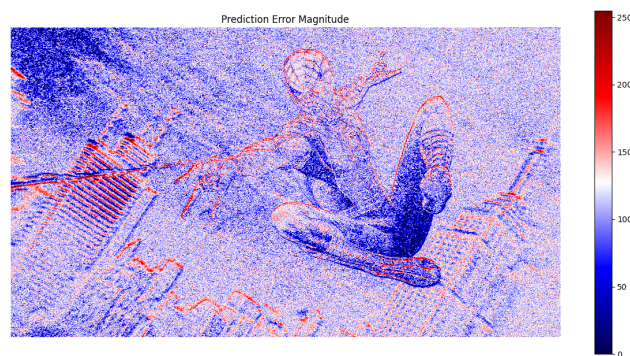


Figura 2: Rappresentazione del modulo dell'errore di predizione.

```
1 # count the occurrences of each prediction error value
2 occ, _ = np.histogram(pred_err, bins = range(-255, 256))
3
4 # calculate the relative frequencies and remove any probability == 0
5 freqRel = occ / np.sum(occ)
6 p = freqRel[freqRel > 0]
7
```

```
8  # calculate the entropy
9  HY = -np.sum(p * np.log2(1/p))
10 print(f"The entropy of the prediction error of {img_file_name} is
    {HY:.3f} bpp")
11 print(f"The compression ratio of {img_file_name} is {8/HX:.4f}\n")
```

capisci se è  
da inserire o  
meno

Task 5

### 3 Conclusioni