

DAISGram (v.1.0)

Scopo:

Realizzare una libreria in linguaggio C++ per effettuare operazioni di elaborazione di immagini (image processing).

Il progetto è diviso in tre parti (consegnate in un'unica soluzione):

- la prima parte in cui dovete implementare un insieme di metodi per la gestione di matrici a 3 dimensioni utile per comprendere alcune operazioni matematiche e di gestione della memoria.
- nella seconda parte dovete implementare alcuni semplici metodi per l'elaborazione di immagini (conversione da colore a scala di grigi, blending).
- nell'ultima parte vi verrà richiesto di implementare l'algoritmo di convoluzione a cui applicherete filtri per ottenere delle trasformazioni dell'immagine.

Per ottenere il massimo punteggio nel laboratorio dovete implementare correttamente tutti i metodi presenti nei file *Tensor.h* e *DAISGram.h*

Sono previste due parti opzionali (*greenscreen* ed *equalize*) che comunque vi esortiamo a realizzare.

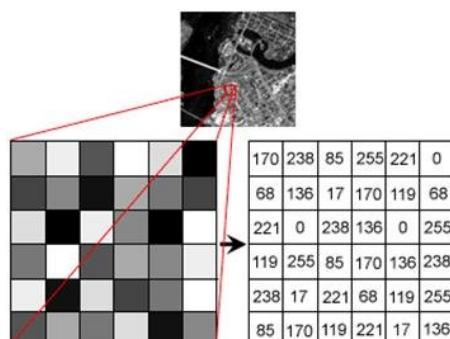
Introduzione al progetto

Le immagini e il computer

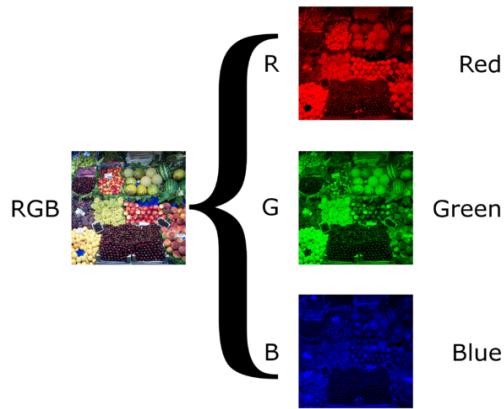
Le immagini sono codificate nel computer tramite griglie ordinate di pixel. Un pixel è l'entità più piccola per descrivere il colore di un'immagine in un determinato punto.

I pixel di un'immagine tipicamente hanno valori compresi tra 0 e 255, pertanto saranno necessari 8 bit per rappresentare un pixel.

Le immagini sono rappresentate in memoria tramite matrici (l'ordine dei pixel è fondamentale). Nel caso di immagini in scala di grigi abbiamo solo un canale (il canale di luminosità).

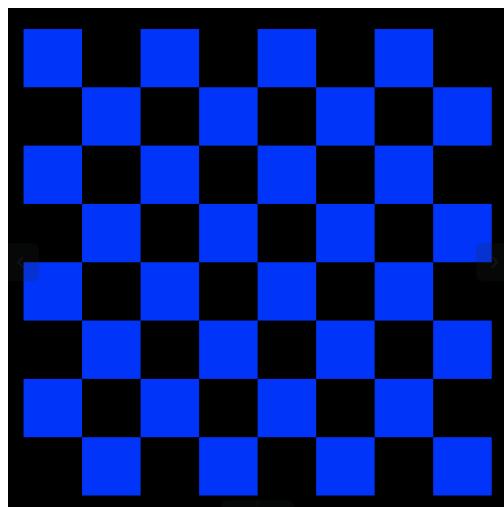


Nel caso di immagini a colori (il nostro caso), per rappresentare i colori visibili vengono utilizzati 3 canali (rosso, verde e blu). Ogni pixel quindi è composto da 3 valori interi (rosso, verde e blu). Pertanto la matrice avrà tre dimensioni: larghezza x altezza x 3. Tipicamente si parla di immagini a 24 bit (8 bit * 3 canali = 24 bit).



La libreria in C++ per leggere e scrivere immagini in formato BITMAP vi viene fornita nel repository del progetto.

Eseguite il comando “**make testbmp**”. Verrà generato un file eseguibile “**test_bmplib**” che, una volta eseguito, creerà una scacchiera colorata.



Bene, siete riusciti a creare delle immagini col vostro PC! (altrimenti bussate sul forum).

Il progetto

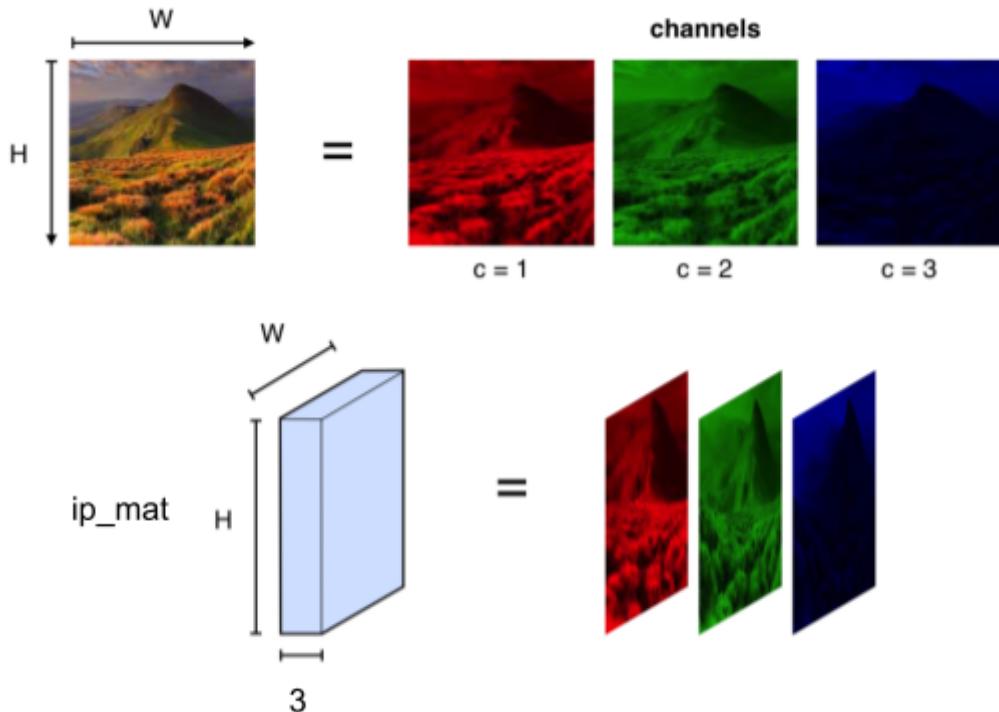
Image Processing

L’image processing, o elaborazione delle immagini, è una branca della computer science nata negli anni ‘60 con il fine di applicare delle trasformazioni alle immagini più o meno in modo automatico. Applicazioni tipiche sono la rimozione del rumore, l’equalizzazione, la conversione di formati, scalatura, ritaglio o il rilevamento di bordi. In sostanza le operazioni basilari che trovate all’interno di Photoshop o GIMP.

PARTE 1: strutture dati, operazioni matematiche e gestione memoria

Per le operazioni successive dobbiamo realizzare uno nuovo tipo di dato chiamato **tensor**. Il tipo **tensor** altro non è che una matrice a 3 dimensioni (altezza x larghezza x canali, vedi figura sotto) in cui saranno memorizzati i pixel dell’immagine e sui quali applicheremo varie trasformazioni. Nella conversione da *Bitmap* a *Tensor*, ai pixel viene fatto un casting a *float*

in quanto le operazioni di elaborazione avranno bisogno di lavorare con dati in virgola, positivi e negativi. Dovremo quindi abbandonare i pixel [0,255] per un po'.



Ci appoggeremo alla seguente classe, che dovrete implementare:

```
class Tensor{
private:
    float * data;
    int c;
    int r;
    int d;
public:
    Tensor();
    ~Tensor();
. . .
}
```

La matrice a tre dimensioni sarà memorizzata nella variabile *data* e il suo accesso avverrà tramite overloading dell'operatore () passando *tre indici*: *riga*, *colonna* e *profondità*.

Vi viene lasciata libertà sulla struttura di memoria (la variabile *data* della struttura di cui sopra) purchè **NON** utilizziate oggetti Vector... in sostanza dovete gestire voi la memoria.

Vanno implementati gli operatori matematici (+ - * /), di selezione (i ,j , k) e quello di assegnamento (=)

Vedete il file *Tensor.h* per maggiori dettagli.

PARTE 2: Operazioni semplici con immagini

Le operazioni su immagini vengono eseguite per mezzo della classe *DAISGram* che contiene all'interno un attributo *data* di tipo *Tensor*.

Dovrete implementare i seguenti metodi (maggiori dettagli in DAISGram.h):

Image Brightening:

Aumenta il valore di tutti i pixel di una costante *bright*. L'effetto è alzare la luminosità complessiva dell'immagine.

```
DAISGram brighten(float bright)
```



Conversione a scala di grigi:

```
DAISGram grayscale();
```

Converte un'immagine a colori in una a scala di grigi



Image Blending:

```
DAISGram blend(DAISGram rhs, float alpha);
```

Fonde due immagini insieme tramite combinazione convessa.

Blend = $\alpha * A + (1-\alpha) * B;$

alpha è in [0,1]

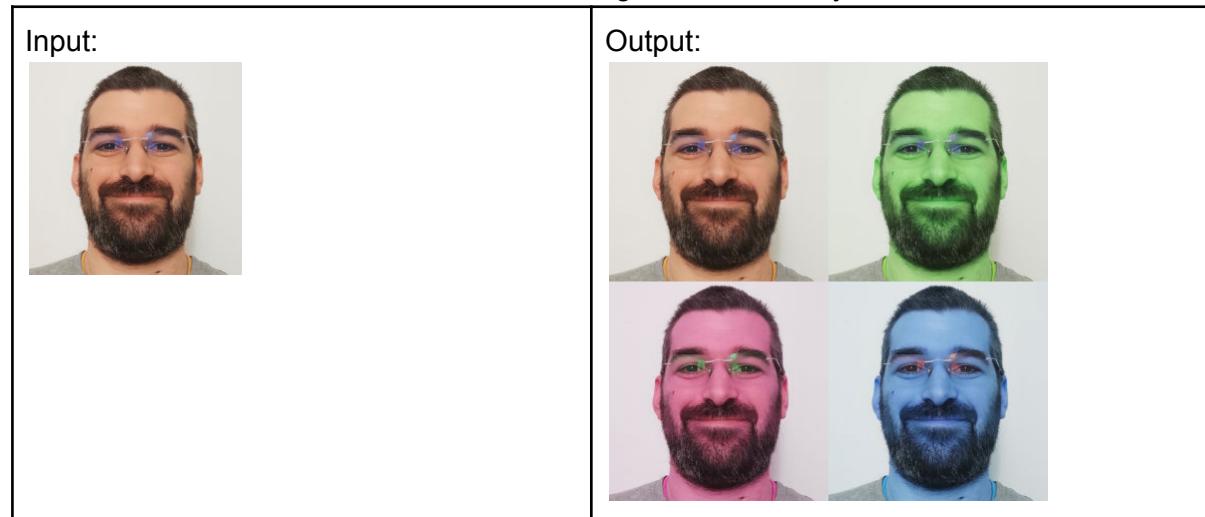
Le immagini "a" e "b" devono avere le stesse dimensioni, altrimenti l'operazione non è possibile (usate le eccezioni).

Immagine A	Immagine B	Blend (alpha = 0)	Blend (alpha = 0.25)	Blend (alpha = 0.5)	Blend (alpha = 0.75)	Blend (alpha = 1)
						

Andy Warhol:

```
DAISGram warhol();
```

Crea una nuova DAISGram contenente l'immagine in stile “Andy Warhol”.



L'immagine risultante è una composizione di 4 immagini così realizzate:

- In alto a sinistra viene replicata l'immagine originale
- In alto a destra, a partire dall'immagine originale, viene invertito il canale Rosso con il canale Verde
- In basso a sinistra, a partire dall'immagine originale, viene invertito il canale Verde con il canale Blu
- In basso a destra, a partire dall'immagine originale, viene invertito il canale Rosso con il canale Blu

L'immagine finale avrà quindi dimensione doppia dell'originale (a parte nel numero di canali) in quanto le 4 immagini saranno concatenate per formarne una unica come da esempio sopra.

PARTE 3: Convoluzione e filtraggio

La convoluzione è una delle tecniche di analisi di segnali maggiormente applicate, ed è inoltre largamente applicata nel contesto di immagini. E' uno dei metodi di processamento di immagini più semplici ma potenti e rappresenta uno dei fondamenti della visione artificiale, nell'analisi di segnali, nelle moderne reti di intelligenza artificiale e nelle reti profonde (deep learning).

La convoluzione, non è altro che una somma pesata dei valori dell'immagine rispetto a quelli di un filtro, chiamato *kernel*. Il kernel è una matrice di valori reali (tipicamente sono matrici quadrate a dimensioni dispari non molto grandi, come una 3x3 o 5x5) che definiscono “l'importanza” dei pixel sottostanti. In funzione della configurazione dei valori del filtro abbiamo diversi effetti sull'immagine di input.

Il kernel abbiamo detto essere una matrice di *float*, possiamo quindi utilizzare la stessa struttura *Tensor* anche per memorizzare il filtro.

Come funziona al lato pratico? prendiamo ad esempio la figura qui sotto. Abbiamo un'immagine 5x5x1 e un filtro 3x3x1:

7	2	3	3	8
4	5	3	8	4
3	3	2	8	4
2	8	7	2	7
5	4	4	5	4

*

1	0	-1
1	0	-1
1	0	-1

=

6		

$7 \times 1 + 4 \times 1 + 3 \times 1 +$
 $2 \times 0 + 5 \times 0 + 3 \times 0 +$
 $3 \times -1 + 3 \times -1 + 2 \times -1$
 $= 6$

Fig 1: esempio di calcolo della convoluzione tra immagine (sinistra), filtro (centro). Il risultato è a destra.

Sovrapponiamo il kernel all'immagine in alto a sinistra (prima posizione) e calcoliamo la convoluzione (somma dei prodotti):

7	2	3	3	8
4	5	3	8	4
3	3	2	8	4
2	8	7	2	7
5	4	4	5	4

*

1	0	-1
1	0	-1
1	0	-1

=

6		

$7 \times 1 + 4 \times 1 + 3 \times 1 +$
 $2 \times 0 + 5 \times 0 + 3 \times 0 +$
 $3 \times -1 + 3 \times -1 + 2 \times -1$
 $= 6$

I valori dei pixel dell'immagine sotto il filtro sono:

IMG:	7	2	3	4	5	3	3	3	2
------	---	---	---	---	---	---	---	---	---

Quelli del filtro:

K:	1	0	-1	1	0	-1	1	0	-1
----	---	---	----	---	---	----	---	---	----

Il risultato della convoluzione è la somma dei prodotti, quindi:

$$7 \times 1 + 2 \times 0 + 3 \times -1 + 4 \times 1 + 5 \times 0 + 3 \times -1 + 3 \times 1 + 3 \times 0 + 2 \times -1 = 6$$

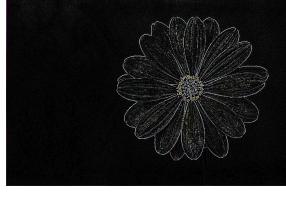
Questo sarà il valore del pixel in alto a sinistra nell'immagine risultante, ovvero quello al centro del kernel. La stessa operazione viene effettuata su tutta l'immagine (vedi animazione) facendo scorrere il filtro di una posizione alla volta su tutte le righe e colonne.

L'operazione di convoluzione è implementata in Tensor.h

```
Tensor convolve(const Tensor &f);
```

e riceve in input un filtro (Tensor f), applica il padding all'oggetto (vedi sotto) e ne calcola la convoluzione.

Dovete implementare 4 filtri che applicherete all'immagine per mezzo della convoluzione:

Sharpen	Edge	Emboss	Smoothing
Matrice 3x3, permette di enfatizzare i dettagli	Matrice 3x3, calcola i contorni dell'immagine	Matrice 3x3, aggiunge un senso di profondità o rilievo all'immagine	Matrice h x h, permette di rimuovere il rumore. $c = 1/(h \times h)$ Ad esempio $w = 3$ e $h = 3 \Rightarrow c = 1/9$
Valori: $\begin{matrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{matrix}$	Valori: $\begin{matrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{matrix}$	Valori: $\begin{matrix} -2 & -1 & 0 \\ -1 & 1 & 1 \\ 0 & 1 & 2 \end{matrix}$	Valori: $\begin{matrix} c & c & c \\ c & c & c \\ c & c & c \end{matrix}$
Metodo: <code>DAISGram sharpen();</code>	Metodo: <code>DAISGram edge();</code>	Metodo: <code>DAISGram emboss();</code>	Metodo: <code>DAISGram smooth(int h);</code>
			
Note:	Note: convertite l'immagine in scala di grigi prima di applicare questo filtro. Tale operazione va fatta all'interno del metodo <code>DAISGram edge();</code>		

Problema 1: le immagini sono rappresentate tramite una matrice a 3 dimensioni ma il filtro è a 2 dimensioni.

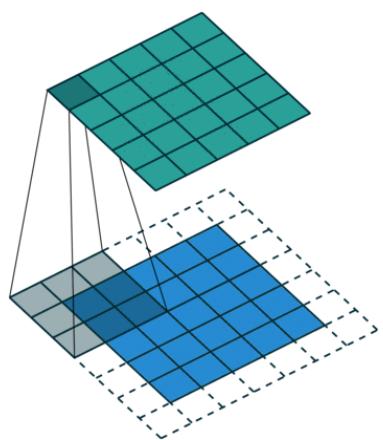
possiamo quindi: 1) avere filtri diversi per ogni canale (uno per il canale rosso, uno per il verde ed uno per il blu) oppure 2) utilizziamo lo stesso filtro per tutti i canali.

Nel nostro caso vogliamo un filtro diverso per ogni canale, nel caso in cui lo stesso filtro sia applicato a più canali, questo si dovrà copiare sui relativi canali del filtro. In sostanza, filtro e immagine devono avere la stessa *depth*.

Problema 2: Cosa fare sui bordi?

Come potete notare, l'immagine risultante dalla convoluzione è più piccola dell'immagine originale perdendo l'informazione sui bordi. Ad esempio nella figura sopra otteniamo un'immagine con dimensione 3x3x1 (vedi parte destra della Figura 1).

Ci sono varie tecniche per risolvere questo problema e ottenere un'immagine delle dimensioni originali dopo la convoluzione. Nel nostro caso useremo una tecnica chiamata *padding*. Nel padding si estende l'immagine con un bordo di larghezza opportuna di pixel tutti a zero. In questo modo, a convoluzione applicata, otterremo un'immagine della stessa dimensione dell'input.



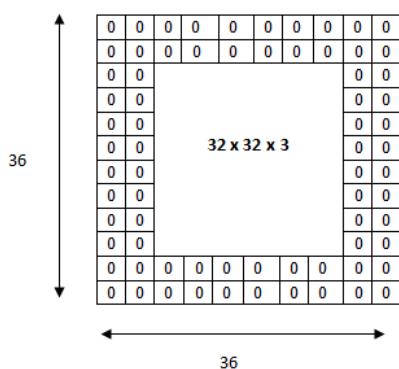
Nella figura a sinistra abbiamo: in blu l'immagine (a singolo canale), in bianco il padding, in grigio il filtro e in verde l'immagine risultante. Notate come la dimensione dell'immagine verde sia la medesima dell'immagine blu.

Ma quanto bordo aggiungere? Dipende dalle dimensioni del filtro!

In generale il padding si calcola come:

$P = \lceil (F-1)/2 \rceil$ mantenendo la parte intera. Dove P è il valore del padding risultante ed F è la dimensione del filtro.

Quindi un filtro 3x3 avrà come padding orizzontale $(3-1)/2 = 1$ e verticale $(3-1)/2 = 1$. Un filtro 5x5 avrà come padding orizzontale $(5-1)/2 = 2$ e verticale $(5-1)/2 = 2$.



Se il filtro è di dimensione 3x3 estenderemo quindi l'immagine di 1 pixel per lato, ad esempio se un'immagine è a 283x362x3, l'immagine risultante con il padding sarà di 285*364x3 (1 pixel sul lato sinistro, 1 sul lato destro, 1 pixel in alto e 1 sotto).

L'operazione di padding viene effettuata, ovviamente, prima della convoluzione.

Problema 3: Una volta calcolata la convoluzione i valori hanno un range diverso da [0,255]!
Come potete vedere nella figura 1, i valori risultanti dalla convoluzione non sono in [0,255].
Prima di convertire la *salvare* un’immagine bisogna convertire i valori del campo *data* nel
range di cui sopra. Pertanto dobbiamo “scalare” il nostro Tensore.

Dovete implementare due metodi (vedete il dettaglio nel file Tensor.h):

```
void clamp(float low, float high);  
void rescale(float new_max=1.0);
```

Il metodo di `rescale` porta i valori del Tensore in [0,new_max] tramite questa operazione:
$$T = (T - \text{min}(T)) / (\text{max}(T) - \text{min}(T));$$

dovranno quindi essere moltiplicati per *new_max* in modo da essere nel range [0,*new_max*].

Il metodo `clamp` vincola l'intervallo di valori tra un valore minimo e uno massimo.

Parti Opzionali:

Sono presenti due parti opzionali del progetto.

Green- Screen (o chroma-key)

Il green screen è una tecnica cinematografica che permette di sostituire uno sfondo a tinta uniforme con uno desiderato. E' tutt'ora largamente utilizzato in tantissime produzioni (Matrix, Avengers etc...).



L'idea alla base è quella che, date due immagini (una di primo piano e una di sfondo), vengano sostituiti i pixel di un certo colore nell'immagine in primo piano con i corrispondenti pixel dell'immagine di sfondo.

Dovete implementare il metodo

```
DAISGram greenscreen(DAISGram & bkg, int rgb[], float threshold[]);
```

che riceve in input un background (delle stesse dimensioni dell'oggetto attuale), il colore di sfondo (come vettore $\text{rgb}[3]=\{0,128,00\}$) e un vettore di soglia ($\text{threshold}[3]=\{10,20,10\}$) che decide per ogni canale l'intervallo intorno al colore selezionato.

Nell'esempio di cui sopra andremo a scambiare i pixel tra primo piano e sfondo se questa condizione è soddisfatta per tutti e 3 i canali:

```
pixel(i,j,0)>=(rgb[0]-threshold[0]) && pixel(i,j,0)<=(rgb[0]+threshold[0]) ...
```



Equalizzazione Immagine:

L'equalizzazione dell'immagine è una tecnica che permette di utilizzare tutto il range di valori che i pixel possono acquisire. In questo modo il range di luminosità viene stirato rendendo le immagini più vive. `DAISGram equalize();`

Il metodo è ben descritto in questo [link](#).

Consideriamo l'equalizzazione indipendente per canale, quindi dato un canale:

1. Creare un istogramma della distribuzione delle intensità (vettore di 256 elementi)
2. Calcolare la funzione cumulativa (*cdf*) dell'istogramma.
3. Calcolare il minimo della funzione cumulativa (cdf_{min})

Dato un valore v di luminosità, la sua versione equalizzata $h(v)$ è:

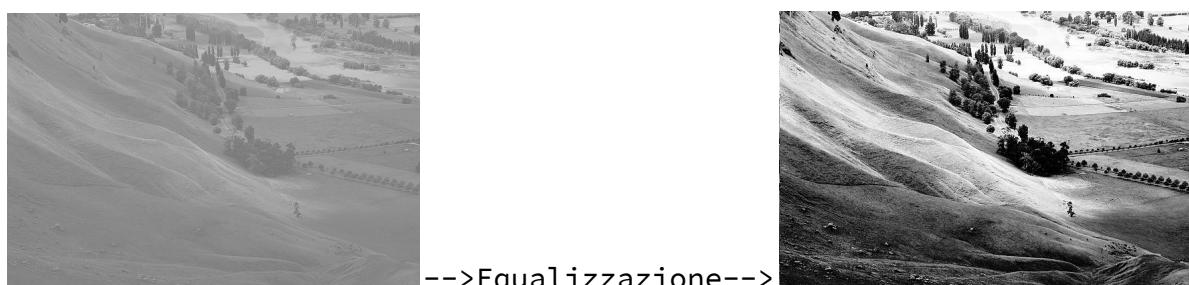
$$h(v) = \text{round} \left(\frac{cdf(v) - cdf_{min}}{(M \times N) - 1} \times (L - 1) \right)$$

dove M ed N sono le dimensioni dell'immagine mentre L è il numero di valori di intensità (256).

Ripetete la stessa operazione indipendentemente per i 3 canali.

Per fare delle prove, convertite prima l'immagine in scala di grigi e poi applicate l'equalizzazione. Se lo fate su immagini a colori potrebbe succedere (quasi sicuramente) che i colori vengano falsati.

La conversione in scala di grigi NON includetela all'interno del metodo `DAISGram equalize();`



Gestione Errori

Nel caso in cui vengano passati dei parametri sbagliati, ad esempio le dimensioni di due Tensor non sono concordi, oppure Tensor è a NULL, dovete generare un errore e bloccare l'esecuzione. Gli errori sono gestiti tramite eccezioni che trovate dentro il file `dais_exc.h`.

Consegna del progetto

Trovate nel [repository del progetto](#) i seguenti file:

- tensor.h: l'intestazione dei metodi da implementare per il calcolo tensoriale
- DAISGram.h: l'intestazione dei metodi da implementare per il calcolo tensoriale
- libbmp.cpp e libbmp.h: la libreria per leggere e scrivere bitmap
- test_bmplib.cpp: tester per la libreria bmp
- main.cpp: main file con cui poter giocare con la libreria DAISGram e Tensor
- le immagini di questo documento con cui fare delle prove.
- makefile

Il file "main.cpp" contiene il main per elaborare le immagini con le chiamate ai vari metodi di cui sopra (ovviamente senza implementazione). Potete utilizzarlo per fare dei test con i filtri/metodi che avete implementato e vedere subito i risultati. Se lo eseguite senza parametri visualizzerà la guida.

Dovete creare un repository privato

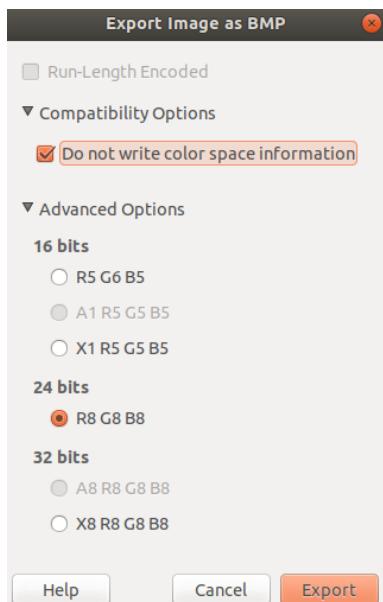
The screenshot shows the GitHub repository creation interface. At the top, it says "Owner * xwasco" and "Repository name * labprog2020-2021". Below that, a message says "Great repository names are short and memorable. Need inspiration?". Under "Description (optional)", there is a text input field. At the bottom, there are two radio button options: "Public" (disabled) and "Private" (selected). The "Private" option is described as "You choose who can see and commit to this repository."

e fare l'import dal repository della consegna:

The screenshot shows the GitHub repository import page. It has sections for "Quick setup", "...or create a new repository on the command line", and "...or push an existing repository from the command line". The bottom section, "...or import code from another repository", is highlighted with a red box. It contains the text "You can initialize this repository with code from a Subversion, Mercurial, or TFS project." and a "Import code" button. A pro tip at the bottom says "ProTip! Use the URL for this page when adding GitHub as a remote."

Il software dovrà funzionare indipendentemente dalle immagini che vi vengono fornite, quindi vi esorto a provarlo con immagini BITMAP a piacere. Per convertire immagini in formato bitmap potete utilizzare GIMP o altro software per l'editing di immagini.

Onde evitare problemi di compatibilità ricordatevi di NON includere informazioni sui colori nella bitmap ed esportate a 24bit. Questo si può fare in GIMP esportando in bitmap (.bmp) e selezionando le seguenti opzioni.



La consegna è fissata al 15 Giugno 2021 23:59.

Consegna:

Dovrete implementare correttamente tutti i metodi presenti nel file *tensor.h* e *DAISGram.h* escluse/incluse le parti opzionali. Per la consegna dovete fornirci l'indirizzo di un repository github.

Compilate il [foglio gruppi](#) indicando i dati richiesti (nome del gruppo, url del vostro repository e i numeri di matricola dei membri del gruppo).

Il file *tensor.h* e *DAISGram.h* che consegnate dovrà rispettare le chiamate a funzione date, quindi **NON** cambiate le intestazioni. Se avete dubbi contattateci.

Suggerimenti:

- iniziate ad implementare i metodi di gestione della memoria. Usate il tool valgrind.
- successivamente implementate quelli relativi ad operazioni matematiche con overload dei parametri
- poi passate alle operazioni semplici sulle immagini (scala di grigi, blending, brighten, etc...)
- successivamente lavorate sulla convoluzione e sui filtri
- POI sulle parti opzionali.

Valutazione:

- Parte 1: Implementazione gestione memoria e operazioni matematiche: **30%**
- Parte 2: Implementazione delle funzioni semplici (scala di grigi, blending, brighten, corruption): **30%**
- Parte 3: Implementazione della convoluzione e dei filtri: il **45%** del progetto

Progetti con punteggio superiore al 100% potranno ottenere la lode.

- Il lavoro in gruppo è molto consigliato in quanto potete dividervi agilmente i compiti una volta coordinati con le strutture dati. ATTENZIONE: la discussione verterà individualmente su TUTTO il progetto.

- **Se il progetto NON compila (è presente anche un solo errore) non sarete ammessi all'orale**

Plagio:

Il plagio non è tollerato pertanto tutti i progetti coinvolti saranno annullati e le parti opzionali diventeranno obbligatorie per i soggetti coinvolti.