Elaborazione delle Immagini

Sara Angeretti
@Sara1798

Fabio Ferrario
@fefabo

2023/2024

Indice

Capitolo 1

Introduzione a MatLab

1.1 Appunti video lezione 2020 da Moodle

1.1.1 Ambienti di lavoro

Command Window e Workspace

- Command Window: finestra di comando, dove andiamo effettivamente a scrivere i comandi.
- Workspace: variabili in memoria, vediamo le variabili che abbiamo istanziato, mi dà un feedback immediato di quello che ho fatto.

Variabili

```
>> pippo = 10
pippo =
    10
>>
```

Questo vuol dire che istanziare una variabile assegnandole anche un valore numerico, dopo che ho premuto invio mi viene creata in memoria la variabile che ho creato (lo vedo nel workspace che l'ho creata). Però scrivendo così mi viene anche stampato a schermo il comando che ha eseguito.
Nel workspace vedo che ho una variabile chiamata pippo che ha valore 10, size 1x1 e class double.
Questo vuol dire che ha dimensione 1x1, cioè che è una singola cella (MatLab lavora con dati che sono matrici). Inoltre, double è il valore base, di default, quasi tutto viene fatto in virgola mobile (ci possono essere forzature se necessario).
Ci sono operazioni più complesse però di semplici assegnamenti, per cui potrebbe tornare utile evitare di stampare a schermo, perché non è assolutamente necessario vedere live l'output del comando che viene eseguito. Perciò:

```
>> pluto = 10;
>>
```

1.1.2 Lista di comandi utili

¿¿ **clear:** pulisce il workspace, cancella tutte le variabili che ho istanziato.

¿¿ **clc:** pulisce il command window, cancella tutti i commenti che ho scritto.

freccia su tastiera: va a riprendere l'*history* dei miei comandi nella command window, che apre in una finestrella pop-up.

1.1.3 Operazioni matematiche

Addizione

```
>> a = 2;
>> b = 1;
>> c = a + b;
```

Sottrazione

```
>> a = 2;
>> b = 1;
>> c = a - b;

ans =

    1
```

Prodotto

```
>> a = 2;
>> b = 1;
>> c = a * b;

ans =

    2
```

Divisione

```
>> a = 2;
>> b = 1;
>> c = a / b;

ans =

    2
```

Elevamento a potenza

```
>> a = 2;
>> a^3;

ans =
```

```
    8
```

MatLab richiede che il valore di un'operazione venga associato ad una variabile. Se non la creo io, lo fa MatLab per me (qua la variabile "`ans`"). Visibile nel *Workspace*.
Questa variabile può essere riutilizzata.

```
>> a + ans;

ans =

    10
```

Però ocio che ad ogni operazione verrà sovrascritta.

Altre operazioni

Si può operare in tanti modi con gli scalari su MatLab; per vedere le operazioni possibili, si deve consultare la documentazione.

1.1.4 La documentazione

1. Tasto "*Help*" in "Resources"
2. In Command Window possiamo chiederlo direttamente a MatLab:
   - ▷ comando "`help`", mi dà dei suggerimenti sull'ultima operazione eseguita.
   - ▷ comando "`help max`", (dove "`max`" è un esempio di operazione che dà in output il massimo numero in un array di numeri dati in input) mi dice quale è la sintassi con tutte le sue alternative dell'operazione richiesta.
   - ▷ comando "`lookfor max`", (dove "`max`" è un esempio di parte del nome di un'operazione che non ci ricordiamo nella sua interezza) mi dà una lista di possibili operazioni che potrebbero essere quella che stiamo cercando e che contengono il pezzo di nome che gli ho assegnato.

1.1.5 Gli array

Sintassi e operazioni sui vettori

Vettori riga: i vettori sono rappresentati come concatenazione di valori, secondo la sintassi:

```
>> v = [1,2,3,4,5,6];
>> v

v =

    1   2    3    4    5    6
```

```
>>
```

Nel workspace vedremo:

Name	Value	Size	Class
v	[1,2,3,4,5,6]	1x6	double

In questo caso è un **vettore riga**.
Le *parentesi quadre* si usano per **concatenare** i valori, le *virgole* per separare i valori in un *vettore riga*, i *punti e virgola* per separare i valori in un *vettore colonna*.

Vettori colonna: se invece volessimo un **vettore colonna**:

```
>> w = [10;20;30;40];
>> w

w =

    10
    20
    30
    40

>>
```

Nel workspace vedremo

Name	Value	Size	Class
w	[10;20;30;40]	4x1	double

N.B.: sono ***indicizzabili***. Per indicizzare si usano le *parentesi tonde* e, cosa importantissima, *in MatLab* ***non esiste l'indice 0****, perciò si parte da 1 e non da 0*. Se si scrive lo 0 come indice, dà errore.
Se volessimo ad esempio il terzo elemento del vettore v:

```
>> v(3)

ans =

    3

>>
```

Assegnare un valore: questa indicizzazione la possiamo usare anche in assegnazione:

```
>> v(3) = 100;
>> v

v =

    1   2     100     4     5     6

>>
```

Posso lavorare anche su *serie di cellette* invece di una celletta singola:

```
>> v = (1:3);

ans =

1   2     100

>> v = (1:3);

ans =

    4     5     6

>>
```

Se volessi, senza sapere quanto è lungo un vettore (c'è un modo per saperlo, ma non lo vediamo ora), prendere tutti gli elementi da un certo indice fino alla fine, posso fare:

```
>> v(4:end)

ans =

    4     5     6

>>
```

Scoprire la lunghezza di un vettore: con il comando:

```
>> size(v)

ans =

    1   6

>>
```

Ovvero una riga e 6 colonne. La convenzione (è sempre così) è prima righe e poi colonne, perciò se volessi conoscere solo le righe:

```
>> size(v,1)

ans =

    1

>>
```

se volessi conoscere solo le colonne:

```
>> size(v,2)

ans =

    6

>>
```

Somma: come sappiamo da GAL, se sommo due vettori devono avere la stessa dimensione, altrimenti non posso sommarli.

Prodotto matriciale: come sappiamo da GAL, se moltiplico due vettori devono avere gli stessi indici interni, altrimenti non posso moltiplicarli. Es.: noi abbiamo `v 1x6` e `w 4x1`, non posso moltiplicarli perché $6 \neq 4$.
Una cosa che potrei fare per ovviare il problema sarebbe selezionare un sotto vettore da `v` che abbia la stessa dimensione di `w`:

```
>> v(1:4)*w

ans =

    3210

>>
```

Per chi non ha ancora visto GAL, questo è un prodotto scalare, ma perché `ans` vale `3210`? Praticamente ho due matrici $m \times p$ e $p \times n$, la dimensione della matrice prodotto sarà $m \times n$. Quindi qua ho un vettore riga 1×4 e un vettore colonna 4×1, il prodotto sarà un vettore riga 1×1, che è quello che vediamo in `ans`. Inoltre il prodotto fra:

una matrice

a_1	a_2	a_3
a_4	a_5	a_6

e una matrice

b_1	b_2
b_3	b_4
b_5	b_6

mi dà una matrice 2×2 con i valori:

$$\begin{array}{ll} a_1 * b_1 + a_2 * b_3 + a_3 * b_5 & a_1 * b_2 + a_2 * b_4 + a_3 * b_6 \\ a_4 * b_1 + a_5 * b_3 + a_6 * b_5 & a_4 * b_2 + a_5 * b_4 + a_6 * b_6 \end{array}$$

È quello che succede se:

```
>> z = v(1:4)

z =

    1   2     100     4

>> w*z

ans =

    10   20     1000     40
    20   40     2000     80
    30   60     3000     120
    40   80     4000     160

>>
```

Prodotto puntuale : .* (punto per punto), ovvero moltiplica elemento per elemento, pixel per pixel, lasciandoli al loro posto.

Valore massimo: `max(v)` mi ritorna il valore massimo fra i numeri del vettore. Per visualizzare le varianti, basta scrivere `max` e aprire la parentesi tonda sinistra e in sovrimpressione uscirà la lista delle varianti. In questo caso:

```
>> max(v)

ans =

    100

>>
```

Somma dei valori: `sum(v)` mi ritorna il valore totale della somma dei numeri del vettore. Questo, come il comando precedente, ha un comportamento diverso se si tratta di vettori o di array. In questo caso:

```
>> sum(v)

ans =

    118
```

```
>>
```

Creare matrici di numeri random: `randi(r, l, c)` mi ritorna un array di `l` righe e `c` colonne di numeri scelti randomicamente in un range ampio `r`.

```
>> v = randi(5,1,10)

v =

    1   5   1   4   5   5   1   2   2   5

>>
```

Trovare un valore uguale ad un numero dato in un array: mi ritorna un vettore booleano che MatLab chiama *vettori logici* che mi dice dove la condizione che ho espresso è vera, ovvero il valore del vettore è uguale al numero dato quando c'è 1.

```
>> v == 5

ans =

    1x10 logical array

    0   1   0   0   1   1   0   0   0   1

>>
```

Non è male perché è una semplice riga di istruzione, mi evita cicli espliciti e altra roba.

Trovare i valori maggiori di un numero dato: mi ritorna un *vettore logico* che mi dice dove la condizione che ho espresso è vera.

```
>> v>3

ans =

    1x10 logical array

    0   1   0   1   1   1   0   0   0   1

>>
```

Sono operazioni molto utili. E combinabili!

```
>> sum(v>3)

ans =

    5

>>
```

Mi conta quanti sono gli 1 del vettore logico.
Posso usare variabili di appoggio:

```
>> x = v>3

x =

    0   1   0   1   1   1   0   0   0   1

>> v

v =

    1   5   1   4   5   5   1   2   2   5

>>
```

Selezionare solo alcuni valori: `>> v(x)`

```
ans =

    5   4   5   5   5

>>
```

Praticamente mi seleziona solo i valori di `v` che hanno 1 nel *vettore logico* `x`.
Molto utile per creare le maschere! Posso indicizzare il mio vettore anche usando un altro vettore di numeri:

```
>> v([1,3,4,6])

ans =

    1   1   4   5

>>
```

Questo invece mi ritorna i valori di `v` che mi interessano, che ho chiesto esplicitamente.

Posso selezionare solo i valori che hanno 1 nel vettore logico anche in un altro modo:

```
>> find(x==1)

ans =

    2   4   5   6   10

>>
```

Questo mi ritorna gli *indici* delle cellette di `x` che mi soddisfano questa condizione.
Posso ovviamente poi usarlo per ottenere i corrispondenti valori di `v`.

```
>> v(find(x==1))

ans =

    5   4   5   5   5

>>
```

Notare che mi produce lo stesso risultato della sua variante precedente, `v(x)`. Queste due istruzioni sono equivalenti.

Sintassi e operazioni sulle matrici

Possiamo istanziare una matrice andando a memorizzare tutti i valori che vanno salvati nelle righe e nelle colonne. Le operazioni sono pressoché le stesse che abbiamo visto per i vettori, solo che ora abbiamo due dimensioni e quindi due indici, uno per le righe e uno per le colonne.

Riempire una `3x2` a mano: `>> a=[1,2;3,4;5,6]`

```
a =

    1   2
    3   4
    5   6

>>
```

Trovare il valore di una specifica celletta: ad esempio terza riga prima colonna,

```
>> a(3,1)
```

```
ans =

    5

>>
```

Visualizzare un'intera riga: ad esempio la seconda

```
>> a(2,1:end)

ans =

    3   4

>>
```

In alternativa, ":" indica "tutti gli elementi", quindi:

```
>> a(2,:)

ans =

    3   4

>>
```

Visualizzare un'intera colonna: funziona come con le righe, ad esempio la prima

```
>> a(:,1)

ans =

    1
    3
    5

>>
```

Matrice trasposta: `>> a'`

```
ans =

    1   3   5
    2   4   6

>>
```

Creare matrici di comodo: utili per creare matrici su cui computare operazioni:

```
>> zeros(3,3)

ans =

    0   0   0
    0   0   0
    0   0   0

>> ones(5,5)

ans =

    1   1   1   1   1
    1   1   1   1   1
    1   1   1   1   1
    1   1   1   1   1
    1   1   1   1   1

>>
```

Creare matrici randomiche: abbiamo visto prima il comando rand, che mi crea numeri randomici double. Se voglio creare una matrice intera, posso usare il comando randi, che mi crea una matrice intera randomica.

```
>> m=rand(5,5)

m =

    0.4314  0.1361  0.8530  0.0760  0.4173
    0.9106  0.8693  0.6221  0.2399  0.0497
    0.1818  0.5797  0.3510  0.1233  0.9027
    0.2638  0.5499  0.5132  0.1839  0.9448
    0.1455  0.1450  0.4018  0.2400  0.4909

>>
```

Creare matrici inverse: `>> inv(m)`

```
ans =

   -9.6261   -0.7953   -70.5587    84.5354   -24.6859
    5.3676    1.3707    43.2380   -50.8728    13.6995
    7.3422    0.6401    42.8965   -51.9585    14.8125
   -1.0754    0.7834     6.1327   - 9.5106     7.8621
   -4.2154   -1.0759   -29.9599    37.1396   -10.6575
```

```
>>
```

Determinante: restituisce il suo valore

```
>> det(m)

ans =

    -0.0014

>>
```

Diagonale: riporta i valori posizionati sulla diagonale principale della matrice

```
>> diag(m)

ans =

    0.4314
    0.8693
    0.3510
    0.1839
    0.4909

>>
```

Sottomatrici: riporta solo i valori della sottomatrice scelta, ad esempio quella fra seconda e terza riga, terza e quarta colonna

```
>> m(2:3,3:4)

ans =
0.6221  0.2399  0.0497
0.1818  0.5797  0.3510  0.1233

>>
```

Capitolo 2

Laboratori Elaborazione delle Immagini 2023-34

2.1 Lab1 13 Ottobre 2023

2.1.1 Il negativo di un'immagine

Serve per rimappare un'immagine. Lo faccio sottraendo la prima immagine a 255 e asegnando il tutto ad una seconda immagine.

```
>> im1 = imread('path\img.format');
>> im2 = 255 - im1;
>> imshow(img2);
```

2.1.2 Mostrare a schermo due immagini

```
>> figure(i);
>> subplot(1,2,1);
>> imshow(im1);
>> subplot(1,2,2);
>> imshow(im2);
```

figure; : crea una finestra, quando ne serve più di una si aggiunge `(i)` con un indice a scelta.
Se non passiamo niente in input, MatLab indicizza da solo, altrimenti glielo diciamo noi l'indice.
Rendo attiva la finestra richiamandola con l'indice scelto.

subplot(r, c, q); : `subplot(n° righe, n° colonne, n° quadrante)`, rende attiva la `figure` e divide la finestra in riquadri (puoi fare più righe ($r > 1$) e più colonne ($c > 1$)) e `q` dice in quale quadrante (della riga scelta con `r`) mettere l'immagine.

Poi abbiamo creato `im3` come somma di `im1` e `im2`, cosa esce? Un'immagine tutta bianca, perché si sommano tutti i valori matematici e sarà tutto 255. Poi, per mostrarla a schermo, se faccio solo `imshow(im3)` andrà a sovrascrivere quella negativa, perché non ho specificato che serve una nuova figure e quindi usa l'ultima attiva. Serve `figure;` per averne una nuova.

2.1.3 Pulizia dello schermo

Ogni volta che eseguo lo script mi si aprono finestre di figure.

`close all;` : chiude in automatico tutte le finestre.

`clear all;` : cancella in automatico tutte le variabili.

2.1.4 Modificare size immagini

```
imresize(clouds, size(moon))
```

2.1.5 Le maschere binarie

Mi individuano una zona di interesse (in questo caso "dove vivono le nuvole") da togliere poi all'immagine originale.
Come? `treeshold` mi ritorna un vettore logico, che posso usare come maschera binaria. Per ora la soglia la troviamo a mano.
Abbiamo provato T = 0.5, ma non bastava. Abbiamo abbassato a 0.25, che poteva andare per quanto riguarda la zona di interesse, ma abbiamo abbassato ancora a 0.125 che mi ha preso ancora un po' di pixel.
Spoiler alert: alla fine abbiamo rimesso 0.25 perché la maschera aveva preso troppi pixel scuri e quando si zoommava sulla foto finale era tutta rovinata con bordi scuri etc.
Abbiamo provato 0 ed è diventato troppo, ha mostrato un effetto quadrettato che mi mostra artefatti/rumore che è tipico della compressione JPEG.
0.125 va più che bene, poi la usiamo per togliere un pezzetto di immagine da quella della Luna.

$1 - mask_clouds$ mi ritorna la maschera binaria invertita, ovvero dove c'è 1 mette 0 e viceversa.

$.*$ moltiplicazione punto punto, dove nella maschera invertita ho 0, si azzera il pixel corrispondente nell'immagine.

Se volessi una maschera più smooth? Così che si nasconda il passaggio tra nuvola e sfondo della nuvola? Applico un filtro di smooth sull'immagine delle nuvole. Non tolgo totalmente il rumore, ma lo riduco localmente.

Immagini double

Prima mi sono dimenticata di segnare che spesso serve avere le immagini in forma double, si fa tipo

```
moon = im2double(imread("Lab1\moon.jpg"));
```

usando quindi il comando

```
im2double();
```

che converte l'immagine da *uint8* in *double*.

2.1.6 Gli istogrammi

`imhist();` mi ritorna l'istogramma dell'immagine.
Dall'istogramma dell'immagine del cavallo vedo che è **ben contrastata**, ovvero uso tutti i valori del range di valori dispobile.
`histeq();` mi ritorna l'immagine equalizzata.

2.1.7 Gamma correction

`imadjust();` mi ritorna l'immagine con la gamma corretta.
È un mapping non lineare, che mi permette di rimappare i valori di un'immagine.

2.1.8 Compiti a casa - 1

Sort of assignment, sono 4 e danno fino a 2 punti in più all'esame (quindi se ne consegno 2 è solo 1 punto in più). Per dire, il primo chiede di scrivere una funzione `myhistogram` che calcola l'istogramma di una immagine a livelli di grigio. Noi sappiamo che un istogramma cumulativo (uint8 tra 0 e 255 monocanale) è un vettore che va da 0 a 255 e mi conta le occorrenze di ogni valore.

2.2 Lab2 20 Ottobre 2023

2.2.1 Es1

Es.1, su immagine `contrast.jpg`.
Punto `a1.`: "Caricate l'immagine 'contrast.jpg' in una variabile im e scalataela tra 0 e 1." vuol dire `imread` e portarla in double con `im2double`.
Punto `b1.`: `crop` mi prende una sottomatrice della mia immagine, che va da (x_1, x_2) a (y_1, y_2), ovvero:`crop = img`$(x_1 : x_2, y_1 : y_2)$
Punto `c1.`: soglia T per trovare i valori scuri.
Ci applichiamo la statistica di **media**, bisogna saper capire quale statistica è meglio.
Ma MatLab opera per colonne; posso linearizzarlo, facendo `crop(:)`, così facendo la media avrà un solo valore.
$media + -k * deviazionestandard$ mi dà un intorno al picco dell'istogramma.
Abbiamo parzialmente automatizzato la soglia.
Punto `d1.`: due maschere light e dark che contengono rispettivamente le regioni chiare e scure di im usando la tecnica delle maschere.

2.2.2 Es2

Filtri lineari

Sono filtri spaziali che possono essere espressi con funzioni lineari. Maschera di convoluzione (?) che prende l'intorno di un pixel e fa la media pesata di tutti i pixel del vicinato. Abbiamo visto a lezione diversi filtri, i più diffusi quelli di smoothing che riducono il rumore (variazione pseudorandomica dell'immagine).

I comandi

`fspecial("filtro", valore)`: funzione che permette di generare diversi tipi di filtri.
In un filtro di media voglio fare la media di `n` valori, se ne ho 25 peso i valori per $\frac{1}{25}$.
`imfilter(input, filtro)`: applica un filtro ad un input.
Più aumento il filtro (5 -¿ 11 -¿ 21) più l'immagine si sfoca ma si crea un bordo perché dove non esistono i pixel vengono messi a 0 (quindi la convoluzione sui bordi mi crea bordi neri).
Ci sono diverse opzioni per evitare ciò: replica i valori, o pescali da in giro, o boh.
Ora dobbiamo creare delle immagini che siano la differenza tra immagine originale e filtrata in valore assoluto. Ma sono array di valori, non usiamo imshow. Usiamo `imagesc` (image scaling) che tratta gli array come fossero immagini e al tempo stesso le tratta con colori fasulli in modo da vedere le differenze. Però spiattella le immagini, perciò aggiungo `axis image` che mi mantiene le proporzioni. `colorbar` mi aggiunge una scala di valori per capire quanto è grande la differenza e dove, in base ai colori fasulli che mi ha messo. E dove lo vedo? Dove ci sono alte frequenze, ovvero nei dettagli, che è dove opera il filtro di smoothing.
Quindi questi valori evidenziati dalle immagini colorati mi dicono **dove ho perso informazione**, cosa che fanno i filtri di smoothing (sono da tonare adeguatamente).
Tutto dipende dalla forza del rumore: se è poco evidente posso limitare la forza del filtro di smoothing.

2.2.3 Es3

Copio e incollo l'es2 perché riciclo il codice in quanto cambio solo il filtro che uso.

Il filtro gaussiano

Due parametri: dimensione spaziale del filtro e la σ.
La sigma calcola quanto sono dispersi i valori della gaussiana rispetto alla sua media.
Essendo distribuzione di probabilità, la sua area?boh senti la registrazione.
I pixel più vicini a quello da calcolare hanno più peso rispetto a quelli più lontani.
Come faccio filtro spaziale su distribuzione gaussiana? Vado a campionarla.
Ha aperto la F3, poi la F5, poi la F7. Nella 5 la sottomatrice centrale è praticamente quella del 3. Stessa cosa per F7. Va da sé che anche F9. Che è succ?
MatLab ha preso una funzione gaussiana di valori discreti e ha campionato la gaussiana, trovato certi valori utili.
Una volta trovata la forma della gaussiana (sigma) bisogna tonare la gaussiana: perciò non ha senso andare troppo oltre un certo valore (in questo caso 5 perché i pixel intorno sono praticamente a 0).
Quando creo un filtro gaussiano a due parametri, la dimensione spaziale del filtro e la sigma, questi due parametri non sono indipendenti ma sono legati tra loro. Di solito, si la dimensione di un filtro Gaussiano si ricava dalla seguente

formula che garantisce di avere tutti i valori utili del filtro:

$$N = 1 + 2 \times [2.5 \times \sigma]$$

`figure, surf(filtro)`: mi fa vedere la forma del filtro.
Il filtro gaussiano è molto meglio del filtro di smooth per smoothare l'immagine perché è tonabile. A parità di dimensione, il gaussiano rovina meno l'immagine del filtro di smooth.

2.2.4 Es4

Parte dicendo "Create un filtro Gaussiano G di dimensione 21x21 con la Sigma adatta." questo lo faccio con la formula di prima

$$N = 1 + 2 \times [2.5 \times \sigma]$$

assegnando 21 a N e risolvendo per σ. Ottengo 4, il minimo sindacale.

2.2.5 Es5

Cosa succede con immagini (Lawrence1) dove c'è palesemente del rumore gaussiano? Rumore distribuito ed evidente ma che non ci dà tanto fastidio.
La riduzione del rumore dipende sempre dal task finale: per dire va bene per ambito cinematografico (la pellicola crea naturalmente questo tipo di rumore quindi è perfetta) mentre per tipo boha ltri scopi mi dà fastidio e lo voglio togliere e avrebbe completamente senso.
Lawrence2 invece salt&pepper, ovvero rumore impulsivo.
S&P è un rumore che non può essere rimosso da un filtro normale, è randomico quindi non segue una legge di distribuzione (perciò i filtri lineari non sono utilizzabili). Il rumore gaussiano può essere rimosso da un filtro gaussiano (o di media, ma gaussiano meglio).

2.2.6 Es6

Facciamo noi, vediamo come l'uso di filtri spaziali semplici possono tornare utili per cose tricky.

2.2.7 Es7

Combinazione di filtri lineari = filtro lineare.
La convoluzione è un filtro lineare.
Riprendiamo es4 e out diventa = $glamour@(I + G) * 0.5$; combino op di convoluzione.

2.2.8 Gli assignment

nearest neighboor campiono le righe e le colonne e replico qualche valore, il problema è decidere quale.
fare a mano il filtro di convoluzione. Anche abusando di cicli for. Nel caso filtro fuori dall'immagine, assumo valore 0 o scelgo un altro meccanismo (tipo scelgo

solo una porzione di immagine su cui riesco a lavorare quinid output immagine più piccola).

terzo, filtro non lineare. Non matrice di pesi ma algoritmo. Per ogni pixel e intorno dare in output il massimo fra i valori dell'intorno. Non lineare perché serve un algoritmo vero e proprio e non ci sono funzioni lineari.

2.3 Lab3

2.3.1 Es.1 - Filtro mediano

Per ogni pixel da processare, prende l-intorno lo analizza e valuta il filtro da applicare.
Non lineare.
Perfetto per rimuovere rumore s&p. Non genera nuovi valori nell'immagine ma rigenera quelli già presenti.
Rimane un pixel di rumore, come mai? Ale ha detto la risposta ma non mi ricordo.
Il f.m. è un buon filtro, ma attenzione: a seconda della dimensione aumento o diminuisco la probabilità di beccare un filtro di rumore.
Con 5x5 invece di 3x3 sparisce il pixel bianco.
Ad aumentare il filtro spariscono i pixel residui di rumore ma si perdono dettagli dell'immagine.
Filtro troppo alti me la rendono "cartoonizzata", mantiene i bordi ma sfoca le singole regioni.

2.3.2 Es.2 - Immagini a colori, spazio RGB

Non sono array **bidimensionali** ma **tridimensionali**.
Un canale colore è la risposta di un filtro che seleziona una certa regione della lunghezza d'onda del visibile (ovvero quanto il dispositivo di acquisizione è stato sensibile nell'acquisizione delle lunghezze elettromagnetiche). Queste risposte sono convertite in numeri.
Es.: rosso, quanto è presente l'onda elettromagnetica della regione che corrisponde al rosso? Si comporta (***in ogni singolo canale***) come il livello di grigio, perché mi calcola l'intensità del colore di quel canale (banalmente, noi lo traduciamo con "quanto colore c'è").
Come facciamo a prendere ogni singolo canale? "Facciamo a fette" l'immagine.

```
R = balls(:,:,1);
G = balls(:,:,2);
B = balls(:,:,3);
```

Perché nella foto del matrimonio il piattino nel canale G è grigio scuro e non bianco? Perché i colori non sono esattamente così come li percepiamo noi, il verde eventualmente lì non è verde puro e quindi il canale G non è prominente in quella regione rispetto agli altri colori.
NB: il bianco non è assenza di informazione, ma esattamente il contrario.
Perché il canale blu non è tutta nera? Perché il blu è contenuto in altri colori, tipo bianco del tovagliolo, grigio del tavolo, ... Invece c'è assenza totale di blu ad es. dove c'è l'arancione per dire.

TIPICA DOMANDA DEGLI SCRITTI.
Ricorda:

nero puro: assenza di tutta l'informazione del colore.

bianco puro: presenza di tutta l'informazione del colore.

grigio: presenza di informazione colore di tutti e tre i canali ma in quantità equa.

Spazio HSV

Ma non basta RGB? Dipende dall'utilizzo che ne facciamo. Danno informazioni che da RGB non posso avere. Ce ne sono decine di spazi colori, noi a teoria ne abbiamo visti alcuni.
HSV codifica dello spazio colore che dà informazioni su tinta, saturazione e valore.

2.3.3 Es.3

YCbCr
Y canale informazione luminanza, ovvero la quantità di luce che c'è in un punto, quanto è forte quel pixel in quella posizione.
Cb dice quanto è blu il colore.
Cr dice quanto è rosso il colore.
Cb e Cr dicono contemporaneamente quanto è blu vs quanto verde c'è. Parlano di un rapporto fra regioni.
`imshow` considera quello che gli do come un'immagine di livelli di grigio o RGB. Quindi reinterpreta i valori che gli passo con colori sfalsati. Noi però abbiamo YCbCr, quindi l'immagine non ha senso. I singoli canali invece vanno bene.
Blu verso giallognolo, rosso verso verde.
Si possono processare le immagini indipendentemente dall'intensità della luce.
Nell'equazione (prodotto matrice per vettore) il primo vettore, a sinistra del +, è l'offset. Ho tutto traslato in un range positivo (lo 0, l'assenza di colore, è pura Y e Cr e Cb a 128, perché da una parte andiamo sul blu e dall'altra sul giallo).

HSV
Trasformazione non lineare di RGB.
H (angolo, codificata come cerchio) - tinta - percezione del colore.
S (valore lineare) - saturazione - quanto è "pieno".
V (valore lineare) - value - intensità luminosa.

Big big nope: non fare istogramma per H. Il valore a 0 e a 360 sono uguali (siamo in un cerchio), bisognerebbe fare un istogramma circolare `(?!)`.

2.3.4 Es.4 - Rumore nelle immagini a colore

Spacchetto nei tre canali poi ricostruisco l'immagine con i canali filtrati tramite il comando:

```
R2 = imfilter(R,GF);
...
```

```
im = cat(3,R2,G2,B2);
```

3 dimensioni perché vuole in quante direzioni e quali.
Posso fare direttamente

```
face3 = imfilter(face,GF);
```

Perché le immagini a colori vengono filtrate per canale in automatico.

2.3.5 Es.5

Rumore spike, sono valori molto netti. Rumore s&p colorato.
Applichiamo filtro mediano (quello che si usa nel caso di s&p). Problema: per matlab è bidimensionale il filtro mediano. Dobbiamo spacchettare nei canali.
Plot-twist: esiste `medfilt3` che fa il filtro mediano in 3 dimensioni.

2.3.6 Es.6

Non ho scritto niente.

2.3.7 Es.7

Filtri di edge, lineari, nella convoluzione otteniamo sia val pos che neg perché approssimano le derivate.
A noi interessa di più l'effetto finale.
Esiste la funzione `edge` che applica un filtro di gradiente e una sogliatura e mi ritorna un'immagine binaria con i bordi.
Il filtro di Prewitt è un filtro derivativo del 1 ordine (derivata prima).
Il filtro di Sobel è un filtro derivativo del 2 ordine (derivata seconda).
Così soglia automatica:

```
im = im2double(imread('mondrian.jpg'));
gray = rgb2gray(im);
ep = edge(gray, "prewitt");
es = edge(gray, "sobel");
figure,
subplot(2,2,1), imshow(im), title('A colori')
subplot(2,2,2), imshow(gray), title('B&N')
subplot(2,2,3), imshow(ep), title('Prewitt')
subplot(2,2,4), imshow(es), title('Sobel')
```

Giocando con la soglia:

```
im = im2double(imread('mondrian.jpg'));
gray = rgb2gray(im);
ep = edge(gray, "prewitt");
es = edge(gray, "sobel");
figure,
subplot(2,2,1), imshow(im), title('A colori')
subplot(2,2,2), imshow(gray), title('B&N')
subplot(2,2,3), imshow(ep), title('Prewitt')
subplot(2,2,4), imshow(es), title('Sobel')
```

Canny invece è un algoritmo, non filtro lineare, e difatti è il migliore finora.
QUANDO SI FA IL PROCESSING DELLE IMMAGINI, ATTENZIONE ALL'ORDINE CON CUI SI FANNO LE COSE! Prima si toglie il rumore, poi si fanno i fltri. Prima i filtri, poi gli edge. Eventualmente, poi tolgo gli edges che non mi interessano.

2.4 Lab4

Binarizzazione e segmentazione e labeling

Le famose maschere, utile per la parte di preprocessing, per andare a trovare aree di interesse.
Diverse tecniche di segmentazione, basate su regole (**soglie**) che possono essere basate su diverse caratteristiche.
Sono simili, binarizzazione trova 2 regioni, la segmentazione ne trova diverse.
Legate alle regioni, c'è anche il labeling delle regioni connesse. Nel caso di immagini binarie, etichetta tutte le diverse regioni disgiunte selezionate all'interno di un'immagine. Anche se la binarizzazione crea solo pixel o 0 o 1, il labeling considera zone di 1 disgiunte come regioni disgiunte.

2.4.1 Es1

Obiettivo: trovare la regione della mano per operare e fare cose. L'immagine è a livelli di grigio quindi l'unica info che abbiamo è l'intensità luminosa.
Suggerimento: guardare l'istogramma per vedere se è per caso bimodale. Lo è, abbiamo un picco abbastanza alto (presumibilmente lo sfondo) e uno più basso di valori non chiari ma meno scuri (Ragionevolmente la mano).
Guardando l'istogramma, possiamo vedere che la soglia è intorno a 33, che normalizzo poi dividendo per 255.
Non è perfetta, ci sono pixel qua e là bianchi nello sfondo. Posso migliorare con filtri mediani?
La soglia calcolata da noi esce circa 0.13, ma se la calcolo in automatico con la funzione di MatLab `graythresh(hand);` ottengo 0.216 (circa livello 55 nell'istogramma).
La regione c'è, lo sfondo è meglio, la mano è un po' più sporca: viene selezionata anche la zona d'ombra. Non è malissimo ma la nostra soglia è migliore. Anche questa comunque dovrebbe essere sistemabile con qualche ciclo di filtro mediano (ma non sappiamo quanti). Più avanti impareremo della morfologia per fare processing abbastanza bene da saper migliorare senza problemi.
La funzione ?rgaythresh? usa un algoritmo chiamato di Otsu (lo puoi vedere sul pdf del Lab4): va a dividere l'istogramma in due zone, va a massimizzare la distanza e minimizzare la varianza, così che siano più distanti e addensati.
L'ombra visibile nella mano è dovuta alla coda del secondo picco che sta sotto alla soglia di 55 che viene presa dentro per sbaglio.
è un algoritmo molto usato ma appunto c'è il rischio di prendere anche zone che non vogliamo se l'istogramma non è ben bimodale.
Se facessimo labeling, avremmo 2 regioni: la mano e lo spigolo dello sfondo che è abbastanza grande da dar fastidio ma abbastanza piccolo perché in fase di processing possa essere presa ed eliminata.

2.4.2 Es2

Immagine marker simil QRcode.
Problema: l'illuminazione non è uniforme. La binarizzazione dovrebbe separare chiari e scuri. Possiamo trovare una soglia adeguata? Risposta forse no, ma proviamo.
Se andiamo a vedere l'istogramma di markers, vediamo che chiamarlo bimodale è una gran forzatura, abbiamo due picchi e una valle in mezzo. Ragionevolmente la soglia che Otsu ha preso sarà sulle code dei picchi di pixel ma nella zona della valle. Vediamo che è stata presa a 149: di fatto più della metà dei valori di pixel è stata presa come zona di scuri da mettere a neri.
In sintesi, Otsu ha fallito. Vediamo se facciamo meglio noi a mano.
Troviamo una soglia noi, sicuro più bassa di 149. Proviamo 100.
Non va bene, proviamo 55.
Non va: non è bimodale questa immagine.
In pratica, la binarizzazione funziona bene in situazioni di illuminazione adeguata, uniforme. La nostra immagine non va bene: non va bene una soglia globale. Dobbiamo trattare zone diverse in modo diverso.
Proviamo l'algoritmo di sauvola. Nella cartella abbiamo un `file.m` che lo implementa. Però la funzione `imbinarize` usa diversi approcci, tra cui '`adaptive`'.
Noi proviamo sauvola, che vuole in input l'immagine e un intorno. Cosa fa Sauvola? Quello che farebbe Otsu, ma pixel per pixel in base al suo intorno.
Cosa è successo? Ha preso gli edge, perché l'intorno è molto piccolo. Ha provato a trovare una soglia per separare quel 9 (3x3) o 25 (5x5) valori di pixel.
Quando l'intorno casca in una zona uniforme, non trova edge, e quindi non trova una soglia e quindi fa o tutto chiaro o tutto scuro (qua tutto chiaro). Quindi trova soglie adattative ma è importantissima la soglia. Deve essere pari alla più piccola zona che vogliamo segmentare (banalmente, qua un quadratino simil QRcode) perché se l'intorno è abbastanza grande, posso trovare i pixel che voglio analizzare senza cascare in zone uniformi.
Proviamo una 5x5.
Proviamo una 11x11.
Proviamo una 21x21. I quadrotti cominciano ad annerirsi, ma non vanno bene ancora.
Proviamo una 31x31. Il risultato è migliore di Otsu ma vediamo se possiamo migliorare.
Proviamo una 41x41. Quelli di luce ci sono, ma quello con la chiazza di illuminazione davanti è ancora corrotto.
Proviamo una 51x51. Quello con l'illuminazione è migliore ma ancora non va benissimo.
Ho messo io 71 perché mi piace il numero.
Perché non provo direttamente con una soglia alta? Proviamo 151x151. Finisco nel problema opposto.
Se la soglia di Sauvola inizia a diventare troppo grande, tende a diventare una soglia globale.
è molto efficiente, il problema è toonarlo bene.
Vediamo cosa farebbe `imbinarize(immagine, metodo)`.
Sicuro non usa Saurola, non è bruttissimo il risultato ma non è proprio ottimale se voglio esattamente le regioni. Se mi interessano solo i quadrotti, ha fatto un ottimo lavoro.

Se vogliamo usare `imbinarize` nel progetto, va anche bene, ma dobbiamo spiegare bene l'algoritmo e cosa c'è dentro (ovvero, come possiamo vedere nella documentazione, che usa Otsu e un algoritmo Bradley-Roth), quindi (per tutte le funzioni non basilari che usiamo) gli algoritmi che usa, l'idea che c'è alla base...
Se vogliamo usare Sauvola, troviamo il pdf doc/spiegazione nello zip.
In generale, voglio leggere un articolo (es. quello che mi spiega Bradley-Roth)? Vado su Google Scholar e cerco l'articolo. Abbiamo accesso a tantissime banche dati in quanto universitari.

2.4.3 Es3

Immagine a colori. La portiamo a livelli di grigio, guardo l'istogramma e vedo che è inutilizzabile. Perfetto esempio di istogramma non bimodale, non ho picchi. Di sicuro non riesco a trovare la soglia. Questo perché l'im a livelli di grigio qua non va bene.
Devo usare l'informazione colore, qua dicono YCbCr, ma perché? Dicono per la crominanza, perché i colori della mano sono uniformi e diversi da quelli dello sfondo.
Perché non RGB? Perché non toglie l'informazione sulla intensità della luce dalla crominanza.
Come faccio a capire cosa boh? Guardo l'istogramma dei due canali CB e CR. Quale uso? CR, perché ha un picco più alto.
Ma attenzione, il valore neutro (assenza di cromaticità) sta nel mezzo.
Boh mo si rimangia tutto e fa che non deve essere per forza bimodale l'istogramma per binarizzare. Occorre anche visualizzare le due immagini CB e CR.
Prende la figure con solo CB e con un tool della figure (`Tools > Data Tips`) va a vedere il livello che mi interessa sull'immagine. In CB è 123.
Non sempre tutto quello che sta sopra o sotto la soglia è quello che mi interessa, devo vedere la situazione.
Con `subplot(1,3,3), imshow(CB<123);` va a visualizzare l'immagine con la soglia. Fa la stessa cosa con CR, `subplot(1,3,3), imshow(CB>133);`
Prova a fare con `graythresh` ma la nostra è comunque migliore (può andare bene anche quella automatica ma ovviamente con migliorie da fare). Per inciso, graythresh trova la soglia più grande da assegnare allo sfondo. Per quello anche ad assegnare la soglia al contrario, la mano rimane comunque bianca. Ocio alle immagini con regioni più o meno equivalenti.
Proviamo a vedere lo spazio colore LAB (l luminosità) che ha un unico problema ovvero che calcola valori negativi: è un problema per l'istogramma. Anche a dividere in canali, dobbiamo servirci di imagesc.
`a` è sogliabile? No.
`b` sembra un po' più complicato. `a` sembra più semplice.
Scrive in Command Window:

```
>> figure, imshow(a>5);
>> figure, imshow(a>3);
>> graythresh(a)

ans =
```

```
    0.5078

>> figure, imshow(im2bw(a, 0.5078));
```

L'immagine ha troppo rumore. La soglia automatica è troppo bassa (voglio le regioni di `a` a valori alti). Correggo un po' la soglia.

```
>> figure, imshow(im2bw(a, graythresh(a)*1.10));
>> figure, imshow(im2bw(a, graythresh(a)*1.20));
>> figure, imshow(im2bw(a, graythresh(a)*1.50));
>> figure, imshow(im2bw(a, graythresh(a)*2.00)); % così si rompe, abbiamo dei lim
>> figure, imshow(im2bw(a, graythresh(a)*1.90));
```

Beh diciamo che qua non ha funzionato.
Ora proviamo noi.

2.4.4 Es4

Voglio trovare tutte le regioni dei cartelli blu e verdi, non quelli bianchi o quelli rossi.
Come sono fatti i cartelli verdi e blu? In quale canale immagine a livelli di grigio li trovo? Magari devo combinare info che arrivano da canali diversi. Magari no.
Posso combinare (o **addizionare**) diverse immagini binarie partizionate per aree di interesse.
Sicuro non va bene RGB perché ho provato e perdo informazione di quale cartello era bianco bianco e quale era blu e io ho fatto diventare bianco.
Uno dice che usa YCbCr e LAB insieme. Ho provato a mettere tutto in una figure 3x3 e *ATTENZIONE* `cb` va **benissimo** per i cartelli blu.
Quindi, cb (e b sogliato per prendere valori bassi bassi) va bene per i cartelli blu, cr (e a sogliato per prendere valori bassi bassi) va bene per i cartelli verdi.
In sintesi, RGB fa cagare. Lo tolgo.
Lavora su cb e cr.
Per curiosità, ha provato lui al volo HSV. Spoiler alert, inutile. La saturazione potrebbe essere utile per separare regioni colorate da regioni neutre. L'immagine della mano a colori per esempio. A casa prova.

2.4.5 Es5

Mostra come fare labeling componenti connesse per analisi di immagini.
Vogliamo trovare la regione della moneta da 50cent.
Proviamo a lavorare in bw? Prova a trovare solo le regioni delle monete, l'immagine binaria che le individua. Lo sfondo è bello chiaro quindi presumo sia circa 255, provo con 250 e poi con 255 e va benissimo il massimo.

```
% bw = im2bw(gray);
bw = gray < 255;
imshow(bw);
```

`bwlabel` è la funzione che fa labeling. `bwlabel` prende in input un'immagine binaria e etichetta le regioni bianche. Mi dà un array di interi, con lo sfondo etichettato 0 e gli altri valori sono le nostre regioni connesse. Per visualizzare il risultato uso `imagesc`, che dà come zone colorate e varie regioni: ogni etichetta

un colore. Ocio, MatLab fa cose sue e finisce a processare immagini e etichettare per colonne. Comunque, a noi non interessa l'ordine ma il valore. A noi interessa solo quella con etichetta 5. Come faccio a selezionarla? Faccio una maschera binaria che seleziona solo quella regione.
Uno gli fa che uint8 funziona anche senza concatenare, quindi:

```
%out = coins.*uint8(cat(3,mask,mask,mask));
%uno gli fa che uint8 funziona anche senza concatenare
out = coins.*uint8(mask);
```

2.4.6 Es6

Vogliamo trovare la regione della moneta più piccola.
Fa maschera con soglia 240 e normalizza.

```
mask = im2bw(coins,240/255);
```

A me esce una maschera al contrario: devo mettere `not(...)` prima.

```
mask = not(im2bw(coins,240/255));
```

Nello scritto ci verrà chiesto di risolvere un problema di elaborazione: dobbiamo descrivere il procedimento passo passo. Ogni. Singolo. Passaggio.
Voglio trovare le regioni:

```
labels = bwlabel(mask);
```

Devo capire quanti e quali etichette sono state messe dentro labels:

```
nlabels = max(max(labels));
```

Perché `max(max(labels));`? Perché lavora per colonne.

```
for r = 1 : nlabels
area = sum(sum(labels == r))
end
```

Mi dà l'area della r-esima regione. NB: anche sum lavora per colonne.
Ha aperto la doc per far vedere quante features possiamo usare per descrivere proprietà delle regioni (regionprops). Ha provato a scriverlo nella Command Window e le impacchetta in un array. Il min di questo array è l'area minima fra le aree visualizzate, ma `[M,I]=min([a.Area])` mi dà il valore minimo e l'indice a cui sta. Ora lo scrive nel codice.

```
% aree = [regionprops(labels,'Area')];
% [M,minlabel]=min(aree);
```

QUeste due righe qua sopra sono equivalenti a tutto quello che abbiamo fatto noi, da dove ho messo quelle due righe in poi.

2.4.7 Esercizi aggiuntivi

Sono solo 2 ma molto tosti, per il secondo ci fa "per semplificare fate solo la 4 connessione".

2.5 Lab5

Oggi usiamo funzioni. Il file dove viene salvata la funzione matlab deve avere lo stesso nome della funzione.

2.5.1 Es1

La prima funzione che vediamo calcola descrittori locali. Data cioè un'img in input dato un intorno quadrato di pixel, applica un vettore di features (un descrittore) ad ogni tassello. Possono essere sovrapposti, tstep mi dice di quanto spostarmi.
Posso avere un n° di tasselli inferiore al n° di pixel dell'immagine, o fare un step 1 e calcolare per ogni intorno di pixel. L'importante è che la funzione ritorni un array di valori riga.
L'iea è: prendo la mia img, usando quetsa f posso calcolare un qualcosa nell'intorno di pixel (se step è 1, è esattamente il singolo pixel, altrimenti anche un intorno di uno separato).
Lo script "`compute_local_descriptors.m`" già pronto calcola la dimensione del tassello ed estende l'immagine in modo da avere una cornice di pixel replicati per non cascare fuori da qualcosa che non esiste.
Fatto il crop dell'immagine, calcolo i descrittori locali.
`struct` mi permette di impacchettare più dati in un valore solo: nello script vediamo le righe 32-34:

```
out.descriptors = descriptors;
out.nt_rows = nt_rows;
out.nt_cols = nt_cols;
```

che mi danno i field descrittori, n° di righe e n° di colonne della struct `out`.
Ora, dobbiamo scrivere noi una funzione `compute_average_color` in un nuovo script che mi ritorna il colore medio di un intorno di pixel.
La funzione `mean` calcola la media di un array, ma mi ritorna un solo valore e noi vogliamo un vettore.
Visto che matlab lavora su vettori colonne, se facessimo un array con una colonna per R, una per G e una per B riusciamo a far calcolare la media di tutto l'array in una sola istruzione. Questo usando la funzione `reshape` che prende in input l'immagine, l'area (righe*colonne) e `ch` che è il numero di canali.
`reshape` è un jolly utile per speed up algoritmi molto lenti.
Ci siamo dimenticati un breve controllo all'inizio per evitare di perdere info all'immagine: NB. togli la riga 20 con fprintf.
Ora, nuovo script.

```
out = compute_local_descriptors(im,5,5,@compute_average_color);
```

Praticamente da im prendiamo un tassello di dim 5x5 e uno step di 5 così che siano adiacenti (uno di fianco all'altro, coprono una sola volta tutta l'immagine), per passare una funzione ad un'altra funzione uso la chiocciola.
Per accedere ad un campo interno alla variabile, usiamo la dot notation.
Vediamo che facendo 5 dim tasselli adiacenti avremo 60 righe e 90 colonne. Perché non è più partizionato? Non ho capito sta parte.

labels etichetta dati con stesso clustering. kmeans fa clustering, prende in input dati messo per riga e n° di cluster.
Mi suggerisce 5: 4 caramelle colorate + sfondo. Otteniamo un vettore di double (che in verità sono numeri interi) 5400x1.
img_labels scala l'immagine in base ai cluster in modo che siano confrontabili.
imresize(img_labels): devo replicare i valori non generarne di nuovi. Metodo nearest perché non voglio interpolare nulla.

```
img_labels = imresize(img_labels,[rows,cols],"nearest");
```

Ora andiamo a vedere immagine di partenza e immagine etichette per vedere che è successo. Non usiamo imshow ma imagesc perché vogliamo vedere le etichette.
kmeans ha una visualizzazione randomica: ogni volta che lo runno cambia su qualcosa. C'è un modo per renderlo deterministico, ma di suo è stocastico, quindi randomico. Sostanzialmente però prende i dati e li divide in 5 gruppi.
Notiamo che abbiamo un po' di problemi sui bordi, perché prob i tasselli sui bordi cascano un po' da una parte un po' dall'altra a seconda di dove si è spostata la media. Diventa ancora più evidente con un intorno 9x9 dei tasselli, con step 9 ovviamente per mantenerli adiacenti.
Nulla ci vieta di andare a fare cluster del singolo pixel, ma è quasi autolesionismo (quasi-cit Gigi). Che succ se provo? "Sbarello in senso opposto": ho troppa informazione puntuale per fare un clustering.
Ricapitolando:

tassello, intorno troppo grande: prendo troppe schifezze (come con il filtro mediano)

tassello, intorno troppo piccolo: non ho abbastanza info per fare un clustering, informazione troppo puntuale (l'unico che si salva è lo sfondo)

Quindi processare localmente l'immagine è utile ma bisogna trovare un compromesso sulla dimensione della regione. Se l'unico descrittore è la media, forse conviene cambiare spazio colore. Proviamo con questo setup a usare YCbCr.
Abbiamo fatto un po' di prove e vediamo sempre la stessa cosa: processare pixel-based quindi un intorno = 1, otteniamo un risultato **molto rumoroso**.
con un intorno ¿ 1 vediamo un'analisi statistica della distribuzione di colore su ampie aree, ovvero l'intorno del pixel.
Se davvero proprio voglio un'informazione puntuale, devo creare tanti tasselli quanti sono i pixel ma di opportune dimensioni. Perciò in questo caso non 1x1 ma 5x5, uno per pixel, con uno step = 1 sperando che tutti i pixel d'intorno siano simili.
è mooolto lento, otteniamo una tassellatura più densa e piena di informazione.
Ha ancora messo insieme rosso e arancione.
Recap: clustering tecnica automatica che raggruppa info simili fra loro in modo molto agnostico.
Dobbiamo sapere (almeno per il kmeans) a priori il numero di regioni che vogliamo ottenere. Ci sono algoritmi che lo trovano in modo automatico.

2.5.2 Es2

Stessa cosa, ma dobbiamo provare noi con la mela. Ancora una volta suggerisce di usare il valore medio di rgb.

Qua la luce dà problemi, crea un riflesso sulla mela e viene considerata come regione a parte. Stessa nota di prima: forse conviene cambiare spazio colore. Proviamo ancora con YCbCr.
Ma ancora non basta!! Manca qualche informazione: es. la `TEXTURE`. Magari la rugosità dell'arancia e della pera che sono diverse da quella della mela, aiuta.

2.5.3 Es3

Proviamo a segmentare per texture. Spoiler alert solo la luminosità dà problemi. Sta clusterizzando valori interi (immagine b&w, ho i livelli di grigio): fa schifo. Più o meno clusterizza ma non bene. Fa casino perché il pattern è più piccolo del tassello. O contrario. Passiamo da 5x5 –¿ 7x7 –¿ 11x11 con step 11 –¿ 11x11 con step 5.
Ricordiamolo, tutto questo solo con intensità luminosa.
Proviamo ad aggiungere classi. sto cercando di raggruppare per regioni omogenee, ma metti che ho zone di schifezze, posso sbatterle in una regione sola? Si chiama "classe di rigetto".
Cvd, solo il colore non basta. Serve un descrittore di texture, che **non** si può calcolare sul singolo pixel ma serve l'intorno, essendo legata a pattern ovvero variazioni di valori. Il descrittore più semplice è la deviazione standard (mi dà info su come sono distribuiti i valori dei pixel in un intorno, in una regione; se la regione è omogenea, deviazione molto bassa, se è eterogenea, deviazione alta).
Creo la funzione `compute_standard_deviation` che è uguale a `compute_average_color` che ha di diverso solamente std invece di mean. Ci assicuriamo nello script che l'input sia di double.
Ancora nada, proviamo a cambiare i descrittori. Cosa usiamo? *l'entropia*. Faccio funzione.
Ancora non basta. Soluzione suggerita in aula: uso più descrittori insieme. Che è quello che dovremo fare noi pure nel progetto. Non è univocamente descritto da nessuno dei descrittori provati ma insieme si aiutano, bisogna trovare la combinazione giusta fra i descrittori.
Come fare ciò? First of all stare **molto** attenti alla scala.
La doc dice che

```
Entropy is a statistical measure of randomness that can be used to characterize t
Entropy is defined as -sum(p.*log2(p)), where p contains the normalized histogram
```

2.5.4 Es4

2.5.5 LBP

Local Binary Pattern.
Non ho fatto in tempo a scrivere. Siamo a circa 1h e 45m di lezione.
NB: è un istogramma.
Ha raggruppato in 59 bin 59 codici sparsi in giro. Sono 11x11 valori, in 59 bin sono circa 3 valori per bin: abbiamo un istogramma molto molto sparso.
Ingrandiamo il tassello: 11x11 –¿ 31x31 con step 2.
Ma ocio: se allargo troppo il tassello, rischio di inglobare regioni diverse.
Domanda legit: ma allora non esiste una soluzione ottima? Risposta: meme di Bugs Bunny, "no". Non è come in matematica "1+1=2", ma "1+1 potrebbe fare 2".

Se l'ob di sto es è trovare dove vivono le texture perché ne devo processare una parte, il nostro risultato potrebbe andare benino rimaneggiandola un po', magari aggiungendo all'LBP l'informazione di intensità luminosa.
Però vale per questo lavoro: non ho garanzia che futuri lavori mai visti riesca a processarle adeguatamente.
In elaborazione, se voglio creare algoritmi particolarmente robusti ed efficienti, devo avere una gran buona conoscenza del dominio. Se non so bene che dati mi arrivano, non riesco a lavorare bene. Vanno sempre chiesti i dati. Se il campione di dati è sufficientemente ampio, posso provare a cercare una soluzione che si spera vada bene in tutti i casi a priori, se mi arrivano un po' alla volta ogni volta sono punto e a capo.
Il problema del nostro algoritmo nel caso specifico è che l'algoritmo è "scemo" ovvero non supervisionato. Se già comincio a dirgli "questi dati sono una regione, quelli sono un'altra" etc inizia ad essere più guidato e supervisionato (classificazione supervisionata argomento next time). Clustering esempio di classificazione non supervisionata (senza etichette).

2.5.6 Es5

Ancora bisogna combinare i descrittori, ma in alcune regioni ho lo stesso colore quindi ocio come li combini.
Es. 5-6 da fare a casa da soli.
Es.7 secondo lui è divertente. Bah. L'idea è trovare le differenze con la segmentazione, quindi in modo automatico.
NB: l'immagine comprende tutte e 4 le vignette, ma quella a sinistra sinistra è quella da tenere e processare le altre tre.
Suggerimento in aula: template_matching. Gli serve il template.
Un altro: binarizzo l'immagine e con labeling di componenti connesse supponendo di avere sempre i bordi trovo tutte le regioni nere in modo da avere la dimensione della vignetta. Ma cosa uso per trovare gli estremi della vignetta? Come sono i tratti del disegno? Connessi (buona parte di loro almeno): una volta fatto il labeling, cerco la regione più grande. Ipotizzando che il numero di pixel neri che riesco a connettere nel bordo sia più grande di tutte le regioni interne, la regione più grande è quella di bordo con tutta la parte interna connessa in teoria (perderò dettagli ma amen). NB: anche il testo fa parte, va processato. Trovo le 4 componenti connesse d'area maggiore (le 4 vignette), ne trovo gli estremi e ritaglio e trovo l'immagine completa in livelli di grigio di cui posso fare la differenza (ordinando per la x).
Un altro modo è: binarizzo, calcolo il n° di pixel neri in ogni riga e ogni colonna. Dove trovo buchi uso per tagliare verticalmente l'immagine, taglio e trovo vignetta per vignetta.

2.6 Lab6

Il lab di oggi (sto recuperando in remoto dal video 2021/22) sarà sulla **classificazione**.

2.6.1 Classificazione

La classificazione è un processo che permette di creare dei modelli matematici in grado di etichettare con un'etichetta semantica dei dati. Questo viene fatto in modo **supervisionato**. La volta scorsa abbiamo visto il *clustering*, processo tramite cui trovare zone uniformi di texture (processo di classificazione *non supervisionata*, ovvero ci diceva "etichetta 1, etichetta 2, etc" senza dirci nulla sulla zona considerata). Con la classificazione supervisionata vedremo "etichetta cane", "etichetta pelle" etc.

Algoritmi di classificazione

Ne vedremo diversi, ma tutti hanno bisogno di una cosa: un **dataset** per dare loro esempi tramite i quali etichetteranno i dati che gli diamo in input.
Partiremo con un'etichettatura semantica delle immagini del nostro dataset. Come? Con tutto quello che abbiamo visto finora, tutti quegli esercizi che ci ritornavano una maschera come risultato (es. tutti i cartelli blu, tutti i cartelli verdi, etc). Useremo tutto per automatizzare quei procedimenti di cui sopra tramite un procedimento di classificazione.
Non siamo noi a dare le soglie ma questi algoritmi le trovano in automatico basandosi sul dataset.

2.6.2 L'obiettivo di oggi

L'obiettivo di oggi è quello di classificare le regioni di pelle e non di pelle (utile per gli algoritmi di face-detection). Cominceremo valutando i classificatori (le immagini del dataset) usando le maschere sotto riportate nel pdf (immagini di groundtruth della classificazione) che qualcuno precedentemente è stato a creare. Queste groundtruth le confronteremo con l'output dei nostri classificatori.
N.B.: **non** le possiamo usare per costruire i classificatori, solo per valutarli.
Il problema allora diventa: se non posso ricorrere a queste immagini per costruire i classificatori, devo usare dei dati esterni. Vado a recuperare i dati di training (che servono per addestrare i classificatori). Noi siamo lavorando su un pixel (una terzina di valori RGB) che corrispondono a pixel di pelle, quindi dobbiamo istruirlo a riconoscere pelle.
Ma non basta: a volte i classificatori devono esaurire il dominio: abbiamo bisogno di esempi di pixel di pelle ma anche di esempi di pixel di non pelle.
N.B.: test $\neq$ training!!

2.6.3 Es1