Cabina fotografica

Federica Del Vecchio

N46004430

Photomaton - Photography Studio and Photography Lab (Paris)

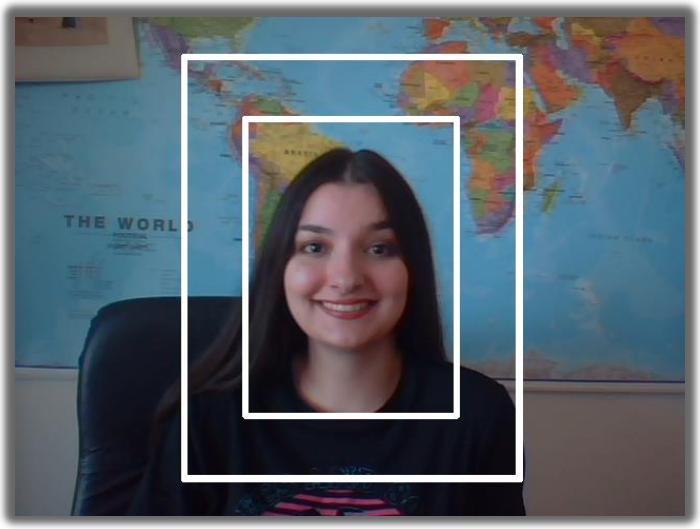
Indice

1 - Introduzione

Una cabina per fototessere rappresenta un complesso sistema multimediale per la creazione, l'elaborazione e la distribuzione di immagini fotografiche "in tempo reale". In questo testo si va a fare uno studio delle parti che compongono tale sistema, le modalità adottate nella gestione dei dati multimediali e una possibile implementazione di esse all'interno di una simulazione sul calcolatore.

In questa cabina per fototessere è possibile scattare più immagini grazie alla presenza di una fotocamera: essa è controllata proprio dall'utente che, dopo aver inserito l'importo richiesto, decide quando scattare le fotografie. Prima di questo, però, viene chiesto al cliente se egli desidera scattare delle fototessere, da usare, ad esempio, nel passaporto o nella patente, oppure semplici fotografie, magari per divertirsi tra amici. Questi due tipi di immagini richiedono due approcci, chiaramente, molto differenti. Nel primo caso, le immagini devono necessariamente presentare uno sfondo bianco e devono essere più nitide e col minor rumore possibile. Nel secondo tipo di foto, invece, non è più obbligatorio soddisfare queste richieste, però è possibile mettere a disposizione una serie di cornici "simpatiche" tra le quali scegliere per abbellire le proprie fotografie.

Il sistema implementa una serie di strategie durante lo scatto. Ad esempio, l'immagine catturata dalla fotocamera viene visualizzata, istante per istante, sopra un display su cui sono sovrapposti dei riquadri per aiutare l'utente a posizionarsi correttamente. L'utente deve spostarsi in modo tale da essere all'interno del rettangolo più esterno poiché la parte al di fuori di esso verrà, in seguito, eliminata. Infatti, le fotografie acquisite verranno, successivamente, ritagliare perché devono avere tutte un rapporto d'aspetto conforme. In aggiunta, può essere presente anche un rettangolo più interno che va a indicare la zona dell'immagine che non verrà, in seguito, coperta da una cornice o sfocata con l'obiettivo di ridurre il rumore.



Durante la fase di scatto delle foto, è possibile anche realizzare la rimozione e la sostituzione dello sfondo, in *real time*.

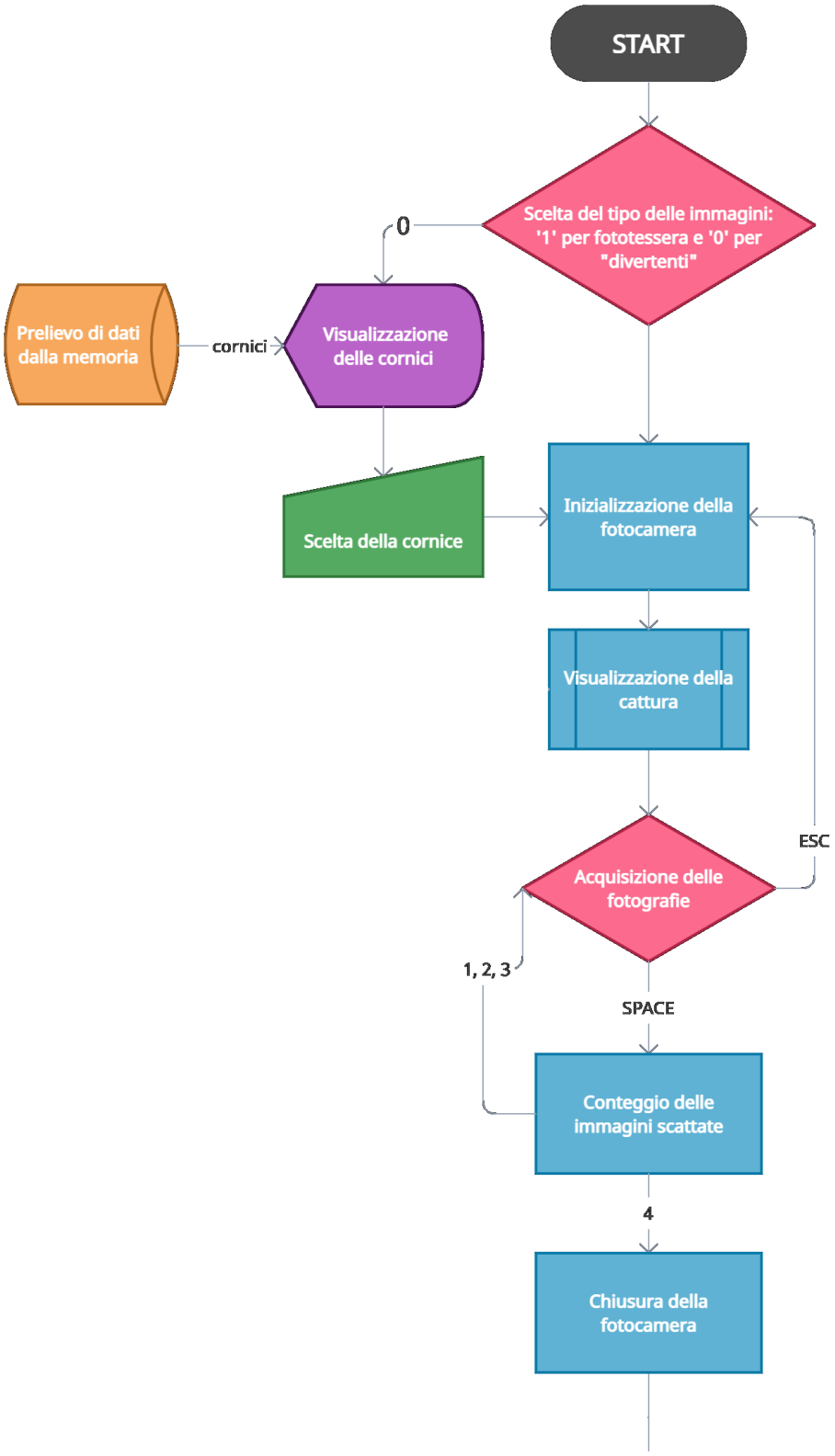
Durante la fase di post-produzione, vengono svolte tutta una serie di importanti elaborazioni su queste immagini catturare: innanzitutto, il ritaglio ma anche la modifica della luminosità, il *denoising* o l'aggiunta delle cornici.

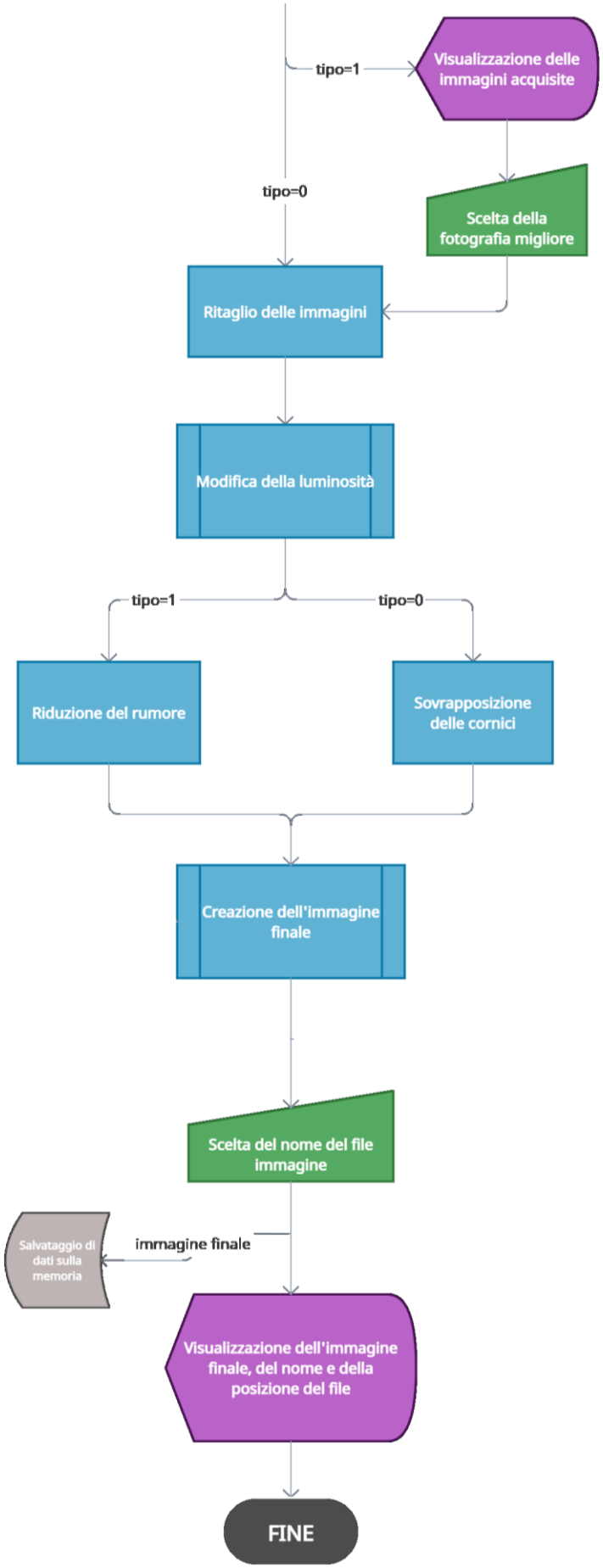
In generale, sia prima della cattura delle immagini che successivamente, è richiesto che l'utente prenda delle scelte. Questo è possibile grazie a una interfaccia, di facile utilizzo, che permette all'utente di interagire con la cabina fotografica e scambiare informazioni con essa che andranno a determinare il comportamento futuro del sistema multimediale. Ad esempio, nel caso di fototessere è richiesto che le immagini da stampare siano tutte uguali quindi, dopo lo scatto, deve essere data la possibilità all'utente di scegliere la propria foto preferita.

Infine, le fotografie modificate devono essere combinate in un'unica immagine che viene stampata dalla cabina stessa. Le foto vengono posizionate ordinatamente su uno sfondo bianco, sufficientemente distanti tra di loro. Sul bordo inferiore possono essere riportate informazioni utili come la data e l'ora attuale e dei suggerimenti sul dove andare a ritagliare.

La simulazione del sistema multimediale è implementata sotto forma di un programma nel linguaggio Python.

3 – Architettura del sistema




tipo=1
Visualizzazione delle immagini acquisite
tipo=0
Scelta della fotografia migliore
Ritaglio delle immagini
Modifica della luminosità
tipo=1
tipo=0
Riduzione del rumore
Sovrapposizione delle cornici
Creazione dell'immagine finale
Scelta del nome del file immagine
Salvataggio di dati sulla memoria
immagine finale
Visualizzazione dell'immagine finale, del nome e della posizione del file
FINE

3 – Specifica e implementazione delle metodologie utilizzate

3.1 – La rimozione e la sostituzione dello sfondo

CVzone è un pacchetto ML che facilita lo svolgimento della *face detection*, dell'*hand tracking*, della *pose estimation* ecc. e, in generale, mette a disposizione numerose funzioni per l'elaborazione delle immagini. Esso è basato sulle librerie OpenCV e MediaPipe. Il sistema sfrutta questo pacchetto per separare lo sfondo dalla persona in primo piano e sostituirlo con il colore bianco. Questo processo deve essere eseguito in tempo reale su ciò che è catturato dalla fotocamera.

Una delle funzioni che viene utilizzata è SelfieSegmentation dal modulo SelfiSegmentationModule; il valore restituito viene memorizzato all'interno della variabile segmentor. Successivamente su segmentor viene chiamata la funzione removeBG, la quale richiede tre parametri in ingresso: il primo argomento è l'immagine in input, il secondo è il colore che rappresenta il nuovo sfondo e, infine, un valore di soglia che deve essere scelto in accordo con l'ingresso. Il *threshold* impostato a 1 corrisponde alla rimozione dell'intera immagine, quindi, tenendo conto di questo, bisogna selezionare il valore ottimale, diverso per ogni contesto.

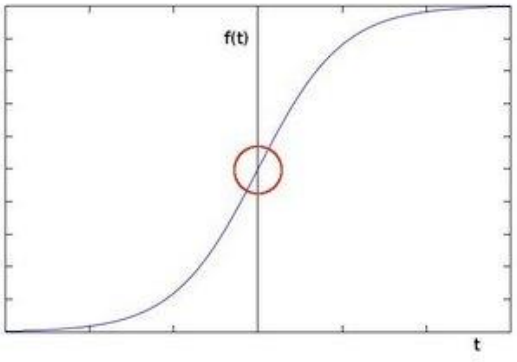
Lo sfondo può essere eventualmente sostituito con un'altra immagine o col *background* stesso ma sfocato. La SelfieSegmentation è in grado di distinguere solamente il viso e il busto di una persona a una distanza minore di 2 metri; viene messa a disposizione anche la funzione BodySegmentation, per riconoscere l'intero corpo e a maggiori distanze. Esistono ulteriori algoritmi di AI capaci di distinguere più persone, o anche animali e oggetti, tutti nella stessa immagine anche se sovrapposti. L'obiettivo principale di questi processi è operare bene anche nelle zone in corrispondenza dei bordi dei soggetti in primo piano che devono essere riconosciuti rispetto allo sfondo.

Il modo in cui opera SelfieSegmentation di CVzone è in pratica equivalente al modello generale della funzione SelfieSegmentation di Mediapipe la quale fornisce due possibili modelli: generale e *landscape*. Il modello generale opera su un tensore 256x256x3 e ne genera uno di tipo 256x256x1 che rappresenta la maschera di

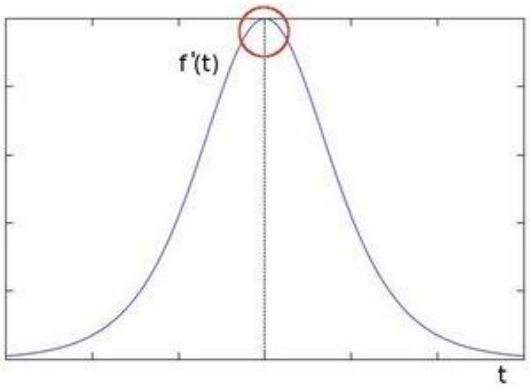
segmentazione, mentre il modello *landscape* opera su un tensore 144x256x3 ed esso è più veloce ma meno accurato. La segmentazione è basata su una *fully convolutional neural network*. Questa rete neurale, grazie a sole operazioni di convoluzione, classifica i pixel dell'immagine di input in una silhouette umana e in uno sfondo e costruisce una maschera dinamica del corpo, che rende possibile l'estrazione dello sfondo. Gli algoritmi di tipo FCN alla base di questi moduli sono spesso protetti e coperti da brevetto, quindi, non sono noti i dettagli della sua implementazione.

In alternativa, la rimozione e la sostituzione dello sfondo si possono implementare dopo lo scatto delle immagini e non durante, attraverso delle tecniche di *image processing*. Sicuramente questi processi richiedono una fase di *edge detection* ovvero di uno step che faccia il riconoscimento dei bordi degli oggetti all'interno di un'immagine. I bordi sono caratterizzati da improvvisi cambiamenti nell'intensità tra pixel vicini. Sono disponibili in OpenCV due importanti algoritmi che fanno proprio questo, ovvero il *Sobel Edge Detection* e il *Canny Edge Detection*. Prima di applicarli è buona pratica utilizzare un filtro gaussiano di sfocatura in modo da lavorare su immagini meno rumorose. Infatti, il *blurring* diminuisce la variazione di intensità tra pixel adiacenti evitando di rilevare, in seguito, bordi molto frastagliati che raramente rappresentano i veri contorni di un oggetto.

Nella figura sottostante è rappresentata una possibile funzione che descrive l'intensità dei pixel di un'immagine a una dimensione. Un bordo viene rilevato in corrispondenza di un "salto" di questa funzione poiché esso rappresenta un repentino cambiamento nell'intensità tra pixel adiacenti.



Una forte variazione nei valori è ancor più evidente se si traccia la derivata prima della funzione dell'intensità: il bordo appare, in questo caso, come un punto di massimo. Ciò implica che i bordi si vanno a rilevare in zone in cui il gradiente è più grande di un particolare valore di soglia. Il gradiente di una funzione in un determinato punto fornisce la direzione e il verso nei quali la funzione cresce più rapidamente.



La *Sobel Edge Detection* mette a disposizione due kernel di dimensioni 3x3, uno dei quali ricava gli sbalzi dell'intensità nella direzione x mentre l'altro fa lo stesso nella direzione y.

$$\begin{bmatrix} -1 & 0 & +1 \\ -2 & 0 & +2 \\ -1 & 0 & +1 \end{bmatrix} \qquad \begin{bmatrix} +1 & +2 & +1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix}$$

$$X - Direction\ Kernel \qquad Y - Direction\ Kernel$$

I gradienti dell'intensità nella direzione x e y, denominati rispettivamente G_x e G_y, si ricavano con le seguenti operazioni.

$$G_x = A * I$$

$$G_y = B * I$$

A e B sono i kernel visti in precedenza; I rappresenta l'immagine di input; $*$ è l'operatore di convoluzione. Il gradiente complessivo, ovvero nella *XY-Direction*, presenta i seguenti valori di grandezza e direzione.

$$G = \sqrt{{G_x}^2 * {G_y}^2}$$

$$\theta = \tan^{-1}(\frac{G_y}{G_x})$$

La sintassi della funzione Sobel in OpenCV è la seguente.

$$cv2.Sobel(image, depth, dx, dy)$$

depth è un valore intero che rappresenta la precisione dell'immagine di output, mentre dx e dy specificano l'ordine della derivata nella rispettiva direzione. Scegliendo opportunamente quest'ultimi parametri è possibile "amplificare" separatamente i bordi nella direzione x o y, in maniera più o meno marcata.

La *Canny Edge Detection* è un metodo alternativo, molto utilizzato perché particolarmente robusto e flessibile. L'algoritmo di Canny consiste in una sequenza di step. Dopo che è stata eseguita una *noise reduction*, l'immagine viene ulteriormente filtrata con un kernel Sobel, sia verticale che orizzontale. L'obiettivo è quello di calcolare la grandezza e la direzione del gradiente per ogni pixel; la direzione viene poi arrotondata al più vicino angolo di 45^o.

Il passo successivo consiste nello svolgere una *non-maximum suppression* dei bordi con l'intento di eliminare i pixel indesiderati ovvero quelli che non fanno parte di un effettivo contorno. Per fare ciò ogni pixel viene comparato ai suoi vicini, nella direzione positiva e negativa del gradiente: se la grandezza del gradiente del pixel considerato è maggiore dei suoi adiacenti non si fa nulla, altrimenti, se è minore, il modulo del gradiente viene impostato a 0.

Infine, le grandezze del gradiente dei pixel, indicate con G, sono messe a paragone con due valori di soglia, uno più grande dell'altro. Se G è maggiore della soglia più elevata i pixel associati vengono marcati come facenti parte di bordi "forti" ovvero predominanti nell'immagine e quindi verranno inclusi nei contorni finali. Se G, invece, è

minore del *threshold* più piccolo i pixel devono essere esclusi in quanto non particolarmente "importanti". I restanti pixel, i cui valori di G ricadono all'interno dell'intervallo tra i due valori di soglia, vengono presi solo se sono adiacenti a dei bordi "forti", altrimenti no.

La sintassi della funzione Canny di OpenCV è la seguente.

$$cv2.Canny(image, threshold1, threshold2)$$

3.2 – La fusione tra immagini

La funzione addWeighted viene messa a disposizione dalla libreria OpenCV e permette di fare il *merge* di due immagini, di uguali dimensioni, in una sola immagine. La sintassi della procedura è la seguente.

$$margeImage = cv2.addWeighted(firstImage, alpha, secondImage, beta, y)$$

firstImage è la prima immagine in ingresso mentre secondImage è la seconda; alpha e beta sono, rispettivamente, i pesi della prima e della seconda immagine; y è una costante aggiuntiva da sommare ai valori dei pixel finali. Sostanzialmente, addWeighted svolge la seguente operazione.

$$mergeImage = alpha \times firstImage + beta \times secondImage + y$$

$firstImage$

$secondImage$

$alpha = 0.5, beta = 0.5, y = 0$

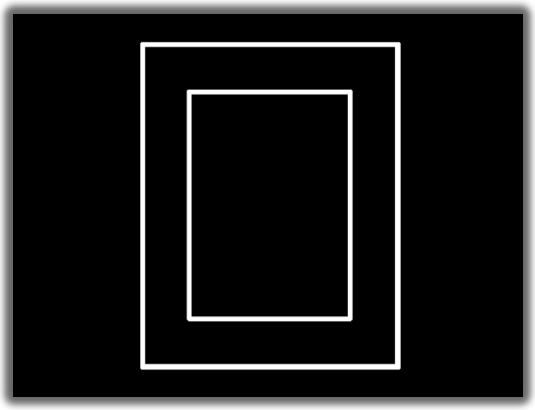
$alpha = 0.5, beta = 0.2, y = 128$

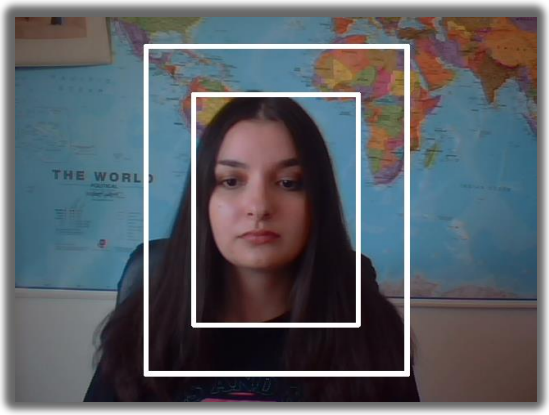
L’immagine finale presenta, ovviamente, le stesse dimensioni delle due in ingresso. Se le immagini in input non sono delle stesse dimensioni, prima della fusione una delle due deve essere opportunamente ridimensionata, ad esempio attraverso la funzione resize di OpenCV applicata nel seguente modo.

$$image = cv2.resize(image, newSize)$$

La funzione addWeighted permette di eseguire un *alpha blending* ovvero, nel fare la fusione, si tiene conto della possibile “trasparenza” di certe zone in una o in entrambe le immagini. Infatti, ai pixel di un’immagine possono essere associati, oltre ai classici canali del colore, anche altri canali che forniscono delle ulteriori informazioni. Un possibile canale aggiuntivo è, appunto, il canale alpha che permette di descrivere la trasparenza dei pixel che compongono un’immagine. I formati per le immagini TIFF, PNG e WebP supportano il canale alpha a 8 bit, quindi, è possibile rappresentare 256 livelli di trasparenza: il valore 0 indica la totale trasparenza di un pixel, 255 indica la completa opacità, i valori intermedi denotano un certo grado di trasparenza. GIF supporta un canale a 1 bit, il che significa che un'area contrassegnata come alpha può essere perfettamente trasparente o meno, senza livelli intermedi. I file di tipo JPEG, invece, non posseggono il canale di trasparenza. Si noti che tutte le immagini nel programma di simulazione sono memorizzate in formato PNG, *Portable Network Graphics.*

Queste considerazioni entrano in gioco nel sistema cabina fotografica quando si va a sovrapporre alle immagini catturate dalla camera una griglia indicativa che aiuta l’utente nello scatto delle fotografie. La griglia non è altro che un’immagine completamente trasparente a meno delle regioni in cui si trovano i due rettangoli.





$$alpha = 0.7; beta = 1; y = 0$$

L'immagine *grid* viene caricata nel programma di simulazione a partire da un file attraverso la funzione imread di OpenCV, la cui scrittura è rappresentata in seguito.

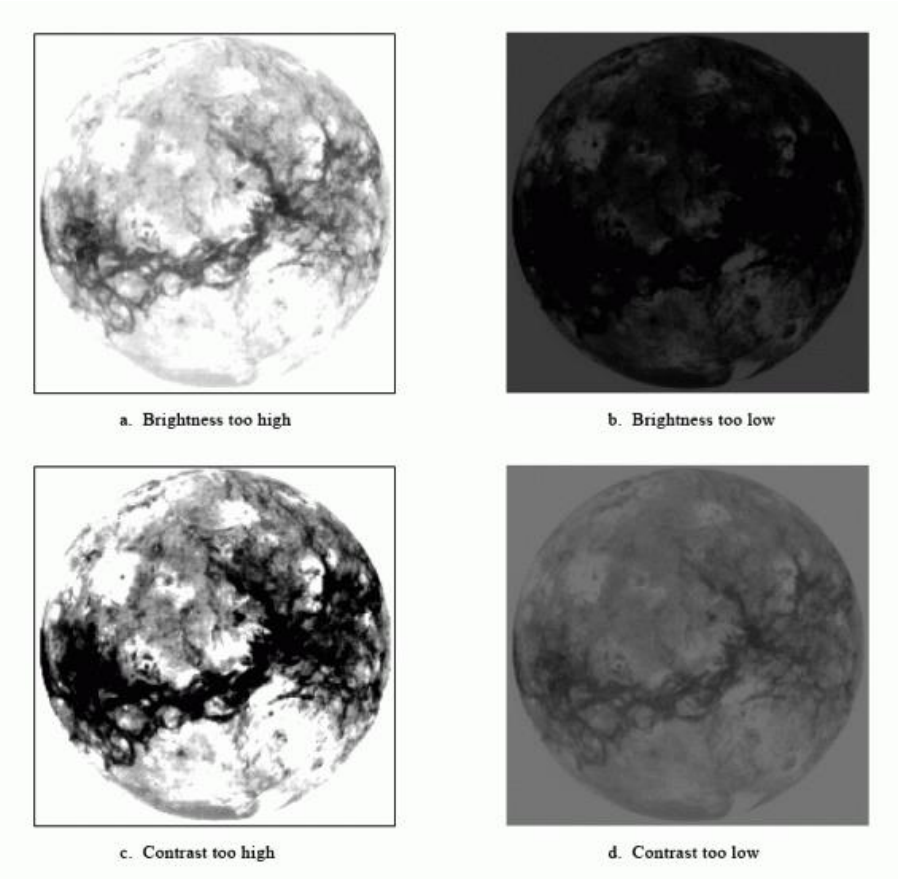
$$image = cv2.imread(pathImage, flag)$$

pathImage è il percorso del file; flag specifica il modo in cui l'immagine deve essere letta e in questo caso bisogna utilizzare cv2.IMREAD_UNCHANGED per considerare anche le informazioni riguardanti la trasparenza.

3.3 – Il Contrasto e la luminosità

Può risultare necessario, dopo aver scattato le foto, doverle renderle più "chiare" o "definite"; ciò è possibile attraverso la modifica della luminosità o del contrasto.

La luminosità si riferisce, appunto, alla luminosità o oscurità complessiva dell'immagine: aumentando la luminosità, ogni pixel diventa più chiaro e viceversa. Il contrasto è la differenza di luminosità tra diverse regioni o oggetti dell'immagine: aumentando il contrasto, le aree chiare diventano più chiare mentre quelle scure diventano più scure.



Nel modulo Python OpenCV, non esiste una funzione specifica che permette di modificare direttamente il contrasto o la luminosità dell'immagine ma esistono degli stratagemmi che si possono applicare; uno di questi consiste nell'utilizzare la funzione addWeighted nel seguente modo.

$$cv2.addWeighted(pathImage, contrast, pathImage, 0, brightness)$$

L'operazione che viene, sostanzialmente, svolta è la seguente.

$$newImage = alpha \times oldImage + beta$$

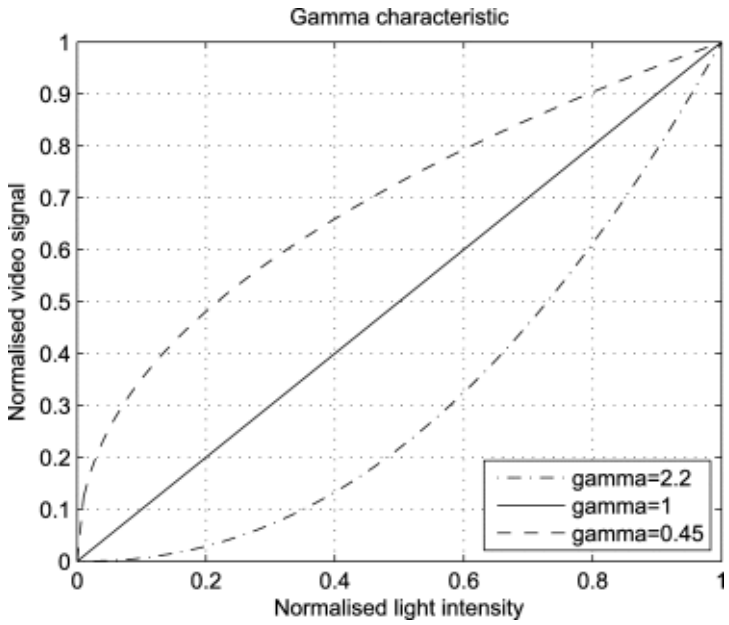
alpha rappresenta il contrasto e beta la luminosità. Per alpha uguale a 1 e beta a 0 non viene apportato alcun cambiamento nell'immagine; per alpha compreso tra 0 e 1 vi è una diminuzione del contrasto, mentre per alpha maggiore di 1 si ha un aumento del contrasto; infine, il range tipico in cui si prende beta va da -127 a +127, ovviamente un valore di beta alto corrisponde a una elevata luminosità e viceversa.

Nella simulazione non è implementata questa soluzione ma quella della *gamma correction* che permette di cambiare la luminosità su scala logaritmica e non lineare, in accordo alla percezione visiva. Infatti, l'occhio umano non funziona in maniera identica ai dispositivi di acquisizione delle immagini digitali. Infatti, l'essere umano è più sensibile nel percepire i cambiamenti che si verificano nei toni scuri rispetto a cambiamenti simili che si verificano nei toni chiari, mentre le fotocamere, scanner ecc. lavorano in modo lineare. La correzione gamma svolge la traduzione tra sensibilità digitale e sensibilità umana.

A livello fisico i colori di una immagine che deve essere visualizzata, ad esempio su un monitor, sono in origine segnali elettrici caratterizzati da una determinata tensione. In realtà, la luce percepita relativa a questi segnali non è proporzionale alla tensione, come ci si aspetterebbe, ma alla tensione elevata a una certa potenza detta gamma. Questa è proprio la causa della distorsione di come appaiono i colori, motivo per cui, prima della trasmissione allo schermo, sono effettuate operazioni come la seguente.

$$O = \left(\frac{I}{255}\right)^{\frac{1}{\gamma}} \times 255$$

I è il valore di input del pixel, O il valore di output e γ è proprio la gamma. Nell'esempio precedente si è considerato il semplice caso in cui i valori che possono assumere i canali dei pixel sono compresi tra 0 e 255.



La correzione gamma può essere implementata grazie all'utilizzo di una tabella di *lookup* (LUT), la quale mappa i valori di input dei pixel sui valori di output. La tabella si costruisce sfruttando funzioni messe a disposizione dalla libreria NumPy. Questa libreria permette di considerare le immagini come vettori e matrici e lavorarci sopra in maniera efficace e veloce. I passaggi da svolgere per creare la LUT sono i seguenti.

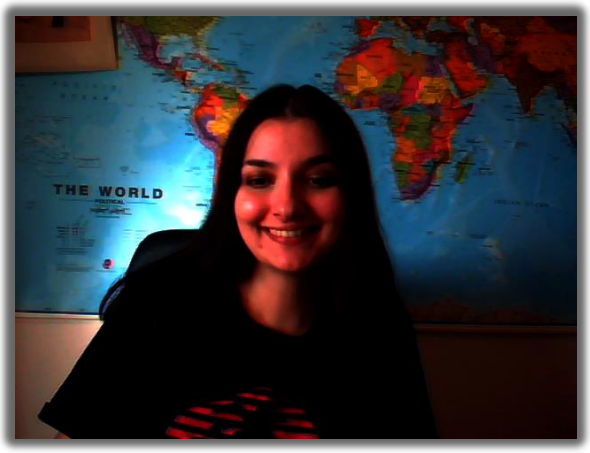
$$values = numpy.arange(0, 256)$$

$$lut = numpy.uint8(255 * numpy.power((values/255.0), gamma))$$

Il metodo arange restituisce i valori equidistanti entro un determinato intervallo. La seconda istruzione, invece, svolge una conversione dei dati specificati nel tipo numpy.uint8 ovvero *unsigned 8-bit integer*. La funzione power esegue una semplice elevazione a potenza. In conclusione, si fa una *lookup table transform* applicando la funzione LUT, fornita da OpenCV.

$$image = cv2.LUT(image, lut)$$

Il valore di gamma si prende normalmente uguale a 2.2 ma, nel caso del sistema cabina fotografica, si può scegliere in modo tale da aumentare o diminuire la luminosità delle proprie fotografie a piacere. Se il valore di gamma è minore di 1 l'immagine sarà più scura, se gamma è maggiore di 1 l'immagine sarà più chiara, altrimenti, per gamma uguale a 1 non si osserva nessun cambiamento.

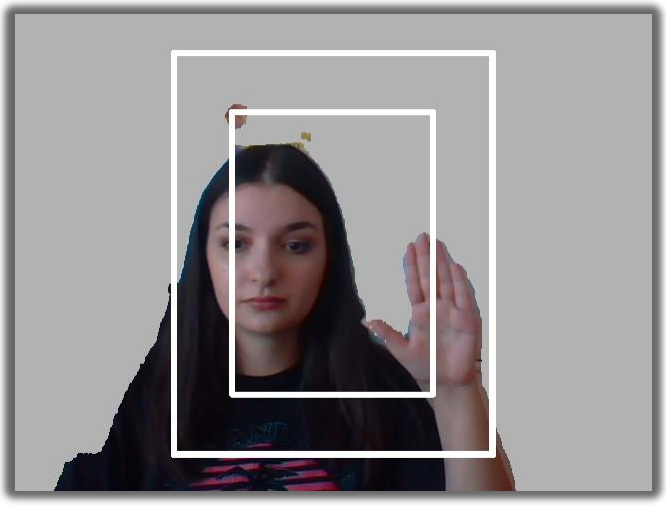


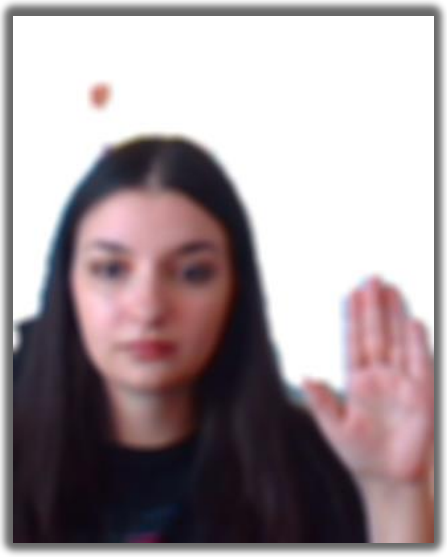
$\gamma = 3$

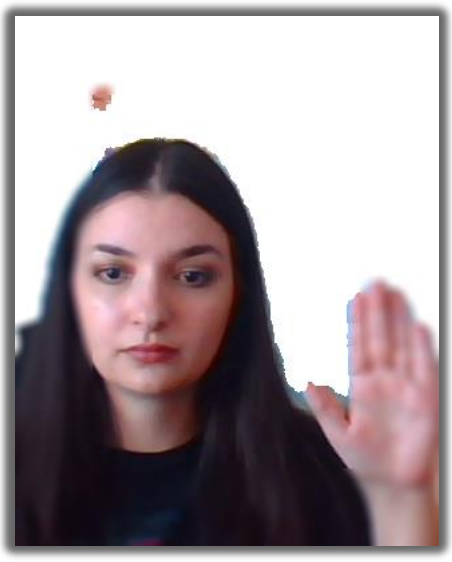
$\gamma = 0.3$

3.4 – La sfocatura gaussiana

Per la riduzione del rumore nelle immagini uno dei possibili modi consiste nell'applicare un semplice algoritmo di sfocatura gaussiana. Nel sistema cabina fotografica, il filtro di *blur* viene utilizzato solamente sulle parti esterne dell'immagine in modo da lasciare intatta l'area centrale dove, presumibilmente, si trova il viso di una persona.

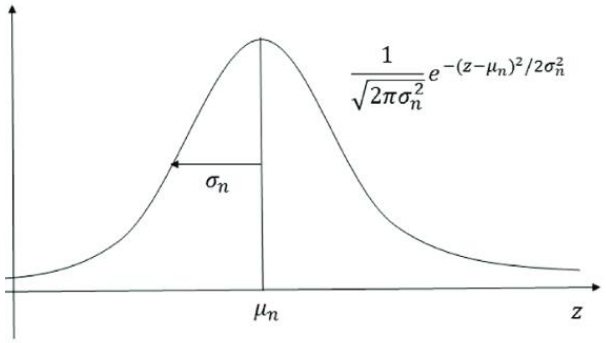


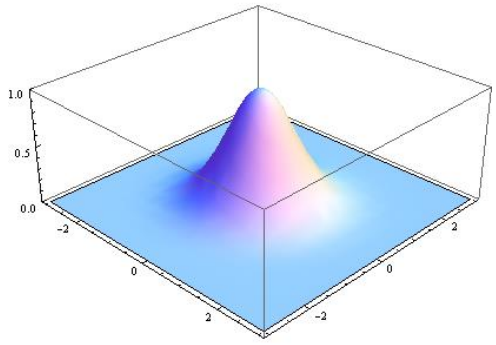




La sfocatura si realizza andando a prendere come nuovo valore di un pixel la media dei valori dei pixel circostanti e del vecchio valore del pixel stesso. Ovviamente più è grande il range dei pixel vicini che si vanno a considerare maggiore è l'effetto di *blur*. Sarebbe irragionevole calcolare una semplice media perché, essendo le immagini continue, più vicini sono i punti più stretta è la relazione tra di essi; quindi, è meglio eseguire una media ponderata nella quale si dà peso maggiore ai pixel più vicini. La distribuzione normale o gaussiana è il modello di distribuzione del peso che si usa in questo caso.

La funzione di densità della distribuzione normale è anche chiamata funzione gaussiana e il grafico è caratterizzato da una forma simmetrica "a campana", sia in una che in due dimensioni. σ è la deviazione standard, mentre σ^2 è la varianza.





La formula bidimensionale della funzione gaussiana è la seguente.

$$G(x,y) = \frac{1}{2\pi\sigma^2}\, e^{-\frac{x^2+y^2}{2\sigma^2}}$$

Questa funzione è quella che viene usata per calcolare il "peso" che deve avere ogni punto nella media. Si consideri un esempio in cui le coordinate del punto centrale sono (0,0) e il raggio del *blur* è 1.

(-1,1)	(0,1)	(1,1)
(-1,0)	(0,0)	(1,0)
(-1,-1)	(0,-1)	(1,-1)

Fissato σ a 1.5, la matrice sottostante mostra il peso che deve assumere ogni punto.

0.0453542	0.0566406	0.0453542
0.0566406	0.0707355	0.0566406
0.0453542	0.0566406	0.0453542

La somma dei nove valori rappresentati nella tabella è 0.4787147; siccome, però, si vuole che questa somma sia uguale a 1 si divide ogni valore proprio per 0.4787147.

0.0947416	0.118318	0.0947416
0.118318	0.147761	0.118318
0.0947416	0.118318	0.0947416

Si consideri la semplice scala dei *gray-value* che va da 0 a 255. In seguito, per esempio, sono mostrati dei possibili valori di grigio associati ai pixel.

12	15	26
37	23	8
23	19	31

Ogni valore di grigio di un punto viene, quindi, moltiplicato per il "peso" corrispondente a quella posizione.

1.1368992	0.177477	2.4632816
4.377766	3.398503	0.946544
2.1790568	2.248042	2.9369896

Sommando i risultati soprastanti si ottiene il nuovo valore del pixel centrale ovvero 19.8645592; ovviamente questo processo si deve ripetere per tutti i punti dell'immagine. Se un pixel è sul bordo una possibile soluzione consiste nel copiare i punti esistenti da un lato nelle posizioni corrispondenti dall'altro per simulare una matrice completa.

OpenCV mette a disposizione una funzione che svolge questo procedimento nella sua interezza; la sintassi di GaussianBlur è la seguente.

$$cv2.GaussianBlur(pathImage, kernelSize, sigmaX)$$

kernelSize è la dimensione del kernel e specifica il grado di sfocatura; sigmaX è una variabile che rappresenta la deviazione standard del kernel gaussiano in direzione x. Nel programma di simulazione si ha un kernelSize uguale a (9,9) e come sigmaX si è scelto cv.BORDER_DEFAULT.

3.5 – La sovrapposizione di un'immagine sopra un'altra

La funzione cvtColor di OpenCV permette di convertire un'immagine da uno spazio di colore a un altro.

$$cv2.cvtColor(pathImage, code)$$

code è il codice che definisce quale tipo di conversione si richiede, Questa operazione è utilizzata per convertire le immagini scattate dall'utente che sono nel *colorspace* BGR, come di default per OpenCV, in BGRA; ovvero oltre ai tre canali legati al colore, blu, verde e rosso, si ha anche il canale alpha. Per fare ciò viene scelto come codice della tabella di conversione cv2.BGR2BGRA. Dopo la conversione dello spazio di colore, il valore del canale di

trasparenza per ogni pixel viene impostato uguale a 1 per indicare che l'immagine è perfettamente opaca.

Questo processo va ad "aggiungere" il canale alpha alle immagini scattate dalla fotocamera ed è necessario perché, in seguito, verranno fatte operazioni che comprendono sia queste che altre immagini, le quali possiedono "naturalmente" il canale alpha. Saltare questo passaggio può generare errori dovuti al fatto che le funzioni vanno a lavorare su immagini che presentano diverse caratteristiche.

A questo punto è necessario osservare come svolgere la completa sovrapposizione di un'immagine di *foreground* che presenta delle trasparenze sopra a un'immagine perfettamente opaca di *background*. L'operazione che, in pratica, deve essere eseguita è seguente.

$$overlayImage = alpha \times foregroundImage + (1 - alpha) \times backgroundImage$$

Con alpha ci si riferisce al canale di trasparenza dell'immagine di *foreground*; quindi, con alpha uguale a 0 il pixel dell'immagine finale è uguale a quello dello sfondo, con alpha uguale a 1 il pixel è quello dell'immagine in primo piano, con alpha compreso tra 0 e 1 il pixel non è altro che una combinazione.

Questo procedimento viene usato nel sistema cabina fotografica immaginato per poter sovrapporre completamente delle "cornici divertenti" sulle fotografie. Le cornici sono delle immagini che presentano alcuni pixel opachi e altri completamente trasparenti; queste sono, quindi, proprio le immagini di *foreground*, mentre gli scatti rappresentano il *background*.

Nel programma di simulazione, viene creato un array NumPy della stessa dimensione dell'immagine dello sfondo e tutti i valori vengono impostati a 0; per fare ciò viene usato il metodo zeros_like la cui sintassi è descritta successivamente.

$$shapes = numpy.zeros\_like(backgroundImage, dataType)$$

Il tipo del dato è, in questo esempio, numpy.uint8. Inoltre, si memorizzano le dimensioni dell'immagine di *overlay* in delle variabili; l'operazione si può eseguire nel seguente modo.

$$height\,, width = foregroundImage.shape[:2]$$

Adesso viene creata una maschera utilizzando il vettore appena generato, *shapes*, e l'immagine da sovrapporre. I passaggi da svolgere sono i successivi.

$$shapes[backgroundImage.shape[0] - height:, backgroundImage.shape[1] - width:] = foregroundImage$$

$$mask = shapes.astype(bool)$$

La funzione astype svolge una copia dell'array facendo, allo stesso tempo, il *casting* a un altro tipo specificato, in questo caso il tipo booleano. Infine, tenendo conto di questa *mask*, si applica addWeighted con la seguente scrittura.

$$backgroundImage[mask] = cv2.addWeighted(backgroundImage, 0, foregroundImage, 1, 0)[mask]$$

4 – Bibliografia

MediaPipe Selfie Segmentation [https://google.github.io/mediapipe/solutions/selfie_segmentation.html]

OpenCV Background Subtraction, 2022 [https://www.delftstack.com/howto/python/opencv-background-subtraction/]

Edge Detection Using OpenCV [https://learnopencv.com/edge-detection-using-opencv/#how-are-edges-detected]

OpenCV Documentation, *Adding (blending) two images using OpenCV* [https://docs.opencv.org/3.4/d5/dc4/tutorial_adding_images.html]

OpenCV Documentation. *Changing the contrast and brightness of an image!* [https://docs.opencv.org/3.4/d3/dc1/tutorial_basic_linear_transform.html]

Adrian Rosebrock, *OpenCV Gamma Correction*, 2025 [https://pyimagesearch.com/2015/10/05/opencv-gamma-correction/]

Ruan Yifeng, *Algoritmo di sfocatura gaussiana*, 2012 [http://www.ruanyifeng.com/blog/2012/11/gaussian_blur.html]

OpenCV Documentation, *Smoothing Images* [https://docs.opencv.org/4.x/d4/d13/tutorial_py_filtering.html]

Alpha blending and masking of images with Python, OpenCV, NumPy, 2019 [https://note.nkmk.me/en/python-opencv-numpy-alpha-blend-mask/]

Transparent overlays with Python OpenCV, 2022 [https://www.geeksforgeeks.org/transparent-overlays-with-python-opencv/]

NumPy Documentation [https://numpy.org/doc/stable/index.html]