UNIVERSITA' DEGLI STUDI DI BARI

Scuola di Scienze e Tecnologie

Dipartimento di Informatica

TESI DI LAUREA TRIENNALE IN

SISTEMI COOPERATIVI

Sistema software per l'individuazione e la segmentazione della congiuntiva palpebrale tramite tecniche di machine learning e image analysis

Relatore:

Prof. **Giovanni Dimauro**

Laureando:

Luigi Baldari

ANNO ACCADEMICO 2016/2017

Sommario

Abstract

Esaminare il livello di emoglobina nel sangue è il procedimento standard per diagnosticare l'anemia. In letteratura, vengono descritte molte ricerche che hanno l'obiettivo di diagnosticare l'anemia con metodi non invasivi, come, ad esempio, l'analisi di immagini digitali della congiuntiva per stimare il pallore.

In questo scenario, il presente lavoro di tesi ha l'obiettivo di individuare un procedimento ottimale per l'individuazione e segmentazione automatica di sezioni rilevanti della congiuntiva.

Nel corso del lavoro di tesi è stato studiato ed implementato un algoritmo di object detection che sfrutta tecniche di *computer vision* e *machine learning,* per individuare la regione di interesse. Sono stati poi studiati e implementati algoritmi di *image analysis* per la segmentazione della regione di interesse e l'individuazione definitiva della congiuntiva. Il software progettato e realizzato, ha permesso lo studio di una sequenza ottimale di tecniche di *image analysis* e dei risultati da esse fornito.

L'area di interesse segmentata è stata ottimizzata in termini di correlazione con il valore di Hb (misurato con tecniche di laboratorio tradizionali), con lo scopo di definire anche i parametri da utilizzare nell'individuazione della congiuntiva, per la rilevazione di emoglobina nel sangue con tecniche non invasive.

In conclusione, verranno riportati interessanti risultati sperimentali.

Capitolo 1

Introduzione

1.1 Presentazione del contesto

Si definisce anemia la riduzione patologica dell'emoglobina (Hb) al di sotto dei livelli di normalità, che determina una ridotta capacità del sangue di trasportare ossigeno. L'emoglobina è una proteina presente nei globuli rossi del sangue [1].

L'anemia è il risultato di una vasta gamma di cause, delle quali, la più significativa, è la carenza di ferro (*Iron deficiency Anemia* o IDA). Si stima che circa il 50% dei casi di anemia sia dovuta alla carenza di ferro. Si stima, inoltre, che l'anemia sia un problema di salute che interessi circa il 30% della popolazione mondiale.

Un alto tasso di anemia nella popolazione ha conseguenze sia sul tasso di mortalità che sullo stato fisico e mentale di chi ne è affetto.

La concentrazione di Hb è l'indicatore più affidabile per diagnosticare l'anemia. La normale distribuzione di Hb varia a seconda dell'età, sesso e stato fisico. La TABELLA 1.1 riporta i valori minimi di Hb usati per diagnosticare l'anemia.

Età/sesso	Soglia Hb (g/dL)
Bambini (0,5 – 5 anni)	11,0
Bambini (5 – 11,99 anni)	11,5
Ragazzi (12 – 14,99 anni)	12,0
Donne non incinta (≥ 15 anni)	12,0
Donne incinta	11,0
Uomini (≥ 15 anni)	13,0

TABELLA 1.1: soglie Hb

Il metodo più utilizzato attualmente per la rilevazione dell'emoglobina avviene tramite prelievo del sangue. Questo procedimento, oltre che essere dispendioso in termini di tempo, sottopone il paziente al rischio di infezioni. Le persone anemiche, che hanno bisogno di effettuare più prelievi del sangue per monitorare il loro livello di Hb, accusano maggiormente i disagi citati precedentemente [2].

Lo studio di metodi alternativi per la rilevazione dell'emoglobina, ha larghe applicazioni, soprattutto in paesi con risorse limitate.

1.2 Metodi non invasivi

Nel corso degli anni, diversi metodi non invasivi sono stati studiati per la rilevazione dell'emoglobina. Osservare il pallore della congiuntiva palpebrale del paziente, ad esempio, può fornire informazioni utili per la diagnosi dell'anemia [3].

Nello studio svolto da Sheth [4], la presenza del pallore congiuntivale è stato utilizzato per la determinazione del livello di Hb. Gli autori hanno raggiunto il 95% di accuratezza basandosi sul pallore della congiuntiva e considerando una soglia di Hb pari a 90 d/L per escludere o meno la presenza di anemia grave.

Sono state considerate anche altre parti del corpo per la diagnosi dell'anemia. Nella pubblicazione di Spinelli [5], ad esempio, è stato considerato il pallore del palmo della mano invece di quello della congiuntiva, dimostrando che, quest'ultimo, ha una correlazione maggiore. Infatti, considerando altre parti del corpo, bisogna tener conto di altri fattori che potrebbero essere un ostacolo ai fini della valutazione oggettiva dell'emoglobina (come, ad esempio, la pigmentazione della pelle).

Nel lavoro di Sanchez-Carrillo [6], invece, è stato progettato uno strumento colorimetrico per la quantificazione non invasiva dell'emoglobina, con tonalità di colore somiglianti a quelle osservate nella congiuntiva. Si sono ottenuti buoni risultati per la determinazione della concentrazione di emoglobina per i pazienti con valori fino a 13 g/dl.

Nel lavoro di Suner [7] si sono analizzate immagini digitali di congiuntive e, in particolare, le diverse tonalità di rosso, verde e blu presenti nelle foto. L'obiettivo di Suner [7] è quello di implementare uno strumento portabile e affidabile per la stima dell'emoglobina, da utilizzare nei paesi in via di sviluppo o dove le risorse sono scarse.

Le foto di 44 pazienti sono state utilizzate nella fase di sviluppo, e 19 per la fase di valutazione. Le immagini sono state scattate con una fotocamera digitale,

includendo nelle foto una carta fotografica standard con scala di grigi del 18%, per uniformare le diverse condizioni di esposizione della luce.

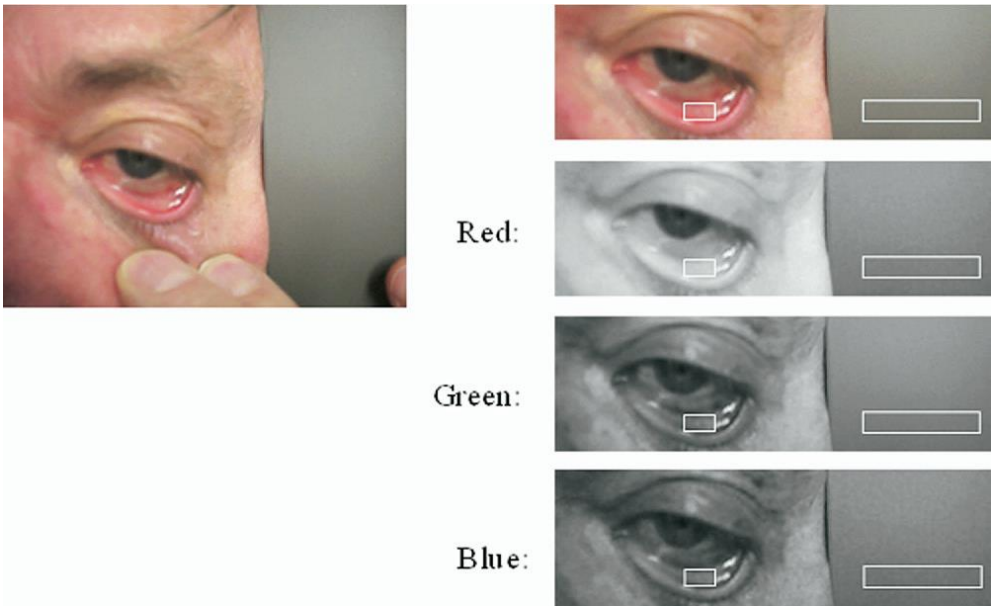

FIGURA 1.1: tratta dalla pubblicazione di Suner [7];
sulla sinistra vi è la foto di un soggetto ottenuta duarante lo studio, in cui si mostra la congiuntiva e la carta fotografica a scala di grigi; sulla destra, vi è l'immagine ritagliata separata nei 3 canali rosso, verde e blu.

Le immagini sono state poi manipolate da un algoritmo costruito dal *National Institutes of Health* (NIH). Nella fase di valutazione, si è cercato di costruire un algoritmo con un'alta correlazione tra il colore della congiuntiva e l'emoglobina misurata.

La Figura 1.1 riassume l'algoritmo utilizzato da Suner [7]. L'algoritmo ha come risultato finale l'emoglobina stimata, che è stata poi comparata con quella misurata tramite tecniche di laboratorio tradizionali. La correlazione tra i due valori è stata misurata tramite la correlazione di Pearson, che è risultata pari a 0.6 (correlazione moderata).

Il sistema valutato nel lavoro di Suner [7] è stato ospitato in un *Personal Digital Assistant* (PDA), e può essere utilizzato in luoghi dove le risorse di laboratorio sono assenti o limitate.

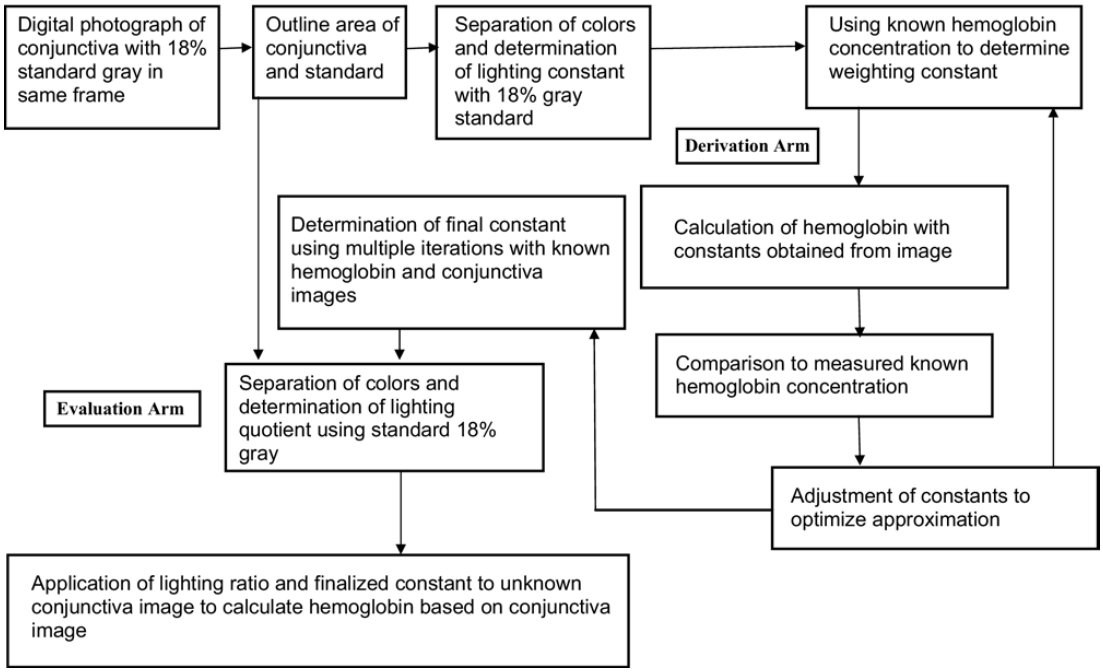

FIGURA 1.2: algoritmo di Suner [7]

Come proseguo dello studio svolto da Dimauro in [23] questo lavoro si focalizzerà sull'individuazione e segmentazione della congiuntiva, in particolare modo su quella palpebrale. La congiuntiva è una membrana mucosa, che ricopre il bulbo oculare e la parte interna delle palpebre; ha la funzione di proteggere il bulbo oculare, soprattutto la cornea (benché la sua faccia anteriore sia sprovvista del rivestimento congiuntivale), nonché di facilitare il suo scorrimento e di quello delle palpebre nelle fasi di ammiccamento, mediante la secrezione della componente mucinica del film lacrimale.

La congiuntiva può essere suddivisa in 3 parti anatomiche:

- Congiuntiva bulbare (o sclerale): è la parte di tonaca congiuntivale che si applica sul bulbo oculare e ricopre la superficie anteriore della sclera, ad eccezione della porzione corneale. Costituita da epitelio pavimentoso, la congiuntiva bulbare poggia debolmente su una lamina propria connettivale lassa. La tonaca congiuntivale bulbare è liscia, molto sottile e così trasparente da lasciare intravedere il colore bianco della sclerotica ed i vasi congiuntivali e ciliari anteriori. In posizione mediale, poi, la congiuntiva tarsale accoglie i puntini lacrimali superiore ed inferiore, che rappresentano l'inizio delle vie lacrimali.

- Congiuntiva palpebrale (o tarsale): costituita da epitelio cilindrico, è una membrana sottile, trasparente, rossa o rosea. Facendo seguito alla cute, la tonaca congiuntivale inizia in corrispondenza del margine libero delle palpebre, quindi riveste la faccia posteriore dei tarsi, ai quali aderisce strettamente.
- Congiuntiva dei fornici: a livello dello spazio tra palpebre e globo oculare, la membrana congiuntivale si piega e riveste i fornici superiore ed inferiore, permettendo libertà ai movimenti del bulbo [24].

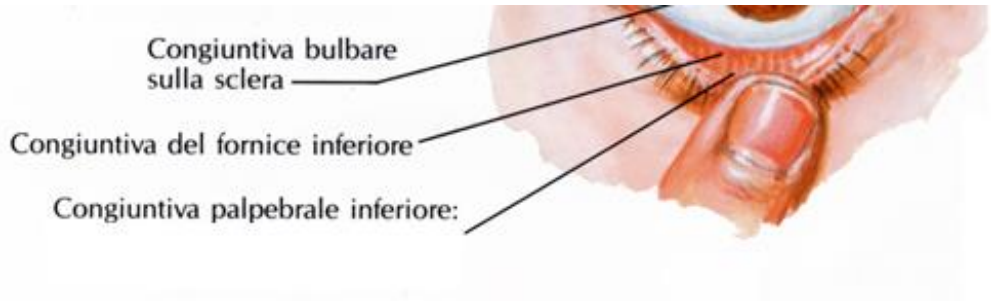

FIGURA 1.3: illustrazione della congiuntiva

1.3 Acquisizione dei dati

La progettazione e la realizzazione di un dispositivo di acquisizione non invasivo, è stata già studiata da Petrosino [8] nel suo lavoro di tesi, e presentato al Festival dell'Innovazione in Sanità Pubblica a Pisa (ottobre 2017).

L'elemento principale del dispositivo è una lente macro che viene adattata alla fotocamera di uno smartphone ed effettua foto in primo piano con zoom elevato. Si è scelta una lente macro con zoom di 10X; essa è provvista di *hood* (cappuccio) satinato che aiuta a prevenire riflessi indesiderati, provenienti da sorgenti luminose esterne. Si è poi scelto di utilizzare un oculare in gomma nera per accomodare gentilmente il dispositivo sul volto del paziente.

Per quanto riguarda l'illuminazione è stata impiegata una scheda USB provvista di 3 led SMD 2835, colore bianco freddo, erogante 22 Lumen, per un consumo totale di 0.2 W. Dopo alcuni test, si è scelto di oscurare 2 dei 3 led presenti sulla scheda (scendendo quindi a circa 7 Lumen effettivi), in quanto, l'eccessiva illuminazione,

produceva un numero elevato di *bright spot* (riflessi) nella fotografia scattata, andando a ridurne l'informazione.

Il corpo illuminante è posizionato tra l'*hood* della lente macro e l'oculare in gomma.

L'alimentazione può essere fornita con due modalità a seconda delle possibilità dell'utilizzatore. Il corpo illuminante è collegato ad una piccola prolunga USB che potrà essere collegata alla fonte di alimentazione via micro-USB (necessario supporto alla modalità OTG) o via USB.

I vari componenti del dispositivo sono stati assemblati con del nastro isolante nero.

Prima di effettuare le foto ai pazienti, si sono eseguite alcune foto di test. Si è arrivati quindi alle seguenti conclusioni:

- Affinché la congiuntiva palpebrale sia ben visibile, al paziente viene chiesto di esporla abbassandola tramite un dito; per far aderire al meglio il dispositivo, viene chiesto di non usare il polpastrello ma di abbassarla posizionando il dito parallelamente sotto la palpebra.
- La luce erogata dal dispositivo viene posizionata in modo da illuminare nella direzione dell'occhio messo a fuoco. Quindi, il led, viene posizionato verso sinistra per l'occhio sinistro e verso destra per l'occhio destro. In questo modo, inoltre, il numero di *bright spot* è ridotto al minimo.

FIGURA 1.4: dispositivo di acquisizione

Tramite il nuovo dispositivo di acquisizione, sono state scattate le foto delle congiuntive sia di pazienti sani che di pazienti anemici (tenendo traccia dei loro valori di Hb). Le foto sono state acquisite con uno smartphone Huawei P9 Lite, un Huawei P7 e un iPhone 6.

Le congiuntive sono state acquisite presso il centro trasfusionale del Policlinico di Bari, l'Istituto Tumori Giovanni Paolo II di Bari e l'ospedale Di Venere Carbonara di Bari. Ad ogni soggetto si è fatto firmare un modulo di consenso informato per la partecipazione allo studio. Successivamente si è chiesto al paziente di abbassare la palpebra inferiore esponendo la congiuntiva.

Si sono acquisite in totale 65 fotografie di cui 12 risultano avere un valore di Hb al di sotto della norma.

Capitolo 2

Concetti chiave

2.1 Computer Vision

La visione è forse il senso più importante che l'uomo possiede; l'occhio raccoglie una banda di radiazioni elettromagnetiche rimbalzate su diverse superfici, e provenienti da fonti luminose diverse. Il cervello poi, elabora queste informazioni formando il quadro della scena come noi lo percepiamo.

Per *Computer Vision* (o Visione Artificiale) si intende l'insieme di processi che mirano a creare un modello approssimato del mondo reale, partendo da immagini bidimensionali. La disciplina studia come abilitare i computer alla comprensione e all'interpretazione delle informazioni visuali presenti in immagini o video.

Possiamo quindi dire che, la Visione Artificiale, si occupa dell'analisi di immagini digitali. L'analisi è finalizzata a scoprire **cosa** è presente nella scena e **dove** si trova. Da questa definizione possiamo distinguere due principali filoni all'interno della Visione Artificiale: *Object Classification* e *Object Detection*.

Per *Object Classification* (o *Object Recognition*) si intende l'abilità del calcolatore nel classificare le immagini, ovvero nel riconoscere all'interno delle immagini la presenza o assenza di determinati oggetti.

Per *Object Detection*, invece, oltre alla classificazione, il calcolatore si deve occupare anche della localizzazione dell'oggetto all'interno della scena.

Gli algoritmi di *Object Detection* hanno diverse applicazioni, come ad esempio: *face detection, pedestrian detection, eye detection* ed altri.

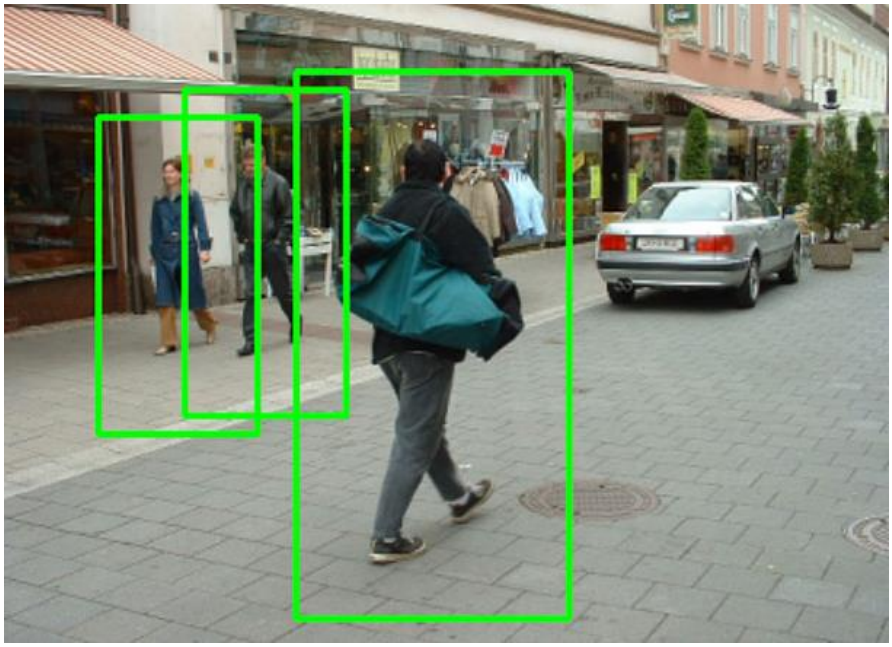

FIGURA 2.1: algoritmo *pedestrian detection* in azione

2.2 Machine Learning

I moderni sistemi di *Computer Vision* sono spesso accompagnati da modelli di apprendimento automatico (*machine learning*).

Ci sono diverse definizioni del *Machine Learning*: "*the field of study that gives computers the ability to learn without being explicitly programmed*" (Arthur Samuel 1959).

Una definizione più formale è la seguente: "*A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P, if its performance at tasks in T, as measured by P, improves with experience E*" (Tom Mitchell 1998).

Nel *machine learning* si tenta di ricreare il processo di apprendimento, tramite algoritmi che danno alla macchina la capacità di imparare senza essere programmata esplicitamente. Si prendono degli esempi, si estraggono i *pattern* (cioè gli schemi che li contraddistinguono) e li si utilizzano per fare previsioni a proposito di nuovi esempi.

Distinguiamo all'interno del *machine learning*, l'apprendimento supervisionato e non supervisionato:

- Nell'apprendimento supervisionato (*supervised learning*), al software viene inizialmente fornito un insieme di dati, costituiti da una serie di coppie input-output contenente la logica di funzione;
- Nell'apprendimento non supervisionato (*non supervised learning*), la logica di funzione non viene fornita inizialmente alla macchina, ma è demandato alla macchina stessa il compito di trovarne una a partire da una serie di dati di input.

2.3 Image Processing

Un'immagine può essere definita come una funzione bidimensionale $f(x, y)$ dove x e y sono le coordinate di ciascun pixel dell'immagine, e l'ampiezza di f per ogni coppia di coordinate (x, y) è chiamata intensità di grigio.

L'*Image Processing* o *Digital Image Processing* è una disciplina che effettua operazioni sulle immagini al fine di ottenere un output, dove ciascuna immagine è vista come una funzione $f(x, y)$.

Possiamo distinguere le operazioni effettuate a seconda del tipo di output che si ottiene: operazioni in cui l'output sono immagini (*image acquisition*, *image enhancement*, *image restoration*, *color image processing*, *wavelets and multiresolution processing*, *image compression*, *morphological processing*) e operazioni in cui l'output sono attributi dell'immagine (*morphological processing*, *segmentation*, *representation & description*, *object recognition*).

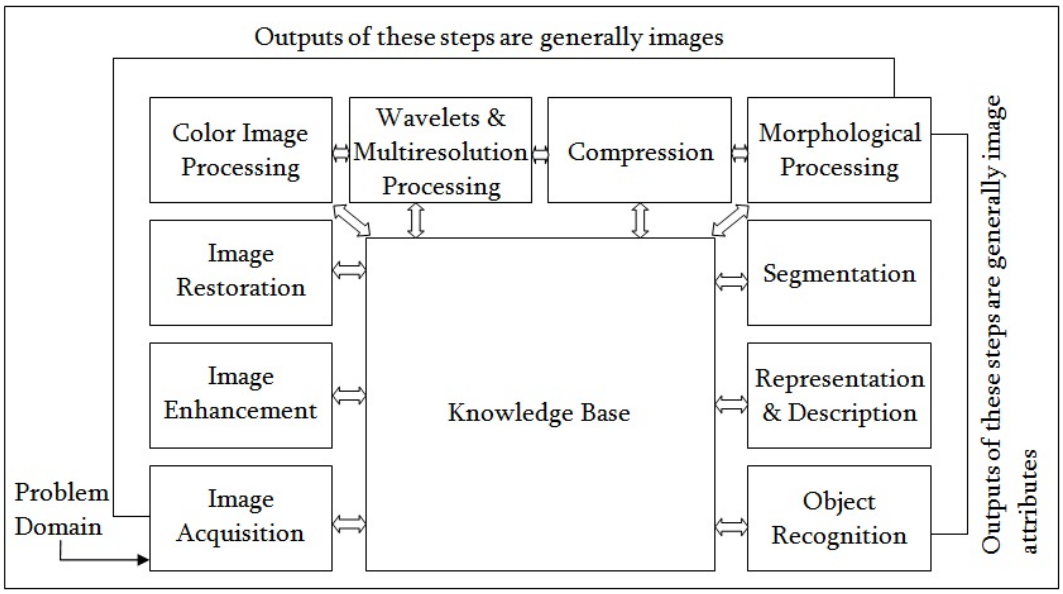

FIGURA 2.2: Image processing

Spesso le operazioni si susseguono una dopo l'altra andando a formare la cosiddetta *image processing pipeline*. Di seguito, una breve introduzione delle operazioni citate precedentemente.

Per *Image acquisition* (acquisizione dell'immagine) si intende il primo step all'interno della catena. L'acquisizione avviene tramite un dispositivo idoneo e di solito comporta una serie di operazioni di *pre-processing*, come il ridimensionamento dell'immagine.

L' *Image enhancement* (miglioramento dell'immagine) è una pratica soggettiva; comprende tutti quegli algoritmi applicati alle immagini, con lo scopo di rendere quest'ultima più gradevole all'occhio umano. Le principali tecniche utilizzate sono: aumento del contrasto, aumento della luminosità, applicazione di filtri e altro.

Il miglioramento dell'immagine è un'operazione simile a quella di restauro (*Image restoration*), con la differenza che, quest'ultima, è una pratica oggettiva in quanto si basa su metodi matematici e probabilistici di *image degradation*.

Le immagini possono essere processate dando maggiore enfasi alla componente del colore (*Color image processing*); con l'avvento dell'uso di immagini digitali via Internet, quest'area ha acquisito un'importanza piuttosto rilevante. I colori all'interno delle immagini possono essere utilizzati per estrarre caratteristiche

informative. Ci sono diversi modelli di colore che si possono usare per rappresentare l'immagine, come RGB, CMYK, CIELAB, HSI, HSV e altri.

Le *Wavelets* (*Wavelets transformation*) sono una trasformata più recente rispetto a quella di Fourier e sono alla base di diversi concetti, come le *Image pyramids* o le *Haar features* (che verranno descritte nel capitolo successivo).

La compressione (*Image compression*), come si può intuire dal nome, implica tutte quelle tecniche usate per ridurre lo spazio di memoria, utilizzato per memorizzare una determinata immagine.

Le operazioni morfologiche (*Morphological processing*) sono uno strumento utilizzato per estrarre dalle immagini componenti che sono utili nella rappresentazione e descrizione di una regione limitata. Viene di solito applicata ad immagini binarie (ovvero immagini che hanno due soli possibili valori per ogni pixel, di solito bianco e nero) e si basa sulla teoria degli insiemi.

La procedura di segmentazione (*Image segmentation*) suddivide l'immagine nelle sue regioni principali o oggetti, in base ad un set di regole precedentemente definite. Di solito gli algoritmi di segmentazione, per individuare aree di interesse, cercano di trovare all'interno dell'immagine discontinuità e similarità. Esempi di algoritmi di segmentazione sono: *global thresholding*, *adaptive thresholding*, *Watershed segmentation*, *canny edge detection*.

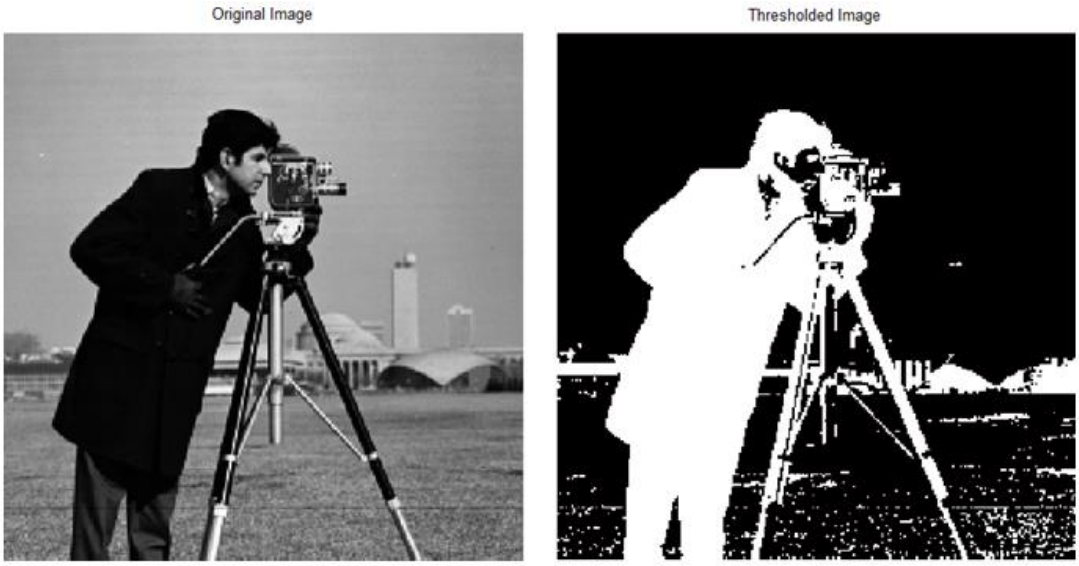

FIGURA 2.3: applicazione algoritmo di thresholding su un'immagine

Segue alla procedura di segmentazione quella di rappresentazione e descrizione (*Representation & description*), atta a descrivere le singole regioni segmentate. Le regioni possono essere descritte con una visione di insieme (tutti i pixel che compongono la regione) o una visione di confine (bordi della regione).

L'ultimo elemento della catena, si occupa infine del riconoscimento di oggetti all'interno della scena (*Object recognition*) [10].

Molte operazioni di image processing si basano sulla convoluzione (in inglese *convolution*). La convoluzione nell'image processing è un'operazione effettuata tra un'immagine e una matrice di dimensioni ridotte (chiamata *convolutional kernel* o *kernel*).

Nel processo di convoluzione per ogni pixel appartenente all'immagine, viene sovrapposto il *kernel* in modo che il "punto di ancoraggio" del kernel (di solito il pixel centrale) sia in corrispondenza del pixel dell'immagine.

Il valore di ogni pixel dell'immagine viene quindi ricalcolato come somma dei prodotti di ciascun elemento sovrapposto del *kernel*, per il rispettivo pixel dell'immagine.

Matematicamente si ha:

$$H(x,y) = \sum_{i=0}^{M_{i-1}} \sum_{j=0}^{M_j-1} I\big(x+i-a_i, y+j-a_j\big)K(i,j)$$

dove I è l'immagine originale, K è il *kernel* e H è la nuova immagine a cui è stata applicata la maschera.

2.4 La libreria OpenCV

OpenCV è una libreria open source utilizzata in campo di *Computer Vision*, *Image Processing*, *Machine Learning* e *Artificial Intelligence*.

La libreria espone delle interfacce in C++, C, Python, Java e supporta Windows, Linux, Mac OS, iOS e Android. OpenCV è stata sviluppata focalizzandosi sull'efficienza computazionale e le applicazioni real-time.

La libreria espone una moltitudine di funzioni, che verranno utilizzate e spiegate durante il lavoro di tesi. Di particolare importanza per il lavoro svolto, sono i seguenti moduli della libreria:

- Core: modulo principale di OpenCV, contiene tutte le strutture dati e le funzioni di base per lavorare sulle immagini;
- Imgproc: definisce una serie di funzioni molto utili per il processamento delle immagini come filtri, trasformazioni geometriche, operazioni morfologiche e molte altre;
- Objdetect: contiene funzioni per implementare algoritmi di *Object Detection* [9].

Capitolo 3

Individuazione automatica della congiuntiva

3.1 Panoramica dell'algoritmo

Una volta acquisite le foto dai pazienti si è passati all'individuazione dell'area di interesse. A questo scopo, si è implementato un algoritmo in grado di riconoscere e localizzare automaticamente la regione di interesse (da qui in poi chiamata ROI), contenente la congiuntiva palpebrale (se presente) all'interno della foto.

Uno degli algoritmi di *object detection* più utilizzato, è senza dubbio quello di Paul Viola e Micheal Jones [11], successivamente migliorato da Rainer Lienhart [12]. Sin dalla sua pubblicazione, l'algoritmo ha riscosso molto successo ed è utilizzato ancora oggi in numerose applicazioni.

L'algoritmo è ampiamente utilizzato grazie alla sua velocità nella fase di *detection*, che lo rende particolarmente adatto in applicazioni *real-time* (a scapito della fase di *training* in cui risulta particolarmente lento).

L'algoritmo si basa su 4 concetti chiave che verranno descritti nei prossimi paragrafi: il primo concetto è una famiglia di particolari *features* (caratteristiche) estratte dalle immagini chiamate *Haar-like features*; il secondo, è una nuova rappresentazione dell'immagine, detta *Integral Image*, che permette di calcolare più velocemente le features precedentemente citate; il terzo, è un algoritmo di apprendimento basato su "AdaBoost", che permette di ridurre drasticamente le features calcolate a quelle che meglio rappresentano l'immagine; il quarto e ultimo concetto chiave è un nuovo modo di combinare una serie di classificatori detti *stages* o *weak classifiers*, in un unico classificatore detto *strong classifier*.

I singoli classificatori sono applicati successivamente (o a cascata) alla ROI. Quest'ultima, per essere classificata come istanza positiva, deve ottenere un esito positivo da ciascun classificatore. L'algoritmo è adattabile con diversi *training set* di immagini; nel nostro caso il *training set* è composto da foto di congiuntive (*Conjunctiva detection*).

Prima di eseguire l'algoritmo, le immagini sono state ridimensionate in quanto sono state acquisite tramite dispositivi con risoluzioni differenti. Questo passaggio verrà descritto nel paragrafo successivo.

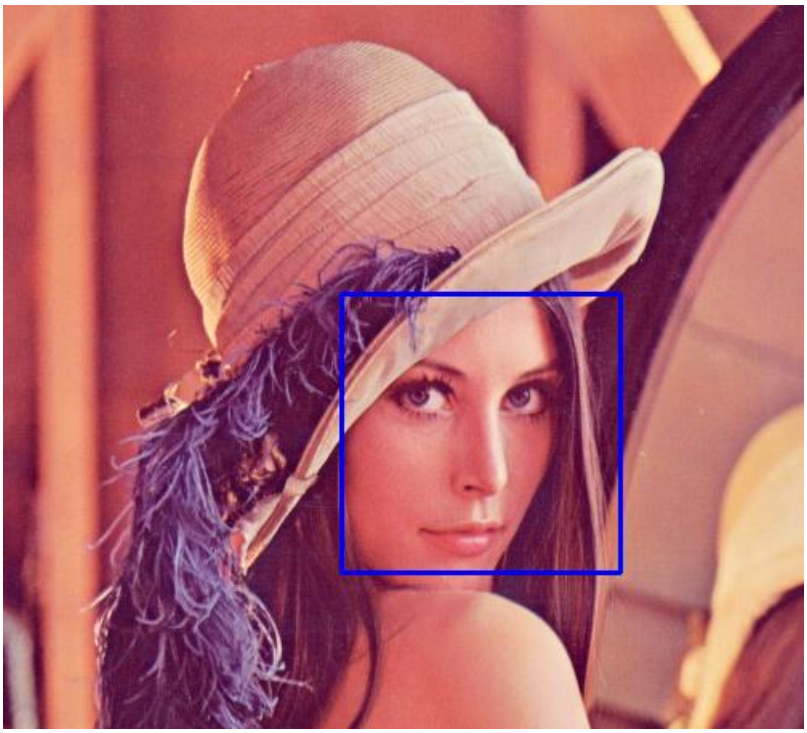

FIGURA 3.1: algoritmo di Viola & Jones (*face detection*)

3.2 Ridimensionamento delle immagini

Nella fase di *training*, l'algoritmo risulta essere particolarmente lento (anche a causa dell'alta risoluzione delle immagini). Inoltre, le immagini delle congiuntive, essendo state acquisite da dispositivi diversi, hanno risoluzioni diverse. Per questi motivi si è deciso di ridimensionare tutte le immagini del *training set*.

Per ciascuna immagine è stato calcolato un fattore, usato per scalare le immagini, mantenendo invariato il rapporto tra larghezza e altezza, e mantenendo questi ultimi al di sotto dei 1000 pixel.

Per effettuare il ridimensionamento è stata utilizzata la funzione *Imgproc.resize* di OpenCV, specificando come metodo di interpolazione quella bilineare (usata di default dalla libreria).

Per interpolazione, si intende un metodo per individuare nuovi punti del piano cartesiano, a partire da un insieme finito di punti dati.

Nell'ambito dell'interpolazione delle immagini, per ogni pixel, si cerca di ottenere la migliore approssimazione possibile del colore e dell'intensità leggendo i valori dei pixel limitrofi.

Nell'interpolazione bilineare si utilizzano i quattro pixel più vicini per stimare l'intensità da assegnare a ciascuna nuova posizione. Supponendo che (x, y) siano le coordinate della posizione a cui si deve assegnare un valore di intensità e che $v(x, y)$, equivalga al valore di intensità da assegnare. Per l'interpolazione bilineare il valore assegnato si ottiene mediante l'equazione:

$$V(x, y) = ax + by + cxy + d$$

dove i quattro coefficienti sono ricavati a partire dai quattro pixel vicini [13].

Si è inoltre scelto di convertire tutte le immagini nel formato BMP perché è un formato non compresso.

3.3 Descrizione dell'algoritmo

L'algoritmo di *Object detection* utilizzato si chiama *Haar feature-based cascade classifiers*. Le *features* sulle quali si basa il classificatore derivano da una particolare trasformata diversa dalla trasformata di Fourier.

La trasformata di Fourier è una trasformata integrale che permette di scrivere una funzione dipendente dal tempo nel dominio delle frequenze.

Tuttavia, la trasformata di Fourier, non è adatta a rappresentare segnali non stazionari (come le immagini), cioè segnali con componenti frequenziali variabili nel tempo (in quanto si perde ogni indicazione temporale del segnale stesso).

La trasformata *wavelet* è un potente strumento per l'analisi e l'elaborazione dei segnali e risulta estremamente efficiente in diversi campi di applicazione, come le immagini. A differenza della trasformata di Fourier, infatti, non viene persa l'informazione temporale.

Una delle più semplici *wavelet* è quella di *Haar*, sulla quale si basano appunto le *Haar features*.

Le *Haar features* hanno un meccanismo simile ai *convolutional kernel* (introdotto nel secondo capitolo).

Il valore delle *Haar features* (FIGURA 3.1) è calcolato come differenza tra la somma dei pixel presenti nelle regioni rettangolari bianche, meno quelli presenti nelle regioni rettangolari neri.

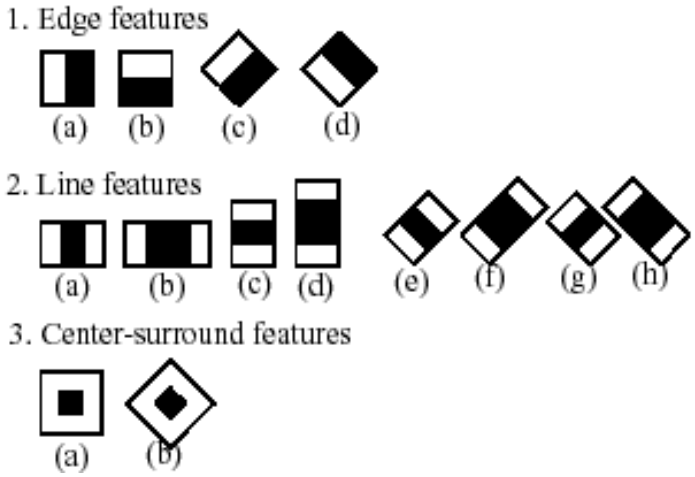

FIGURA 3.2: Haar-like features

Per calcolare le *Haar features* viene fatta scorrere una finestra lungo l'immagine.

L'algoritmo di Viola & Jones usa una finestra di 24X24 pixel e, se si considerano tutti i possibili parametri come posizione, scala e tipo di *features*, si finisce con il calcolare più di 180,000 *features* per ogni finestra.

Le *features* citate precedentemente possono essere calcolate in una maniera efficiente usando una rappresentazione diversa delle immagini, chiamata *integral image*. Quando si passa a questo tipo di rappresentazione vi è la necessità di creare una tabella che mappa l'immagine stessa, detta *summed-area table*. Nella tabella il valore di ogni pixel (x, y), è dato dalla somma dei pixel presenti in alto a sinistra rispetto a (x, y):

$$ii(x,y) = \sum_{x' \leq x, y' \leq y} i(x', y')$$

Dove $ii(x,y)$ è *l'integral image* e $i(x,y)$ è l'immagine originale. Usando questa rappresentazione, ogni sommatoria di pixel all'interno di un rettangolo può essere calcolata con quattro riferimenti (vedi Figura sotto).

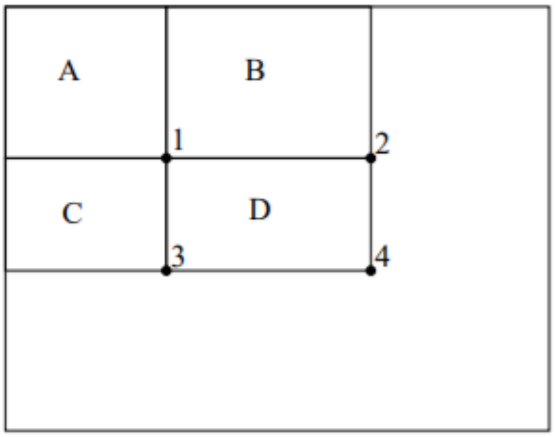

FIGURA 3.3:
La somma dei pixel all'interno del rettangolo D può essere calcolata tramite 4 riferimenti: D = 4 + 1 – (2 + 3)

Anche se le *features* vengono calcolate efficientemente, utilizzarle tutte all'interno del classificatore risulterebbe troppo costoso (perché, come detto precedentemente, sono più di 180,000 per ogni finestra).

Le *features* possono essere ridotte a quelle più discriminanti grazie ad un algoritmo di Boosting chiamato Adaboost.

Per Boosting si intende un metodo generale che crea un classificatore detto *strong classifier* (ovvero un classificatore con maggiore accuratezza) da un certo numero di classificatori, chiamati *weak classifiers* (ovvero classificatori deboli). Questi ultimi devono il loro nome al fatto che, singolarmente, non possono rappresentare l'istanza da classificare. Di solito sono poco più performanti di un classificatore random (ovvero un classificatore con accuratezza pari al 50%).

Adaboost (*Adaptive Boosting*) è un algoritmo di *machine learning*, formulato da Freund & Schapire [14]. L'algoritmo Adaboost è utilizzato sia per selezionare le *features* più discriminanti, che per allenare il classificatore stesso.

Di seguito la descrizione dell'algoritmo Adaboost [11]:

- Dato un *training set* di immagini (x_1, y_1) ... $(x_n\, y_n)$ dove $y_i = 0, 1$ rispettivamente per gli esempi negativi e positivi;
- Vengono inizializzati i pesi $w_{1,i} = \frac{1}{2\mathrm{m}}, \frac{1}{2\mathrm{l}}$, per $y_i = 0, 1$, dove m ed l sono il numero di esempi negativi e positivi;
- Per ogni stage $t = 1$ ... T:
  - Vengono normalizzati i pesi: $w_{t,i} = \frac{w_{t,i}}{\sum_{j=1}^{n} w_{t,j}}$;
  - Per ogni *feature*, j, si allena un classificatore h_j che è limitato ad utilizzare una singola *feature*. L'errore è valutato tramite: $\epsilon_j = \sum_i w_i \, |h_j(x_i) - y_i|$;
  - Viene scelto il classificatore, h_t, che minimizza l'errore ϵ_t;
  - Si modificano i pesi: $w_{t+1,i} = w_{t,i}\beta_t^{1-e_i}$ dove $e_i = 0$ se l'esempio x_i è classificato correttamente, $e_i = 1$ in caso contrario; $\beta_t = \frac{\epsilon_t}{1-\epsilon_t}$;
- Il modello finale, detto *strong classifier*, è rappresentato tramite la seguente formula: $h(x) = \begin{cases} 1 & \sum_{t=1}^{T} \alpha_t h_t(x) \geq \frac{1}{2}\sum_{t=1}^{T} \alpha_t \\ 0 & altrimenti \end{cases}$ dove $\alpha_t = \log\frac{1}{\beta_t}$.

I singoli classificatori, vengono ricavati dalle varie iterazioni nella fase di *training* dell'algoritmo Adaboost. I classificatori più semplici, sono richiamati all'inizio per scartare la maggior parte delle finestre che non contengono l'istanza da classificare, mentre, quelli più complessi, sono richiamati in seguito per raggiungere una bassa percentuale di *false positive* (FP).

Un risultato positivo ottenuto dal primo classificatore innesca la chiamata del secondo classificatore; un risultato positivo del secondo innesca la chiamata del terzo e così via. Un risultato negativo da ogni classificatore innesca l'immediato scarto della finestra (o *sub-window* nell'immagine). I classificatori, quindi, vengono applicati a cascata sull'immagine di interesse (da qui il nome dell'algoritmo *Cascade classifier*).

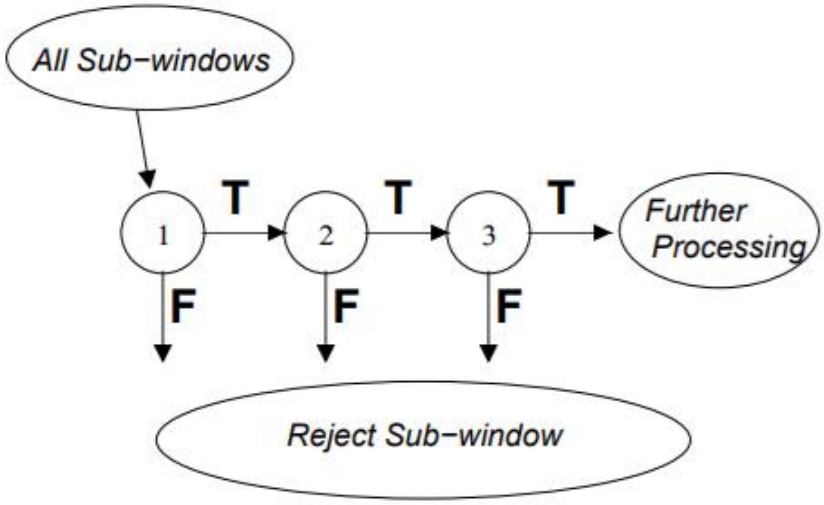

FIGURA 3.4: Classificatore a cascata

3.4 Allenamento del classificatore

La libreria OpenCV fornisce diversi *tools* per allenare il classificatore descritto precedentemente. Di seguito verranno descritti nel dettaglio tutti i passaggi eseguiti nella fase di *training*.

Per prima cosa, le immagini del *training set*, sono state disposte in due cartelle differenti, dove in una sono state poste le immagini contenenti la ROI (da qui in avanti chiamate immagini positive) e nell'altra, quelle non contenenti la ROI (da qui in avanti chiamate immagini negative o immagini di *background*).

Le immagini positive sono tutte le immagini scattate tramite il dispositivo di acquisizione contenenti la congiuntiva. Sono state utilizzate 116 foto come positive.

Le immagini negative, invece, devono contenere tutto ciò che le foto possono avere come sfondo, eccetto la congiuntiva. Essendo state scattate con il dispositivo di acquisizione, le foto negative possono solo contenere la parte circoscritta dal dispositivo di acquisizione attorno all'occhio, eccetto la congiuntiva stessa.

Per questo motivo si sono scattate ulteriori foto da utilizzare come foto di *background*, senza chiedere al soggetto di mostrare la congiuntiva (che non deve essere assolutamente esposta nelle foto negative).

Per ottenere più foto negative, sono state modificate le foto positive tramite Photoshop. La sezione contenente la congiuntiva, è stata selezionata e sostituita con una media dei pixel di una zona di pelle prelevata dalla foto del soggetto. In questo modo, le foto positive non contengono più la ROI, e possono essere utilizzate come negative.

Si sono raccolte in totale 162 foto negative.

Ai fini dell'allenamento del classificatore bisogna indicare al calcolatore l'esatta posizione della ROI, all'interno dalle immagini positive. Questa fase è stata svolta tramite il *tool* fornito dalla libreria OpenCV **opencv_annotations.exe**:

opencv_annotation.exe -a=file.txt -i=positive_images_folder.

Il *tool* serve per selezionare manualmente le regioni di interesse che saranno poi date in pasto al classificatore. Una volta lanciato il comando verranno aperte una dopo l'altra le immagini presenti nella cartella "*positive_images_folder*". Per ogni immagine, il tool, permette di selezionare la/e regione/i d'interesse che verrà/verranno poi trascritta/e in "*file.txt*".

Le regioni di interesse all'interno del file verranno memorizzate come: *[filename] [# of ROI] [[x y width height] [... 2nd object] ...]*

[filename] [# of ROI] [[x y width height] [... 2nd object] ...]

...

dove (x, y) rappresenta l'angolo in alto a sinistra della regione di interesse selezionata (l'origine $(0,0)$ è data dall'intersezione in alto a sinistra rispetto all'immagine); *width* ed *height*, invece, rappresentano rispettivamente la larghezza e altezza del rettangolo d'interesse tracciato.

Successivamente, si è passati alla creazione dei campioni da usare nel classificatore, tramite il *tool* **opencv_createsamples.exe**:

opencv_createsamples.exe -info file.txt -vec vector.vec -num 116 -w 120 -h 48

Il *tool* richiama la funzione *cvhaartraining.cpp#cvCreateTestSamples*, che ritaglia le regioni di interesse specificate con *opencv_annotation.exe* dalle immagini

positive, le ridimensiona secondo i parametri *-w* e *-h* specificati (che rappresentano rispettivamente la larghezza e altezza della regione di interesse) e li converte in un file VEC (specificato nel parametro *-vec*).

L'altezza e larghezza specificati devono rispecchiare le proprietà dell'oggetto da localizzare. In questo caso, l'altezza delle regioni selezionate contenenti la congiuntiva, è circa un terzo della larghezza.

Successivamente, si è passati alla fase finale, ovvero quella dell'allenamento vero e proprio. Per il training è stato utilizzato il *tool* **opencv_traincascade.exe**:

opencv_traincascade.exe -data cascade -vec vector.vec -bg bg.txt -numPos 116 -numNeg 162 -numStages 21 -w 120 -h 48 -minHitRate 0.999 -mode ALL

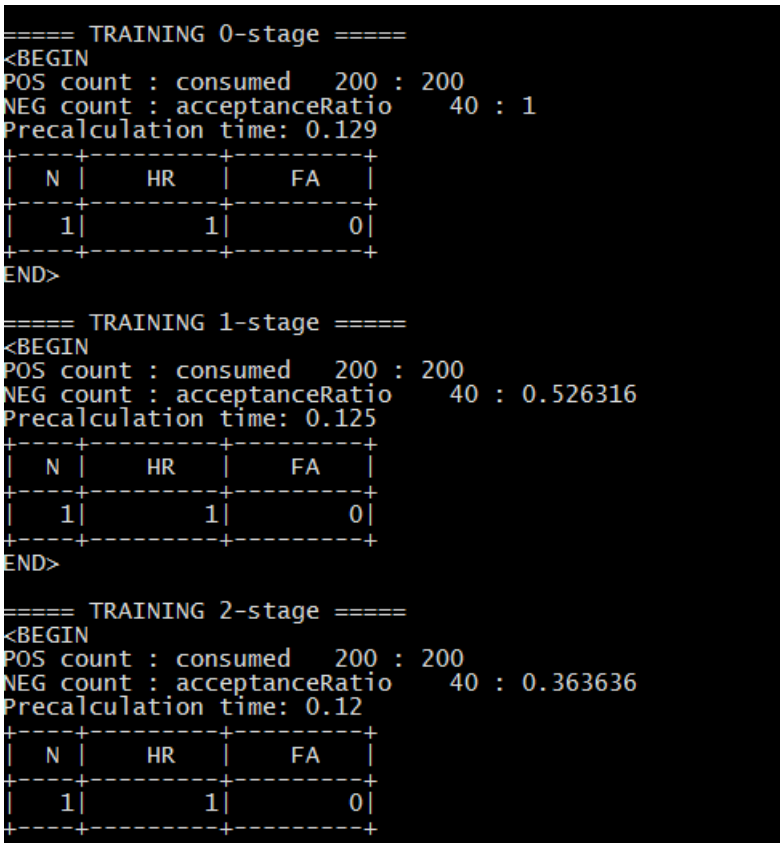

FIGURA 3.5: esempio schermata mostrata durante l'esecuzione di *opencv_traincascade.exe*

Di seguito la spiegazione dei parametri specificati:

- -*data*: cartella dove verranno memorizzati i risultati. Il risultato finale è un file XML (*cascade.xml*) che contiene il classificatore finale;
- -*vec*: file VEC creato con il tool *opencv_createsamples*;
- -*bg*: file contenente i percorsi assoluti delle immagini negative;
- -*numPos*: numero di campioni positivi utilizzati per ogni *stage*;
- -*numNeg*: numero di campioni negativi utilizzati per ogni *stage*;
- -*numStages*: numero di *stages* da allenare. Di solito vengono utilizzati 14-25 *stages* per *training set* con poche immagini [21]. Sono stati utilizzati 21 *stages*;
- -*w*, -*h*: sono rispettivamente la larghezza e altezza dei campioni (devono essere uguali a quelli utilizzati in *opencv_createsamples*);
- -*minHitRate*: si intende la percentuale minima di istanze classificate correttamente per ogni *stage*; per ottenere la percentuale totale bisogna elevare *minHitRate* per il numero totale di *stage* [22]. Per ottenere allora una percentuale totale del 98% (percentuale consigliata) bisogna impostare *minHitRate* a 0.999;
- -*mode*: si intende il set di *Haar-like features* da utilizzare; specificando l'attributo *ALL* vengono utilizzate il set di *features* esteso specificato da Lienhart [12].

La fase di allenamento ha impiegato 15 ore su un computer hp con processore Intel i7-6700HQ 2.60 Ghz, 16 GB RAM.

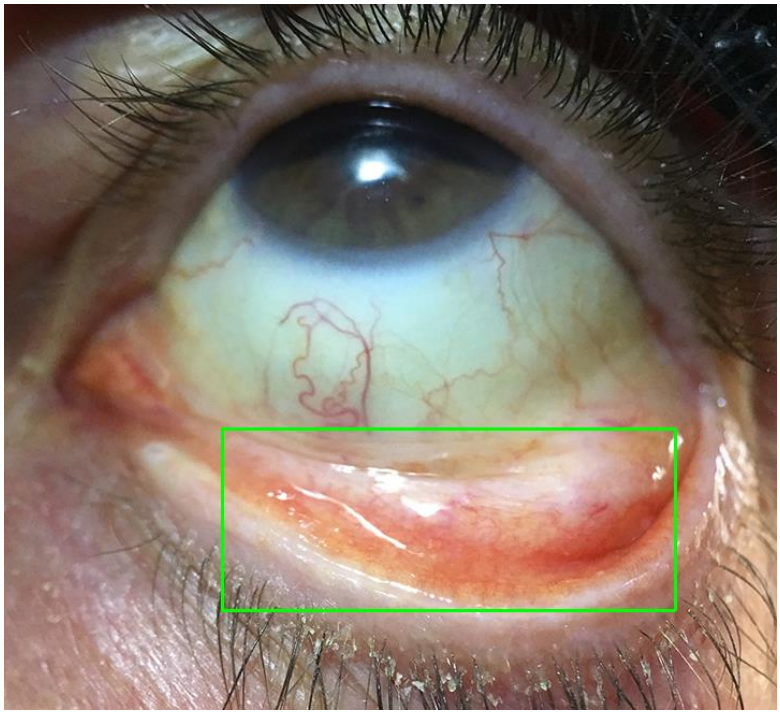

FIGURA 3.6: algoritmo *Conjunctiva detection* in azione

3.5 Test del classificatore

Il classificatore è stato infine testato su delle immagini create tramite lo script in Perl fornito da Naotoshi Seo [15].

Lo script prende in input una serie di immagini positive (precedentemente ritagliate a mano) e una serie di immagini di *background*. Dopodiché, in maniera casuale, applica delle distorsioni alle immagini positive (rotazione su asse delle x, y, z e deviazione dell'intensità) e le incolla su quelle di *background*, trascrivendo la posizione in un file di testo.

Il classificatore è stato quindi valutato in termini di *Precision* e *Recall*.

$$Precision = \frac{TP}{TP + FP}$$

$$Recall = \frac{TP}{TP + FN}$$

La metrica *Precision* misura, di tutte le istanze classificate positive, quante effettivamente sono esatte.

La metrica *Recall* misura, di tutte le istanze che dovrebbero essere classificate come positive, quante effettivamente sono classificate come tali.

Per avere una visione di insieme del classificatore, inoltre, si è valutato quest'ultimo con la metrica F_1 score che prende in considerazione le due metriche precedentemente citate:

$$F_1 = 2\,\frac{Precision * Recall}{Precision + Recall}$$

Le tre misure hanno un range che va da 0 a 1, dove 0 è il risultato peggiore, 1 è quello migliore.

Il classificatore è stato valutato su 100 immagini di test, riportando i seguenti risultati:

- *Precision* = 0.875
- *Recall* = 0.863
- F_1 = 0.869

Capitolo 4

Segmentazione della congiuntiva palpebrale

4.1 Image processing pipeline

Dopo aver individuato la ROI tramite l'algoritmo di *Object detection*, descritto nel capitolo precedente, si è passati all'implementazione di algoritmi *di image analysis* al fine di segmentare la congiuntiva palpebrale.

Lo scopo è quello di trovare una sequenza ottimale di algoritmi di *image analysis*, affinché l'area segmentata venga ottimizzata in correlazione con i valori di Hb.

I passi della sequenza di algoritmi applicati, sono, in ordine:

- Applicazione algoritmo di sfocatura per ridurre il rumore: $Gaussian\ blurring$ tramite *kernel* di $(9X9)$;
- Applicazione algoritmo di sogliatura per ottenere un'immagine binaria: $Otsu\ thresholding$;
- Applicazione algoritmo di *Morphological processing* con lo scopo di aprire un varco tra regioni connesse da un sottile ponte di pixel: $Morphological\ Opening$ tramite *structuring element* di $(5X5)$;
- Rilevamento dei contorni della regione più grande nell'immagine binaria, escludendo i "fori" all'interno della regione (tramite l'algoritmo di Suzuki [17]).

FIGURA 4.1: esempio congiuntiva segmentata automaticamente con la sequenza scritta sopra

La sequenza è stata individuata con l'ausilio di un software (da me progettato nel corso del lavoro di tesi) che ha permesso la visione in tempo reale dei risultati forniti dagli algoritmi.

Nei prossimi paragrafi verranno descritti gli algoritmi implementati, compresi quelli che non sono stati utilizzati all'interno della sequenza.

Il software è stato un valido strumento per lo studio e la visione dei risultati e sarà descritto in dettaglio nel prossimo capitolo.

4.2 Spazio di colore CIELAB

Lo spettro elettromagnetico è l'insieme di tutti i tipi di radiazioni elettromagnetiche. Queste possono essere descritte in termini di flusso di particelle senza massa (chiamate fotoni) che viaggiano alla velocità della luce, percorrendo onde sinusoidali. Ciascun fotone contiene una certa quantità di energia.

I differenti tipi di radiazioni possono essere definiti in termini di energia (J), lunghezza d'onda (m) o frequenza (Hz). A seconda di questi parametri, distinguiamo le onde radio, microonde, raggi infrarossi, le onde appartenenti allo spettro visibile, raggi ultravioletti, raggi X e raggi Gamma [10].

	Lunghezza d'onda (m)	Frequenza (Hz)	Energia (J)
Radio	$> 1 * 10^{-1}$	$< 3 * 10^{9}$	$< 2 * 10^{-24}$
Microonde	$1 * 10^{-3} - 1 * 10^{-1}$	$3 * 10^{9} - 3 * 10^{11}$	$2 * 10^{-24} - 2 * 10^{-22}$
Infrarossi	$7 * 10^{-7} - 1 * 10^{-3}$	$3 * 10^{11} - 4 * 10^{14}$	$2 * 10^{-22} - 3 * 10^{-19}$
Spettro visibile	$4 * 10^{-7} - 7 * 10^{-7}$	$4 * 10^{14} - 7{,}5 * 10^{14}$	$3 * 10^{-19} - 5 * 10^{-19}$
Raggi UV	$1 * 10^{-8} - 4 * 10^{-7}$	$7{,}5 * 10^{14} - 3 * 10^{16}$	$5 * 10^{-19} - 2 * 10^{-17}$
Raggi X	$1 * 10^{-11} - 1 * 10^{-8}$	$3 * 10^{16} - 3 * 10^{19}$	$2 * 10^{-17} - 2 * 10^{-14}$
Raggi Gamma	$< 1 * 10^{-11}$	$> 3 * 10^{19}$	$> 2 * 10^{-14}$

TABELLA 4.1: regioni dello spettro elettromagnetico

Se un raggio contiene una radiazione con una singola lunghezza d’onda, si dice che è monocromatico, se contiene radiazioni con più lunghezze d’onda, si dice policromatico. L'insieme delle lunghezze d’onde che compongono un raggio policromatico (e le loro rispettive luminosità) è detto spettro.

L'occhio umano è capace di vedere degli irraggiamenti la cui lunghezza d'onda è compresa tra 380 e 780 nanometri. Sotto i 380 nm si trovano i raggi ultravioletti, mentre i raggi infrarossi hanno una lunghezza d'onda superiore ai 780 nm. L'insieme delle lunghezze d'onda visibili dall'occhio umano è detto spettro visibile.

Nel 1931, la CIE (Commissione Internazionale dell'Illuminazione) ha elaborato il sistema colorimetrico che rappresenta i colori secondo la loro cromaticità (assi x e y) e la loro luminanza (asse Y).

Il diagramma di cromaticità (o diagramma cromatico), è il risultato della sovrapposizione di tre distribuzione di intensità spettrale, appartenenti a 3 colori primari (Rosso, Verde e Blu). I vari colori visibili dall’occhio umano sono individuabili nel diagramma CIE della figura sotto.

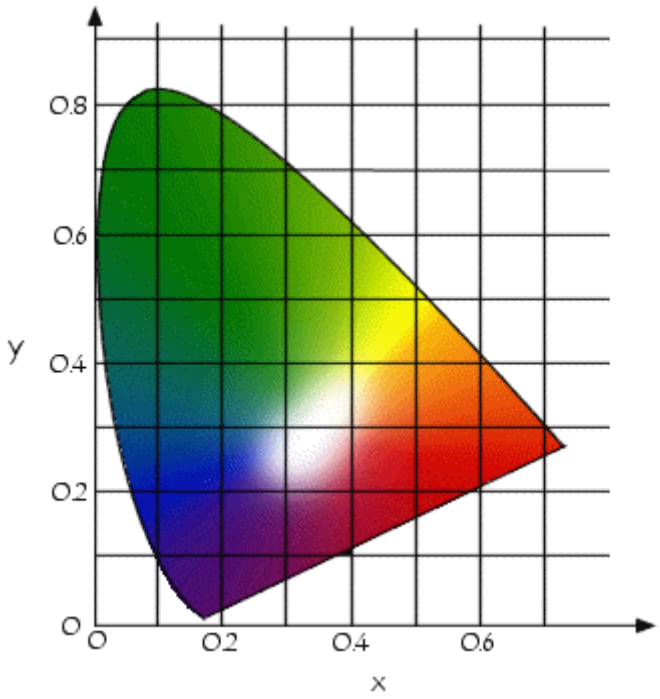

FIGURA 4.2: diagramma CIE

Viene quindi detto “spazio dei colori” la rappresentazione matematica di un insieme di colori. Ne esistono diversi tipi, fra i quali i più conosciuti sono RGB, HSI, HSV, CMYK e altri.

Lo spettro dei colori che una periferica di visualizzazione permette di visionare è detto *gamut* o spazio colorimetrico. I colori che non appartengono al *gamut* sono detti colori fuori gamma.

Tuttavia, questa modalità di rappresentazione puramente matematica, non tiene conto dei fattori fisiologici della percezione del colore dall'occhio umano.

Un'immagine si forma sulla retina grazie alla cornea (la copertura traslucida dell'occhio) e all'iride (che chiudendosi permette di dosare la quantità di luce). Questa è composta da piccoli bastoncelli (rods) e coni (cones).

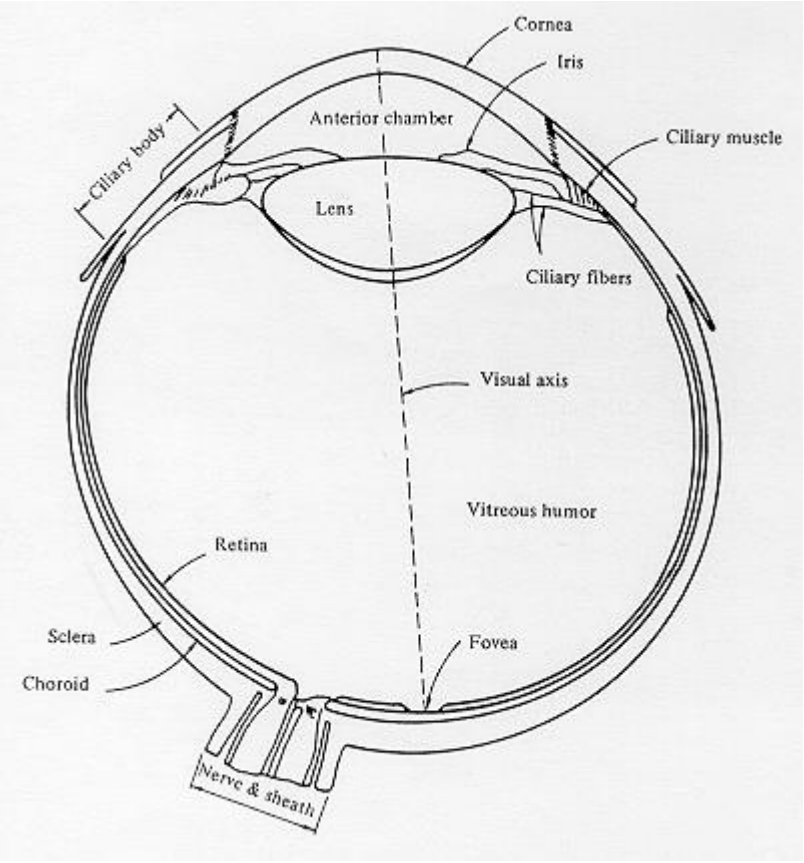

FIGURA 4.3: struttura dell’occhio

I bastoncelli, formati da una pigmentazione detta rodopsina e posti alla periferia della retina, permettono di percepire la luminosità e il movimento (visione scotopica). I coni, invece, posti in una zona detta fovea, permettono di diversificare

i colori (visione fotopica). Esistono tre tipi di coni: sensibili al rosso, al verde e al blu; se attivati simultaneamente, la luce percepita risulta essere bianca.

Nel 1976, per rimediare alle lacune del modello xyY, la CIE elabora il modello colorimetrico L*a*b (anche conosciuto con il nome di CIELAB), in cui un colore è individuato da tre valori.

Lo spazio di colori CIELAB usa due dimensioni per la rappresentazione del colore (a* e b*) e una dimensione per la rappresentazione della luminosità (L*). In (a*, b*) i valori pari a 0 rappresentano i grigi. I colori rosso e verde sono rappresentati rispettivamente con a* > 0 e a* < 0; analogamente le componenti giallo e blu sono rappresentati rispettivamente con b* > 0 e b* < 0.

La modalità Lab copre l'intero spettro visibile dall'occhio umano e lo rappresenta in modo uniforme. Esso permette, quindi, di descrivere l'insieme dei colori visibili, indipendentemente da qualsiasi tecnologia grafica. In questo modo esso comprende la totalità dei colori RGB e CMYK, ragione per cui software come PhotoShop usano questa modalità per passare da un modello di rappresentazione ad un altro [16].

OpenCV permette di acquisire un'immagine nel formato predefinito BGR (blue, green, red) tramite il modulo *Imgcodecs,* e di convertire l'immagine in diversi modelli di colore, tramite il modulo *Imgproc*.

Per la lettura delle immagini si è utilizzato la funzione *Imgcodecs.imread(String filename),* mentre, per la conversione allo spazio di colori CIELAB, si è utilizzato la funzione *Imgproc.cvtColor(Mat src, Mat dst, int code)*, specificando come parametro *code* la costante *Imgproc.COLOR_BGR2Lab*.

La funzione *Core.split(Mat src, List<Mat> channels)* è utilizzata per ottenere all'interno di una lista i singoli canali dello spazio di colori CIELAB. La singola componente a* è stata facilmente ottenuta tramite un'operazione di "get" sulla lista: *channels.get(1).*

La conversione dallo spazio di colori BGR a CIELAB, secondo la documentazione di OpenCV, viene effettuata tramite i seguenti passaggi:

$$\begin{bmatrix} X \\ Y \\ Z \end{bmatrix} = \begin{bmatrix} 0.412453 & 0.357580 & 0.180423 \\ 0.212671 & 0.715160 & 0.072169 \\ 0.019334 & 0.119193 & 0.950227 \end{bmatrix} * \begin{bmatrix} R \\ G \\ B \end{bmatrix}$$

$$X = {}^{X}/_{X_n} \; dove \; X_n = \; 0.950456$$

$$Z = {}^{Z}/_{Z_n} \; dove \; Z_n = 1.088754$$

$$L = \begin{cases} 116 * \; Y^{1}/_{3} - 16 \; per \; Y > \; 0.008856 \\ 903{,}3 * Y \; per \; Y \; \leq 0.008856 \end{cases}$$

$$a = 500\big(f(X) - f(Y)\big) + 128$$

$$b = 200\big(f(Y) - f(Z)\big) + 128$$

dove

$$f(t) = \begin{cases} t^{1}/_{3} \; per \; t > \; 0.008856 \\ 7{,}787 \; per \; t \; \leq \; 0.008856 \end{cases}$$

L'output dei valori è il seguente: $0 \leq L \leq 100, -127 \leq a \leq 127, -127 \leq b \leq 127$. I valori sono convertiti infine come: $L = L * \frac{255}{100}, a = a + 128, b = b + 128$.

I valori di L* hanno un range di $[0 \; - \; 255]$, mentre quelli di a* e b* hanno valori compresi tra $[1 \; - \; 255]$.

Le operazioni di segmentazione da qui in poi verranno eseguite sulla singola componente a* dell'immagine (da qui in poi chiamata immagine).

4.3 Image pre-processing

Prima di effettuare le operazioni di *thresholding* (descritti nel prossimo paragrafo), le immagini vengono di solito sottoposte ad operazioni di *pre-processing*. Queste sono molto utili per eliminare eventuali rumori e/o accentuare i contorni. Gli algoritmi di *pre-processing* implementati nel software sono: equalizzazione dell'istogramma, *Gaussian Blurring*.

L'equalizzazione dell'istogramma è una procedura utilizzata per accentuare il contrasto dell'immagine e, quindi, facilita l'individuazione dei contorni.

Per equalizzare un istogramma bisogna calcolare la funzione PMF (*Probability Mass Function*), ovvero la frequenza di ogni valore di intensità diviso il numero di occorrenze totali:

$$h(i) = \frac{numero\ di\ pixel\ con\ intensità\ i}{numero\ totale\ di\ pixel}$$

Successivamente, viene calcolata la funzione CDF (*Cumulative Distributive Function*): la funzione cumulativa prende il primo valore di PMF così com'è, il secondo come somma del primo più il secondo, il terzo come somma del primo + secondo + terzo e così via.

$$H(j) = \sum_{i=0}^{j} h(i)$$

Infine, per calcolare il nuovo istogramma (equalizzato), ciascun valore di CDF viene moltiplicato per il corrispettivo valore di intensità originale, trovando così il nuovo valore di intensità.

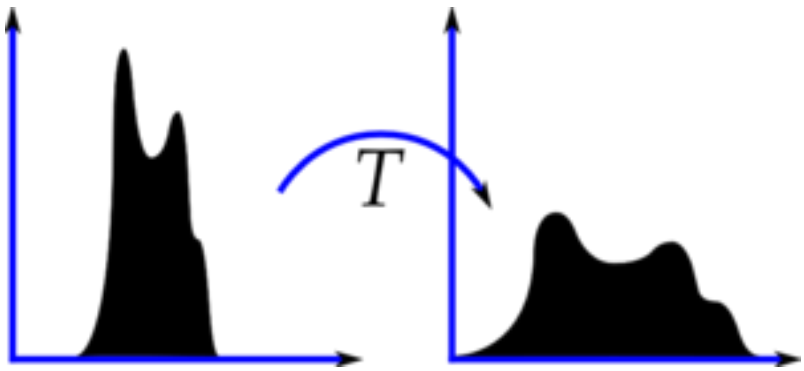

FIGURA 4.4: equalizzazione dell'istogramma

L'algoritmo *Gaussian Blurring* (sfocatura gaussiana), è un filtro appartenente alla famiglia dei *low pass filter,* ovvero filtri che eliminano le frequenze più alte all'interno di un'immagine. Le fotografie possono essere piuttosto rumorose, il che significa che possono esserci piccole irregolarità che possono influenzare il processo di segmentazione. Un modo comune di sbarazzarsi del rumore è quello di

processare l'immagine tramite un algoritmo di sfocatura, in modo da "lisciare" il contenuto.

L'algoritmo è quindi utilizzato per ridurre il rumore e il livello di dettaglio. Il filtro usa una matrice che approssima la distribuzione Gaussiana in due dimensioni:

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

Una volta che il *kernel* è stato calcolato, la sfocatura Gaussiana può essere implementata tramite il metodo di convoluzione (introdotto nel secondo capitolo).

4.4 Thresholding

L'operazione di sogliatura (in inglese chiamata *thresholding*) è uno dei più semplici metodi per segmentare un'immagine. Il metodo prende come input un'immagine a livelli di grigio e restituisce come output un'immagine binaria.

Per immagine binaria, si intende un'immagine che ha due possibili valori per ogni pixel (di solito bianco o nero). Durante il processo di *thresholding*, i singoli pixel all'interno dell'immagine sono catalogati come "pixel oggetto" se il valore è maggiore di una certa soglia (*threshold*) o come "pixel di sfondo" se il valore è al di sotto della soglia:

$$g(x,y) = \begin{cases} 1 \; se \; f(x,y) > T \\ 0 \; se \; f(x,y) \leq T \end{cases}$$

Il concetto chiave all'interno degli algoritmi di *thresholding* è la scelta del valore di soglia. É possibile impostare un valore manualmente o utilizzare algoritmi che calcolano automaticamente il valore.

É stato implementato un algoritmo di *threshold* manuale (dove il valore della soglia è specificato dall'utente) e uno con scelta della soglia automatica.

Il metodo *Otsu* è un metodo di sogliatura automatica per immagini bimodali. Un'immagine bimodale è un'immagine che ha un istogramma associato che presenta due mode (ovvero due picchi distinti tra di loro che rappresentano distintamente lo sfondo dalla scena).

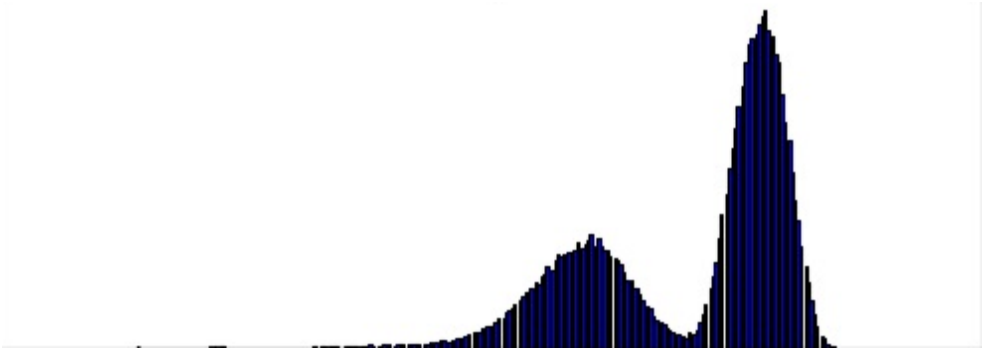

FIGURA 4.5: esempio di istogramma bimodale

L'immagine bimodale si presenta quindi sull'istogramma con due distribuzioni ben distinte. La binarizzazione di *Otsu* aiuta ad ottenere il valore che meglio separa le due mode.

Il metodo *Otsu* è un algoritmo che itera su tutti i possibili valori di *threshold* e calcola per ogni valore una misura chiamata varianza intra classe, definita come somma pesata delle varianze delle due classi (che sono rispettivamente la somma dei pixel di *background* e *foreground*). L'obiettivo è quello di trovare il valore che minimizza la varianza intra classe.

Per ogni valore di *threshold* t, la varianza intra classe è definita come:

$$\sigma_w^2(t) = q_1(t)\sigma_1^2(t) + q_2(t)\sigma_2^2(t)$$

$$dove\ q_1(t) = \sum_{i=1}^{t} P(i) \quad \& \quad q_2(t) = \sum_{i=t+1}^{I} P(i)$$

$$\mu_1(t) = \sum_{i=1}^{t} \frac{iP(i)}{q_1(t)} \quad \& \quad \mu_2(t) = \sum_{i=t+1}^{I} \frac{iP(i)}{q_2(t)}$$

$$\sigma_1^2(t) = \sum_{i=1}^{t} [1 - \mu_1(t)]^2 \frac{P(i)}{q_1(t)} \quad \& \quad \sigma_2^2(t) = \sum_{i=t+1}^{I} [1 - \mu_2(t)]^2 \frac{P(i)}{q_2(t)}$$

dove $P(i)$ rappresenta la probabilità di occorrenza del pixel con intensità i.

4.5 Morphological operations

Una volta ottenuta l'immagine binaria tramite le operazioni di *thresholding*, le regioni individuate possono avere delle imperfezioni. Lo scopo principale delle operazioni morfologiche è quello di rimuovere le imperfezioni basandosi sulla struttura e la forma dell'immagine.

Le operazioni morfologiche sono operazioni non lineari che utilizzano una matrice chiamata *structuring element*. Di questi elementi bisogna specificarne l'origine, che di solito si fa coincidere con il pixel centrale (motivo per il quale si scelgono sempre matrici di dimensione dispari). Per effettuare l'operazione si utilizza lo stesso principio dei *convolutional kernel* (convoluzione).

1	0	0
0	**1**	0
0	0	1

FIGURA 4.6: esempio di *structuring element*

Le operazioni morfologiche utilizzano la teoria degli insiemi. I concetti fondamentali della teoria sono due:

- **Riflessione**: viene indicata con $\hat{B} = \{w|w = -b, per\ b\ \epsilon B\}$; ovvero, se B è l'insieme di pixel che rappresentano un oggetto di un'immagine, allora $\hat{B}$, è l'insieme dei punti in B le cui coordinate (x, y) sono state sostituite da $(-x, -y)$;
- **Traslazione**: di un insieme B tramite un punto $z = (z_1, z_2)$ viene indicata con $(B)_z$ ed è definita come $(B)_z = \{c|c = b + z, \text{per b } \epsilon B\}$; ovvero, per ogni pixel (x, y) la traslazione tramite z è data da $(x + z, y + z)$.

Le due operazioni precedenti sono alla base delle operazioni morfologiche di *erosion, dilation, opening e closing.*

L'operazione di ***erosion***, viene utilizzata per eliminare o assottigliare gli oggetti di un'immagine binaria; matematicamente se A e B sono due insiemi, l'erosione di A attraverso B è indicata con

$$A \ominus B = \{z|(B)_z \subseteq A\}$$

Quindi, l'erosione di A attraverso B, è l'insieme di tutti i punti tali che B traslato sia contenuto in A. B deve essere completamente contenuto in A, quindi non deve avere elementi contenuti nello sfondo.

L'operazione di ***dilation***, serve per far accrescere le regioni individuate da un'immagine binaria. La dilatazione di A attraverso B è così definita:

$$A \oplus B = \{z|[(\hat{B})_z \cap A] \subseteq A\}$$

Quindi, la dilatazione di A con B, è l'insieme di tutte le traslazioni tali che $(\hat{B})$ e A si sovrappongono almeno un elemento.

Le operazioni di **dilation** ed **erosion** sono spesso combinate assieme nell'*image processing*; in particolare, se si fa seguire la dilatazione all'erosione si parla di apertura (**opening**):

$$A \circ B = (A \ominus B) \oplus B$$

Mentre, se accade l'inverso, si parla di chiusura (**closing**):

$$A \bullet B = (A \oplus B) \ominus B$$

L'operazione di *opening*, è chiamata così perché ha come effetto quello di aprire un varco tra oggetti connessi da un sottile ponte di pixel.

L'operazione di *closing*, invece, ha come effetto quello di rendere più omogenee le sezioni di contorno colmando le interruzioni ed eliminando piccoli vuoti.

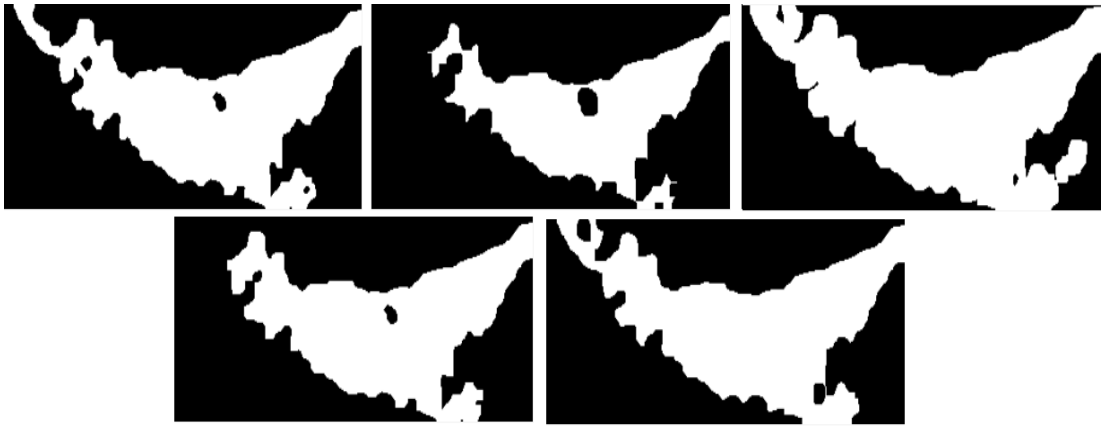

FIGURA 4.7: da in alto a sinistra verso destra, la figura mostra l'immagine binaria originale di una congiuntiva; l'immagine dopo l'operazione di *erode*; l'immagine dopo l'operazione di *dilate*; l'immagine dopo l'operazione di *opening*; l'immagine dopo l'operazione di *closing*.

4.6 Rilevamento dei contorni

Il rilevamento dei contorni viene effettuato tramite l'algoritmo di Suzuki [17] che sfrutta alcune proprietà dei pixel (descritte di seguito).

Ciascun pixel $p(x, y)$ ha 4 vicini definiti come $(x+1, y), (x-1, y), (x, y+1), (x, y-1)$. Questo insieme di pixel è chiamato *4-neighbors of P*, ed è denotato come $N_4(P)$.

I 4 vicini diagonali di $p(x, y)$ sono $(x+1, y+1), (x+1, y-1), (x-1, y+1), (x-1, y-1)$. Questo insieme è denotato come $N_D(P)$.

L'insieme dei pixel formati da $N_4(P)$ e $N_D(P)$ sono denotati come $N_8(P)$ ovvero gli 8 vicini di P.

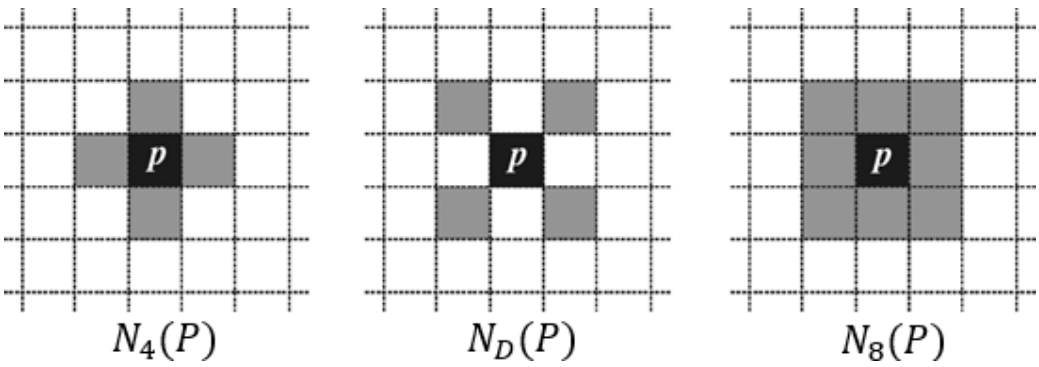

FIGURA 4.8: la figura mostra $N_4(P)$, $N_D(P)$ e $N_8(P)$

In un'immagine binaria, due pixel sono adiacenti se sono vicini e hanno lo stesso valore (0/1). Dati due pixel p e q con lo stesso valore di intensità distinguiamo 3 tipi di adiacenza:

- 4-adiacenza: se $q \in N_4(p)$;
- 8-adiacenza: se $q \in N_8(p)$;
- M-adiacenza:
    - Se $q \in N_4(p)$ oppure,
    - Se $q \in N_D(p)$ e $N_4(p) \cap N_4(q)$ non ha pixel con lo stesso valore di intensità.

Un percorso (in inglese *path*) dal pixel p al pixel q (di coordinate (x_p, y_p) e (x_q, y_q)) è una sequenza di pixel aventi coordinate $(x_0, y_0), (x_1, y_1) \dots (x_n, y_n)$ dove:

- $(x_0, y_0) = (x_p, y_p)$ e $(x_n, y_n) = (x_q, y_q)$
- (x_i, y_i) e (x_{i+1}, y_{i+1}) sono adiacenti per $(1 \leq i \leq n)$

Il numero n è detta lunghezza del percorso. Inoltre, se $(x_0, y_0) = (x_n, y_n)$ il percorso è chiuso. La definizione di percorso, ovviamente, va definita rispetto ai 3 tipi di adiacenza.

Dato un sottoinsieme S di pixel dell'immagine, due pixel $p, q \in S$ sono detti connessi in S, se esiste un percorso da p a q in S. L'insieme dei pixel connessi costituisce una **componente connessa** di S. Se esiste una sola componente connessa in S, allora S viene detto insieme connesso.

Un sottoinsieme R di pixel dell'immagine, viene detto regione, se è un insieme connesso.

Le relazioni di adiacenza possono essere estese alle regioni: due regioni R_i e R_j sono adiacenti se l'unione delle due regioni è connesso, in caso contrario sono dette disgiunte.

L'unione di tutte le regioni di un'immagine viene detta scena (o *foreground*), mentre, il loro complemento, viene detto sfondo della scena (o *background*).

Il **bordo** (in inglese *boundary* o *contour*) di una regione R è l'insieme dei punti di R adiacenti al suo complemento. Distinguiamo due tipi di bordi:

- Bordo interno, ovvero i pixel appartenenti alla regione R adiacenti con il suo complemento;
- Bordo esterno, costituito invece dai pixel di *background* adiacenti a R.

L'algoritmo di Suzuki [17], permette di estrarre i bordi delle regioni aggiungendo una descrizione topologica alle strutture estratte.

Le informazioni da estrarre riguardano due tipi di bordi: i bordi esterni e i bordi dei "fori" o "buchi" (in inglese *holes*). Se una componente connessa S, formata da pixel di background, contiene la scena, allora S è lo sfondo, altrimenti, è un foro all'interno della scena.

L'algoritmo è implementato nella libreria OpenCV tramite la funzione *Imgproc.findContours(Mat image, List<MatOfPoint> contours, Mat hierarchy, int mode, int method)*, dove *image* è l'immagine binaria dove si vuole eseguire l'algoritmo, *contours* è una lista vuota dove verranno memorizzati i contorni rilevati, *hierarchy* è una matrice vuota dove verranno memorizzate le informazioni topologiche dei contorni formando una struttura gerarchica.

Il parametro *method* definisce il modo in cui i contorni verranno approssimati, mentre, il parametro *mode*, definisce il modo in cui verrà descritta la struttura topologica dei contorni, nella matrice *hierarchy*. Specificando *RETR_TREE* come valore di *mode*, è possibile ottenere l'intera gerarchia dei contorni (ovvero dal bordo più esterno fino ai fori interni).

La segmentazione della congiuntiva nel software implementato, a seconda dell'input dell'utente, può considerare:

- Tutti i contorni delle regioni rilevate escludendo i fori dalla regione di *foreground* (a);
- Tutti i contorni delle regioni rilevate includendo i fori nella regione di *foreground* (b);

- Solo il contorno della regione più ampia includendo i fori nella regione di *foreground* (c);
- Il contorno della regione più ampia escludendo i fori dalla regione di *foreground* (d).

Nelle immagini della congiuntiva, i contorni dei fori interni alle regioni di foreground, corrispondono ai *bright spot*.

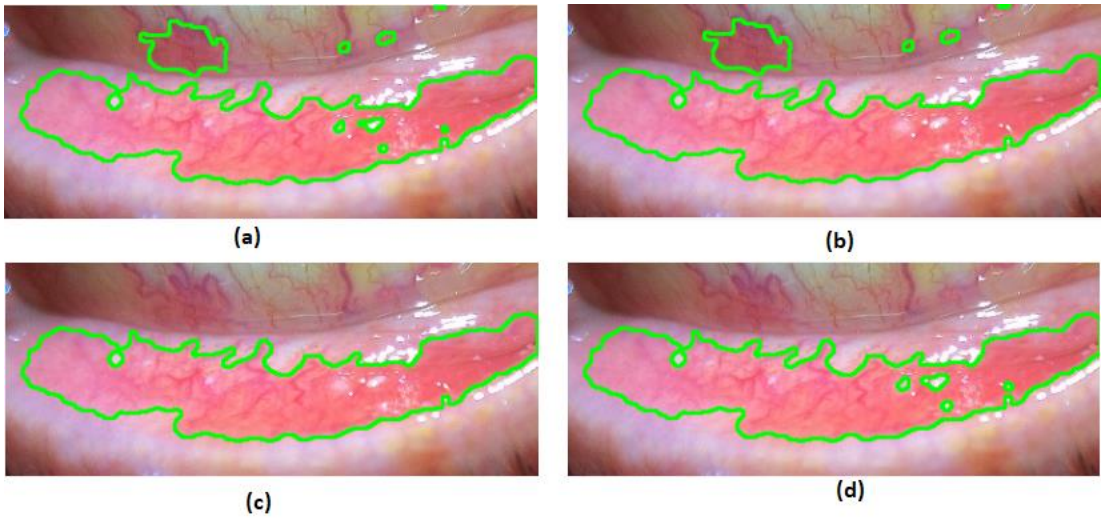

FIGURA 4.9: rilevamento dei contorni secondo le specifiche scritte sopra

4.7 Ellipse fitting

Una volta ottenuti i contorni di interesse questi possono essere utilizzati come insieme di curve che uniscono tutti i punti (o pixel) presenti sul bordo. I contorni, inoltre, sono uno strumento molto utile per algoritmi di *shape analysis*.

Data la forma della congiuntiva, la figura che meglio si adatta a contenerla è l'ellissi. L'adattamento di una conica (in inglese *conic fitting*) è una procedura molto comune in algoritmi di *computer vision* ed *image analysis*.

OpenCV fornisce un'implementazione dell'algoritmo descritto da Fitzgibonn [18]. Quest'ultimo, descrive il problema di *conic fitting* come:

- Dati un set di punti $P = \{x_i\}_{i=1}^{n}$, dove $x_i = (x_i, y_i)$;
- Data una famiglia di curve $C(a)$;

- Data una metrica $\delta(C(a), x)$ che descrive la distanza da un punto x alla curva $C(a)$;
- Trovare il valore a_{min} che minimizza $\epsilon^2(a) = \sum_{i=1}^{n} \delta(C(a), x_i)$. La curva $C(a_{min})$ è allora la curva che meglio si adatta ai dati.

L'algoritmo è implementato tramite la funzione *Imgproc.fitEllipse(MatOfPoint2f contours),* dove *conoturs* è la matrice contenente i punti del bordo appartenenti ad una regione (sulla quale si vuole adattare l'ellissi).

Affinché l'algoritmo abbia esito positivo, il bordo deve essere formato da almeno 5 punti.

Capitolo 5

Software sviluppato

5.1 Panoramica del Software

Il software sviluppato ha permesso sia lo studio degli algoritmi implementati che i risultati da essi forniti. I prossimi paragrafi descriveranno nel dettaglio il funzionamento del software, dal punto di vista delle sue funzionalità.

Le funzionalità principali sono due (in corrispondenza del menù "File"), ovvero:

- Localizzazione automatica della congiuntiva (il cui funzionamento è stato descritto nel capitolo 3);
- Segmentazione della congiuntiva palpebrale (il cui funzionamento è stato descritto nel capitolo 4).

Le due funzioni corrispondono rispettivamente alle due voci presenti nel menù ("Congiuntiva detection" e "Open Conjunctiva image to segment").

Il software è stato scritto in Java (versione 8) con l'ausilio della libreria JavaFX per il design e implementazione dell'interfaccia grafica. Per l'implementazione degli algoritmi di *Object detection* e *Image analysis* è stata utilizzata la libreria OpenCV (versione 3.3). Le funzionalità del software, inoltre, sono state testate tramite la libreria JUnit.

Il software è stato testato su una macchina con sistema operativo Windows 10.

5.2 Localizzazione automatica della congiuntiva

La localizzazione automatica della congiuntiva viene avviata in corrispondenza della voce di menu "Congiuntiva detection" (vedi figura sotto).

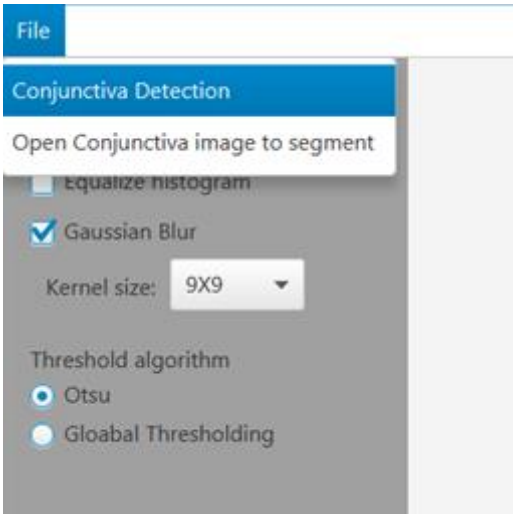

FIGURA 5.1: voce del menu “Conjunctiva detection”

Selezionando la voce del menu, il software dà la possibilità di scegliere una o più foto, a cui applicare l’algoritmo di *object detection*, che cerca di individuare automaticamente la congiuntiva. Se il classificatore ha esito positivo, viene tracciato un rettangolo in corrispondenza della regione di interesse individuata. In caso contrario, il software dà la possibilità all’utente di tracciare manualmente la congiuntiva, facendo tracciare la regione di interesse (tramite click e trascinamento del mouse).

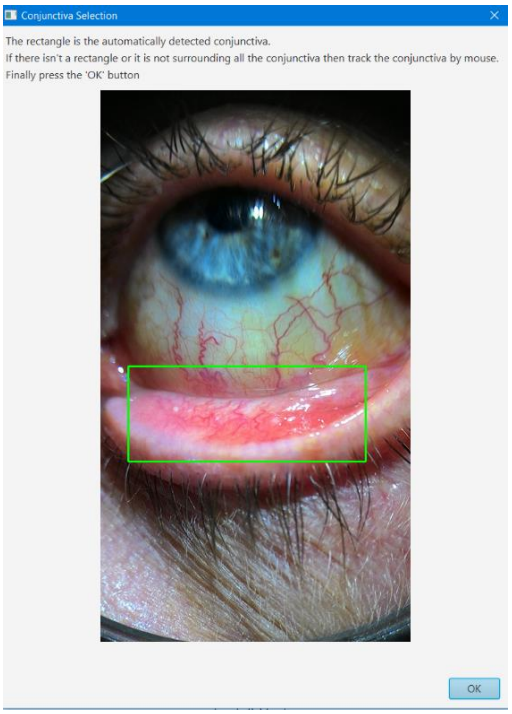

FIGURA 5.2: congiuntiva individuata automaticamente

Al termine della fase di *detection*, il software dà la possibilità di selezionare la cartella in cui si desidera salvare le congiuntive individuate.

5.3 Segmentazione della congiuntiva palpebrale

Sono stati implementati tutti gli algoritmi di segmentazione descritti nel capitolo 4. Gli algoritmi implementati corrispondono alla *sidebar* presente sul lato sinistro dell'interfaccia grafica, divisi per tipologia di algoritmo. Per default, sono selezionati gli algoritmi che permettono di implementare la sequenza descritta nel capitolo precedente.

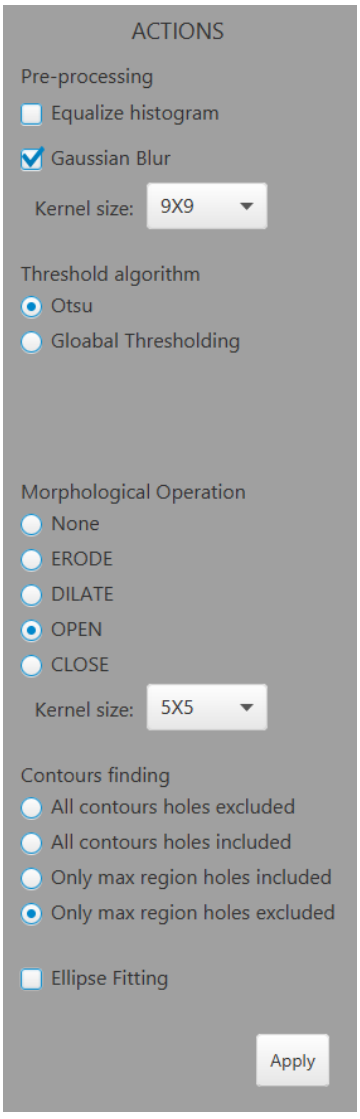

FIGURA 5.3: sidebar del software

La pressione del bottone “Apply”, permette di eseguire la sequenza di algoritmi specificati nella *sidebar* alla/e immagine/i selezionata/e. Per l’implementazione della sequenza specificata nella FIGURA 5.3, ad esempio, sarà necessario il seguente script di codice:

```
Mat mat = Imgcodecs.imread(pathOfImageToRead);
AstarSegmentation as = new AstarSegmentation(mat);
as.setAstarComponent(false);
as.gaussianBlur(new Size(9, 9));
as.setBinaryImageByOtsu();
as.morphologyOperation(2, new Size(5, 5));
as.regionFilling(2);
```

Per selezionare una o più immagini a cui applicare gli algoritmi è necessario selezionare la voce del menu “Open Conjunctiva image to segment” (vedi figura sotto).

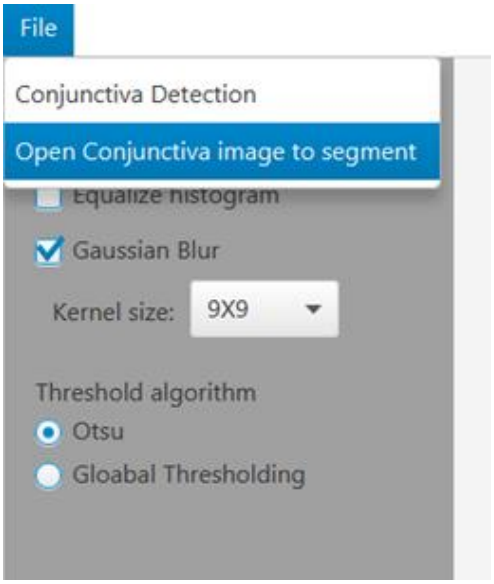

FIGURA 5.4: Voce del menu “Open”

Selezionando la voce “Open Conjunctiva image to segment”, il software permette di selezionare una o più immagini di congiuntive (di cui è stata già individuata la ROI tramite la funzionalità “Congiuntiva detection”), a cui viene applicata la sequenza di algoritmi di *image processing* specificata nella *sidebar*. Selezionando una sola immagine il software visualizza il prospetto dell’immagine sotto.

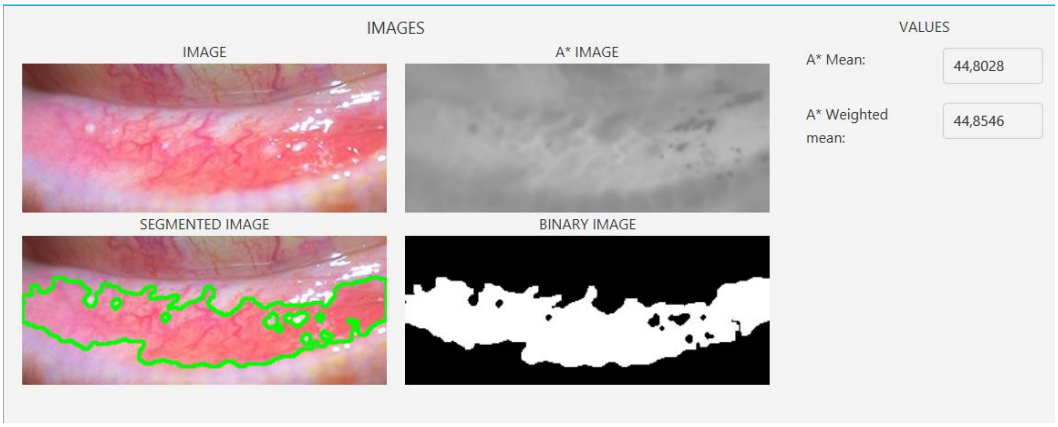

FIGURA 5.5: applicazione algoritmi di segmentazione su un'immagine

Sul lato sinistro sono presenti in ordine, da sinistra verso destra, le seguenti immagini:

- L'immagine originale selezionata (*image*);
- La componente a* estratta dall'immagine (*a* image*);
- L'immagine segmentata (*segmented image*);
- L'immagine binaria (*binary image*).

Sul lato destro, nella sezione *values*, sono presenti la media e la media pesata di a* estratte dalle immagini (il calcolo delle medie verrà descritto nel prossimo capitolo).

Il range di a* [1;255] è stato convertito nel range [-128;127) seguendo le specifiche descritte nella documentazione della classe Android ColorUtils [19]. É stata utilizzata questa convenzione per uniformare i valori con quelli ottenuti da lavori effettuati da altri tesisti (che utilizzano appunto questo range per i valori di a*), in modo da essere direttamente confrontabili.

É possibile, inoltre, selezionare più immagini. Al momento della selezione di più immagini (sempre tramite la funzionalità "Open Conjunctiva image to segment") il software richiede, inoltre, la selezione di un file con formato CSV dove vengono indicati i valori di HB, di ciascuna immagine selezionata. Il file deve essere scritto con la seguente formattazione:

[nome file;valore hb]

[nome file;valore hb]

…

Nel caso in cui i nomi delle immagini scritte nel file CSV non corrispondessero alle immagini selezionate, verrà sollevata un'eccezione. In caso contrario, verrà tracciato un grafico in cui vengono tracciati i valori medi di a* con i corrispondenti valori di Hb (vedi figura sotto).

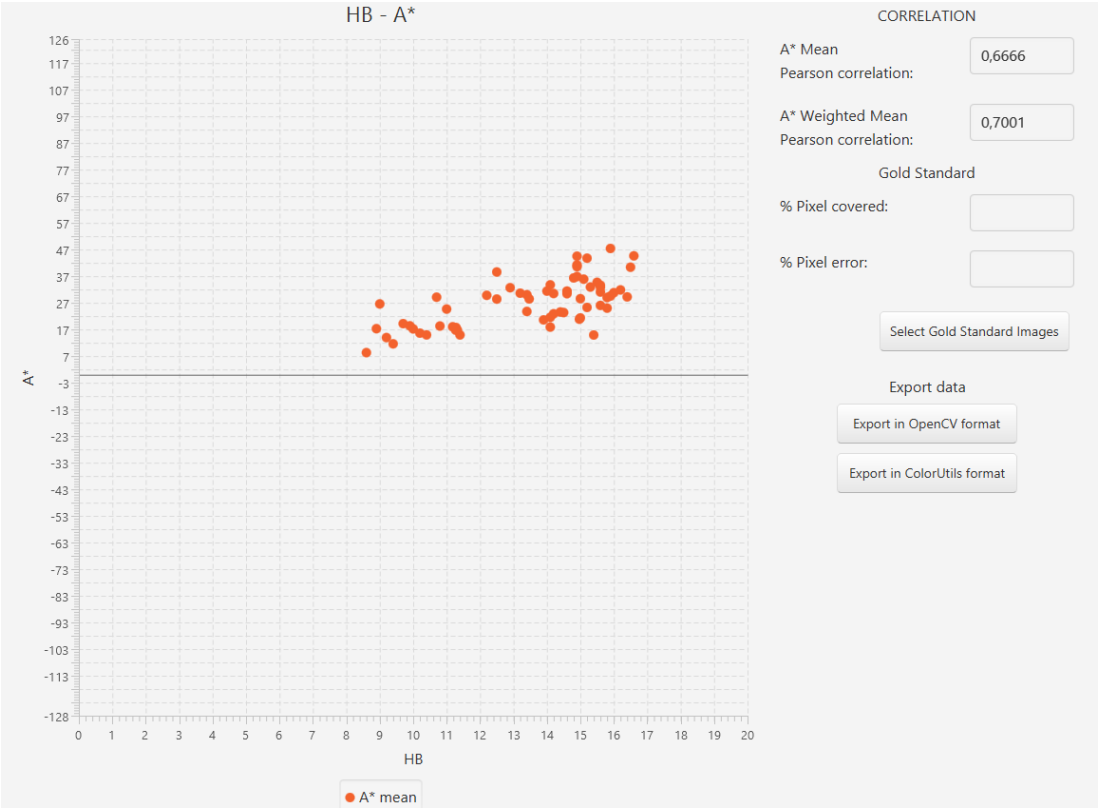

FIGURA 5.6: grafico visualizzato e correlazione con i valori di Hb

Sulla parte destra sono presenti le correlazioni tra i valori di a* e i valori di Hb (il cui calcolo verrà descritto nel prossimo capitolo). La visione e aggiornamento dei valori in tempo reale, in base agli algoritmi selezionati, ha permesso di ottimizzare il processo di segmentazione in base alla correlazione con i valori di Hb.

Il software, inoltre, permette di ottimizzare i risultati anche in base a delle immagini considerate "Gold standard". Per gold standard si intendono delle immagini in cui, l'individuazione dell'area di interesse (la congiuntiva palpebrale), è stata precedentemente fatta a mano (tramite un software di photo editing come Photoshop). Sono delle immagini binarie in cui si è manualmente selezionata l'area

di interesse impostando questi pixel in bianco e, la restante parte, considerata come sfondo, in nero.

É possibile selezionare le immagini Gold standard tramite il pulsante “Select Gold Standard Images”. Le immagini gold standard, ovviamente, devono corrispondere alle immagini selezionate inizialmente (in modo da poter effettuare un confronto). In caso contrario, verrà sollevata un’eccezione.

Le metriche utilizzate per misurare la bontà del processo di segmentazione, in correlazione con le foto considerate gold standard, sono due:

- % di pixel correttamente considerati nella ROI;
- % di pixel erroneamente considerati nella ROI.

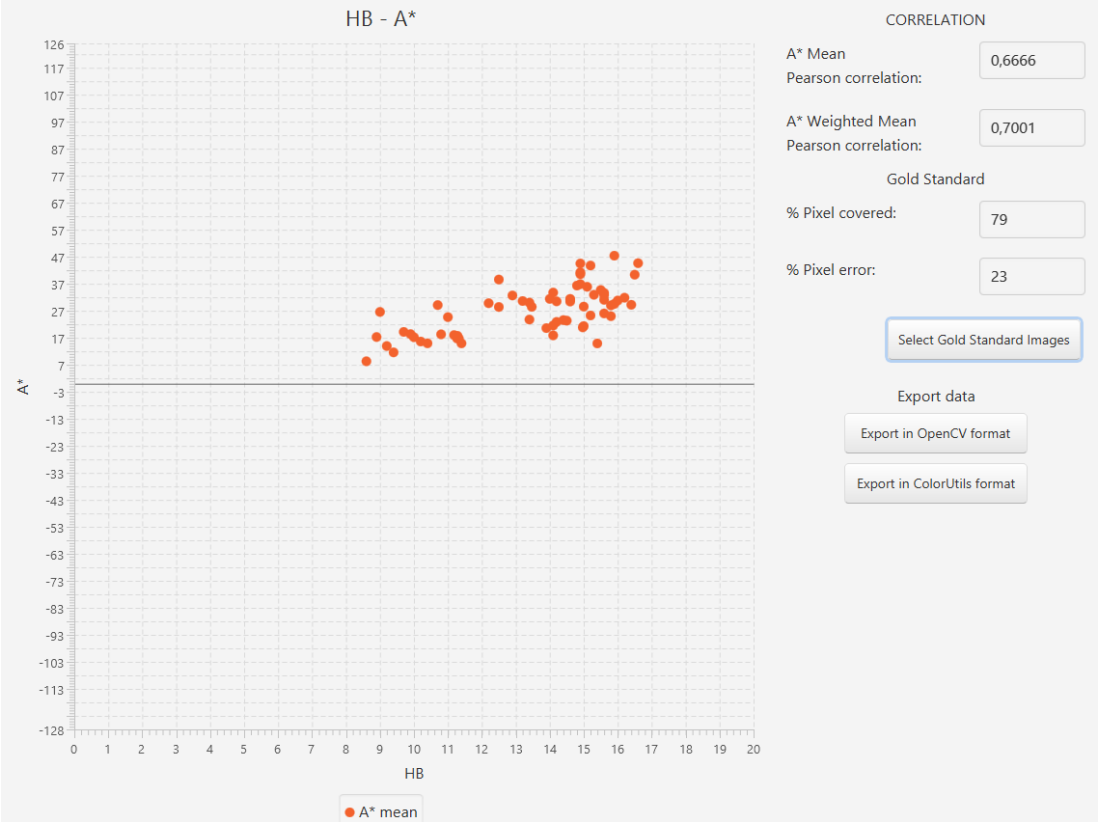

FIGURA 5.7: grafico visualizzato e correlazione con le foto Gold standard

Sempre nella parte destra, vi è una sezione dedicata all’esportazione dei dati presenti nel grafico, in un file CSV. È possibile esportare i dati sia nel formato OpenCV (ovvero con i valori di a* nel range [1-255]), che nel formato di ColorUtils

specificato nella documentazione Android [19] (ovvero con i valori di a* nel range [-128;127)).

Capitolo 6

Conclusioni

6.1 Correlazione di Pearson

La correlazione tra i valori di a* estratti dalle immagini e i relativi valori di Hb, è stata misurata tramite la correlazione di Pearson.

La correlazione indica la tendenza che hanno due variabili a variare insieme (covariare). La relazione è di tipo lineare se, rappresentata sugli assi, ha la forma di una retta.

In statistica, l'indice di correlazione di Pearson tra due variabili statistiche X e Y, è così definito:

$$\rho_{XY} = \frac{\sigma_{XY}}{\sigma_X \sigma_Y}$$

Dove σ_{XY} è la covarianza tra X e Y; σ_X, σ_Y sono le due deviazioni standard. L'indice esprime un'eventuale relazione di linearità tra le due variabili. La covarianza tra due variabili statistiche è definita come:

$$\sigma_{XY} = \frac{1}{N}\sum_{i=1}^{N}(x_i - \bar{x})(y_i - \bar{y})$$

Dove $\bar{x}$ e $\bar{y}$ rappresentano le medie delle variabili X e Y. La deviazione standard di una variabile X è così definita:

$$\sigma x = \sqrt{\frac{\sum_{i=1}^{N}(x_i - \bar{x})^2}{N}}$$

Il coefficiente assume valori compresi nel range [-1;1]. A seconda del valore assunto, distinguiamo tre tipi di relazione:

- Le variabili X e Y si dicono direttamente correlate se $\rho_{XY} > 0$;
- Le variabili X e Y si dicono non correlate se $\rho_{XY} = 0$;
- Le variabili X e Y si dicono inversamente correlate se $\rho_{XY} < 0$.

Distinguiamo inoltre, per le variabili direttamente correlate:

- Correlazione debole se $0 < \rho_{XY} < 0{,}3$;
- Correlazione moderata se $0{,}3 < \rho_{XY} < 0{,}7$;
- Correlazione forte se $\rho_{XY} > 0{,}7$ [20].

6.2 Correlazione con i valori di Hb

Come primo risultato è stata calcolata la correlazione di Pearson tra i valori di a* medi estratti da ogni foto e i corrispettivi valori di Hb. Le correlazioni vengono aggiornate nella sezione *values* del software descritto nel capitolo precedente.

Eseguendo una semplice binarizzazione di *Otsu* sulle immagini delle congiuntive, inoltre, il grafico tracciato dal software ha messo in evidenza alcune caratteristiche dei dati:

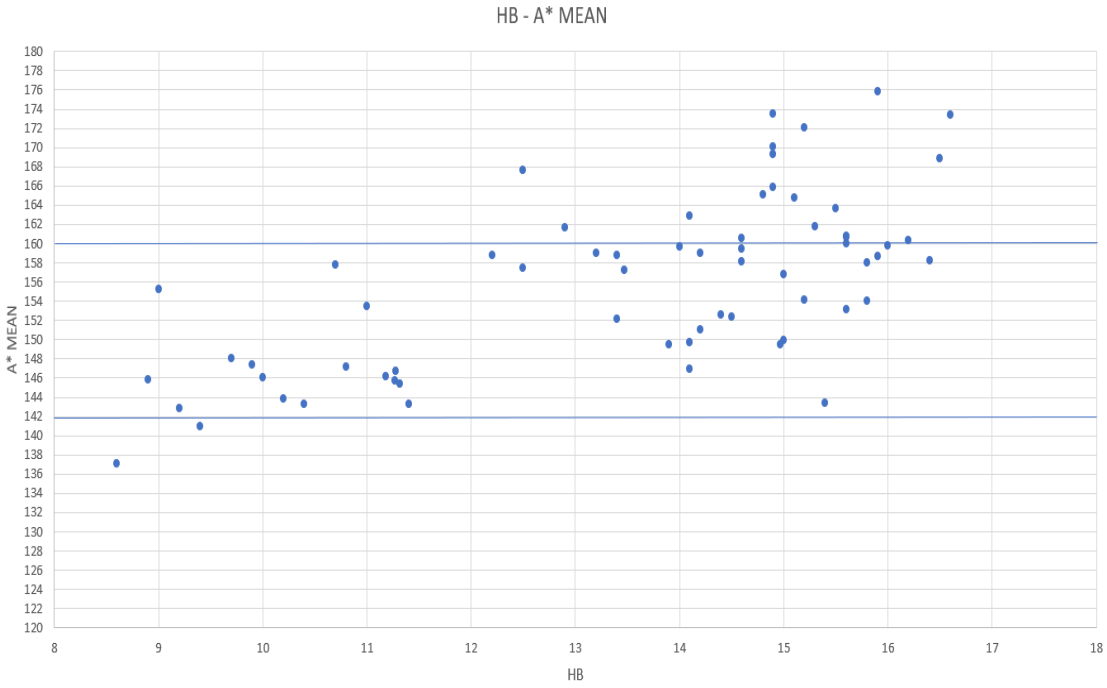

FIGURA 6.1: dati del software tracciati su un grafico Excel

Come si evince dal grafico, i valori di a* medi nel *range* [142;160) non sono discriminanti, in quanto, sia le foto di pazienti con basso valore di Hb, che quelle con alti valori di Hb, hanno valori di a* medi appartenenti a questo *range*.

Si è così pensato di dividere i valori di a* in due gruppi e assegnare a ciascuno di essi un peso in base alla loro importanza:

- Per i valori di a* appartenenti al *range* $[142; 160)$, si è assegnato un peso di $0{,}01$, in quanto non sono discriminanti (per il motivo descritto precedentemente);
- Per i valori di a* maggiori uguali di 160 e i valori minori di 142, si è assegnato un peso uguale a 1. É stato scelto un peso maggiore in quanto, il grafico mostra che: le persone con valori di Hb maggiori, tendono ad avere valori di a* medi maggiori uguali di 160, e le persone con valori di Hb minori, tendono ad avere valori di a* medi minori di 142. I valori di a* presenti in questo range, allora, sono maggiormente discriminanti.

É stata quindi calcolata la media pesata con la seguente formula:

$$\bar{a}_w = \frac{\sum_{i=1}^{n} a_i p_i}{\sum_{i=1}^{n} p_i}$$

Dove p_i rappresentano i pesi assegnati ai singoli valori a_i.

Sono state quindi ottenute, applicando alle foto la sequenza degli algoritmi descritta nel capitolo 4, le seguenti correlazioni:

- Correlazione di Pearson tra i valori di Hb con la semplice media dei valori di a*: **0,66**;
- Correlazione di Pearson tra i valori di Hb con la media pesata dei valori di a*: **0,70**.

È stata quindi trovata una correlazione forte tra media pesata dei valori di a* ed emoglobina misurata. Nella tabella sotto, sono riportati, per ciascuna foto di congiuntiva palpebrale utilizzata nella correlazione, il valore di emoglobina (Hb), il valore della media dei valori di a* (*a* mean*) e il valore della media pesata di a* (*a* weighted mean*).

HB	*a* mean*	*a* weighted Mean*
9.4	140.5981	139.4119907
10.4	143.9628	140.3478957
10.2	144.6287	140.0036998
12.5	157.4077	163.776276
9.9	147.358	141.4706278
13.4	152.7575	162.5758429
14.6	159.4305	165.8968895
11	153.6616	162.6162617
14.2	159.4982	164.1636726
9.7	148.1829	153.1285981
9.2	142.9536	139.9139018
8.9	146.2885	141.6305902
10	146.21	142.3869554
11.4	143.9431	140.2453796
10.8	147.2841	143.3275474
11.27	145.7557	139.8276181
10.7	158.1132	165.9279866
11.32	145.8134	140.278762
11.28	146.734	149.8780771
13.47	157.4592	162.8361078
14.97	149.8637	149.8211965
11.18	146.9939	140.7402727
9	155.5794	163.9145398
8.6	137.3049	136.7186908
14.4	152.5591	160.6517288
15	157.5544	165.3738922
14.1	150.5529	157.8421095
15	150.3358	151.107459
15.6	155.0284	163.1616652
15.4	143.9184	139.4539851

16	159.8007	165.8302735
15.9	158.5362	166.5302869
13.9	149.5911	158.5474063
15.3	161.9296	166.0038287
15.2	154.2746	163.0350386
12.9	161.6678	165.6412438
14	160.406	164.6853445
14.9	165.8733	167.671472
15.1	164.8765	168.0329233
14.9	170.234	170.476275
14.9	173.509	173.5508845
12.5	167.5766	167.8731922
15.9	176.4062	176.4639632
15.2	172.7511	172.900477
16.6	173.6353	174.3779505
16.5	169.372	169.7416102
15.8	154.0152	160.3261478
16.2	160.8431	164.2992939
15.6	161.8419	164.5042746
16.4	158.2285	165.3980858
15.6	162.5997	166.8573391
14.6	160.1049	164.8144642
15.5	163.6901	166.1649415
14.2	151.8846	159.0109893
14.5	152.3436	159.3438923
13.4	159.0333	162.8467951
14.1	146.9046	146.8655018
15.8	158.0034	162.8491109
13.2	159.6198	163.5295379
14.1	162.7419	165.8426439
15.6	160.063	164.351991

14.6	160.4017	164.84636
12.2	158.8124	164.3050615
14.9	169.5558	169.6889529
14.8	165.3411	167.6059307

TABELLA 6.1: dati estratti dalle foto di congiuntive

6.3 Risultati ottenuti in termini di % dell'area segmentata

Come detto nel capitolo precedente, il software permette di vedere in tempo reale la percentuale di area segmentata rispetto a delle foto considerate "Gold standard", ovvero immagini binarie in cui si è precedentemente delineata l'area di interesse. Tramite la sequenza di algoritmi di image analysis implementata, si sono ottenuti i seguenti risultati:

- % pixel correttamente considerati nella ROI: **79 %**;
- % pixel erroneamente considerati nella ROI: **23 %**.

6.4 Considerazioni finali

Il lavoro svolto ha avuto un duplice scopo: trovare un metodo universale per l'individuazione e segmentazione automatica della congiuntiva palpebrale, e trovare una correlazione maggiore con i valori di Hb.

A seguito del lavoro, si può dire che gli obiettivi prefissati sono stati raggiunti, in quanto: si è implementato un algoritmo di *Object Detection* per l'individuazione automatica della congiuntiva e, inoltre, si è implementato una catena di algoritmi di image analysis al fine di segmentare la congiuntiva palpebrale. Il processo di segmentazione è stato ottimizzato da due differenti punti di vista:

- In base alla correlazione con i valori di Hb: si è ottenuto una correlazione forte (0,7) tra i valori di a* estratti dallo spazio di colori CIELAB e i valori di Hb;
- In termini di percentuale di area correttamente segmentata: circa l'80%.

Si è inoltre prodotto un software che è stato un valido strumento per lo studio degli algoritmi implementati e per la visione dei risultati da essi forniti. Questo può essere utilizzato per eventuali sviluppi futuri.

Sviluppi futuri

I principali sviluppi futuri previsti, sono:

- Acquisizione di nuove foto di congiuntive;
- Migliorare la performance del classificatore sviluppato;
- Allenare il classificatore sviluppato con un training set più ampio;
- Adattare il classificatore alla fotocamera dello smartphone utilizzato con il dispositivo di acquisizione: in questo modo la classificazione avverrà in *real-time*. Essendo l'algoritmo di classificazione molto veloce nella face di *detection*, può essere utilizzato nella fase di acquisizione delle foto per avere un *feedback* in tempo reale (ovvero, se la congiuntiva è correttamente inquadrata o meno);
- Implementazione di nuovi algoritmi di *image analysis* nel software;
- Trovare una correlazione più alta tra valori di a* e i valori di Hb;
- Aumentare la percentuale di area correttamente segmentata dal software;
- Rendere il software portabile su altri sistemi operativi oltre Windows 10.

Bibliografia e sitografia

[1] https://it.wikipedia.org/wiki/Anemia

[2] B. De Benoist, E. McLean, I. Egli, and M. Cogswell, “Who global database on anaemia,” Geneva: WHO, pp. 1993–2005, 2008.

[3] http://www.istitutopalatucci.it/libri/scienze/semeiotica%20libro.pdf

[4] T. N. Sheth, N. K. Choudhry, M. Bowes, A. S. Detsky et al., “The relation of conjunctival pallor to the presence of anemia,” Journal of general internal medicine, vol. 12, no. 2, pp. 102–106, 1997

[5] M. G. N. Spinelli, J. M. P. Souza, S. B. de Souza, and E. H. Sesoko, “Reliability and validity of palmar and conjunctival pallor for anemia detection purposes,” Revista de Saúde Pública, vol. 37, no. 4, pp. 404– 408, 2003.

[6] C. I. Sanchez-Carrillo, T. de Jesus Ramirez-Sanchez, B. J. Selwyn et al., “Test of a noninvasive instrument for measuring hemoglobin concentration,” International journal of technology assessment in health care, vol. 5, no. 04, pp. 659–667, 1989.

[7] S. Suner, G. Crawford, J. McMurdy e G. Jay “Non-invasive determination of hemoglobin by digital photography of palpebral conjunctiva”, 2007

[8] Petrosino, L'algoritmo SLIC SuperPixels per l’acquisizione assistita e l’analisi della congiuntiva palpebrale

[9] https://opencv.org/

[10] Rafael C. Gonzalez, Richard E. Woods “Digital Image Processing”, Prentice Hall, New Jersey, 2001Second Edition

[11] Paul Viola and Michael J. Jones. Rapid Object Detection using a Boosted Cascade of Simple Features. IEEE CVPR, 2001

[12] Rainer Lienhart and Jochen Maydt. An Extended Set of Haar-like Features for Rapid Object Detection. IEEE ICIP 2002, Vol. 1, pp. 900-903, Sep. 2002

[13] http://www.dmi.unict.it/~fstanco/lezioni_IEM_2007_2008/Lez%2007%20interpolazione.pdf

[14] Yoav Freund and Robert E. Schapire. A decision-theoretic generalization of on-line learning and an application to boosting. In Computational Learning Theory: Eurocolt ’95, pages 23–37. Springer-Verlag, 1995.

[15] http://note.sonots.com/SciSoftware/haartraining.html

[16] http://it.ccm.net/contents/722-la-codifica-cie-lab-l-a-b

[17] Suzuki, S. and Abe, K., Topological Structural Analysis of Digitized Binary Images by Border Following. CVGIP 30 1, pp 32-46 (1985)

[18] Andrew W. Fitzgibbon, R.B.Fisher. A Buyer’s Guide to Conic Fitting. Proc.5th British Machine Vision Conference, Birmingham, pp. 513-522, 1995

[19] https://developer.android.com/reference/android/support/v4/graphics/ColorUtils.html#RGBToLAB(int, int, int, double[])

[20] https://it.wikipedia.org/wiki/Indice_di_correlazione_di_Pearson

[21]http://www.computer-vision-software.com/blog/2009/11/faq-opencv-haartraining/

[22] Paul Viola, Michael J. Jones. Robust real-time face detection. International Journal of Computer Vision, 57(2):137–154, 2004.

[23] Vito antonio Bevilacqua, Giovanni Dimauro, Francesco maria Marino, Antonio Brunetti, Fabio Cassano, Antonio Di Maio, Enrico Nasca, Gianpaolo Francesco Trotta, Francesco Girardi, Angelo Ostuni and Attilio Guarini, “A novel approach to evaluate blood parameters using computer vision techniques” 2016

[24] http://www.my-personaltrainer.it/salute-benessere/congiuntiva.html

Ringraziamenti

Giunto alla fine del percorso triennale, cominciato quasi per caso e diventato una vera passione, ci sono dei ringraziamenti che mi sento in dovere di fare. Non è stato solo il mio impegno che mi ha fatto raggiungere questo traguardo, ma anche il supporto di alcune persone.

In primis vorrei ringraziare la mia famiglia, per l'incoraggiamento e il sostegno morale ed economico che mi hanno dato sin da piccolo, durante il mio percorso di studi. Un doveroso ringraziamento va al prof. Giovanni Dimauro, per la disponibilità dimostrata durante tutto il lavoro di tesi.

Desidero inoltre ringraziare tutti i miei amici, con cui ho condiviso gioie e delusioni. Un grazie va al mio amico Davide, che mi ha incoraggiato a intraprendere il corso di studi in Informatica e con il quale ho cominciato gli studi.

Ringrazio tutte le persone che mi hanno ispirato e che hanno contribuito ad accrescere le mie passioni. Nella vita si diventa migliore soprattutto grazie agli insegnamenti ricevuti e alle persone incontrate.

Come ultimo ringraziamento, non per importanza, vorrei ringraziare la mia ragazza Claudia, per l'affetto, i momenti condivisi e i piccoli gesti di ogni giorno.

Listato codice

Di seguito il codice delle classi principali del software sviluppato.

Classe AstarSegmentation.java

```
/**
 * class to implement image analysis algorithms
 * on digital photos
 *
 * @author Luigi
 *
 */
public class AstarSegmentation {
	private final static Scalar defaultColor = new Scalar(0, 255, 0);
	private final static int defaultTickness = 1;

	public final static int
	MORPH_ERODE = 0,
	MORPH_DILATE = 1,
	MORPH_OPEN = 2,
	MORPH_CLOSE = 3;

	public final static int
	FILL_ALL_REGIONS = 0,
	FILL_ONLY_MAX_REGION_WITH_ITS_CHILDREN = 1,
	FILL_ONLY_MAX_REGION_WITHOUT_ITS_CHILDREN = 2;

	private Mat image;		//BGR source image
	private Mat astarComponent;
	private Mat binaryImage;

	/**
	 * Constructor
	 * @param path to image
	 * @throws FileNotFoundException
	 */
	public AstarSegmentation(String path) throws FileNotFoundException {
		File f = new File(path);
		if(!f.isFile()){
			throw new FileNotFoundException();
		}
		image = Imgcodecs.imread(path, Imgcodecs.IMREAD_COLOR);	//read in BGR format
	}

	/**
	 * Constructor
	 * @param image
	 * @throws InvalidArgumentException if the image is null
	 */
	public AstarSegmentation(Mat image) throws InvalidArgumentException{
		if(image == null || image.rows() == 0 || image.cols() == 0){
			throw new InvalidArgumentException("Image reading error");
		}
		this.image = image;
	}
```

```
/**
 *
 * @return BGR original image
 */
public Mat getImage() {
	return image;
}


/**
 * set the a* component in CIEL*a*b* color space
 * original image will not be modified
 * @param equalize: if true a* histogram will be equalized
 * @return this
 */
public AstarSegmentation setAstarComponent(boolean equalize){
	Mat src = image.clone();
	Imgproc.cvtColor(src, src, Imgproc.COLOR_BGR2Lab);

	ArrayList<Mat> channels = new ArrayList<>();
	Core.split(src, channels);

	if(equalize){
		Imgproc.equalizeHist(channels.get(1), channels.get(1));
	}

	this.astarComponent = channels.get(1);

	return this;
}

/**
 * perform gaussian blur on astarComponent image
 * @param structSize size of structuring element
 * @return true if gaussian blur has been performed
 * @throws PreliminaryOperationException will be thrown if setAstarComponent() has not been called
	before
 * {@link #setAstarComponent(boolean)}
 */
public boolean gaussianBlur(Size structSize) throws PreliminaryOperationException{
	if(astarComponent == null){
		throw new PreliminaryOperationException("You have to perform
			setAstarComponent() before");
	}
	if(structSize == null || structSize.height < 1 || structSize.width < 1){
		return false;
	}

	Imgproc.GaussianBlur(astarComponent, astarComponent, structSize, 0);

	return true;
}

/**
 *
 * @return a copy of a* component if it is not null
 */
public Mat getAstarComponent() {
	return astarComponent == null? null : astarComponent.clone();
}
```

```
/**
 * perform a global threshold to get a binary image from a* component
 * @param threshold: pixel value with a* intensity greater than it will be converted to white;
 *                   the other pixels will be converted to black
 * @return this
 * @throws PreliminaryOperationException will be thrown if setAstarComponent() has not been called
        before
 * {@link #setAstarComponent(boolean)}
 */
public AstarSegmentation setBinaryImage(int threshold) throws PreliminaryOperationException {
        if(astarComponent == null){
                throw new PreliminaryOperationException("You have to perform
                        setAstarComponent() before");
        }
        Mat astar = astarComponent.clone();

        //apply a simple threshold
        Imgproc.threshold(astar, astar, threshold, 255, Imgproc.THRESH_BINARY);

        this.binaryImage = astar;
        return this;
}

/**
 * perform Otsu thresholding to get a binary image from a* component
 * @return this
 * @throws PreliminaryOperationException will be thrown if setAstarComponent() has not been called
                before
 * {@link #setAstarComponent(boolean)}
 */
public AstarSegmentation setBinaryImageByOtsu() throws PreliminaryOperationException{
        if(astarComponent == null){
                throw new PreliminaryOperationException("You have to perform
                        setAstarComponent() before");
        }
        Mat astar = astarComponent.clone();

        //Apply otsu
        Imgproc.threshold(astar, astar, 0, 255, Imgproc.THRESH_BINARY + Imgproc.THRESH_OTSU);

        this.binaryImage = astar;
        return this;
}

/**
 *
 * @return a copy of binary image if it is not null
 */
public Mat getBinaryImage() {
        return binaryImage == null? null : binaryImage.clone();
}

/**
 * Perform a morphology operation on the binary image
 * @param indexOperation: index of operation to perform; 0, 1, 2, 3 stay respectively for ERODE,
                DILATE, OPEN, CLOSE
 * @param structSize: the size of the structuring element by which to perform the operation
 * @return false if structSize null or < 1 or indexOperation is not in the range specified; true otherwise
 * @throws PreliminaryOperationException will be thrown if setBinaryImage() has not been called
                before
```

```
 * {@link #setBinaryImage(int)}
 */
public boolean morphologyOperation(int indexOperation, Size structSize) throws
                PreliminaryOperationException {
        if(binaryImage == null){
                throw new PreliminaryOperationException("You have to perform setBinaryImage()
                        before");
        }
        if(structSize == null || structSize.height < 1 || structSize.width < 1){
                return false;
        }

        Mat kernel = Imgproc.getStructuringElement(Imgproc.MORPH_RECT, structSize);
        if(indexOperation == 0){
                Imgproc.morphologyEx(binaryImage, binaryImage, Imgproc.MORPH_ERODE,
                kernel);
        }
        else if(indexOperation == 1){
                Imgproc.morphologyEx(binaryImage, binaryImage, Imgproc.MORPH_DILATE,
                kernel);
        }
        else if(indexOperation == 2){
                Imgproc.morphologyEx(binaryImage, binaryImage, Imgproc.MORPH_OPEN, kernel);
        }
        else if(indexOperation == 3){
                Imgproc.morphologyEx(binaryImage, binaryImage, Imgproc.MORPH_CLOSE, kernel);
        }
        else{
                return false;                   //index is not in the range allowed
        }

        return true;
}

/**
 * Perform region filling operation: try to find contours in the binary image and then fill only a specified
        subset
 * @param indexOperation: index of operation to perform; 0, 1, 2 stay respectively for
        FILL_ALL_REGIONS, FILL_ONLY_MAX_REGION_WITH_ITS_CHILDREN,
 *      FILL_ONLY_MAX_REGION_WITHOUT_ITS_CHILDREN
 * @return false if indexOperation is not in the range specified; true otherwise
 * @throws PreliminaryOperationException will be thrown if setBinaryImage() has not been called
                before
 * {@link #setBinaryImage(int)}
 * @link {@link ContourUtils}
 */
public boolean regionFilling(int indexOperation) throws PreliminaryOperationException {
        if(binaryImage == null){
                throw new PreliminaryOperationException("You have to perform setBinaryImage()
                        before");
        }

        if(indexOperation == 0){        //all the contours are taken into account (holes included)
                List<MatOfPoint> contours = ContourUtils.getAllContours(binaryImage);
                ContourUtils.fillRegion(binaryImage, contours, 255);
        }
        else if(indexOperation == 1){ //only biggest contour (and its sub-contours) are taken into
                                                account (holes included)
                List<MatOfPoint> contours =
                        ContourUtils.getMaxContourAndItsChildren(binaryImage);
                ContourUtils.fillRegion(binaryImage, contours, 255);
```

```
                List<MatOfPoint> contoursToDelete = ContourUtils.getMinContours(binaryImage);
                ContourUtils.fillRegion(binaryImage, contoursToDelete, 0);
        }
        else if(indexOperation == 2){  //only biggest contour is taken into account (holes excluded)
                List<MatOfPoint> contours = ContourUtils.getAllContoursExceptMax(binaryImage);
                ContourUtils.fillRegion(binaryImage, contours, 0);
        }
        else{
                return false;           //as default in binary image all the contours are taken into
                                        account (holes excluded)
        }

        return true;
}



/**
 * Try to fit an ellipse in binary image
 * @return false if it's not possible to fit an ellipse; true otherwise
 * @throws PreliminaryOperationException will be thrown if setBinaryImage() has not been called
                                                before
 * {@link #setBinaryImage(int)}
 */
public boolean ellipseFitting() throws PreliminaryOperationException {
        if(binaryImage == null){
                throw new PreliminaryOperationException("You have to perform setBinaryImage()
                                before");
        }

        MatOfPoint maxContour = ContourUtils.getMaxContour(binaryImage);
        if(maxContour == null){                 //there are no contours
                return false;
        }

        MatOfPoint2f contourMax = new MatOfPoint2f(maxContour.toArray());

        if(maxContour.size().height >= 5){                      //to fit an ellipse at least 5 point are
                                                                        necessary
                RotatedRect minEllipse = Imgproc.fitEllipse(contourMax);

                Mat temp = Mat.zeros(binaryImage.rows(),  binaryImage.cols(), 0);
                Imgproc.ellipse(temp, minEllipse, new Scalar(255), -1);

                double[] b = {0};
                for(int i = 0; i < binaryImage.rows(); i++){
                        for(int j = 0; j < binaryImage.cols(); j++){
                                if(temp.get(i, j)[0] != 255){
                                        binaryImage.put(i, j, b);
                                }
                        }
                }

                return true;
        }

        return false;
}
```

```
/**
 * Segment the original image according to binary image
 * @return the segmented image
 * @throws PreliminaryOperationException will be thrown if setBinaryImage() has not been called
                                                before
 * {@link #setBinaryImage(int)}
 */
public Mat getSegmentedImage() throws PreliminaryOperationException{
        if(binaryImage == null){
                throw new PreliminaryOperationException("You have to perform setBinaryImage()
                                                                before");
        }

        Mat src = image.clone();
        Mat hierarchy = new Mat();
        ArrayList<MatOfPoint> contours = new ArrayList<>();

        Imgproc.findContours(binaryImage, contours, hierarchy, Imgproc.RETR_TREE,
                                        Imgproc.CHAIN_APPROX_SIMPLE);
        Imgproc.drawContours(src, contours, -1, defaultColor, defaultTickness);

        return src;
}

/**
 * Segment the original image according to binary image
 * @param color the color by which draw the region of interest
 * @param tickness of color
 * the segmented image
 * @throws PreliminaryOperationException will be thrown if setBinaryImage() has not been called
                                                before
 * {@link #setBinaryImage(int)}
 */
public Mat getSegmentedImage(Scalar color, int tickness) throws PreliminaryOperationException{
        if(binaryImage == null){
                throw new PreliminaryOperationException("You have to perform setBinaryImage()
                                                                before");
        }
        if(color == null || color.val == null){
                color = defaultColor;
        }
        if(tickness <= 0){
                tickness = defaultTickness;
        }

        Mat src = image.clone();
        Mat hierarchy = new Mat();
        ArrayList<MatOfPoint> contours = new ArrayList<>();

        Imgproc.findContours(binaryImage, contours, hierarchy, Imgproc.RETR_TREE,
                                        Imgproc.CHAIN_APPROX_SIMPLE);
        Imgproc.drawContours(src, contours, -1, color, tickness);

        return src;
}

/**
 * Return mean value of astar component not equalized
 * @param threshold: condsider olny pixel major than threshold
 * @return mean value
```

```
 * @throws PreliminaryOperationException will be thrown if setBinaryImage() has not been called
                                                before
 * {@link #setBinaryImage(int)}
 */
public double getMeanValue(int threshold) throws PreliminaryOperationException {
        if(binaryImage == null){
                throw new PreliminaryOperationException("You have to perform setBinaryImage()
                                                before");
        }

        return AstarUtils.getMeanValue(getNotEqualizedAstarComponent(), binaryImage, threshold);
}


/**
 * Calculate the weighted mean of not equalized astar component.
 *
 * @param boundaryPixels: array of pairs where each pair is a range of pixel (left bounday included,
        right excluded) considered in weighted mean (with the corresponding weight specified in
        weights array)
 * @param weights used in mean
 * @return the weighted mean
 * @throws PreliminarOperationException will be thrown if setBinaryImage() has not been called before
 * @throws InvalidArgumentException if boundaryPixels or weights array are null/empty; if there is an
        element alone in pairs array;
 * if for each pair is not specified a weight; if a left boundary is greather than its right boundary
 * {@link #setBinaryImage(int)}
 */
public double getWeightedMeanValue(int[] boundaryPixels, double[] weights) throws
        PreliminaryOperationException, InvalidArgumentException {
        if(binaryImage == null){
                throw new PreliminaryOperationException("You have to perform setBinaryImage()
                                        before");
        }

        if(boundaryPixels == null || weights == null || boundaryPixels.length == 0 || weights.length
        == 0){
                throw new InvalidArgumentException("Empty parameters are not allowed");
        }

        if((boundaryPixels.length % 2) != 0){
                throw new InvalidArgumentException("Boundary pixels array must constains pairs
                (length must be even)");
        }

        if((boundaryPixels.length / 2) != weights.length){
                throw new InvalidArgumentException("The number of boundary pixel pairs must be
                equal to weights length");
        }

        for(int i = 0; i < boundaryPixels.length; i++){
                if((i % 2 == 0) && boundaryPixels[i] > boundaryPixels[i+1]){
                        throw new InvalidArgumentException("Left boundary pixel pairs must be
                                minor equal to right boundary pixel");
                }
        }

        return AstarUtils.getWeightedMeanValue(getNotEqualizedAstarComponent(), binaryImage,
                boundaryPixels, weights);
}
```

```
	/**
	 *
	 * @return the not equalized astar component
	 */
	private Mat getNotEqualizedAstarComponent(){
		Mat src = image.clone();
		Imgproc.cvtColor(src, src, Imgproc.COLOR_BGR2Lab);

		ArrayList<Mat> channels = new ArrayList<>();
		Core.split(src, channels);

		return channels.get(1);
	}

}
```

Classe astar.segmentation.Performance.java

```
/**
 * Class used to test segmentation algorithm
 *
 * @author Luigi
 *
 */
public class Performance {
	private Mat goldStandard;	//segmented image hand made; white pixel = foreground (ROI), black =
					background
	private Mat segmentedImage;		//automatically segmented image

	/**
	 * create a new instance of Performance
	 * @param goldStandard
	 * @param segmentedImage
	 */
	public Performance(Mat goldStandard, Mat segmentedImage){
		this.goldStandard = goldStandard;
		this.segmentedImage = segmentedImage;
	}

	/**
	 *
	 * @return percentage of pixels covered
	 */
	public double getPerformance(){
		int n = 0;
		int pixelCovered = 0;
		for(int i = 0; i < goldStandard.rows(); i++){
			for(int j = 0; j < goldStandard.cols(); j++){
				if(goldStandard.get(i, j)[0] == 255){
					n++;
				}

				if(goldStandard.get(i, j)[0] == 255 && segmentedImage.get(i, j)[0] == 255){
					pixelCovered++;
```

```
                                        }
                                }
                        }

                        if(n == 0){
                                n = 1;
                        }

                        return pixelCovered / (double) n * 100;
        }

        /**
         *
         * @return percentage of pixel wrongly considered in ROI
         */
        public double getError(){
                int n = 0;
                int errorPixel = 0;
                for(int i = 0; i < goldStandard.rows(); i++){
                        for(int j = 0; j < goldStandard.cols(); j++){
                                if(segmentedImage.get(i, j)[0] == 255){
                                        n++;
                                }

                                if(goldStandard.get(i, j)[0] != 255 && segmentedImage.get(i, j)[0] == 255){
                                        errorPixel++;
                                }
                        }
                }

                if(n == 0){
                        n = 1;
                }

                return errorPixel / (double) n * 100;
        }
}
```

Classe ConjunctivaDetection.java

```
/**
 * Class used to perform Conjunctiva detection in the image
 *
 * @author Luigi
 *
 */
public class ConjunctivaDetection{
        private CascadeClassifier classifier;
        private Mat src;
        private Rect rectROI;

        /**
         * Create a new instance of Cascade Classificator
         * @param cascadeFilePath

         * @throws FileNotFoundException
         */
```

```
public ConjunctivaDetection(String cascadeFilePath) throws FileNotFoundException{
	File f = new File(cascadeFilePath);
	if(! f.exists()){
		throw new FileNotFoundException();
	}

	classifier = new CascadeClassifier(cascadeFilePath);
}

/**
 * set the image where to detect the conjunctiva; if size is over the limit it will be resized
 * @param image
 * @throws InvalidArgumentException
 */
public void setSrc(Mat image) throws InvalidArgumentException{
	src = image.clone();
	if(image.size().width > ConjunctivaUtils.limit || image.size().height > ConjunctivaUtils.limit){
		src = ConjunctivaUtils.resize(image);
	}

}

/**
 *
 * @return the image
 */
public Mat getSrc(){
	return src;
}


/**
 * Try to find ROI (Region Of Interest)
 * @param image
 * @return ROI if there is; A rect that surround all the image otherwise
 * @throws InvalidArgumentException
 */
public void setROI(Mat image) throws InvalidArgumentException{
	//resize
	setSrc(image);

	MatOfRect detections = new MatOfRect();
	classifier.detectMultiScale(src, detections);

	double maxArea = 0;
	Rect rectROI = null;
	for (Rect r : detections.toArray()) {
		if((r.width * r.height) >= maxArea){
			maxArea = r.width * r.height;
			rectROI = new Rect(r.x, r.y, r.width, r.height);
		}
	}

	if(rectROI == null){
		//rect will be all the image
		rectROI = new Rect(0, 0, src.cols(), src.rows());
	}

	this.rectROI = rectROI;
}
```

```
	/**
	 *
	 * @return the ROI
	 */
	public Rect getROI(){
		return rectROI;
	}

	/**
	 *
	 * @return the Mat ROI
	 */
	public Mat getMatROI(){
		return new Mat(src, rectROI);
	}

}
```

Classe AstarViewController.java

```
/**
 * Controller of AstarView.fxml
 * Main controller of application
 *
 * @author gigib
 *
 */
public class AstarViewController implements Initializable{
	private static final String GLOBAL_THRESHOLD_ID = "global";

	public static final String NO_MORPH_ID = "noMorph";
	public static final String ERODE_ID = "erode";
	public static final String DILATE_ID = "dilate";
	public static final String OPEN_ID = "open";
	public static final String CLOSE_ID = "close";

	public static final String NO_REGION_FILLING_ID = "noRegionFill";
	public static final String FILL_ALL_REGIONS_ID = "region0";
	public static final String FILL_ONLY_MAX_REGION_WITH_ITS_CHILDREN_ID = "region1";
	public static final String FILL_ONLY_MAX_REGION_WITHOUT_ITS_CHILDREN_ID = "region2";


	private static final String structSizeDelimitator = "X";
	private static final ObservableList<String> structSizeOptions =
			FXCollections.observableArrayList(
					"3"+structSizeDelimitator+"3",
					"5"+structSizeDelimitator+"5",
					"7"+structSizeDelimitator+"7",
					"9"+structSizeDelimitator+"9",
					"11"+structSizeDelimitator+"11",
					"13"+structSizeDelimitator+"13",
					"15"+structSizeDelimitator+"15"
					);
	private static final String structSizeOptionDefault = "5"+structSizeDelimitator+"5";

	private static int[] boundary = {129, 142, 142, 160, 160, 255};
	private static double[] weights = {1, 0.01, 1};
```

```
@FXML
private CheckBox equalizeHistogram;
@FXML
private CheckBox gaussianBlur;
@FXML
private HBox gaussianParameter;
@FXML
private ToggleGroup thresholdAlgorithm;
@FXML
private VBox globalParameters;
@FXML
private Slider threshold;
@FXML
private Label thresholdValue;
@FXML
private ToggleGroup morphologicalOperations;
@FXML
private HBox morphKernel;
@FXML
private ComboBox<String> morphStructComboBox;
@FXML
private ComboBox<String> gaussianStructComboBox;
@FXML
private ComboBox<Integer> gaussianSigma;
@FXML
private ToggleGroup regionFilling;
@FXML
private CheckBox ellipseFitting;
@FXML
private Button submit;
@FXML
private HBox content;

private boolean equalize;
private boolean blur;
private Size gaussianStructSize;
private boolean otsu = false;
private int thresholdNumber;
private int morphOp;
private int regionFill;
private boolean ellipse;
private Size morphStructSize;
private Mat matImage;
private AstarSegmentation astarSegmentation;
private AstarImages imagesContent = new AstarImages();
private ScatterChartHb scatterChartContent = new ScatterChartHb();
private List<File> images;
private List<String> imageNames;
private List<Double> hbValues;
private List<File> goldStandardFileList;

private String exportOCV = "";
private String exportCU = "";

private Main mainApp;


/**
 * set the algorithms selected in the left sidebar
 */
private void setSegmentationValues(){
```

```
            RadioButton m = (RadioButton) morphologicalOperations.getSelectedToggle();
            RadioButton r = (RadioButton) regionFilling.getSelectedToggle();
            RadioButton a = (RadioButton) thresholdAlgorithm.getSelectedToggle();

            equalize = equalizeHistogram.isSelected();
            blur = gaussianBlur.isSelected();
            if(blur){
                        gaussianStructSize =
                        getSizeFromComboBox(this.gaussianStructComboBox.getValue());
            }

            if(!a.idProperty().getValue().equals(GLOBAL_THRESHOLD_ID)){
                        otsu = true;
            }
            else{
                        otsu = false;
                        thresholdNumber =
                        GuiUtils.getAstarValueInOpenCVFormatFromCU((int)this.threshold.getValue());
            }

            morphOp = getMorphologicalOperationIndex(m.idProperty().getValue());
            morphStructSize = getSizeFromComboBox(this.morphStructComboBox.getValue());


            regionFill = getContoursFindingOperationIndex(r.idProperty().getValue());
            ellipse = ellipseFitting.isSelected();
}

/**
 *
 * @param id
 * @return the morphological operation selected in the left sidebar
 */
private int getMorphologicalOperationIndex(String id){
            if(id.equals(ERODE_ID)){
                        return 0;
            }
            else if(id.equals(DILATE_ID)){
                        return 1;
            }
            else if(id.equals(OPEN_ID)){
                        return 2;
            }
            else if(id.equals(CLOSE_ID)){
                        return 3;
            }
            else{
                        return -1;
            }
}

/**
 *
 * @param id
 * @return the set of contours to find in the binary image
 */
private int getContoursFindingOperationIndex(String id){
            if(id.equals(FILL_ALL_REGIONS_ID)){
                        return 0;
            }
            else if(id.equals(FILL_ONLY_MAX_REGION_WITH_ITS_CHILDREN_ID)){
```

```
                return 1;
        }
        else if(id.equals(FILL_ONLY_MAX_REGION_WITHOUT_ITS_CHILDREN_ID)){
                return 2;
        }
        else{
                return -1;
        }
}

/**
 *
 * @param size
 * @return the size of structuring element selected in the combobox
 */
private Size getSizeFromComboBox(String size){
        String[] xy = size.split(structSizeDelimitator);

        return new Size(Integer.parseInt(xy[0]), Integer.parseInt(xy[1]));
}



@Override
public void initialize(URL url, ResourceBundle res){
        //set behaviour of label that track slider value
        threshold.valueProperty().addListener(new ChangeListener<Number>() {
                @Override
                public void changed(ObservableValue<? extends Number> arg0, Number oldValue,
                        Number newValue) {
                        thresholdValue.textProperty().setValue(""+newValue.intValue());
                }
        });

        //if "none" morphological operations is selected than size of kernel is not visible
        morphologicalOperations.selectedToggleProperty().addListener(new
                ChangeListener<Toggle>() {

                @Override
                public void changed(ObservableValue<? extends Toggle> observable, Toggle
                oldValue, Toggle newValue) {
                        RadioButton r = (RadioButton) newValue;
                        if(getMorphologicalOperationIndex(r.idProperty().getValue()) == -1){
                                morphKernel.setStyle("visibility:hidden;");
                        }
                        else{
                                morphKernel.setStyle("visibility:visible;");
                        }

                }
        });

        //populate strcut size of morph operations combobox
        morphStructComboBox.setItems(structSizeOptions);
        morphStructComboBox.setValue(structSizeOptionDefault);

        //if "none" morphological operations is selected than size of kernel is not visible
        gaussianBlur.selectedProperty().addListener(new ChangeListener<Boolean>() {

                @Override
```

```
		public void changed(ObservableValue<? extends Boolean> observable, Boolean
		oldValue, Boolean newValue) {
			if(!newValue){
				gaussianParameter.setStyle("visibility:hidden;");
			}
			else{
				gaussianParameter.setStyle("visibility:visible;");
			}
		}
});

//populate struct size of gaussian blur combobox
gaussianStructComboBox.setItems(structSizeOptions);
gaussianStructComboBox.setValue(structSizeOptionDefault);

//if Otsu treshold is selected than slider of global threshold is not visible
thresholdAlgorithm.selectedToggleProperty().addListener(new ChangeListener<Toggle>() {

		@Override
		public void changed(ObservableValue<? extends Toggle> observable, Toggle
		oldValue, Toggle newValue) {
			RadioButton r = (RadioButton) newValue;
			if(!r.idProperty().getValue().equals(GLOBAL_THRESHOLD_ID)){
				globalParameters.setStyle("visibility:hidden;");
			}
			else{
				globalParameters.setStyle("visibility:visible;");
			}

		}
});

//set the behavior of gold standard button
scatterChartContent.getGoldButton().setOnAction(new EventHandler<ActionEvent>() {
		@Override
		public void handle(ActionEvent event) {
			try {
				if(selectGoldStandardImages()){
					submitActionChart();
				}
			} catch (GoldenStandardException e){
				GuiUtils.showError(Thread.currentThread(), e);
			}
		}
});

scatterChartContent.getExportOCVButton().setOnAction(new EventHandler<ActionEvent>() {
		@Override
		public void handle(ActionEvent event) {
			File selectedDirectory =
			FileChooserUtils.getDirectoryChooser(mainApp.getPrimaryStage());
			if(selectedDirectory == null){
				return;
			}

			String path = selectedDirectory.getAbsolutePath()+
				GuiUtils.PATH_SEPARATOR+
				"exportInOpenCVFormat"+getDate()+".csv";
			export(path, "file name;hb;A* mean;A* weighted Mean;\n"+exportOCV);
		}
```

```
		});

		scatterChartContent.getExportCUButton().setOnAction(new EventHandler<ActionEvent>() {
			@Override
			public void handle(ActionEvent event) {
				File selectedDirectory =
				FileChooserUtils.getDirectoryChooser(mainApp.getPrimaryStage());
				if(selectedDirectory == null){
					return;
				}

				String path = selectedDirectory.getAbsolutePath()+
						GuiUtils.PATH_SEPARATOR+
						"exportInColorUtils"+getDate()+".csv";
				export(path, "file name;hb;A* mean;A* weighted Mean;\n"+exportCU);
			}

		});
}

/**
 *
 * @return current data as string
 */
private String getDate(){
		DateFormat dateFormat = new SimpleDateFormat("dd-MM-yyyy");
		Date date = new Date();
		return dateFormat.format(date);
}

/**
 * save the export data file
 * @param path where to save file
 * @param str content
 */
private void export(String path, String str){
		File file = new File(path);
		PrintWriter out = null;
		try {
			out = new PrintWriter(file);
		} catch (FileNotFoundException e) {
			GuiUtils.showError(Thread.currentThread(), e);
		}
		out.print(str);
		out.close();

		FileChooserUtils.showSuccessAlert("Export file has been saved in " + path);
}

/**
 *
 * @return an intance of AstaSegmentation that performs image analysis algorithms specified in the left
 		sidebar
 * @throws FileNotFoundException
 * @throws PreliminaryOperationException
 * @throws InvalidArgumentException
 */
private AstarSegmentation getAstarSegmentation() throws FileNotFoundException,
PreliminaryOperationException, InvalidArgumentException {
		AstarSegmentation astarSegmentation = new AstarSegmentation(matImage);
		astarSegmentation.setAstarComponent(equalize);
```

```
		if(blur){
			astarSegmentation.gaussianBlur(gaussianStructSize);
		}

		if(otsu){
			astarSegmentation.setBinaryImageByOtsu();
		}
		else{
			astarSegmentation.setBinaryImage(thresholdNumber);
		}

		if(morphOp != -1){
			astarSegmentation.morphologyOperation(morphOp, morphStructSize);
		}
		if(regionFill != -1){
			astarSegmentation.regionFilling(regionFill);
		}
		if(ellipse){
			astarSegmentation.ellipseFitting();
		}

		return astarSegmentation;
	}

	/**
	 * set the images in the gui when only one image is selected
	 * @throws PreliminaryOperationException
	 */
	private void setImages() throws PreliminaryOperationException {
		this.imagesContent.setImage(GuiUtils.getImageFromMat(astarSegmentation.getImage()));
		this.imagesContent.setAstarImage(
			GuiUtils.getImageFromMat(astarSegmentation.getAstarComponent()));
		this.imagesContent.setBinaryImage(
			GuiUtils.getImageFromMat(astarSegmentation.getBinaryImage()));
		this.imagesContent.setSegmentedImage(
			GuiUtils.getImageFromMat(
				astarSegmentation.getSegmentedImage(GuiUtils.color,
					GuiUtils.tickness)));
	}

	/**
	 * set the a* values in the gui when only one image is selecteds
	 * @throws PreliminaryOperationException
	 * @throws InvalidArgumentException
	 */
	private void setValues() throws PreliminaryOperationException, InvalidArgumentException {
		this.imagesContent.setAMean(getAstarMeanValue(astarSegmentation));
		this.imagesContent.setAWeightMean(getAstarMeanWeightedValue(astarSegmentation));
	}

	/**
	 *
	 * @param astarSegmentation
	 * @return a* mean value
	 * @throws PreliminaryOperationException
	 */
	private double getAstarMeanValue(AstarSegmentation astarSegmentation) throws
		PreliminaryOperationException{
		return GuiUtils.getAstarValueInCUFormatFromOpenCV(astarSegmentation.getMeanValue(0));
	}
```

```
/**
 *
 * @param astarSegmentation
 * @return a* weighted mean value
 * @throws PreliminaryOperationException
 * @throws InvalidArgumentException
 */
private double getAstarMeanWeightedValue(AstarSegmentation astarSegmentation) throws
        PreliminaryOperationException, InvalidArgumentException{
        return          GuiUtils.getAstarValueInCUFormatFromOpenCV(
                        astarSegmentation.getWeightedMeanValue(boundary, weights));
}


/**
 * the the pearson correlations in the gui when multiple images are selected
 * @param hbV
 * @param asmValues
 * @param aswmValues
 */
private void setCorrelation(double[] hbV, double[] asmValues, double[] aswmValues){
        PearsonsCorrelation p = new PearsonsCorrelation();
        scatterChartContent.setPearsonCorrelations(p.correlation(hbV, asmValues),
                                                        p.correlation(hbV, aswmValues));
}

/**
 * action of button "Apply" in the gui when only one image is selected
 * @throws FileNotFoundException
 * @throws PreliminaryOperationException
 * @throws InvalidArgumentException
 */
private void submit() throws FileNotFoundException, PreliminaryOperationException,
InvalidArgumentException{
        if(matImage == null){
                return;
        }

        setSegmentationValues();
        astarSegmentation = getAstarSegmentation();
        setImages();
        setValues();
}

/**
 * action of button "Apply" in the gui when multiple images are selected
 * @throws FileNotFoundException
 * @throws PreliminaryOperationException
 * @throws InvalidArgumentException
 */
private void submitChart() throws FileNotFoundException, PreliminaryOperationException,
InvalidArgumentException{
        if(images == null || imageNames == null || hbValues == null){
                return;
        }

        setSegmentationValues();

        XYChart.Series<Number, Number> aStarMeanSerie = new XYChart.Series<>();
        aStarMeanSerie.setName("A* mean");
```

```
		double[] hbV = new double[hbValues.size()];
		double[] asmValues = new double[hbValues.size()];
		double[] aswmValues = new double[hbValues.size()];

		HashMap<String, AstarSegmentation> map = new HashMap<>();

		exportOCV = "";
		exportCU = "";
		for(int i = 0; i < images.size(); i++){
			File f = images.get(i);

			String name = f.getName();
			matImage = GuiUtils.getMatFromFile(f);

			AstarSegmentation as = getAstarSegmentation();

			double hb = hbValues.get(imageNames.indexOf(name));
			double asm = getAstarMeanValue(as);
			double aswm = getAstarMeanWeightedValue(as);

			asmValues[i] = asm;
			aswmValues[i] = aswm;
			hbV[i] = hb;

			aStarMeanSerie.getData().add(new XYChart.Data<Number, Number>(hb, asm));

			map.put(name, as);
			exportOCV += name + ";" + hb + ";" + as.getMeanValue(0) + ";" +
				as.getWeightedMeanValue(boundary, weights) + "\n";
			exportCU += name + ";" + hb + ";" + asm + ";" + aswm + "\n";
		}

		setCorrelation(hbV, asmValues, aswmValues);
		scatterChartContent.setSeries(aStarMeanSerie);

		if(goldStandardFileList != null){
			setGoldPercentages(map);
		}
	}

	/**
	 * set gold standard section in the gui
	 * @param map
	 */
	private void setGoldPercentages(HashMap<String, AstarSegmentation> map){
		double sumC = 0;
		double sumE = 0;
		int n = 0;

		Iterator<Entry<String, AstarSegmentation>> it = map.entrySet().iterator();
		while (it.hasNext()) {
			Map.Entry<String, AstarSegmentation> pair = (Map.Entry<String,
			AstarSegmentation>)it.next();
			for(File goldFile : goldStandardFileList){
				if(pair.getKey().equals(goldFile.getName())){
					AstarSegmentation as = (AstarSegmentation) pair.getValue();
					Performance p = new
					Performance(Imgcodecs.imread(goldFile.getAbsolutePath()),
					as.getBinaryImage());
```

```
                                        sumC += p.getPerformance();
                                        sumE += p.getError();

                                        n++;
                                }

                        }
                }

                if(n == 0){
                        n = 1;
                }

                scatterChartContent.setGoldStandard(sumC / n, sumE / n);
}



/**
 * set the image selected and the behavior of "Apply" button when
 * only one image is selected by "Open" menu function
 * @param originalImage
 * @throws FileNotFoundException
 * @throws PreliminaryOperationException
 * @throws InvalidArgumentException
 */
public void setImage(Mat originalImage) throws FileNotFoundException,
                PreliminaryOperationException, InvalidArgumentException {
                if(originalImage == null){
                        return;
                }
                matImage = originalImage;
                submit.setOnAction(new EventHandler<ActionEvent>() {

                        @Override
                        public void handle(ActionEvent event) {
                                submitAction();

                        }
                });
                submitAction();
}

/**
 * set the images selected and the behavior of "Apply" button when
 * multiple images are selected by "Open" menu function
 * @param images
 * @param imageNames
 * @param hbValues
 * @throws InvalidArgumentException
 * @throws FileNotFoundException
 * @throws PreliminaryOperationException
 */
public void setMultipleImages(List<File> images, List<String> imageNames, List<Double> hbValues)
throws InvalidArgumentException, FileNotFoundException, PreliminaryOperationException{
                if(images == null || imageNames == null || hbValues == null){
                        return;
                }
                disableGoldStandard();

                this.images = images;
```

```
		this.imageNames = imageNames;
		this.hbValues = hbValues;

		submit.setOnAction(new EventHandler<ActionEvent>() {

			@Override
			public void handle(ActionEvent event) {
				submitActionChart();
			}
		});
		submitActionChart();
}

/**
 * display the dialog opened when "Conjunctiva detection" menu funcion is selected
 * @param originalImage
 * @param rectROI
 * @return ROI from image
 */
private Mat getMatROI(Mat originalImage, Rect rectROI){
		Mat matROI = null;
		if(originalImage != null && rectROI != null){
			ConjunctivaCroppingDialog dialog = new ConjunctivaCroppingDialog(originalImage,
                                                                                    rectROI);
			matROI = dialog.show();
		}
		return matROI;
}

/**
 * behavior of multiple detections
 * @param images
 * @param srcList
 * @param roiList
 * @throws FileNotFoundException
 * @throws PreliminaryOperationException
 * @throws InvalidArgumentException
 */
public void setMultipleImagesToDetect(List<File> images,List<Mat> srcList, List<Rect> roiList)
			throws FileNotFoundException, PreliminaryOperationException,
				InvalidArgumentException {
		if(images == null || srcList == null || roiList == null){
			return;
		}

		List<Mat> matROIList = new ArrayList<>();
		for(int i = 0; i < images.size(); i++){
			Mat matROI = getMatROI(srcList.get(i), roiList.get(i));
			if(matROI == null){
				return;
			}
			matROIList.add(matROI);
		}

		saveDetections(images, matROIList);
}

/**
 * save detection in a directory choosed from user
 * @param images
 * @param matROIList
```

```
 */
private void saveDetections(List<File> images, List<Mat> matROIList) {
	File selectedDirectory = FileChooserUtils.getDirectoryChooser(mainApp.getPrimaryStage());
	if(selectedDirectory == null){
		return;
	}

	boolean status = true;
	for(int i = 0; i < matROIList.size(); i++){
		if(! Imgcodecs.imwrite(selectedDirectory.getAbsolutePath()+
					GuiUtils.PATH_SEPARATOR+
					images.get(i).getName(), matROIList.get(i))){
			status = false;
		}
	}

	if(!status){
		FileChooserUtils.showErrorAlert();
	}
	else{
		FileChooserUtils.showSuccessAlert("Detections have been saved");
	}
}

/**
 * perfrom "Select Gold Standard Images" button behavior
 * @return
 * @throws GoldenStandardException
 */
private boolean selectGoldStandardImages() throws GoldenStandardException{
	List<File> list = FileChooserUtils.getMultipleFilesFromChooser(mainApp.getPrimaryStage());

	if(list == null){
		return false;
	}

	if(list.size() < 2 || images == null){
		throw new GoldenStandardException("Golden standard images have to be equals to
					images selected");
	}

	if(list.size() != images.size()){
		throw new GoldenStandardException("Golden standard images have to be equals to
					images selected");
	}

	for(File goldFile : list){
		boolean status = false;
		for(File f : images){
			if(f.getName().equals(goldFile.getName())){
				status = true;
			}
		}

		if(status == false){
			throw new GoldenStandardException("Golden standard images have to
					be equals to images selected");
		}
	}

	this.goldStandardFileList = list;
```

```
            return true;

}


/**
 * action of "Apply" button when single image is selected
 */
private void submitAction(){
            GuiUtils.loadingAlert.show();
            Platform.runLater(new Runnable() {

                        @Override
                        public void run() {
                                    try{
                                                submit();
                                    }catch (FileNotFoundException | PreliminaryOperationException |
                                                InvalidArgumentException e) {
                                                GuiUtils.loadingAlert.close();
                                                GuiUtils.showError(Thread.currentThread(), e);
                                    }
                                    setContent(imagesContent.getContent());
                                    enableButtons();
                                    GuiUtils.loadingAlert.close();
                        }
            });
}

/**
 * action of "Apply" button when multiple images are selected
 */
private void submitActionChart(){
            GuiUtils.loadingAlert.show();
            Platform.runLater(new Runnable(){

                        @Override
                        public void run() {
                                    try {
                                                submitChart();
                                    } catch (FileNotFoundException | PreliminaryOperationException |
                                                InvalidArgumentException e) {
                                                GuiUtils.loadingAlert.close();
                                                GuiUtils.showError(Thread.currentThread(), e);
                                    }
                                    setContent(scatterChartContent.getScatterChartBox());
                                    enableButtons();
                                    GuiUtils.loadingAlert.close();
                        }
            });
}

/**
 * enable apply button
 */
private void enableButtons(){
            submit.setDisable(false);
}

/**
 * delete golden standard images selected
 */
```

```
	private void disableGoldStandard(){
		goldStandardFileList = null;
		if(scatterChartContent != null){
			scatterChartContent.setEmptyGoldStandard();
		}
	}

	/**
	 * set the main content of gui
	 * @param node
	 */
	private void setContent(Node node) {
		content.getChildren().clear();
		content.getChildren().add(node);
	}


	public void setMain(Main main) {
		this.mainApp = main;

	}

}
```

Classe Main.java

```
/**
 * MAIN of JavaFX application
 *
 * @author Luigi
 *
 */
public class Main extends Application {
	private Stage primaryStage;
	private BorderPane rootLayout;
	private AstarViewController astarViewController;
	private MenuViewController menuViewController;

	@Override
	public void start(Stage primaryStage) throws IOException{
		System.loadLibrary(Core.NATIVE_LIBRARY_NAME);


		Thread.setDefaultUncaughtExceptionHandler(GuiUtils::showError);   //exception are
                                                                                  handled by GuiUtils.showError

		this.primaryStage = primaryStage;
		this.primaryStage.setTitle(GuiUtils.GUI_TITLE);

		initRootLayout();
	}

	/**
	 * init the gui
	 * @throws IOException
	 */
	private void initRootLayout() throws IOException {
		//loading
```

```
            FXMLLoader loader = new FXMLLoader();
            loader.setLocation(Main.class.getResource("view/MenuView.fxml"));
            rootLayout = (BorderPane) loader.load();
            menuViewController = loader.getController();
            menuViewController.setMain(this);

            FXMLLoader astarViewLoader = new FXMLLoader();
            astarViewLoader.setLocation(Main.class.getResource("view/AstarView.fxml"));
            rootLayout.setCenter((ScrollPane) astarViewLoader.load());
            astarViewController = astarViewLoader.getController();
            astarViewController.setMain(this);

            menuViewController.setAstarView(astarViewController);

            // Show the scene containing the root layout.
            Scene scene = new Scene(rootLayout);
            scene.getStylesheets().add(getClass().getResource("bootstrap2.css").toExternalForm());
            scene.getStylesheets().add(getClass().getResource("bootstrap3.css").toExternalForm());
            primaryStage.setScene(scene);
            primaryStage.show();
    }

    /**
     *
     * @return the primatyStage
     */
    public Stage getPrimaryStage() {
            return primaryStage;
    }

    /**
     * launch main
     * if errors occur then print the stack trace
     * @param args
     */
    public static void main(String[] args) {
            try{
                    launch(args);
            } catch (Exception e) {
                    e.printStackTrace();
            }
    }
}
```