Melanocytic image segmentation using fuzzy c-clustering in C++

RELATORE: Ch.mo Prof. Enoch Peserico Stecchini Negri De Salvi
CORRELATORE: Dott. Alberto Silletti
LAUREANDO: Denis Altomare

Corso di laurea in Ingegneria dell'Informazione

A.A. 2011-2012

UNIVERSITÀ DEGLI STUDI DI PADOVA
DIPARTIMENTO DI INGEGNERIA DELL'INFORMAZIONE
CORSO DI LAUREA IN INGEGNERIA DELL'INFORMAZIONE

TESI DI LAUREA

Melanocytic image segmentation using fuzzy c-clustering in C++

RELATORE: Prof. Enoch Peserico Stecchini Negri De Salvi

CORRELATORE: Dott. Alberto Silletti

LAUREANDO: Denis Altomare

A.A. 2011-2012

Indice

Sommario **1**

Introduzione **3**

1 Il melanoma **5**

1.1 Eziologia 6

1.2 Epidemiologia 7

1.3 Regola ABCDE 8

1.4 Prevenzione 9

2 Strumenti utilizzati **11**

2.1 Il linguaggio C++ 11

2.2 La CImg Library 11

2.3 Il framework JNI 12

3 Algoritmo **13**

3.1 Calcolo delle componenti principali: PCA 13

3.2 Istogramma 19

3.3 Segmentazione 21

3.4 Identificazione del nevo e calcolo dell'area 30

3.5 Prestazioni 40

Conclusioni **43**

Bibliografia **45**

Elenco delle figure **46**

Sommario

Questo elaborato illustrerà un algoritmo per la segmentazione di immagini di nevi al fine di individuare delle anomalie degli stessi ed aiutare il dermatologo ad identificare possibili melanomi.
Si parlerà del melanoma in generale, descrivendone le caratteristiche, i sintomi e i metodi per prevenirlo.
Successivamente verranno analizzati tutti i passaggi che compie l'algoritmo per segmentare il nevo, partendo dall'immagine a colori fino al raggiungimento di quella in bianco e nero.

Introduzione

Nel corso degli ultimi anni sono notevolmente aumentati i casi di persone affette dal tumore cutaneo chiamato melanoma. Esso detiene il triste primato di essere la neoplasia a prognosi peggiore se non diagnosticata precocemente.
Per poter prontamente individuare i melanomi è bene tenere sotto controllo l'evoluzione di ogni singolo nevo, questo comporta un lavoro ripetitivo a carico del dermatologo che deve controllare e registrare periodicamente la dimensione, il colore e la forma di ognuno di essi.
Tale processo richiede una maggior attenzione qualora venga effettuato su individui che presentano un elevato numero di nevi. Al fine di aiutare il medico ad esaminarli si potrebbe automatizzare la procedura, per esempio, attraverso la creazione di un software in grado di mettere a confronto le diverse foto di uno stesso nevo scattate in periodi differenti al fine di cogliere gli eventuali cambiamenti.
Per applicare tale procedura è fondamentale la segmentazione delle immagini di nevi. La segmentazione consiste nella partizione di un'immagine in regioni significative al fine di identificare, in maniera corretta, il nevo della pelle.
In questa tesi descriverò un algoritmo che prende come input un'immagine a colori di un nevo e ne fa la segmentazione restituendo come output un'immagine in bianco e nero nella quale viene evidenziato il nevo (in nero) dalla pelle (in bianco) e due valori numerici che indicano il perimetro e l'area del nevo.

La trattazione si articolerà come segue:

Il **primo capitolo** ha carattere introduttivo e parlerà del melanoma dal punto di vista medico descrivendone le caratteristiche, i sintomi ed i metodi e gli accorgimenti per prevenirlo.

Nel **secondo capitolo** ci sarà una breve descrizione degli strumenti usati.

Il **terzo capitolo** descriverà in tutte le sue parti l'algoritmo utilizzato per ottenere la segmentazione dell'immagine. Verrà poi data una breve analisi delle prestazioni e

dei colli di bottiglia dell'algoritmo.

Capitolo 1

Il melanoma

La pelle è l'organo più esteso del nostro corpo ed è formata da tre strati: uno strato più superficiale, l'epidermide, uno strato intermedio, il derma e uno strato più profondo, il tessuto sottocutaneo o grasso.
L'epidermide è formato principalmente da cellule basali che fungono da barriera (contro batteri, funghi, parassiti, calore, radiazioni UV...), e da cellule squamose che permettono lo scambio di ossigeno e nutrienti e melanociti. I melanociti si trovano nella parte più bassa dell'epidermide, sono delle cellule che producono la melanina, un pigmento che da alla pelle il suo colore naturale e che permette, nel momento in cui ci si espone al sole, di abbronzarsi.
Questo pigmento naturale (presente anche nei capelli e in alcune parti dell'occhio) protegge dagli effetti dannosi dei raggi solari. In condizioni normali i melanociti possono dar luogo ad agglomerati scuri visibili sulla superficie della pelle e noti come nei (nevi è il termine medico).
La crescita incontrollata delle cellule basali, delle cellule squamose e dei melanociti dà origine ai tumori della pelle, nello specifico questi ultimi sono all'origine nel melanoma. L'arma più efficace per sconfiggere il melanoma è la diagnosi precoce, infatti se non si è ancora sviluppato in profondità il tumore può essere asportato senza grosse difficoltà e le probabilità di sopravvivenza sono elevate. Se invece il tumore è più profondo e ha raggiunto altri organi generalmente non è curabile e si cerca di mantenerlo limitato (attraverso chemioterapia, immunoterapia, chirurgia ecc...) [1].

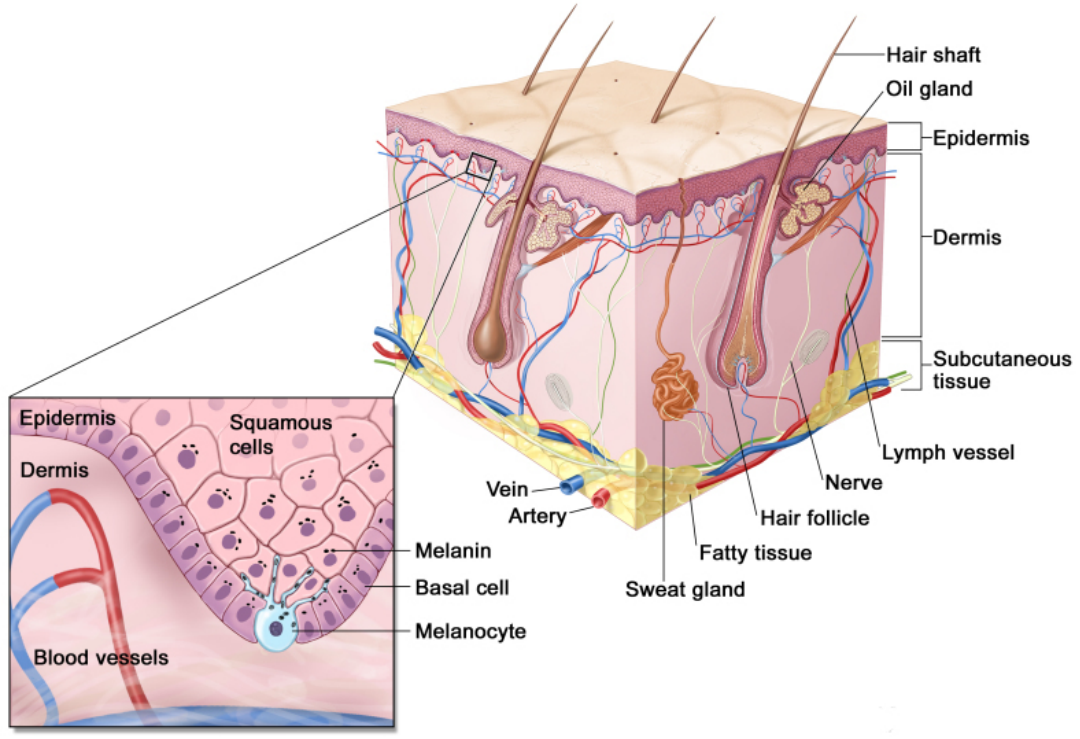


Figura 1.1: La pelle è formata dall'epidermide, dal derma e dal tessuto sottocutaneo. I melanociti sono nel livello delle cellule basali situate nella parte più profonda dell'epidermide.

1.1 Eziologia

Questa neoplasia può insorgere *de novo* o su un nevo preesistente, tuttavia si ritiene che la maggior parte dei melanomi si sviluppi *de novo* su aree di cute normale. Le lesioni che hanno una elevata probabilità (che si aggira attorno al 10%) di sviluppare melanoma sono i nevi congeniti giganti[1].
Si stima che l'80% dei melanomi sia causata dai danni provocati dai raggi ultravioletti (luce solare e lampade abbronzanti), infatti è stato dimostrato che le esposizioni episodiche ed intense portano ad un fattore di rischio maggiore. Non si parla solo della luce solare ma anche dalle radiazioni provenienti dalle lampade abbronzanti.

[1]I nevi congeniti sono nevi presenti alla nascita o che insorgono nelle prime settimane (o mesi) di vita. Con il tempo assumono una superficie quadrettata o verrucosa, spesso rilevata sul piano cutaneo, talvolta con componente pilifera. Vengono classificati come giganti quando hanno diametro superiore ai 20cm.

Il rischio aumenta notevolmente per le persone maggiormente predisposte alle scottature (cioè per le persone di fototipo I[2]).
Altro elemento da prendere in considerazione è l'età in cui sono avvenuti gli eccessi di esposizione ai raggi ultravioletti: è stato dimostrato che le probabilità di contrarre questa neoplasia in età adulta sono maggiori se durante l'infanzia l'esposizione è stata particolarmente significativa.
Altri fattori di rischio sono la presenza di un elevato numero di nevi di piccole dimensioni o alcuni di grandi dimensioni, la presenza di nevi displastici[3], l'uso di contraccettivi orali [3] e la presenza di melanomi maligni nell'anamnesi personale e familiare.

1.2 Epidemiologia

L'esposizione al sole ha sicuramente una incidenza notevole sulla comparsa dei melanomi, infatti come visto precedentemente gli individui più colpiti sono quelli caratterizzati da fototipo I.
Si è potuto, inoltre, evincere come i casi siano pressochè raddoppiati negli ultimi 20 anni e ciò risulta essere legato a diversi fattori tra i quali hanno grande rilievo la diminuzione della capacità schermante dell'atmosfera, causata dal progressivo ingrandimento del buco dell'ozono, ed il cambiamento delle abitudini di vita, nella fattispecie l'aumento delle esposizioni al sole acute ed intermittenti. Tale aumento della casistica sembra essere destinato a continuare poiché nei paesi occidentali è sempre più diffusa la moda dell'abbronzatura [4].
Le zone del corpo più colpite sono le gambe e le braccia nelle donne e la zona del tronco, il viso ed il collo negli uomini. Le zone che più raramente sono colpite dai melanomi sono: la bocca, l'iride o la retina. [1]. Questa neoplasia interessa in modo più consistente le donne anche se la prognosi risulta essere migliore, rispetto a quella degli uomini. [5].
Gli studi degli ultimi anni hanno evidenziato che il melanoma si presenta con maggior frequenza tra i 30 ed i 60 anni mentre risulta essere più raro in età infantile ed adole-

[2]In dermatologia, il fototipo di una persona è determinato dalla qualità e dalla quantità di melanina presente in condizioni basali nella sua pelle. Vengono distinti sei tipi di fototipo, a seconda delle caratteristiche dell'individuo nella fattispecie i soggetti di fototipo I sono caratterizzati da carnagione molto chiara, spessissimo con efelidi, capelli di colore biondo o rosso, occhi chiari [2].

[3]Il nevo displastico appare di grandezza variabile (generalmente maggiore di 6 mm), con bordo irregolare e colore variabile dal bruno scuro al rossiccio, principalmente si localizzano sul tronco, addome e braccia.

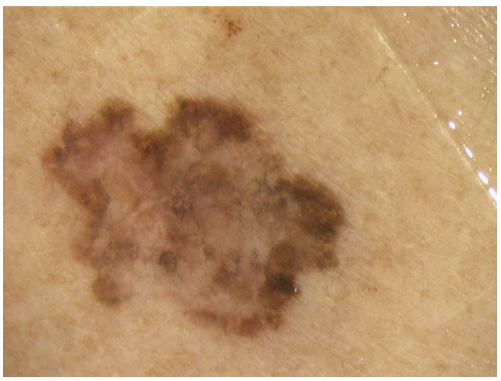


Figura 1.2: Esempi di melanoma.

scenziale.

1.3 Regola ABCDE

Il sistema ABCDE è un'insieme di sintomi tipici del melanoma, se anche uno solo di questi sintomi è presente bisogna consultare tempestivamente un medico.
Le caratteristiche di questo sistema sono:

- **A**simmetria: i melanomi solitamente si presentano come asimmetrici, con metà della macchia cutanea più grande dell'altra.
- **B**ordi: i bordi del melanoma sono irregolari a carta geografica, al contrario di quelli dei nevi.
- **C**olore: spesso il melanoma è policromo ovvero presenta colori diversi come nero, bruno, rosso e rosa.
- **D**imensione: una lesione cutanea sospetta, di diametro superiore ai 6 millimetri deve essere verificata da uno specialista.
- **E**voluzione: la lesione cutanea che tende a modificare la propria forma, colore e superficie è da ritenersi sospetta e da verificare.

1.4 Prevenzione

Per prevenire l'insorgere di questo tumore è consigliato rispettare le consuete norme di igiene solare quindi è bene evitare di esporsi al sole nelle ore centrali della giornata nel periodo estivo e bisogna, inoltre, utilizzare creme solari con livello di protezione alto anche per esposizioni di breve durata (queste regole vanno rispettate pedissequamente in età pediatrica), accertarsi che le creme solari utilizzate schermino sia dai raggi UVA che dai raggi UVB.
La diagnosi precoce si è rivelata l'arma più importante per combattere questa malattia altrimenti mortale. Infatti il melanoma cutaneo ha una prognosi strettamente dipendente dallo spessore raggiunto nella pelle al momento della sua diagnosi e asportazione (se il melanoma è rimasto ancora confinato agli strati cutanei superficiali, la prognosi è generalmente buona, altrimenti le probabilità di sopravvivenza sono molto più basse). Pertanto si dovrebbe effettuare un autoesame con cadenza regolare (preferibilmente mensile) seguendo il sistema ABCDE. In caso di nevi sospetti, è necessario rivolgersi tempestivamente al proprio medico di base, che saprà valutare la necessità di una visita specialistica.

Capitolo 2

Strumenti utilizzati

Prima di descrivere nel dettaglio l'implementazione dell'algoritmo viene fornita una breve carrellata degli strumenti utilizzati.

2.1 Il linguaggio C++

Come si vedrà nel corso di questa tesi alcune fasi dell'algoritmo sono computazionalmente pesanti quindi è stato scelto di implementare tali fasi attraverso il linguaggio di programmazione C++[1] per rendere più veloce possibile l'esecuzione del programma.
Questa scelta è stata presa poichè le implementazioni precedenti (Matlab[2] prima e Java[3] dopo) si sono rivelate significativamente inefficienti.

2.2 La CImg Library

Per la gestione delle immagini è stato scelto di utilizzare la CImg Library[4] poichè è semplice e leggera. Questa libreria infatti non ha bisogno di ulteriori librerie esterne e viene fornita interamente in un unico file di intestazione.
Le funzionalità offerte da CImg Library soddisfano pienamente le necessità dell'implementazione dell'algoritmo trattato in questa tesi.

[1] `http://www.cplusplus.com/`
[2] `http://www.mathworks.it/products/matlab/`
[3] `http://www.oracle.com/us/technologies/java/overview/index.html`
[4] `http://cimg.sourceforge.net/`

2.3 Il framework JNI

Il framework JNI[5] (Java Native Interface) permette l'utilizzo di frammenti di codice nativo (che risiede in delle librerie condivise) in programmi scritti in java.
L'ambiente di lavoro scelto per scrivere e compilare la libreria condivisa, in questo caso la DLL (Dynamic Link Library) è l'IDE (Integrated Development Environment) Microsoft Visual C++[6].
La scelta di utilizzare JNI è stata fatta per sfruttare la velocità del C++ anche se a scapito della portabilità (per poter far girare il software su altre piattaforme è necessario ricompilare le librerie condivise).
Inoltre è in programma di utilizzare NDK[7] (Native Development Kit), tool analogo a JNI per poter utilizzare lo stesso codice anche su sistemi android.

[5] `http://java.sun.com/docs/books/jni/html/intro.html`
[6] `http://www.microsoft.com/visualstudio/en-us/products/2008-editions`
[7] `http://developer.android.com/tools/sdk/ndk/index.html`

Capitolo 3

Algoritmo

Lo scopo dell'algoritmo è di ottenere a partire da un'immagine RGB (Red Green Blue) di un nevo un'altra immagine in bianco e nero delle stesse dimensioni della prima che evidenzia il nevo stesso (nero sul nevo, bianco sul resto della pelle).
L'algoritmo deve, inoltre, fornire la lunghezza del bordo e l'area del nevo. La suddivisione di immagini in regioni significative prende il nome di segmentazione.
I passi principali dell'algoritmo sono:

- Calcolo delle componenti principali dell'immagine;
- Generazione dell'istogramma;
- Segmentazione dell'istogramma;
- Individuazione del nevo;

Ad ognuno di questi punti verrà dedicato un paragrafo che spiegherà nel dettaglio il loro significato e i metodi utilizzati per raggiungere l'obiettivo.

3.1 Calcolo delle componenti principali: PCA

La PCA (Principal Component Analysis), nota anche come trasformata di Karhunen-Loeve [7], è definita come una trasformazione ortogonale che trasforma i dati in un nuovo sistema di coordinate tale per cui la variabile con più grande varianza si trova sulla prima coordinata (chiamata prima componente principale), la variabile con la seconda più grande varianza si trova sula seconda coordinata (chiamata seconda componente principale), e così via [8].

Se i dati sono abbastanza correlati si noterà che le ultime componenti hanno varianza molto bassa quindi portano informazione trascurabile e possono essere scartate.
Nel nostro caso la trasformazione parte dallo spazio RGB (quindi tridimensionale) e arriva nello spazio delle componenti principali (P_1, P_2 e P_3) dove si nota chiaramente che la terza componente porta una quantità di informazione trascurabile dunque verrà scartata (si veda Figura 3.1).
Per effettuare la trasformazione bisogna trovare il valore medio di ciascuna delle componenti dei dati (in questo caso le componenti sono 3: red, green e blue) e per ogni dato (in questo caso per ogni pixel) calcolare la deviazione delle sue componenti rispetto alla media. Questo significa semplicemente sottrarre alle componenti dei pixel la relativa componente della media.
Per ottenere le componenti PCA basta moltiplicare la matrice delle deviazioni per la matrice di cambiamento di base.
La matrice di cambiamento di base si può costruire in pochi passi:

- calcolare la matrice di covarianza dei dati;
- calcolare gli autovalori ed autovettori;
- la matrice di cambaimento di base è la matrice formata daglii autovettori ordinati secondo i relativi autovalori decrescenti;

Di seguito viene riportato una parte del codice per il calcolo delle componenti principali:

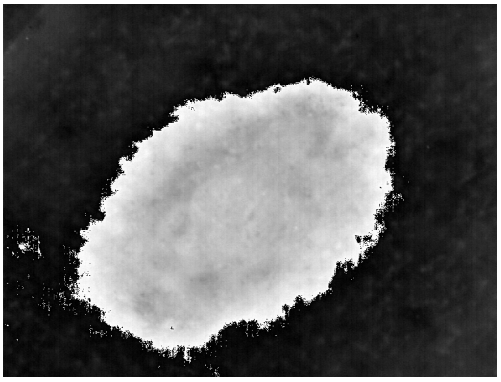
```
/**
 *  PCA
 *
 *  This procedure takes in input the RGB image and calculates its
 *  principal components.
 *  The principal components are stored in the PCA object and it can be
 *  accessed with the get methods.
 *    input:
 *      CImg<float>* img: RGB image
*/
PCA::PCA(CImg<float>* img) {
  // takes the dimensions of the image
  rows = img->height();
```

```
cols = img->width();

// reading the pixels from the matrix in row-major,
// treating them as column vectors in a three dimensional space,
// a new 3x(rows*columns) matrix is then created

float** rgb = rowize(img);

// calculating the average of the colors of the image
float average[3];
average[0] = average[1] = average[2] = 0;
for (int i = 0; i < rows * cols; i++) {
  average[0] += rgb[0][i];
  average[1] += rgb[1][i];
  average[2] += rgb[2][i];
}
average[0] /= rows * cols;
average[1] /= rows * cols;
average[2] /= rows * cols;

// shifts all the elements to have zero mean
for (int i = 0; i < rows * cols; i++) {
  rgb[0][i] -= average[0];
  rgb[1][i] -= average[1];
  rgb[2][i] -= average[2];
}

// calculation of the covariant matrix of the data
// covariance = (rgb*rgb')
float** cov;
cov = square(rgb, 3, rows * cols);

// calculation of the trasformation matrix for change the basis
// from rgb to pca
```

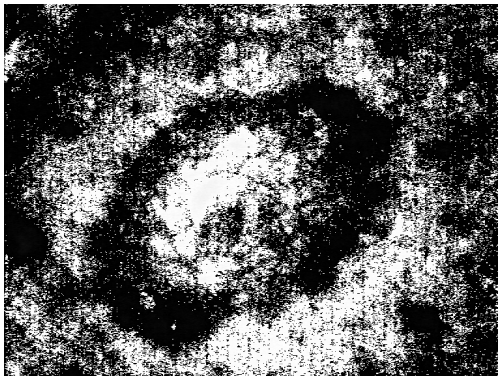
```
float** trasf;
float** eigVal;

// obtaining eigenvector and eigenvalues of the covariance matrix
// assigns to the iegVal variable the eigenvalues in decreasing order,
// then assigns to the trasf variable the eigenVectors sorted be
// decreasing eigenvalue.
eigen_decomposition(cov, trasf, eigVal);

trasf = transpose(trasf, 3, 3);

// changement of the basis from RGB to PCA
float **pcaVector = matrixProduct(trasf, rgb, 3, 3, 3, rows * cols);

// now pcaVector[i] contains the i-th pca components in a row
// derowize function transforms an 1x(rows*cols) matrix
// in a (rows)x(cols) matrix putting parts of the
// first matrix of "cols" length in each row of the second matrix.
p1Matrix = derowize(pcaVector[0], rows, cols);
p2Matrix = derowize(pcaVector[1], rows, cols);
p3Matrix = derowize(pcaVector[2], rows, cols);

// calculation of the principal components of the healthy skin.
// assumption: the mole is located in the center of the original
// image so the four pixels in the verticles of the image rappresents
// healthy skin.
// for the calculation of the rgb components it will take the average of
// the component of those 4 pixels
float healthyRgb[3];           //rgb components of the healty skin
// each CImg object rappresents an image in 3 spatial dimensions
// (width, height, depth) and 1 hyperspectral dimension (dim).
healthyRgb[0] = (img->atXYZC(0, 0, 0, 0) +
                 img->atXYZC(img->_width, 0, 0, 0) +
                 img->atXYZC(0, img->_height, 0, 0) +
```

```
                    img->atXYZC(img->_width, img->_height, 0, 0)
                   ) / 4;
  healthyRgb[0] -=  average[0];
  healthyRgb[1] = (img->atXYZC(0, 0, 0, 1) +
                    img->atXYZC(img->_width, 0, 0, 1) +
                    img->atXYZC(0, img->_height, 0, 1) +
                    img->atXYZC(img->_width, img->_height, 0, 1)
                   ) / 4;
  healthyRgb[1] -=  average[1];
  healthyRgb[2] = (img->atXYZC(0, 0, 0, 2) +
                    img->atXYZC(img->_width, 0, 0, 2) +
                    img->atXYZC(0, img->_height, 0, 2) +
                    img->atXYZC(img->_width, img->_height, 0, 2)
                   ) / 4;
  healthyRgb[2] -= average[2];

  // changing the basis for the healthy skin from rgb to pca
  healthyPca = new float[3];
  healthyPca[0] = trasf[0][0] * healthyRgb[0] +
                  trasf[0][1] * healthyRgb[1] +
                  trasf[0][2] * healthyRgb[2];
  healthyPca[1] = trasf[1][0] * healthyRgb[0] +
                  trasf[1][1] * healthyRgb[1] +
                  trasf[1][2] * healthyRgb[2];
  healthyPca[2] = trasf[2][0] * healthyRgb[0] +
                  trasf[2][1] * healthyRgb[1] +
                  trasf[2][2] * healthyRgb[2];
}
```

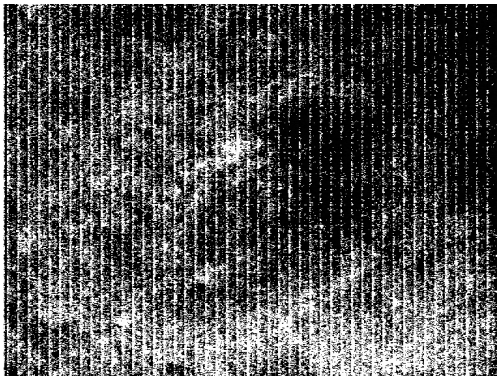
(a) Immagine originale (RGB).



(b) Componente principale 1.



(c) Componente principale 2.



(d) Componente principale 3.

Figura 3.1: Esempio di analisi PCA.

3.2 Istogramma

Il secondo passo dell'algoritmo è la creazione dell'istogramma delle prime due componenti principali (la terza componente viene scartata perchè ha varianza molto bassa quindi porta informazione trascurabile). Poichè l'istogramma è a due dimensioni si può rappresentare con una matrice nella quale l'indice di riga rappresenta il valore della componente P_1 e l'indice di colonna rappresenta il valore della componente P_2.
Le celle della matrice contengono il numero di pixel con quelle caratteristiche (pixel le cui prime due componenti principali assumono esattamente i valori delle coordinate), quindi l'elemento in posizione (i, j) indicherà il numero di pixel che hanno componente $P_1 = i$ e componente $P_2 = j$. Poichè le componenti principali sono immagini in gradazioni di grigio a 256 livelli (8 bit), la dimensione della matrice istogramma sarà 256 x 256.
Il calcolo dell'istogramma è molto semplice, di seguito viene riportato il codice:

```
/** Function makeHist
 *
 *  returns the 2-dimensional histogram of the first and
 *  the second principal components
 *
 *  input:
 *    p1: an CImg object that contains the first principal component
 *        p1 values goes from 0 to 255
 *    p2: an CImg object that contains the second principal component
 *        p2 values goes from 0 to 255
 *  output:
 *    returns an CImg object 256x256 that contains the 2-dimensional
 *    histogram of the (first and second) principal components
 */
CImg<float>* Utilities::makeHist(CImg<float> *p1, CImg<float> *p2) {
  CImg<float>* histImg = new CImg<float> (256, 256, 1, 1);
  // initializes the histogram matrix
  float **hist = new float*[256];
  for (int i = 0; i < 256; i++) {
    hist[i] = new float[256];
    for (int j = 0; j < 256; j++) {
```

```
      hist[i][j] = 0;
    }
  }

  int rows = p1->height();
  int columns = p1->width();
  float p1pixel, p2pixel;

  for (int i = 0; i < rows; i++)
  {
    for (int j = 0; j < columns; j++)
    {
      p1pixel = p1->atXYZC(j, i, 0, 0);
      p2pixel = p2->atXYZC(j, i, 0, 0);
      hist[(int) p1pixel][(int) p2pixel]++;
    }
  }
  histImg = convertToCImg(hist, 256, 256);
  return histImg;
}
```

Gli istogrammi ottenuti partendo da immagini di nevi hanno sempre la stessa forma allungata a "fagiolo" (simile a quella in Figura 3.2) che si estende verticalmente ed è caratterizzata da due regioni più intense.
Queste regioni stanno ad indicare una forte concentrazione di pixel con caratteristiche simili, in generale, la regione situata più in alto ha dei valori di P_1 bassi, la regione situata più in basso ha dei valori di P_1 alti. Mentre per quanto riguarda la componente P_2 l'intero "fagiolo" rientra nello stesso range.
Il significato di questa distribuzione di valori si può interpretare analizzando le componenti principali (Figura 3.2), in particolare la componente P_1: i pixel con elevata intensità di P_1 (quindi quelli chiari) sono i pixel di nevo, quelli con bassa intensità (quindi quelli scuri) rappresentano invece la pelle sana.

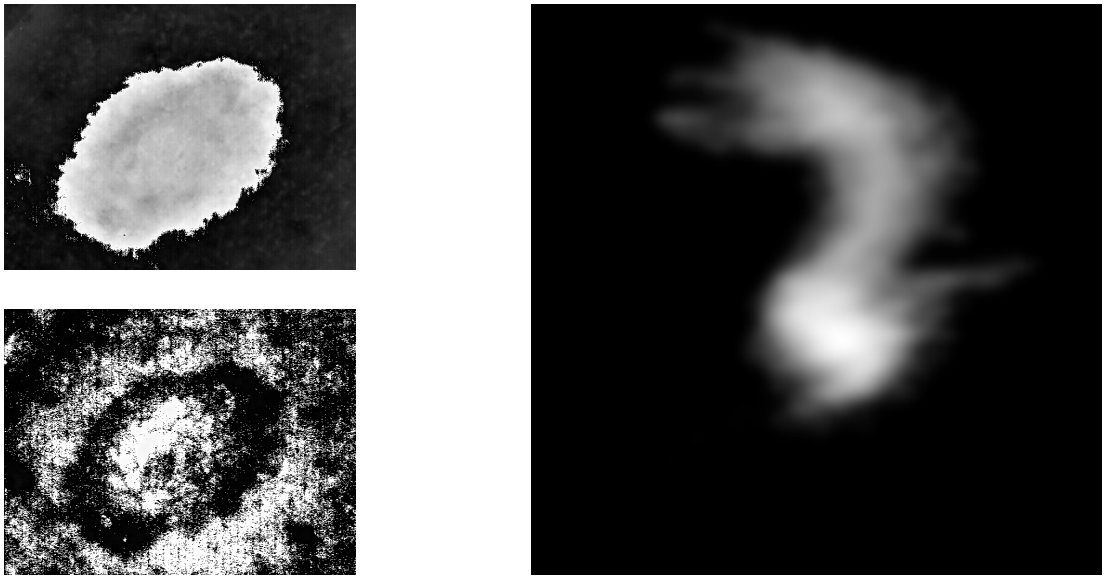


Figura 3.2: Sulla sinistra sono state riportate componenti principali P_1 e P_2 di Figura 3.1, sulla destra l'istogramma ottenuto da esse.

3.3 Segmentazione

Il principale obiettivo della segmentazione è partizionare un immagine in 'C' regioni che devono essere omogenee secondo uno o più criteri.
In alcune situazioni può accadere che non si riesca ad assegnare qualche pixel ad una regione piuttosto che ad un altra perchè il criterio scelto per verificare l'omogeneità non ha una netta transizione in corrispondenza dei bordi. Per gestire queste situazioni viene inserito il concetto di fuzzy[1] [9] nel processo di segmentazione.
Nel nostro caso l'immagine (l'istogramma) deve essere partizionata in due regioni, una che rappresenta il nevo e una che rappresenta la pelle.

Segmentazione fuzzy c-means

Nella segmentazione fuzzy ad ogni pixel è assegnato un valore di membership relativo ad ognuna delle 'C' regioni. La funzione membership assume valori compresi tra 0 e 1 riflettendo il grado di appartenenza del pixel rispetto alla sua classe (chiamata anche

[1]La logica fuzzy è una logica in cui si può attribuire a ciascuna proposizione un grado di verità compreso tra 0 e 1. Con grado di verità o valore di appartenenza si intende quanto è vera una proprietà: questa può essere, oltre che vera (corrispondente al valore 1) o falsa (corrispondente al valore 0) come nella logica classica, anche pari a valori intermedi.

cluster).
Ogni cluster è rappresentato da un centroide e il valore di membership viene calcolato mediante una funzione che misura il livello di similarità del pixel rispetto al centroide del cluster, sotto il vincolo che la somma dei valori di membership di un pixel rispetto a tutti i centroidi deve essere unitaria.
Secondo il metodo fuzzy c-means [10] la partizione fuzzy ottima è quella che minimizza questa funzione obiettivo:

$$J_m = \sum_{i=1}^{N}\sum_{j=1}^{C} u_{ij}^m |x_i - c_j|^2 \qquad , \qquad 1 \leq m < \infty \tag{3.1}$$

Dove:

- m è un numero reale non inferiore a 1. Rappresenta la fuzziness, generalmente è fissato a 2;
- N è il numero di pixel dell'immagine da segmentare;
- x_i è l'i-esimo pixel dell'immagine da segmentare;
- $|\cdot|$ è la distanza euclidea;
- C è il numero dei cluster;
- c_j è il cluster j-esimo, $(1 \leq j \leq C)$;
- u_{ij} è il grado di membership di x_i rispetto al cluster c_j;

Il partizionamento può essere effettuato calcolando iterativamente i centroidi dei cluster e la matrice membership applicando le seguenti formule:

$$c_j = \frac{\sum_{i=1}^{N} u_{ij}^m \cdot x_i}{\sum_{i=1}^{N} u_{ij}^m} \qquad u_{ij} = \frac{1}{\sum_{k=1}^{C} \left(\frac{|x_i - c_j|}{|x_i - c_k|}\right)^{\frac{2}{m-1}}} \tag{3.2}$$

L'iterazione termina quando $max_{ij}\{|u_{ij}^{(k+1)} - u_{ij}^{(k)}|\} < \epsilon$ dove k è il passo di iterazione. Questa procedura converge ad un minimo locale della funzione obiettivo J pertanto la partizione trovata risulta corretta solo se le posizioni iniziali dei centroidi sono abbastanza vicine alle posizioni finali.

Variante del fuzzy c-means

In questo algoritmo viene utilizzata una variante al metodo fuzzy c-means che per segmentazioni di nevi risulta molto precisa [11].
La funzione obiettivo da minimizzare è la funzione inerzia, l'algoritmo iterativamente calcola membership, inerzia e sposta i centroidi verso zone di inerzia minore. Queste iterazioni terminano quando viene raggiunto uno stato stazionario ossia quando l'inerzia ha raggiunto un punto di minimo locale.
La funzione membership è una potenza dell' inverso della distanza quindi pixel distanti dal centroide avranno membership minore mentre la funzione inerzia è proprio l'inerzia fisica:

$$U_{k,x} = \frac{1}{d(x, c_k)^{\alpha}} \tag{3.3}$$

$$\forall k : \underset{c_k}{\operatorname{argmin}} \qquad I_k = \int_A (U_{k,x})^{\beta} d(x, c_k)^{\gamma} h(x)^{\lambda} \, \mathrm{d}x. \tag{3.4}$$

Dove:

- A è l'immagine;
- h è l'istogramma bidimensionale;
- x è un punto nello spazio dei colori bidimensionale;
- c_k è il centroide del cluster k nello spazio dei colori, $k \in \{1, 2, ..., C\}$;
- $U_{x,k}$ é la fuzzy membership di x rispetto al cluster k;
- $\alpha, \beta, \gamma, \lambda$ sono dei valori scalari;

Di seguito viene riportato il pezzo di codice che esegue le iterazioni che portano al raggiungimento dello stato stazionario:

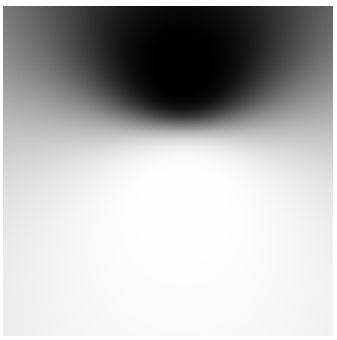
```
//    CLUSTERING MAIN LOOP
// when stability will be true the clustering main loop terminates
bool stability = false;
// inertia of the point focused and of the 4 adjacent points
float M_neighbor[2][5];
// offsets: center, up, left, down, right
const short NEIGHBOR[2][5] = {
  {0, -1, 0, 1, 0},
  {0, 0, -1, 0, 1}
};

for (int h = 0; h < 500 && stability == false; h++) {
  // h counts the numbers of loops.
  // If it grows too much the funcion terminates.

  // sets to zero the neighbors inertia
  for (int k = 0; k < NUM_CLUSTER; k++)
    for (int n = 0; n < 5; n++)
      M_neighbor[k][n] = 0;

  // moves the centers toward the lower inertia area
  // - calculates the inertia for all the neighbors
  //   (and the current center).
  // - finds the neighbor with the lower inertia.
  //   this point will be the new center.

  // inertia of the neighbors
  for (int k = 0; k < NUM_CLUSTER; k++) {
    for (int n = 0; n < 5; n++) {
      center[0] = clus[0][k] + NEIGHBOR[0][n];
      center[1] = clus[1][k] + NEIGHBOR[1][n];
      for (int y = 0; y < 256; y++) {
```

```
      for (int x = 0; x < 256; x++) {
        // calculates the square of the distance
        // because are used only powers of the distance.
        // the exponets of those powers will be halved
        distance = (pow((float)(y - center[0]), 2) +
                    pow((float)(x - center[1]), 2));
        M_neighbor[k][n] += pow(hist[y][x], ro) *
                            pow(distance, beta/2) * U[y][x][k];
      }
    }
  }
}

// finds the neighbor with lower inertia
int minInertiaPoint;
// boolean variable that checks if the centers are moving
bool isMoving[2] = {true, true};
for (int k = 0; k < NUM_CLUSTER; k++) {
  minInertiaPoint = 0;
  for (int n = 1; n < 5; n++) {
    if (M_neighbor[k][n] < M_neighbor[k][minInertiaPoint]) {
      minInertiaPoint = n;
    }
  }
  clus[0][k] += NEIGHBOR[0][minInertiaPoint];
  clus[1][k] += NEIGHBOR[1][minInertiaPoint];
  if(minInertiaPoint == 0)
    isMoving[k] = false;
}

//if the two centers aren't moving is reached a stationary state.
if(isMoving[0] == false && isMoving[1] == false) {
  stability = true;
}
```

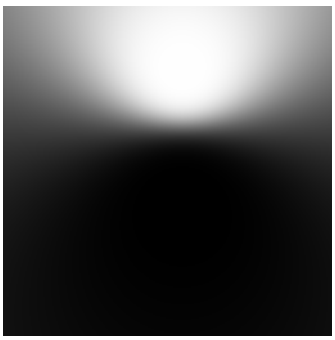
```
    // recalculates the membership
    for (int x = 0; x < 256; x++) {
      for (int y = 0; y < 256; y++) {
        sum = 0;
        for (int k = 0; k < NUM_CLUSTER; k++) {
          center[0] = clus[0][k];
          center[1] = clus[1][k];
          distance = (pow((float)(y - center[0]), 2) +
                      pow((float)(x - center[1]), 2));
          if (distance == 0)
            distance = (float)0.0001;
          if (hist[y][x] > threshold) {
            U[y][x][k] = pow(distance, -k1/2);
            sum += U[y][x][k];
          }
        }
        if (sum == 0)
          sum = 1;

        for (int k = 0; k < NUM_CLUSTER; k++)
          U[y][x][k] /= sum;
      }
    }
  }
}
```

Dal codice riportato precedentemente si può notare che i centroidi vengono spostati di un pixel per volta fino al raggiungimento del punto di minimo della funzione inerzia. Più è grande la distanza tra le coordinate iniziale e finali dei centroidi più iterazioni dell'intero ciclo dovranno essere eseguite (il ciclo può essere eseguito svariate decine di volte). Ad ogni iterazione viene calcolata l'inerzia per ciascun vicino del centroide e, dopo aver spostato il centroide nel vicino di inerzia minima, viene calcolata anche la funzione membership. Il calcolo dell'inerzia e della mebership relativo ad un punto coinvolge tutti i punti dell'istogramma dunque è evidente che la segmentazione ed, in

(a) Istogramma.



(b) Membership relativa al primo cluster.



(c) Membership relativa al secondo cluster.

Figura 3.3: Questo è il risultato del calcolo della membership. In Figura 3.3(b) la membership del cluster con componente P_1 più alta e in Figura 3.3(c) la membership del cluster con componente P_1 più bassa.

particolare, lo spostamento dei centroidi risulta molto dispendiosa dal punto di vista prestazionale.
Pertanto l'assegnazione delle posizioni iniziali dei centroidi incide particolarmente sul numero delle iterazioni svolte e quindi anche sulle prestazioni dell'algoritmo.

Ottimizzazione: posizionamento iniziale dei centroidi

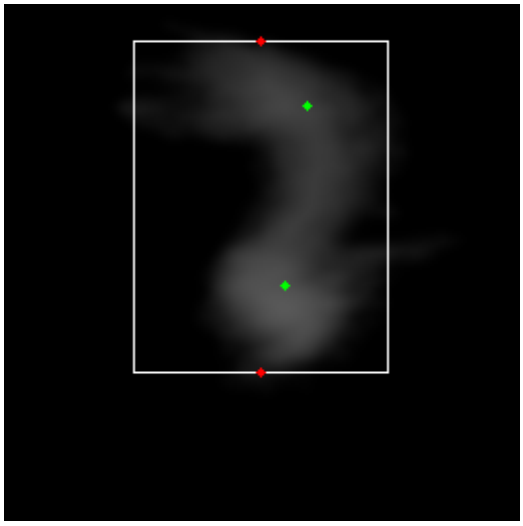
Come detto precedentemente un punto cruciale per ottenere delle buone prestazioni nel calcolo della segmentazione è quello di avere come input dell'agoritmo delle coordinate iniziali dei centroidi il più vicino possibile a quelle che si otterranno come output, tanto più le coordinate iniziali sono vicine a quelle finali tanto minore sarà il numero delle iterazioni svolte nell'algoritmo.
In questa applicazione l'istogramma si presenta sempre con una forma allungata verticale (vedi Figura 3.2) e i due centroidi si trovano sulle estremità di questa figura. Dunque, per assegnare delle coordinate ai centroidi prima di eseguire la vera e propria segmentazione, come primo approccio si può pensare di selezionare tutti i pixel dell'istogramma di valore non trascurabile, prendere in considerazione il rettangolo più piccolo che li contiene e assegnare come coordinate iniziali il valore medio della componente P_2 del rettangolo per entrambi i centroidi, mentre per quanto riguarda la componente P_1, il valore minimo per il primo centroide e il valore massimo per il secondo centroide (in Figura 3.4(a) in rosso è segnata la posizione iniziale dei centroidi calcolata come de-

scritto precedentemente, in verde è segnata la posizione finale ottenuta con la variante dell'algoritmo fuzzy c-means).
Vediamo ora un altro approccio che fornisce coordinate migliori e che quindi riduce notevolmente il numero di iterazioni svolte per il calcolo dei centroidi.
Inanzitutto, anche in questo metodo, si tiene in considerazione il più piccolo rettangolo che contiene tutti i pixel di valore non trascurabile, dopodichè si calcolano i massimi relativi e li si ordinano per componente P_1 crescente. A ciascun punto di massimo M viene assegnato un punteggio $h(M) + \alpha \cdot i$, dove α è una costante, i è la posizione di M secondo l'ordinamento crescente di P_1 e $h(M)$ è il valore dell'istogramma in corrispondenza del punto di massimo M. É facile vedere che più grande è α più rilevanza viene data alla posizione di M rispetto al valore dell'istogramma.
Le coordinate del punto di massimo relativo con punteggio più alto saranno le coordinate assegnate come valore iniziale al primo centroide.
Con lo stesso procedimento (ma con ordinamento decrescente secondo la componente P_1) si assegnano le coordinate al secondo centroide.
É evidente che secondo metodo fornisce coordinate migliori (basti pensare che i centroidi siano sempre interni al rettangolo più piccolo che contiene tutti i pixel non trascurabili dell'istogramma), dall'esempio di Figura 3.4 si può notare che con il secondo approccio le distanze tra i punti iniziali e finali sono più che dimezzate.

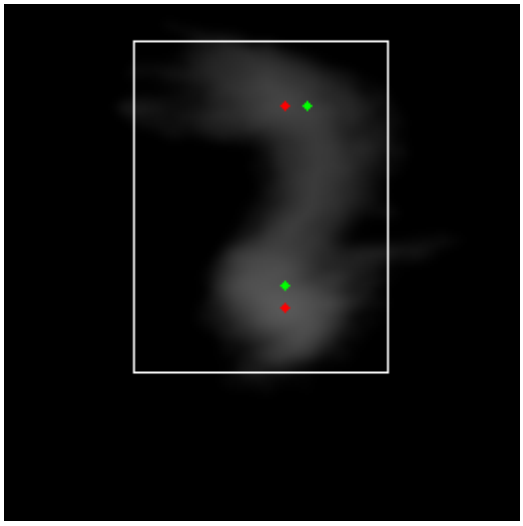
Calcolo della clustered image

Per completare la procedura di segmentazione non rimane altro che ricavare la clustered image[2].
Innanzitutto l'istogramma viene diviso nettamente in base alla membership. Per ogni punto di esso si considera la membership relativa ai due centroidi, la più alta determina l'appartenenza del pixel a quel cluster. Si ottiene in questo modo la cosiddetta clusered region, una matrice della stessa dimensione dell'istogramma (256x256) che ha 0 in corrispondenza del primo cluster e 255 in corrispondenza del secondo (vedi Figura 3.5).
A partire dalla clustered region, seguendo la stessa mappa che è stata usata per costruire l'istogramma, si può disegnare la clustered image.
Ogni punto dell'immagine originaria viene mappato su un unico punto della clustered region tramite le sue componenti principali (proprio come è stato costruito l'istogram-

[2]Per clustered image si intende l'immagine ottenuta tramite una segmentazione. In questo caso, la clustered image è un'immagine dove pixel del nevo sono tutti neri e i pixel della pelle sono tutti bianchi.



(a) Primo approccio per il posizionamento iniziale dei centroidi.



(b) Secondo approccio per il posizionamento iniziale dei centroidi.

Figura 3.4: Il rettangolo bianco contiene tutti i pixel considerati non trascurabili (ovvero sia i pixel con intensità maggiore ad una certa soglia).
I punti rossi identificano le coordinate che sono state assegnate inizialmente ai centroidi mentre i punti verdi sono posizionati in corrispondenza delle coordinate dei centroidi che si ottengono con la variante dell'algoritmo fuzzy c-means.
É evidente che usando primo metodo (Figura 3.4(a)) le distanze tra le posizioni iniziali e finali dei centroidi sono maggiori rispetto a quelle calcolate con il secondo metodo (Figura 3.4(b)). L'utilizzo del secondo metodo comporta una drastica riduzione delle iterazioni compiute dall'algoritmo di segmentazione con un conseguente miglioramento delle prestazioni.

Figura 3.5: Clustered region.

ma), a quell'elemento (della figura originale) viene assegnato lo stesso valore che ha la sua immagine nella clustered region cioè 0 (nero) oppure 255 (bianco).
La clustered image ha bisogno ancora di un'ultima elaborazione. Infatti essa può presentare delle piccole macchioline, queste sono dovute a diversi fattori tra i quali: rumore presente nella foto originale, approssimazioni varie nel calcolo della PCA, anche il fatto stesso di scartare la terza componente (che per quanto sia incorrelata alle altre due è pur sempre, anche se in piccolissima parte, correlata con le altre componenti (si veda la Figura 3.1)), approssimazioni in fase di segmentazione.
Quindi, la clustered image non è altro che un'immagine con diverse macchie nere, alcune delle quali hanno delle macchie bianche all'interno (si veda la Figura 3.6). Ora rimane solo da identificare la macchia che rappresenta il nevo e scartare le altre.

3.4 Identificazione del nevo e calcolo dell'area

Partendo dall'immagine originale (Figura 3.1(a)), mediante la procedura di segmentazione, è stata ottenuta una seconda immagine che è una costellazione di chiazze nere e bianche (Figura 3.6). Questa immagine però non rappresenta ancora il risultato desiderato, infatti l'obiettivo è quello di avere un'unica chiazza nera completamente connessa (quindi senza macchioline bianche al suo interno) che rappresenti in modo univoco il nevo.
Quindi, non rimane che trovare un metodo per identificare correttamente il nevo in questa seconda immagine.
Per fare ciò sono state fatte le seguenti considerazioni:

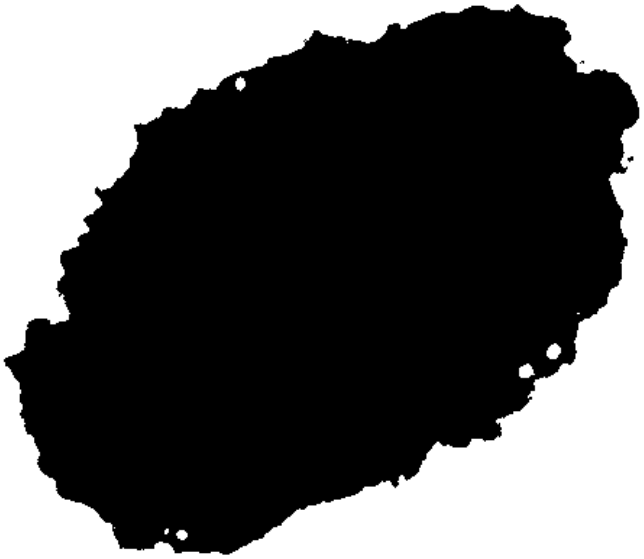


Figura 3.6: Clustered image.

- Le macchie bianche sono sempre all'interno di quelle nere e, poichè si assume che il nevo sia completamente connesso, queste macchie non vengono prese in considerazione.

- Bisogna decidere quale tra le chiazze nere è il nevo. Il criterio di scelta è l'area, pertanto verranno esaminate tutte le chiazze e quella di area maggiore sarà classificata come nevo mentre le altre verranno scartate.

L'algoritmo prevede di scandire l'immagine per trovare tutte le macchie, ogniqualvolta ne viene trovata una ne calcola l'area e il perimetro.
All'avvio della procedura viene istanziata una matrice M delle stesse dimensioni dell'immagine che tiene traccia dei pixel da non visitare durante la scansione. Tutti gli elementi di questa matrice vengono inizializzati al valore 0, ad ogni macchia trovata durante la scansione verrà associato un numero crescente (1 per la prima, 2 per la seconda e così via) che sarà assegnato ai pixel appartenenti a quella macchia.

La scansione viene fatta per righe a partire da un vertice. Per ciascun pixel $c_{i,j}$ vengono esaminati il pixel stesso e l'elemento $M_{i,j}$ corrispondente nella matrice di supporto M (ovvero quello con le stesse coordinate).
Se $M_{i,j}$ ha valore diverso da 0 o se $c_{i,j}$ è bianco si passa al successivo poichè il pixel non va esaminato oppure perchè non è un pixel di bordo. Se invece l'elemento $M_{i,j}$ ha valore 0 e il pixel $c_{i,j}$ è nero significa che è stato trovato il bordo di una macchia quindi la scansione viene interrotta e si procede con il calcolo del perimetro e dell'area della macchia.
La procedura per il calcolo del perimetro segue il bordo della macchia registrando tutte le coordinate dei suoi punti in un vettore, il perimetro è dunque la lunghezza di questo vettore.
Dopo aver trovato il bordo della macchia ne viene calcolata l'area e, contemporaneamente, tutti gli elementi di M in corrispondenza ai pixel appartenenti alla macchia vengono marcati con l'indice della macchia in questione (in questo modo viene segnalato che i pixel appartenenti alla chiazza sono già stati visitati e quindi devono essere ignorati in fase di scansione). Dopo questa fase di calcolo dell'area si riprende la scansione per righe dal pixel in cui si era sospesa e, al termine quest'ultima è finalmente possibile individuare la chiazza di area maggiore quindi quella che identifica il nevo.

Calcolo dell'area

Il calcolo dell'area non è banale e viene descritto di seguito. Viene eseguito dopo aver calcolato il perimetro e ha in input il vettore V contenente le coppie di valori (x, y) che rappresentano coordinate dei punti del bordo.
Siano x_m e x_M le coordinate di ascissa, tra quelle presenti nel vettore, minore e maggiore rispettivamente (quindi relative ai pixel di bordo più a sinistra e più a destra). Analogamente siano y_m e y_M le coordinate di ordinata, tra quelle presenti nel vettore, minore e maggiore rispettivamente (quindi relative ai pixel di bordo più in alto e più in basso).
Viene istanziata una matrice B di larghezza $w = x_M - x_m + 2$ e altezza $h = y_M - y_m + 2$ nella quale verranno mappati i punti del bordo. Ogni elemento della matrice B viene inizializzato al valore 0. Per ogni elemento del bordo di coordinate (i, j) viene impostato l'elemento $B_{i-x_m+1,j-y_m+1}$ ad un valore diverso da 0 (nella fattispecie l'indice della macchia descritto precedentemente).
La matrice B rappresenta il più piccolo rettangolo che contiene la macchia nel quale

solo il bordo della macchia in questione è considerato (se ne può vedere la rappresentazione in Figura 3.7). Questo rettangolo è necessario per migliorare le prestazioni della prossima fase dell'algoritmo che prevede di contare tutti i pixel esterni alla macchia (e interni al rettangolo). Successivamente vengono marcati i pixel esterni al perimetro nella matrice B. La marcatura di questi pixel viene fatta esplorando il rettangolo a partire da un vertice. Dopo aver visitato un pixel vengono visitati i suoi adiacenti che non siano stati già visitati in precedenza o che non siano appartenenti al perimetro della macchia e si procede in questo modo fino all'esaurimento di tutti i pixel.
La funzione si conclude con l'aggiornamento della matrice M e nel conteggio dei pixel interni al perimetro.

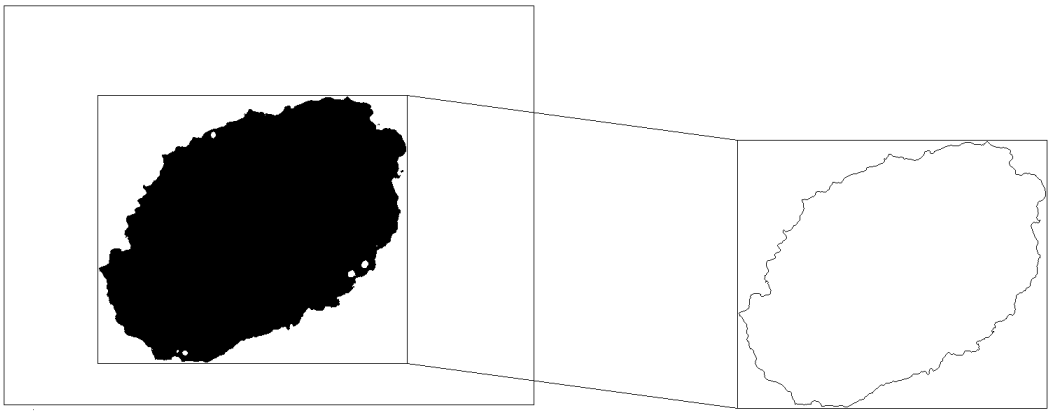


Figura 3.7: Questa immagine illustra il procedimento per il calcolo dell'area. Una volta calcolato il perimetro viene preso un rettangolo che lo contiene (senza che i bordi del rettangolo "tocchino" quelli della macchia). Viene avviata una funzione che marca nella matrice B tutti i pixel esterni al perimetro della macchia. Successivamente utilizza la matrice B per segnare nella matrice M quali sono i pixel interni al nevo e contemporaneamente li conteggia. Non bisogna poi dimenticare di considerare i pixel appartenenti al perimetro quindi è necessario aggiungere la lunghezza del vettore che contiene le coordinate del perimetro.

Il codice della funzione relativa a questo conteggio è riportato di seguito (le denominazioni di B e di M sono rispettivamente borderMap e filledNevus):

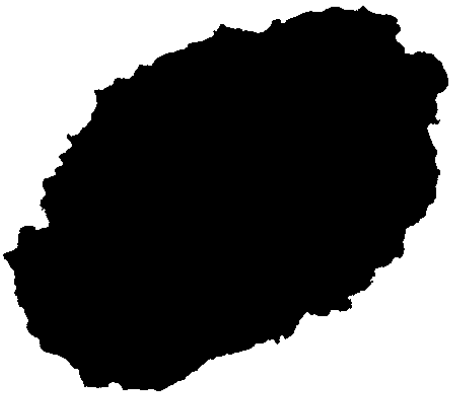
```
/* Function calculateInternalArea
 *
 *  marks each contiguous black pixel of the borderMap starting from the
 *  pixel in 0,0 position
 *  then marks with "indexOfNevus" value the elements in filledNevus
 *  that corresponds to the elements zero marked in borderMap
 *
 *  input:
 *    rows, columns: dimension of borderMap
 *    borderMap: matrix with 0 on every element except the border element.
 *               this is the smallest rectangle that can contains the spot
 *    indexOfNevus: mark of the spot
 *    offsetY, offsetX: offsets in pixels of the firsts pixels of the
 *                      borderMap respect the filledNevus matrix
 *  output:
 *    filledNevus: for each black pixel in borderMap mark its correspondent
 *                 in filledNevus with indexOfNevus the correspondance is:
 *                 borderMap[y+1]*[x+1]<=>filledNevus[y+offsetY][x+offsetX]
 *    returns the number of internal pixel at the border
 *
 */
void fuzzylib::calculateInternalArea(int rows, int columns,
 char** borderMap, char indexOfNevus, char** &filledNevus,
 int offsetY, int offsetX) {
 // queue with the y coordinate of the elements to be visited
  queue< int > yQueue;
  // queue with the x coordinate of the elements to be visited
  queue< int > xQueue;
  char** visited = initMatrix(rows, columns, 0);  // map of the visited element
  char** inQueue = initMatrix(rows, columns, 0);  // map of the queued element

  // inserts the first value: 0,0
```

```
int xElem = 0;
int yElem = 0;
yQueue.push(yElem);
xQueue.push(xElem);
inQueue[yElem][xElem] = 1;

// coordinates of the neighbors
int xUp, yUp;
int xDn, yDn;
int xLf, yLf;
int xRg, yRg;

bool isInternal;

while(!xQueue.empty()) {
  // takes the head
  yElem = yQueue.front();
  xElem = xQueue.front();
  yQueue.pop();
  xQueue.pop();

  // check if the coordinates are "legal"
  isInternal = (yElem >= 0) && (yElem < rows) &&
               (xElem >= 0) && (xElem < columns);

  if( isInternal && (visited[yElem][xElem] == 0) &&
    (borderMap[yElem][xElem] == 0) ) {
    //checks that is internal, not visited yet and black

    // marks that the element as visited
    visited[yElem][xElem] = indexOfNevus;
    // marks the element in the borderMap
    borderMap[yElem][xElem] = indexOfNevus;
```

```
// takes neighbors coordinates
yUp = yElem-1;
xUp = xElem;
yDn = yElem+1;
xDn = xElem;
yLf = yElem;
xLf = xElem-1;
yRg = yElem;
xRg = xElem+1;

// enqueues the neighbors that are internal, black, not visited
// and not in queue yet
if( yUp >= 0 ){
  if( (borderMap[yUp][xUp] == 0) && (visited[yUp][xUp] == 0) &&
      (inQueue[yUp][xUp] == 0) ) {
    yQueue.push(yUp);
    xQueue.push(xUp);
    inQueue[yUp][xUp] = 1;
  }
}

if( xLf >= 0 ){
  if( (borderMap[yLf][xLf] == 0) && (visited[yLf][xLf] == 0) &&
      (inQueue[yLf][xLf] == 0) ) {
    yQueue.push(yLf);
    xQueue.push(xLf);
    inQueue[yLf][xLf] = 1;
  }
}
if( xRg < columns ){
  if ( (borderMap[yRg][xRg] == 0) && (visited[yRg][xRg] == 0) &&
      (inQueue[yRg][xRg] == 0) ) {
    yQueue.push(yRg);
    xQueue.push(xRg);
```

```
          inQueue[yRg][xRg] = 1;
        }
      }
      if( yDn < rows ){
        if( (borderMap[yDn][xDn] == 0) && (visited[yDn][xDn] == 0) &&
            (inQueue[yDn][xDn] == 0) ) {
          yQueue.push(yDn);
          xQueue.push(xDn);
          inQueue[yDn][xDn] = 1;
        }
      }
    }
  }

  // counts the area pixels and updates filledNevus
  int internalArea = 0;
  for(int y = 0; y < rows; y++){
    for(int x = 0; x < columns; x++){
      if(borderMap[y+1][x+1] == 0) {    // +1 because the margin
        filledNevus[y + offsetY][x + offsetX] = indexOfNevus;
        internalArea++;
      }
    }
  }
  return internalArea;
}
```

La marcatura dei pixel esterni alla macchia nella matrice B è stata fatta procedendo iterativamente e tenendo traccia dei pixel da visitare con il supporto di una coda. É stato preferito questo approccio piuttosto che quello ricorsivo perchè meno oneroso in termini di overhead e, in caso di istanze di grandi dimensioni, il numero di chiamate ricorsive può far crashare l'applicazione.
É stata scelto l'utilizzo di una coda (piuttosto che di uno stack) perchè nel corso della procedura si mantiene di dimensioni più contenute, questo è dovuto al fatto che la coda serve con politica fifo mentre lo stack serve con politica lifo.



(a)

(b)

Figura 3.8: In Figura 3.8(a) è riportata l'immagine ottenuta dalla funzione di identificazione del nevo descritto precedentemente sull'immagine di Figura 3.1(a). Oltre a questa immagine la funzione produce come output due numeri interi: il numero di pixel che compongono il nevo cioè l'area e la lunghezza del vettore contenente il bordo cioè la lunghezza del perimetro.
In Figura 3.8(b) è evidenziato il rusultato ottenuto riportando la sovrapposizione dell perimetro trovato sull'immagine d'origine.

Nel primo caso vengono serviti i pixel che sono da più tempo in coda, questo fa sì che i pixel che vengono accodati ad ogni ispezione saranno serviti in breve tempo (quindi verranno rimossi dalla coda al più presto).
Nel secondo caso invece, viene data la priorità agli ultimi pixel che sono stati accodati. Questo significa che ad ogni ispezione solo uno dei pixel che viene accodato sarà servito a breve termine mentre gli altri rimarranno in attesa (e quindi nello stack) ancora a lungo. In particolare si avrà una crescita della pila fino a quando non ci saranno più pixel da accodare, dopodichè lo stack si svuoterà velocemente.
Una volta completata questa fase è facile identificare la macchia di area maggiore nella matrice M poichè ogni macchia è marcata con il proprio indice e si conosce l'indice di quella più grande.
L'immagine di output è costruita colorando di nero i pixel in corrispondenza degli elementi di M marcati con l'indice della chiazza più grande e colorando di bianco gli altri. Con l'identificazione del nevo l'algoritmo è terminato. Come output si hanno dunque l'immagine così ottenuta a sfondo bianco e nevo nero (un esempio è riportato in Figura 3.8(a)), l'area ed il perimetro del nevo espressi come numero di pixel.

3.5 Prestazioni

Le prestazioni dell'algoritmo sono influenzate da vari fattori. Il calcolo delle componenti principali dipende solo dalla dimensione dell'immagine ed esso non rappresenta un collo di bottiglia per l'algoritmo.
La segmentazione lavora sull'istogramma e questo rappresenta un punto importantissimo per quanto concerne le prestazioni poichè la rende indipendente dalle dimensioni dell'istanza su cui lavora. La fase più onerosa della segmentazione consiste in un ciclo (descritto nel paragrafo 3.3) iterato un numero di volte pari alla distanza tra la posizione iniziale e finale dei 2 centroidi. In questo ciclo è presente il calcolo dell'inerzia dell'istogramma per ciascun centroide e per i relativi vicini il che consiste nel calcolo dell'integrale 3.4 per ciascuno di essi. Questo integrale è fatto sull'istogramma quindi comporta due ulteriori cicli nidificati da 256 iterazioni ciascuno per la visita degli elementi dell'istogramma. La segmentazione compie un numero di iterazioni proporzionale alla distanza tra la posizione finale dei centroidi e quella iniziale (si può dire quindi che le sue prestazioni dipendono da quanto sono state buone le predizioni dei centroidi).
Il calcolo della clustered region e della clustered image sono computazionalmente leg-

gere e non pesano sulle prestazioni globali.
L'ultima fase dell'algoritmo ossia l'identificazione del nevo dipende dalla dimensione dell'immagine e dal numero di macchie nere presenti nella clustered image. Questo fa sì che in caso di immagini molto disturbate e di grandi dimensioni questa fase rappresenti il collo di bottiglia dell'algoritmo. Questa fase sicuramente è quella che ha più margini di miglioramento ed è quella sulla quale bisogna puntare per migliorare le prestazioni globali dell'algoritmo.

Conclusioni

L'implementazione dell'algoritmo ha richiesto la conoscenza di base del framework JNI per l'integrazione in programma in Java e di diversi aspetti dell'elaborazione di immagini digitali quali l'analisi delle componenti principali e la segmentazione, ha inoltre permesso di approfondire la conoscenza del linguaggio C++.
Durante l'implementazione del software è sempre stata tenuta in considerazione che esso doveva essere il più veloce possibile. I risultati ottenuti sono decisamente soddisfacenti per quanto riguarda immagini di dimensioni contenute, lo sono un pò per immaggini più grandi.
Il degradamento delle prestazioni per istanze di grandi dimensioni è imputabile all'ultima fase dell'algoritmo che comunque presenta margini di miglioramento.
In futuro è previsto l'uso del tool NDK per rendere disponibile il software anche su piattaforme android.

Bibliografia

[1] http://www.ncbi.nlm.nih.gov/pubmedhealth/PMH0032763/

[2] http://en.wikipedia.org/wiki/Phototype

[3] Catherine M. Poole; DuPont Guerry, *Melanoma: prevention, detection, and treatment*, Yale University, Yale University Press, 2005

[4] U. Veronesi, M. Testori, *Il melanoma: diagnosi e trattamento specialistico*, Elsevier, 2000

[5] http://seer.cancer.gov/statfacts/html/melan.html

[6] http://en.wikipedia.org/wiki/Principal_component_analysis

[7] Fukunaga, K., *Introduction to Statistical Pattern Recognition*, Accademic Press, New York,1971

[8] Jasit S. Suri, David L. Wilson Swamy Laxminarayan, *Handbook of Biomedical Image Analysis, Volume II: Segmentation Models, Part B*, Kluwer Academic/Plenum Publishers, New York, 2005

[9] http://en.wikipedia.org/wiki/Fuzzy_logic

[10] Jasit S. Suri, David L. Wilson Swamy Laxminarayan, *Handbook of Biomedical Image Analysis, Volume II: Segmentation Models, Part A*, Kluwer Academic/Plenum Publishers, New York, 2005

[11] A. Silletti, E. Peserico, A. Mantovan, E. Zattra, A. Peserico, A. Belloni Fortina, *Variability in human and automatic segmentation of melanocytic lesions*, 31st Annual International Conference of the IEEE EMBS, Minneapolis, 2009

Elenco delle figure

1.1 Anatomia della pelle 6
1.2 Melanoma 8

3.1 Analisi PCA 18
3.2 Istogramma 21
3.3 Segmentazione: membership 27
3.4 Segmentazione: posizionamento iniziale dei centroidi 29
3.5 Clustered region 30
3.6 Clustered image 31
3.7 Calcolo dell'area 34
3.8 Risultato della segmentazione 39