Università degli Studi di Bari

Corso di Laurea Magistrale in Informatica

Elaborazione di Immagini e Tecniche Fuzzy

Fuzzy Clustering per la segmentazione di immagini in ImageJ

Studenti:
Antonio Vergari
Francesco Tangari

Docente:
Prof.ssa Laura Caponetti

1 febbraio 2011

Indice

Premessa, introduzione e obiettivi **1**
Segmentazione 1
Clustering 4
Organizzazione del documento 6

K-Means **7**
Algoritmo di Lloyd 7
KMeans Plugin: Refactoring 8
Estensione a feature generiche 10
Scelta spazio colore 10
Criteri di inizializzazione 18
Modalità di visualizzazione 20

Fuzzy C-Means (*FCM*) **22**
Una nota di carattere numerico 23
Equivalenza fra Fuzzy C-Means e K-Means 23
FCM Plugin: Refactoring 23
Criteri di inizializzazione 26
Criteri di terminazione 26
Modalità di visualizzazione 27

Spatial Fuzzy C-Means (*SFCM*) **31**
Equivalenza fra Spatial Fuzzy C-Means e Fuzzy C-Means 31
Scelta della funzione spaziale 32
SFCM Plugin: Refactoring 32

Validazione **38**

Riferimenti Bibliografici **42**

Indice

Premessa, introduzione e obiettivi

Il progetto, realizzato nell'abito del corso di *Eleborazione di Immagine e Tecniche Fuzzy*, ha l'obiettivo di implementare un insieme di algoritmi, fra loro affini, per la segmentazione di immagini a colori. Le sezioni che tratteranno, in questo documento, di *Image Processing* riguarderanno la *segmentazione* e la rappresentazione del *colore* sotto forma di codifica in differenti spazi; gli argomenti di algoritmica e teoria Fuzzy saranno invece trattati puntualmente nelle sezioni successive.

Segmentazione

Segmentare una immagine vuol dire raggruppare i suoi pixel in maniera tale da ottenere delle aree di interesse che siano significative per l'immagine in questione e per l'utente che applica la tecnica in relazione ad un determinato dominio o applicazione.

L'interesse potrebbe riguardare esclusivamente una particolare regione dell'immagine, come un oggetto in primo piano tale da rendere trascurabile qualsiasi cosa sullo sfondo, o il riconoscimento di una regione anomala che non segue un determinato pattern specificato dalle altre. Segmentare una immagine, dunque non vuol dire necessariamente associare una etichetta a ciascun elemento possibilmente riconoscibile[1] in una immagine digitale, e spesso diventa sinonimo di *estrazione* proprio di quelle aree intorno alle quali ruota l'interesse per uno specifico dominio.

Sembrerebbe che da questo ridimensionamento del task si possa ottenere una facilitazione, maggiore semplicità pratica, purtroppo la segmentazione rimane uno dei compiti più ardui dell' image processing. Sovente un passo di segmentazione precede successive fasi di elaborazione o comprensione di una immagine ed allora gli errori e le imprecisioni qui commesse possono propagarsi nelle seguenti compromettendo tutto il processo. Altre volte ancora, di fronte ad alcuni obiettivi, sebbene semplici su carta, si è costretti a confrontarsi con problemi legati alla cattiva acquisizione della immagine da segmentare o anche all'impossibilità di riuscire a descrivere una immagine tramite delle feature che consentano il risultato segmentato aspettato.

Se riduciamo la nostra prospettiva alle sole problematiche dipendenti direttamente dal task di segmentazione possiamo, sommariamente, elencarne le maggiori, o quantomeno le più frequenti per chi ha a che fare con la segmentazione di immagini:

- **Alta frammentazione delle regioni**, ovvero il risultato del task restituisce una immagine segmentata in un determinato numero di regioni, una o più delle quali è

[1] Anche un essere umano avrebbe non pochi grattacapi a dividere precisamente una immagine in tutti e soli i suoi elementi costituenti, sia perché la nozione di costituente potrebbe non essere oggettiva sia perché molte regioni potrebbero sovrapporsi o complementarsi dando origine a nuove aree *sfuocate*, nell'accezione di *fuzzy*

a sua volta suddivisa in tante micro aree fra loro non adiacenti ed immerse nelle rimanenti regioni.

- **Errata assegnazione di sotto-regioni**, che si verifica quando una sotto area, percepita come appartenente ad una regione viene assegnata ad una altra o anche quando due aree vengono riconosciute in luogo di una sola o specularmente quando due aree distinte vengono raggruppate sotto la stessa etichetta, sintomo che si stanno descrivendo gli elementi base del processo di segmentazione in maniera errata o ad un livello di dettaglio non conforme con le aspettative

- **Instabilità in presenza di rumore** e più in generale in presenza di immagini iper dettagliate, quando il raggruppamento in regioni omogenee e adiacenti può ricondurre al primo problema, la sovra-segmentazione, o quando le immagini da segmentare sono state acquisite o pre-elaborate in maniera impropria. In queste circostanze si richiederebbe all'algoritmo di segmentazione di essere *robusto* e portare a risultati sperati prescindendo dai dati su cui opera.

- **Difficoltà nel segmentare semanticamente**, quando cioè si vorrebbero estrarre regioni che, in relazione alla conoscenza di dominio, possano venire riconosciute semanticamente ma sempre automaticamente. L'operazione di segmentazione deve quindi essere compiuta tenendo conto non solo degli elementi dell'immagine e dei descrittori estraibili da essa

Essendo il task di segmentazione al centro del nostro lavoro come progetto, ci siamo dovuti scontrare con le problematiche di cui sopra. Crediamo, e speriamo che anche il lettore se ne ravvederà dalla lettura di questo documento e dall'utilizzo di questo plugin, di essere riusciti ad arginare i primi tre problemi con l'uso combinato di tecniche fuzzy e dell'informazione spaziale associata ad ogni intorno dei pixel dell'immagine.

Non siamo però stati in grado di segmentare semanticamente, come farebbe un essere umano non necessariamente esperto del dominio, le immagini trovate il più delle volte. La risposta che abbiamo fornito di fronte a questa difficoltà irrisolta è l'utilizzo dei soli descrittori relativi all'informazione del colore. La segmentazione attraverso l'uso delle informazioni di *colore* associate ad ogni pixel ha lo scopo di ottenere regioni delle immagini che siano omogenee rispetto a dei descrittori che rappresentano uno dei livelli più bassi ai quali può essere descritta una immagine. La percezione del colore sta ad una immagine come la rappresentazione sintattica sta ad una frase in linguaggio naturale, talvolta il colore è sufficiente a determinare la regione di alcuni fiori da estrarre rispetto un campo verde di erba, ma il più delle volte da sola risulta inefficace o, peggio, può condurre alla sovra-segmentazione o all'errata assegnazione di sotto-regioni. Nelle immagini che verranno utilizzate si farà sempre l'assunzione, dove non specificato altrimenti, che l'informazione del colore, da sola, sia sufficiente a condurre ad una *decente* segmentazione.

In letteratura gli approcci alla segmentazione, grazie ai quali possiamo virtualmente suddividere le tecniche per la segmentazione in due grandi famiglie, si basano sulla rilevazione di *discontinuità* o sulla ricerca di *similarità* all'interno dell'immagine oggetto del task.

Se consideriamo come descrittore per un pixel dell'immagine l'intensità del suo livello di grigio, rilevare discontinuità in essa potrebbe essere realizzato ricercando in tutta l'immagine le zone in cui il valore dell'intensità subisce un'improvvisa e notevole variazione e si potrebbe raggruppare tutti i punti contigui con la stessa intensità in una unica regione. Approcci meno banali e sicuramente più efficienti in letteratura hanno riguardato il rilevare:

- **Punti isolati** che si differenziano particolarmente dal background e questo si può fare utilizzando operatori come il *Laplaciano* applicato puntualmente, in maniera tale da accentuare la differenze con lo sfondo.

- **Linee** espresse come una successione di punti adiacenti in rilievo rispetto allo sfondo; ugualmente per i punti, maschere che mettono in rilievo il contrasto e sono operatori differenziali discretizzati, come quelle di Prewitt, possono essere applicate in filtri per estrarre da una immagine le linee costituenti o in risalto, qualora ve ne fossero.

- **Contorni *(edge)***, la naturale generalizzazione di punti e linee, per l'individuazione del passaggio da una area dell'immagine ad una altra con intensità differente. Ancora una volta, operatori differenziali del primo o di ordini successivi, o operatori compositi maggiormente resistenti al rumore (come LoG) possono essere applicati localmente ad una immagine per filtrarne le aree di discontinuità.

Le tecniche che ricercano la similarità fra le sotto aree, specularmente, andranno ad individuare aree che risultano essere simili. Una breve panoramica su alcune tipologie di tecniche:

- **Thresholding** applicato all'istogramma dell'immagine, con l'obiettivo di trovare dei valori soglia che partizionino l'istogramma secondo le mode (nell'ipotesi che regioni simili possano essere descritte da un picco dell'istogramma), oltretutto operando in maniera efficiente poiché non direttamente sull'immagine. Una tecnica potrebbe essere il *metodo di Otsu* che cerca di determinare un valore soglia minimizzando la varianza intra classe.

- **Region growing** che raggruppa pixel connessi (in un intorno) incrementalmente se soddisfano un determinato predicato, come potrebbe essere il criterio che i pixel adiacenti abbiano un colore fra loro vicino, in regioni. I pixel di partenza, *seed* potrebbero essere scelti secondo un particolare criterio o in maniera random. Immagini con istogrammi multimodali potrebbero essere più facilmente segmentate con una tecnica del genere che con thresholding.

- **Region splitting and Merging** consistente nel suddividere l'immagine in regioni disgiunte se non similari e nel fondere regioni individuate come simili secondo un dato criterio. Strutture dati di supporto come i *quad tree* sono generalmente utilizzate per tenere traccia del partizionamento incrementale.

A questi approcci si aggiunge quello che comprende le tecniche di ***clustering*** e in cui ricadono anche gli algoritmi scelti per questo progetto. Si approfondirà in seguito quanto clusterizzare una immagine in partizioni sia sinonimo di segmentarla in regioni.

Clustering

Numerose tecniche di *clustering* ben si prestano ad essere utilizzate per un task di segmentazione di immagini. Di fatto, clusterizzare un insieme di dati multivariati non vuol dire altro che *partizionare* lo spazio dei dati in maniera da ottenere gruppi di dati (i cluster) che siano fra loro *simili* (ovvero *vicini* secondo una qualche misura di distanza o dissimilarità). Il numero di partizioni deve essere espresso a priori[2], ma ciò è l'unico *suggerimento* che si dà all'algoritmo per ricercare le regioni in cui segmentare una immagine[3].

Tecniche di clustering sono state sviluppate a partire dalla fine degli anni 30[1] e non sarebbe difficile descriverle come analoghe ad alcune tecniche, prima citate, per la segmentazione di immagini: facili parallelismi si possono fare fra l' *agglomerative clustering* e *region growing* o più in generale fra *clustering gerarchico* e *region splitting & merging*, ma la nostra attenzione sarà concentrata sugli algoritmi di clustering *partitivo*, in cui la creazione di partizioni fra i dati è derivante dalla minimizzazione di una particolare *funzione costo* rappresentante i vincoli o i predicati che devono essere verificati per poter dire che due dati siano a sufficienza simili da appartenere alla stessa partizione o dissimili dagli altri appartenenti alle altre partizioni.

Il grado di appartenenza di un dato (nel nostro caso la reppresentazione di un pixel tramite attributi di colore) ad un cluster può essere un valore *crisp*, o 0 o 1, ed ogni dato può venir assegnato ad un unico cluster di appartenenza (*Hard Clustering*). È anche possibile che un dato possa appartenere a diversi cluster, o meglio che non si sia certi della sua assoluta appartenenza ad un cluster piuttosto che agli altri, in tal caso una partizione ottenibile da un algoritmo che tenga in considerazione questa eventualità presenterà valori *compresi* tra 0 e 1, *fuzzy* (nella stessa accezione di *fuzzy logic*), ognuno dei quali indicherà il livello di incertezza ottenuto nell'assegnare un dato ad un cluster.

[2]Il numero di cluster, k, deve essere, inoltre, maggiore di 2 e minore della cardinalità dell'insieme dei dati.

[3]Gli algoritmi di clustering, ideati per l'analisi dei dati multivariati, fanno parte di quegli algoritmi di *unsupervised learning* in apprendimento automatico, ed hanno lo scopo di descrivere in maniera automatica, senza necessità di dati di training, l'insieme dei dati di cui si dispone

Sono proprio tecniche di *Fuzzy Clustering* le due che esploreremo in questo case study; partiremo però dalla segmentazione di immagini utilizzando il colore attraverso un classico algoritmo di hard clustering, *K-Means*, passando per la sua evoluzione fuzzy, *Fuzzy C-Means* ed approdando ad una rivisitazione di quest'ultimo.

L'obiettivo principale di questo progetto è la realizzazione di un plugin per *ImageJ*[2] che implementi una variante dell'algoritmo Fuzzy C-Means che tenga conto non solo della informazione relativa al colore di ogni pixel di una immagine, ma anche di quella riguardante tutti i pixel ad esso spazialmente vicini. In letteratura una tale variante è stata proposta in [3] e tale riferimento è stato criticamente seguito nella realizzazione del progetto. Non si è partiti ex nihilo per la creazione del plugin, si è utilizzato come punto di partenza il lavoro gi fatto per la segmentazione di immagini basata su colore presente nella toolbox in [4]. Un gran lavoro di ristrutturazione ingegneristica (*refactoring*) è stato fatto per accomodare il codice alle esigenze e agli obiettivi del progetto, tra cui:

- **indipendenza dalle API di ImageJ** un algoritmo dipendente dalle API di un software (è il caso del plugin in questione) difficilmente potrà essere riutilizzato in futuro
- **indipendenza dalla rappresentazione dei dati** un pixel verrà rappresentato tramite i valori del colore ad esso associato, tuttavia è bene che l'algoritmo non sappia su che tipo di valori stia operando, nè quanti valori vengano utilizzati per rappresentare ogni singolo pixel
- **estensione per spazi colore differenti da RGB** perché se l'algoritmo sarà effettivamente indipendente dalle feature l'aggiunta di questa funzionalità dovrebbe portare minimo sforzo (implementativo) e massimo rendimento (in termini di utilizzo del plugin)
- **aggiunta di diversi criteri di inizializzazione** poiché l'esito di questi algoritmi di clustering è fortemente influenzato da come e cosa l'algoritmo inizia a ricercare la soluzione
- **aggiunta di diverse modalità di visualizzazione** per poter valutare qualitativamente, ad occhio, in maniera migliore i risultati dell'algoritmo
- **aggiunta di funzioni di validazioni** per poter valutare quantitativamente i risultati di differenti esecuzioni dell' algoritmo
- **scelta fra diverse funzioni spaziali** ovvero differenti modalità di computare l'informazione spaziale associata ad ogni pixel

Organizzazione del documento

Il presente documento servirà da documentazione per il progetto e si andrà ad aggiungere alla *javadoc* generata per il codice prodotto. Lo scopo di questa documentazione digitale sarà quello di descrivere i passi compiuti nella realizzazione dei plugin, spiegando e motivando le scelte compiute, e allo stesso tempo fornendo un background teorico riguardante gli algoritmi trattati ed implementati (come si vedrà i due scopi sono ben intrecciati fra loro). Verranno allegate numerose immagini d'esempio (utilizzate anche all'interno di questo documento) e grafici risultanti dall'analisi dei dati dei test effettuati.

L'ordine di presentazione che verrà seguito è quello cronologicamente, storicamente avvenuto (anche per quanto riguarda la creazione dei plugin). Come già accennato si partirà dall' algoritmo K-Means, si seguirà con Fuzzy C-Means, si terminerà con Spatial Fuzzy C-Means e si proporrà una parte conclusiva riguardante la validazione degli algoritmi di clustering. Per ogni algoritmo trattato, una parte teorica introduttiva precederà una successiva parte ingegneristica che spiegherà quanto fatto e perché alla luce degli obiettivi sopra elencati; inoltre, per quanto possibile, si cercherà di mostrare dei risultati concreti di ogni cambiamento apportato per fornire al lettore un metro qualitativo immediato).

K-Means

Il primo algoritmo che tratteremo è uno dei più noti in letteratura e più usati per fare clustering: **K-Means**. L'idea dietro K-Means risale al 1956, sebbene il nome non sarebbe comparso fino al 1967 [5], e consiste nel determinare un partizionamento che minimizza la distanza all'interno di ogni partizione fra i punti ad essa appartenenti e il *centroide* della partizione, ovvero il punto medio.

In maniera formale, dati:

$$\mathbf{X} = \{\mathbf{x_1}, \mathbf{x_2} \dots \mathbf{x_N}\}, \mathbf{x_1} \dots \mathbf{x_N} \in \mathbb{R}^q$$

un insieme di N dati rappresentanti vettori di q feature, data una partizione di k cluster

$$\mathbf{P} = \{p_1, p_2, \dots p_k\}$$

che rispetti is eguenti vincoli:

$\bigcup_{p \in \mathbf{P}} p = \mathbf{X}$	(l'unione di tutte le partizioni dia l'insieme dei dati)
$\bigcap_{p \in \mathbf{P}} p = \emptyset$	(le partizioni siano fra loro disgiunte)
$\emptyset \subset p_i \subset \mathbf{X}$	(nessun cluster deve essere vuoto o contenere tutti i dati)

e dato un insieme di k centroidi (uno per cluster)

$$\mathbf{V} = \{\mathbf{v_1}, \mathbf{v_2} \dots \mathbf{v_k}\}, \mathbf{v_1} \dots \mathbf{v_k} \in \mathbb{R}^q$$

l'obiettivo è quello di minimizzare la funzione obiettivo

$$J(\mathbf{U}, \mathbf{V}) = \sum_{i=1}^{N} \sum_{j=1}^{k} ||\mathbf{x_i} - \mathbf{v_j}||^2 \tag{1}$$

dove U è una matrice di partizionamento di dimensione $N \times k$ i cui elementi u_{ij} sono pari a 1(qualora l'elemento i-esimo appartenga al cluster j-esimo) o 0 (qualora non appartenga) e la norma al quadrato utilizzata può essere una qualsiasi distanza o misura di dissimilarità (solitamente è la norma euclidea). La notazione matriciale della funzione obiettivo J, è sovente espressa senza la matrice U. L'esplicitarla sin da ora risulterà parecchio conveniente per la trattazione in questo documento, si noti intanto come il vincolo sugli elementi di U rispecchi quanto detto a proposito di K-Means come algoritmo di *hard clustering*.

Algoritmo di Lloyd

Determinare una partizione hard in cluster (la matrice U) e un insieme di centroidi (la matrice V) per minimizzare la funzione J descritta in (1) è un problema **NP-Hard** la cui soluzione sarebbe intrattabile in un tempo ragionevole [4].

[4] Essendo noi interessati ad applicare questo algoritmo alla segmentazione di immagini utilizzano il loro livello sintattico, il colore, siamo alla ricerca di tempi ragionevoli di esecuzione

L'implementazione classica dell'algoritmo è una versione iterativa proposta da Lloyd nel 1956 [6], la quale di fatto è una approssimazione in grado di trovare dei minimi *locali* per J.

Ecco cosa prevede la procedura iterativa:

1. Inizializza in maniera casuale U e V
2. calcola U che minimizzi $J(V)$
3. calcola V che minimizzi $J(U)$
4. se l'algoritmo converge fermati, altrimenti torna al passo 2

Il primo passo corrisponde alla fatidica *inizializzazione* dell'algoritmo, inizializzazioni diverse porteranno ad approssimazioni fra loro diverse e dunque a possibili minimi locali differenti. L'ultimo passo invece ha il compito di determinare se l'algoritmo ha trovato un punto di minimo e viene generalmente realizzato computando la differenza fra la matrice dei centroidi fra una iterata e la precedente, se tale differenza è trascurabile, l'algoritmo termina. I due passi intermedi, invece, provvedono all'aggiornamento di U e V e le regole implementative per gli aggiornamenti sono le seguenti:

$$u_{ij} = \begin{cases} 1 & \text{se } ||\mathbf{x_i} - \mathbf{v_j}||^{\mathbf{2}} < ||\mathbf{x_i} - \mathbf{v_h}||^{\mathbf{2}},\ \forall h \neq j \\ 0 & \text{altrimenti} \end{cases} \tag{2}$$

$$\mathbf{v_j} = \frac{\sum_{i=1}^{N} u_{ij}\mathbf{x_i}}{\sum_{i=1}^{N} u_{ij}} \tag{3}$$

Nella regola di update (2) si assegna ad ogni elemento di U il valore 1 solo se la distanza fra il dato e il centroide identificati dall'elemento corrente è la minore fra le distanze calcolabili fra il dato e gli altri centroidi, mentre la (3) assegna ad ogni nuovo elemento della matrice V il valore medio fra i dati appartenenti al cluster identificato dal centroide.

KMeans Plugin: Refactoring

Il plugin per ImageJ implementato in [4] realizza la versione dell'algoritmo di Lloyd per implementare K-Means. Le assunzioni di base che fa, da cui noi siamo partiti per realizzare il nostro, sono:

- ogni dato da clusterizzare è un pixel espresso tramite le sue componenti RGB o il solo valore di luminosità in scala di grigi
- il criterio di inizializzazione per la matrice V è *K-Means++*[5]

[5]Nella sottosezione *Criteri di inizializzazione* se ne spiegherà il funzionamento in dettaglio

- il criterio di stop é dato dal calcolo della *norma di Frobenius* sulla differenza fra la matrice dei centroidi V fra due iterate
- la metrica utilizzata per misurare la distanza fra un pixel ed un centroide è la *distanza euclidea*
- la matrice U non viene mai esplicitamente computata e salvata in memoria, ogniqualvolta diviene necessario sapere l'appartenenza di un pixel ad un cluster, per aggiornare la matrice dei centroidi, si calcola il centroide ad esso più vicino andando a calcolare la distanza euclidea fra quel dato e tutti i centroidi computati per quell'iterata.

Alcune assunzioni saranno da noi condivise per tutta la durata del progetto (ad esempio la scelta della distanza euclidea come misura), altre saranno successivamente estese (come la rappresentazione di un unico spazio colore), altre ancora, scartate da principio (come il non serbare esplicitamente la matrice di partizionamento in memoria). Nel seguito del documento, estensioni e modifiche saranno nel dettaglio maggiormente trattate.

La fine di questa sezione tratterà di come si è ristrutturato il plugin architetturalmente per consentire il raggiungimento degli obiettivi già elencati ed in particolare per ottenere l'indipendenza dalle librerie di ImageJ [6].

Il plugin originario sebbene strutturato in più di una classe, presenta una forte interconnessione fra le differenti componenti e tutte hanno libero accesso ai pixel della immagine su cui sta operando, ogniqualvolta una funzione necessita di sapere il valore dell' i-esimo dato/pixel la lettura avviene tramite le API di ImageJ sull'immagine. Ancora, quando la classe che implementa l'algoritmo di clustering deve comunicare con l'esterno il proprio stato, essa utilizza le API di ImageJ per scrivere in console. Il nostro obiettivo primario è stato quello di eliminare questo forte accoppiamento ed è risultato nella creazione di tre classi con responsabilità ben distinte:

- **KMeansPlugin** che implementa l'interfaccia *PlugIn* di ImageJ e presenta all'utente la schermata di configurazione dei parametri dell'algoritmo e si occupa della visualizzazione dei risultati della clusterizzazione

- **KMeansManager** controllore dell'algoritmo che si occupa di configurarne i parametri, trasformare i dati in input per astrarre dal formato di ImageJ, eseguirlo, raccogliere i dati risultanti e codificare le nuove immagini clusterizzate tramite le API di ImageJ

- **KMeans** che implementa la logica dell'algoritmo ed opera su dati generici, massimizzando il possibile riuso futuro.Al termine dell'algoritmo di clustering computa la matrice U esplicitamente una sola volta e consente a KMeansManager multiple visualizzazioni a partire proprio dalla matrice, senza ulteriore sforzo computazionale.

[6] una documentazione aggiuntiva, diagrammatica, chiarirà in maggior dettaglio quanto fatto

Implementa il *delegation Pattern* per comunicare il proprio stato all'esterno ed il suo delegato è KMeansManager.

L'architettura così ristrutturata sarà alla base anche delle successive ristrutturazioni per gli algoritmi seguenti, soggetta a minimi cambiamenti[7].

Estensione a feature generiche

Il plugin originario rappresenta di già i pixel come array di feature ma limita il numero delle feature e il loro tipo alle componenti dello spazio colore RGB (qualora invece l'immagine su cui lavora il plugin sia in scala di grigi allora ogni pixel sarà rappresentato da un unico valore, la propria luminanza) e ciò è diretta conseguenza della dipendenza dalle libreria di ImageJ.

Avendo noi, nel nostro progetto, svincolato il plugin da tale dipendenza, possiamo rappresentare un pixel come un vettore di un arbitrario numero di descrittori; con un unico vincolo applicabile, ovvero che i valori dei descrittori siano tutti rappresentabili come numeri reali[8] poiché la scelta della struttura dati che conterrà i dati è caduta su una matrice per ovvie ragioni di efficienza.

La classe KMeans non è addirittura a conoscenza che le righe della matrice dei dati che viene data in pasto all' algoritmo rappresentino dei pixel, e ciò la rende riutilizzabile in altri contesti e domini applicativi oltre a facilitare l'introduzione di ulteriori descrittori; tuttavia le altre due classi, dovendo comunicare con l'utente e le librerie di ImageJ rimangono responsabili della *estrazione* dei descrittori da ogni pixel. Questo comportamento è stato anch' esso oggetto di estensione (come sarà chiaro nella prossimo paragrafo), ma la riutilizzabilità delle classi KMeansManager e KMeansPlugin può avere senso solo all'interno di ImageJ.

Nonostante ciò, consentiamo l'introduzione di feature arbitrarie non generabili all'interno del plugin stesso: si consideri ad esempio uno *stack* in ImageJ composto da cinque slice, le prime tre descrivono le componenti colore classiche, la quarta rappresenta l'immagine originaria a cui è stato applicato un filtro basato sulla media e la quinta una immagine rappresentante l'originale filtrato in base alla varianza. Il quarto e il quinto descrittore per ogni pixel saranno dunque il valore medio e la varianza calcolato in un intorno specificato dal raggio dei filtri utilizzati.

Scelta spazio colore

L'estensione più semplice realizzabile come concreta dimostrazione della estensibilità dello algoritmo è l'introduzione di spazi di colore differenti dal canonico RGB.

[7] Una buona reingegnerizzazione del plugin ha consentito anche di minimizzare le modifiche future

[8] Il vincolo più che limitare la tipologia di descrittori utilizzabili, limiterebbe il formato con cui è possibile rappresentare i valori, una conversione diverrebbe necessaria per attributi categorici ad esempio

Il risultato di un task di segmentazione di una immagine attraverso attributi di colore non potrà che essere determinato dalla scelta dello spazio colore, quindi l'estensione proposta è pi'u che ragionevole per un plugin per ImageJ, consentirà di ottenere e visualizzare con facilità immagini convertite e clusterizzate in spazi colori alternativi, di scegliere, infine, il risultato migliore fra quelli ottenibili.

Si è scelto di supportare la conversione tra tre possibili spazi colore:

- **XYZ** uno spazio colore che, come RGB, tenta di rappresentare il colore avvicinandosi alla percezione biologica per come avviene nell'occhio umano[9]. La componente Y rappresenta la *luminanza*, mentre le altre due componenti la *cromaticità*. Lo spazio RGB è di fatto un sottospazio di XYZ , da ciò interessante vedere come uno spazio di colore affine ad RGB possa condurre a risultati differenti. Per convertire XYZ in RGB si procede con una conversione in due passi, nel primo si ottengono le componenti r, g e b che sottendono una versione lineare dello spazio RGB attraverso la seguente trasformazione lineare[10]:

$$\begin{pmatrix} r \\ g \\ b \end{pmatrix} = \begin{pmatrix} 3.2404542 & -1.5371385 & -0.4985314 \\ -0.9692660 & 1.8760108 & 0.0415560 \\ 0.0556434 & -0.2040259 & 1.0572252 \end{pmatrix} \begin{pmatrix} X \\ Y \\ Z \end{pmatrix}$$

  Successivamente, lo spazio lineare *rgb* verrà rapportato ad *RGB* applicando una particolare correzione a ciascuna componente. La scelta della correzione dipenderà dal valore di ogni singola componente lineare poiché alcuni valori potrebbero non essere rappresentabili nello spazio canonico RGB (essendo XYZ uno spazio più ampio), considerando ad esempio la componente del rosso si avrà che:

$$R = \begin{cases} 1.055 \cdot r^{1/2.4} - 0.055 & \text{se } r > 0.0031308 \\ 12.92 \cdot r & \text{se } r \leq 0.0031308 \end{cases}$$

  Per la trasformazione lineare da RGB a XYZ sarà sufficiente applicare la matrice inversa di quella utilizzata in precedenza:

$$\begin{pmatrix} X \\ Y \\ Z \end{pmatrix} = \begin{pmatrix} 0.4124564 & 0.3575761 & 0.1804375 \\ 0.2126729 & 0.7151522 & 0.0721750 \\ 0.0193339 & 0.1191920 & 0.9503041 \end{pmatrix} \begin{pmatrix} r \\ g \\ b \end{pmatrix}$$

[9] L'occhio umano possiede tre tipi diversi di sensori che ricevono l'impulso luminoso cromatico, differenziati in base alla lunghezza d'onda dello stimolo che sono chiamati a riconoscere, questi sensori sono specializzazioni dei coni le cellule della retina addette al riconoscimento del colore

[10] Lo spazio colore XYZ è, come lo spazio RGB, dipendente dallo schermo o strumento su cui viene visualizzato e con particolare riferimento ad una codifica del colore bianco, a differenti profili RGB, adatti ad una particolare device, e a differenti rappresentazioni del colore bianco corrisponderanno differenti trasformazioni lineari. Quella da noi utilizzata è riferita allo spazio *sRGB* e al bianco ottenuto tramite una fonte di illuminazione *D65*.

Lo spazio XYZ è stato il primo spazio ad essere matematicamente definito[11] e standardizzato dal CIE (*International Commission on Illumination*) ed è servito come punto di riferimento per la definizione dei successivi standard. Per tale ragione risulta semplice utilizzare XYZ come spazio colore intermedio di conversione per il passaggio da RGB ad altri spazi e viceversa.

- **L*a*b***, spazio colore derivabile da XYZ ma con un obiettivo differente, la rappresentazione uniforme della gamma dei colori percepibili dall'occhio umano[12], in maniera tale che a incrementi lineari di una tonalità corrispondano cambiamenti naturali nella percezione del nuovo colore[13].

  Per computare i valori delle componenti dello spazio si utilizza una trasformazione a partire dalle componenti standard XYZ, i cui valori vengono inizialmente rapportati al valori standard calcolati in presenza di una data fonte luminosa[14]:

$$X_r = \frac{X}{0.95047}, \quad Y_r = \frac{Y}{1.0000}, \quad Z_r = \frac{Z}{1.08883}$$

  Successivamente tali valori vengono corretti (proiettati nel nuovo spazio) se sorpassano una determinata soglia definita dal CIE come $\epsilon = 0.008856$, utilizzando un valore costante $\kappa = 903.3$ rapportato alla luminosità qualora fossero al di sotto della soglia. La seguente correzione è uguale per ciascuna componente, di seguito per la X_r:

$$f(X_r) = \begin{cases} X_r^{1/3} & \text{se } X_r > \epsilon \\ (\kappa \cdot X_r + 16)/116 & \text{se } X_r \leq \epsilon \end{cases}$$

  Infine si computano le componenti L^*, a^* e b^* in funzione dei valori corretti precedentemente, scalandoli opportunamente:

$$\begin{aligned} L^* &= 116 \cdot f(Y_r) - 16 \\ a^* &= 500 \cdot (f(X_r) - f(Y_r)) \\ b^* &= 200 \cdot (f(Y_r) - f(Z_r)) \end{aligned}$$

  La trasformazione inversa, è di facile computo.

  In termini di luminosità e cromaticità, L^* specifica il valore di luminanza, a^* codifica il valore cromatico compreso tra il magenta e il verde mentre b^* il valore fra giallo e

[11] I valori per i tre stimoli luminosi da associare alle sue componenti sono stati individuati per via empirica attraverso numerosi esperimenti controllati

[12] Il risultato è però uno spazio troppo grande per essere interamente rappresentato secondo i classici spazi per la visualizzazione a monitor, e di stampa

[13] Per esempio in XYZ differenze di colore non sono egualmente percepite nello spazio geometrico definito dagli assi componenti, per un dato incremento si possono percepire grandi cambiamenti nella regione del magenta ma lievi per la regione del verde, a parità di distanza di coordinate

[14] Utilizziamo come riferimento per la luminosità sempre il bianco standard a *D65*

blu. I valori di L^* sono generalmente positivi e compresi fra $[0, 100]$, mentre a^* e b^* assumono valori nell'intervallo $[-127, +127]$.

Lo standard CIE *L*a*b** è stato pensato per essere indipendente dal dispositivo di visualizzazione e ciò, congiuntamente alla rappresentazione del colore molto vicina a quella umana ad alto livello, lo ha reso standard privilegiato per i programmi di grafica e le applicazioni di image processing. Come si potrà vedere da successivi esempi di clusterizzazioni, sarà lo spazio che, in media, consentirà di raggiungere le segmentazioni a colori più vicine a quelle che verrebbero eseguite da un essere umano.

Tuttavia in questo spazio la componente L^* ha un peso maggiore delle altre e per tale ragione applicando una semplice distanza euclidea talvolta cluster di regioni molto chiare potrebbero venire inglobati in regioni più scure circostanti.

- **HSB**, infine, è uno spazio che punta come il precedente alla rappresentazione del colore per come viene interpretato ad un livello più alto dall'essere umano, e lo fa utilizzando come componenti: H il valore di *tinta*, S rappresentante la *saturazione* (ovvero distanza per una tinta dal corrispondente grigio) e B indicante la *luminosità* (nel senso di distanza verso il bianco o nero). Tinta e saturazione sono le componenti che determinano la cromaticità in questo spazio colore.

  Le affinità con lo spazio colore *L*a*b** si fermano qui, *HSB* non è predisposto per l'uniformità percettiva, è stato invece ideato per consentire un riconoscimento di oggetti più vicino a quello compibile da un essere umano e spesso le tre componenti non vengono processate contemporaneamente essendone sufficienti a volte due, a volte una sola. Verranno da noi utilizzate tutte e i risultati che si otterranno saranno talvolta paragonabili a quelli dello spazio *L*a*b**, altri ancora, il peso della componente saturazione influirà in maniera più spiccata creando regioni che difficilmente appariranno uniformi all'occhio umano.

  La conversione in questo spazio non passerà da XYZ, sar'a diretta a partire da RGB. Si definiscono inizialmente le componenti con valore massimo e minimo in RGB e differenza fra i due:

$$C_{max} = \max(R, G, B) \quad C_{min} = \min(R, G, B) \quad C_{rng} = C_{max} - C_{min}$$

  La componente di luminosità (B) in HSB è semplicemente calcolata come il valore massimo normalizzato in $[0, 1]$:

$$B_{HSB} = \frac{C_{max}}{255}$$

  La componente di saturazione deriva dalla distanza cromatica fra massimo e minimo:

$$S = \begin{cases} \frac{C_{rng}}{255} & \text{se } C_{max} > 0 \\ 0 & \text{altrimenti} \end{cases}$$

Per il calcolo della tinta quanto $C_{max} > 0$ (altrimenti non è definita) si computano prima delle componenti dei colori primari rinormalizzate:

$$R^{'} = \frac{C_{max} - R}{C_{rng}} \quad G^{'} = \frac{C_{max} - G}{C_{rng}} \quad B^{'} = \frac{C_{max} - B}{C_{rng}}$$

e otteniamo un valore preliminare per la tinta compreso in $[-1, 5]$:

$$H^{'} = \begin{cases} B^{'} - G^{'} & \text{se } R = C_{max} \\ R^{'} - B^{'} + 2 & \text{se } G = C_{max} \\ G^{'} - R^{'} + 4 & \text{se } B = C_{max} \end{cases}$$

infine da normalizzare in $[0, 1]$:

$$H = \frac{1}{6} \cdot \begin{cases} H^{'} + 6 & \text{se } H^{'} < 0 \\ H^{'} & \text{altrimenti} \end{cases}$$

Per la conversione inversa si procede a ritroso, de-computando la componente della tinta:

$$H^{'} = (6 \cdot H) \mod 6;$$

e calcolando dei valori intermedi per le componenti dei colori primari:

$$c_1 = \lfloor H^{'} \rfloor, \quad c_2 = H^{'} - c_1$$

$$x = (1 - S) \cdot V, \quad y = (1 - S \cdot c_2) \cdot V, \quad z = (1 - (S \cdot (1 - c_2))) \cdot V$$

utilizzati per calcolare le componenti dei valori primari normalizzati in relazione al valore di c_1:

$$(R^{'}, G^{'}, B^{'}) = \begin{cases} (V, z, x) & \text{se } c_1 = 0 \\ (y, V, x) & \text{se } c_1 = 1 \\ (x, V, z) & \text{se } c_1 = 2 \\ (x, y, V) & \text{se } c_1 = 3 \\ (z, x, V) & \text{se } c_1 = 4 \\ (V, x, y) & \text{se } c_1 = 5 \end{cases}$$

Per scalare queste componenti in $[0, 255]$ si applica:

$$R = min(round(256 \cdot R^{'}), 255)$$

$$G = min(round(256 \cdot G^{'}), 255)$$

$$B = min(round(256 \cdot B^{'}), 255)$$

Sebbene solo tre, le peculiarità di ciascun spazio colore sono di sicuro interesse. Il codice scritto, implementante le trasformate sopra descritte, risulta essere modulare e l'introduzione successiva di aggiuntivi spazi colore e relative conversioni non sarà impresa difficile.

Le possibili tipologie di immagini che il plugin ristrutturato accetterà sono: scala di grigi, colori in RGB o uno *stack* di dimensione arbitraria contenente immagini a livelli di grigio (per l'utilizzo di descrittori addizionali come spiegato nella sottosezione precedente). solo le immagini a colori potranno ovviamente subire una conversione in uno spazio colore differente.

Il compito di convertire ogni singolo pixel è stato affidato ad una nuova classe **ColorSpaceConversion** che mette a disposizione metodi di conversione da e verso lo spazio RGB. La responsabilità di convertire l'intera immagine è affidata alla classe KMeansManager, che ha anche il dovere di riconvertire la matrice contenente i centroidi ottenuta dall'esecuzione dell'algoritmo, qualora l'utente volesse visualizzare il risultato della clusterizzazione tramite i colori dei centroidi dei cluster a cui ogni pixel appartiene; questa modalità di visualizzazione, già presente nel plugin di partenza, è l'unica che richiede la conversione da uno spazio colore alternativo scelto dall'utente in RGB (L'elenco esaustivo di tutte le possibili modalità di visualizzazione e loro relativa spiegazione verrà fornita in una sottosezione successiva).

Di seguito una serie di immagini che mettono a confronto i diversi risultati di clusterizzazioni in spazi colore differenti ottenute tramite il plugin ristrutturato.

(a) Immagine originale

(b) Spazio colore: RGB

(c) Spazio colore: XYZ

(d) Spazio colore: L*a*b*

(e) Spazio colore: HSB

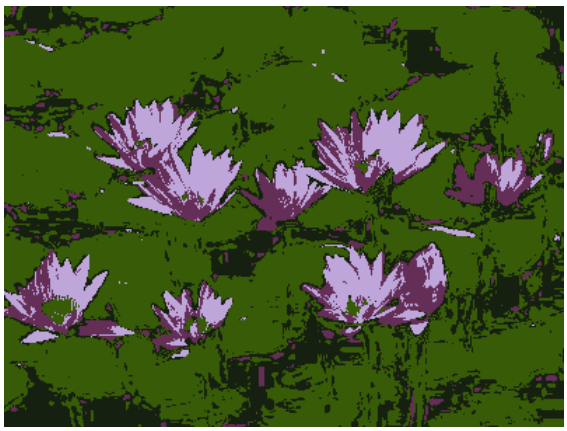
Figura 1: Immagine originale *(a)* elaborata in 4 cluster, con inizializzazione *K-Means++*, in differenti spazi colore in *(b)*, *(c)*, *(d)*, *(e)*.

(a) Immagine originale

(b) Spazio colore: RGB

(c) Spazio colore: XYZ

(d) Spazio colore: L*a*b*



(e) Spazio colore: HSB

Figura 2: Immagine originale *(a)* elaborata in 4 cluster, con inizializzazione *Forgy*, in differenti spazi colore in *(b)*, *(c)*, *(d)*, *(e)*.

Criteri di inizializzazione

Non solo la scelta dei descrittori, dello spazio colore, determina una particolare segmentazione dell'immagine in input, come già detto, l'inizializzazione delle matrici U e V può condurre a differenti minimi locali nell'algoritmo di Lloyd.

Se si considera che la matrice delle partizioni U può non essere esplicitata se non alla fine dell'algoritmo, il problema di scelta dell'inizializzazione si riduce a determinare un insieme iniziale di centroidi.

La maniera forse più semplice ed usata è stata teorizzata da Forgy in [7] e consiste nello scegliere i centroidi in maniera totalmente casuale fra i dati disponibili[15] in base al numero di cluster k scelto. Di certo una scelta che non garantisce una soluzione ottimale.

Fra i criteri alternativi per l'inizializzazione di K-Means [8], spicca **K-Means++** proposto in [9] nel 2007 per il cercare i centroidi seguendo la distribuzione di probabilità pesata dei quadrati delle distanze fra i punti e i centroidi incrementalmente scelti. Di seguito sono riassunti i passi dell'algoritmo di inizializzazione:

1. Scegli il primo centroide v_1 in maniera casuale fra i dati in X. Aggiungilo all'insieme dei centroidi V
2. Per ogni punto $x \in X$ calcola la distanza $D(x, v^*)$, dove $v^* \in V$ è il centroide più vicino al punto fra i centroidi già calcolati
3. Estrai a sorte un nuovo centroide v_i fra i punti x (non ancora scelti) seguendo la distribuzione di probabilità $\frac{D(x,v^*)^2}{\sum_{i=1}^{N} D(x_i,v^*)^2}$
4. Se $|V| < k$ torna al passo 2, altrimenti esegui K-Means con V come insieme dei centroidi

L'algoritmo è ancora, in parte, suscettibile a variazioni dovute alla estrazione casuale del primo centroide fra i dati con distribuzione di probabilità uniforme, tuttavia riesce a migliorare la scelta dei successivi punti considerando la distribuzione pesata delle distanze e prendendo, in definitiva, in considerazione i pixel più distanti dagli altri cluster pesando però tale distanza con la somma di tutte le distanze ed elevandola al quadrato risultando molto meno suscettibile alla presenza di outliers[16].

Il plugin ristrutturato implementa entrambi i criteri di inizializzazione e sebbene talvolta i risultati siano poco dissimili, altre l'inizializzazione ottenuta tramite K-Means++

[15] Una scelta di questo tipo comporta la certezza che alla prima iterata vi saranno distanze fra dati e centroidi pari a zero. Nessun problema per l'algoritmo di Lloyd, ma una possibile fonte di errori numerici per Fuzzy C-Means, come verrà spiegato nelle seguenti sezioni

[16] Considerare semplicemente la distanza indurrebbe alla scelta dei punti più distanti fra loro, in presenza di dati con outlier, sarebbero questi ultimi ad essere sempre scelti

sembra risultare migliore, altre ancora i criteri paiono far convergere l'algoritmo allo stesso minimo locale. Immagini clusterizzate utilizzando entrambi i criteri verrano di seguito mostrate utilizzando il criterio di visualizzazione dei colori dei centroidi, come esempio qualitativo di comparazione fra i criteri su un caso base.

(a) Immagine originale (b) K-Means++(seed: 48)

(c) Forgy (seed: 5) (d) Forgy (seed: 481)

Figura 3: Immagine originale *(a)* elaborata in 4 cluster, spazio colore RGB, con diversi criteri di inizializzazione (*K-Means++*, Forgy) e differenti seed di randomizzazione in *(b)*, *(c)*, *(d)*.

Modalità di visualizzazione

Le molteplici possibilità di visualizzazione di una immagine clusterizzata in base al colore sono state implementate per fornire un rapido strumento valutativo della bontà dell'algoritmo eseguito e della scelta dei parametri fatta. Ognuna presenta dei particolari vantaggi, trasformabili in svantaggi in determinate condizioni.

Le modalità presenti nella versione ristrutturata del plugin K-Means sono quattro:

- **Visualizzazione in scala di grigi** già presente nel plugin originale, colora i cluster in toni di grigio la cui luminosità viene adattata in relazione al numero di clusters. Pe run numero limitato di cluster consente di individuare con precisione i differenti cluster. Fa perdere i colori dell'immagine originale.

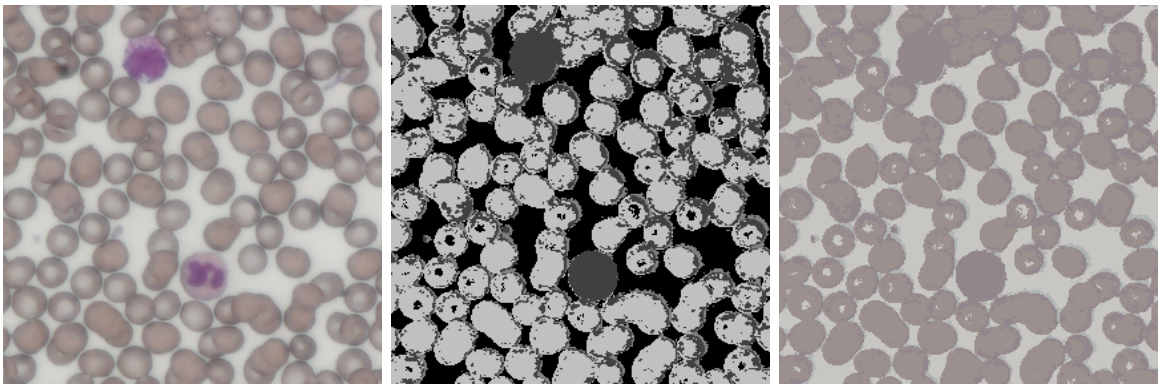
- **Visualizzazione dei colori dei centroidi** già presente nel plugin originale, colora ogni pixel con il colore del centroide del cluster di appartenenza. Qualora venga scelta una conversione in uno spazio colore diverso da RGB viene eseguita una conversione inversa dei colori dei centroidi per consentire la visualizzazione a video dell'immagine. Consente di visualizzare facilmente i cluster con colorazioni ben contrastate, in presenza di colori molto vicini o di *misclustering* la visualizzazione delle diverse regioni diventa più complicata e i colori tendono mediamente verso i grigi.

- **Visualizzazione tramite colori random** Assegna ad ogni cluster una colorazione casuale. Quando la colorazione con il colore dei centroidi non è di particolare aiuto, questo tipo di visualizzazione può invece consentire una facile visualizzazione delle aree distinti o comuni.

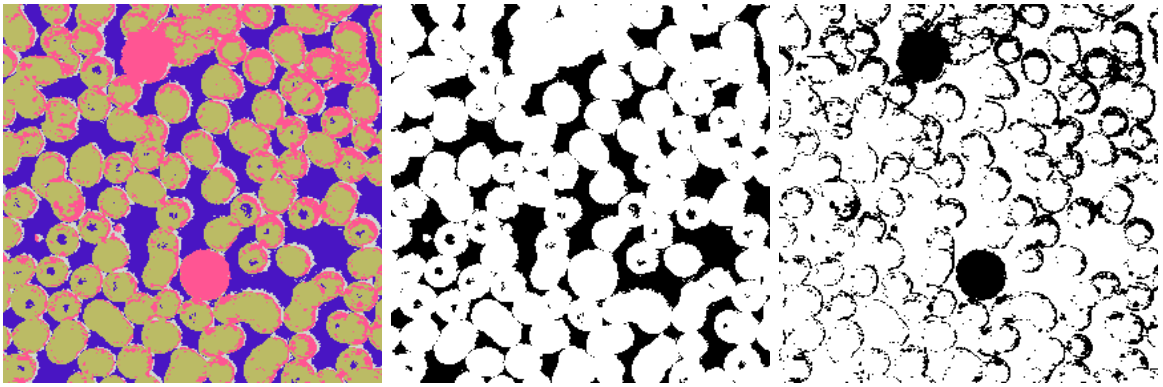
- **Visualizzatione tramite stack binario** Impiega uno stack di immagini binarie, una per ogni cluster, rappresentanti l'appartenenza di ciascun pixel ad un cluster. È la traduzione visuale della matrice di partizionamento U. Consente una grande precisione nella individuazione dell'appartenenza di una zona ad un cluster piottosto che ad un altro. In presenza di molti cluster il riconoscimento può diventare complicato

Il calcolo addizionale che veniva compiuto per ulteriori visualizzazioni di uno stesso risultato nel plugin originario non è più necessario grazie alla introduzione del calcolo della matrice di partizionamento U e della ristrutturazione del codice, l'algoritmo e il calcolo dei cluster più vicini per ogni pixel vengono eseguiti solo una volta, mentre tutte le modalità di visualizzazione potranno essere richiamate in seguito sull'unico output e fornire rappresentazioni differenti dell'immagine clusterizzata.

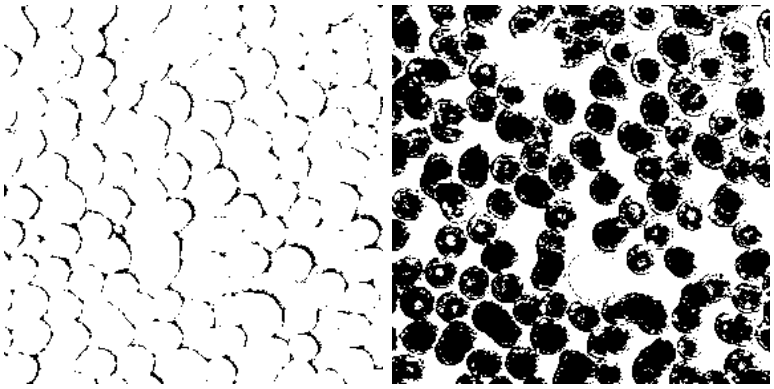
Ancora una volta seguono immagini dimostrative, comparano i diversi metodi di visualizzazioni su una stessa immagine clusterizzata una sola volta con una unica configurazione di parametri.



(a) Immagine originale (b) Scala di grigi (c) Colore centroidi



(d) Colori random (e) Stack binario (primo cluster) (f) Stack binario (secondo cluster)



(g) Stack binario (terzo cluster) (h) Stack binario (quarto cluster)

Figura 4: Immagine originale *(a)* elaborata in 4 cluster, spazio colore HSB, inizializzazione K-Means++ e differenti visualizzazioni in *(b)*, *(c)*, *(d)*, *(e)*, *(f)*, *(g)*, *(h)*.

Fuzzy C-Means (*FCM*)

Fuzzy C-Means è l'estensione fuzzy del classico K-Means, è stato introdotto da J. C. Dunn nel 1973[10] e successivamente migliorato da J. C. Bezdek nel 1981[11].

Esso prevede la modifica del codominio della funzione matriciale U: se precedentemente i suoi elementi u_{ij} assumevo valori 0 o 1, ora avranno valori reali nell'intervallo $[0, 1]$. Tali valori rappresentano il grado di appartenenza, detto anche *membership*, di un dato ad un cluster. Inoltre tali valori devono rispettare i seguenti vincoli:

$$\sum_{j=1}^{k} u_{ij} = 1$$

ovvero la somma di tutte le membership di un dato rispetto a tutti i cluster deve essere uguale a 1, e

$$0 < \sum_{i=1}^{N} u_{ik} < n$$

la somma delle membership dei dati di un cluster deve essere strettamente compresa fra 0 e il numero dei dati.

Questa modifica si riverbera sulla formula della funzione da minimizzare:

$$J(\mathbf{U}, \mathbf{V}) = \sum_{i=1}^{N} \sum_{j=1}^{k} u_{ij}^{m} ||\mathbf{x_i} - \mathbf{v_j}||^2 \tag{4}$$

In essa è anche aggiunto un esponente $m \in (1, \infty)$ che pesa i valori delle membership di U.

L'algoritmo iterativo utilizzato per minimizzare questa funzione è molto simile a quello utilizzato per K-Means, mentre i calcoli da effettuare per ottenere i nuovi valori di U e di V ad ogni iterata cambiano.

Si introduce m nella regola di aggiornamento di V, la cui formula diviene:

$$\mathbf{v_j} = \frac{\sum_{i=1}^{N} u_{ij}^{m} \mathbf{x_i}}{\sum_{i=1}^{N} u_{ij}^{m}} \tag{5}$$

Si modifica l'espressione del calcolo degli elementi di U :

$$u_{ij} = \frac{1}{\sum_{c=1}^{k} \left(\frac{||\mathbf{x_i} - \mathbf{v_j}||}{||\mathbf{x_i} - \mathbf{v_c}||} \right)^{(2/m-1)}} \tag{6}$$

Ispezionando la formula notiamo che u_{ij} in qualche modo pesa la distanza fra x_i e v_j.

Una nota di carattere numerico

Nel momento in cui il numeratore del denominatore dell'equazione (6), rappresentante la distanza del dato dal cluster diventa uguale a 0, ovvero, il dato $i-esimo$ coincide con il $j-esimo$ cluster[17], si possono avere problemi di natura numerica nel calcolo della formula: avremmo $[\frac{0}{0}]$ che è una forma indeterminata, essa condurrebbe alla generazione di **Nan**.

In letteratura le implementazioni classiche non tengono conto di questo caso, sebbene venga esaminato da J. C. Bezdek stesso in [12]; si può far fronte al problema assegnando come valore di membership 1 in corrispondenza del dato e del cluster la cui distanza sia 0, e 0 a tutte le altre membership dello stesso dato. Qualora più cluster avessero distanza 0 da uno stesso dato, assegnamo un valore di membership a ciascuno di essi pari a $1/r$ dove r è il numero di questi cluster.

Equivalenza fra Fuzzy C-Means e K-Means

È possibile dimostrare che Fuzzy C-Means, sotto determinate condizioni, si comporta in maniera uguale a K-Means. Si dimostra che le equazioni (5) e (6), per $m \to 1$, sono equivalenti alle equazioni (2) e (3) presentate precedentemente nella sezione dell'algoritmo di Lloyd.

Considerando l'espressione (6) notiamo che il valore di membership può essere interpretato come un valore inversamente proporzionale al quadrato della distanza fra il dato e il cluster correnti:

$$u_{ij} = \frac{1}{||x_i - v_j||^2}$$

se si divide questa distanza per la somma delle distanze fra il dato e tutti i centroidi, la si normalizza tra 0 e 1. Si veda in ultimo l'esponente m che dà un peso alla distanze, cosicchè i valori piccoli divengano ancora pi ù piccoli e quindi meno influenti. Per $m \to 1$, l'approssimazione è diviene quella della equazione (2), per $m = 2$ invece la normalizzazione delle membership è lineare[18].

Ora considerando l'espressione (5), per $m \to 1$ osserviamo facilmente che l'espressione (5) risulta equivalente alla (3).

FCM Plugin: Refactoring

Similarmente a quanto fatto per il plugin precente, si è riusato quanto implementato per K-Means per realizzare un altro plugin per ImageJ basato su Fuzzy C-Means per come è stato presentato finora.

[17] Questo è sicuramente vero quando la lontananza viene misurata con una distanza, potrebbe non essere verificato utilizzando una misura di dissimilarità

[18] Questo valore è anche il più usato in letteratura in assenza di particolare conoscenza di dominio

Come promesso qualche sezione fa, l'architettura generale è rimasta intaccata, un refactoring superficiale ha portato alla ridenominazione delle tre classi principali, ora divenute: **FCM**, **FCMPlugin**, **FCMManager**. Le classi responsabili della configurazione dei parametri dell'algoritmo e della loro inizializzazione per opera dell'utente hanno dovuto necessariamente tener conto della introduzione dei nuovi parametri e variabili[19] (partendo da m per giungere ai nuovi criteri di terminazione).

La classe FCM, che incapsula Fuzzy C-Means, ha subito le maggiori modifiche in termini di logica interna, i vari step dell'algoritmo sono stati ulteriormente raffinati in metodi separati per aumentarne la leggibilità e comprensibilità ed ovviamente le funzioni delegate all'aggiornamento della matrice delle membership e di quella dei centroidi dei cluster sono state riscritte per implementare le regole (6) e (5). Proprio a proposito della riscrittura di questi metodi va spesa una qualche riga come complemento alla documentazione che accompagna il codice; numerose ottimizzazioni sono state fatte con lo scopo di migliorare i tempi di esecuzione di ogni singola iterata dell'algoritmo[20]. Per ottenere questo risultato si è volentieri deciso di sacrificare maggiore memoria per conservare i risultati di operazioni che altrimenti sarebbero state ripetute: il computo delle distanze fra ogni dato e ogni centroide ad esempio, il calcolo dei valori di ogni elemento di U elevato ad m. Ancora, si è cercato di precomputare la parti invarianti ai cicli (nella espressione (6) il numeratore della frazione al denominatore è indipendente dalla sommatoria e la somma degli inversi delle distanze da tutti i dati può essere calcolata una unica volta per ciascun dato). In altre implementazioni, prese da noi come confronto, questo tipo di ottimizzazioni del codice non risulta presente, ma nonostante lo svantaggio causato dal linguaggio utilizzato[21] si è riusciti ad ottenere una versione del plugin a nostro avviso più che competitiva.

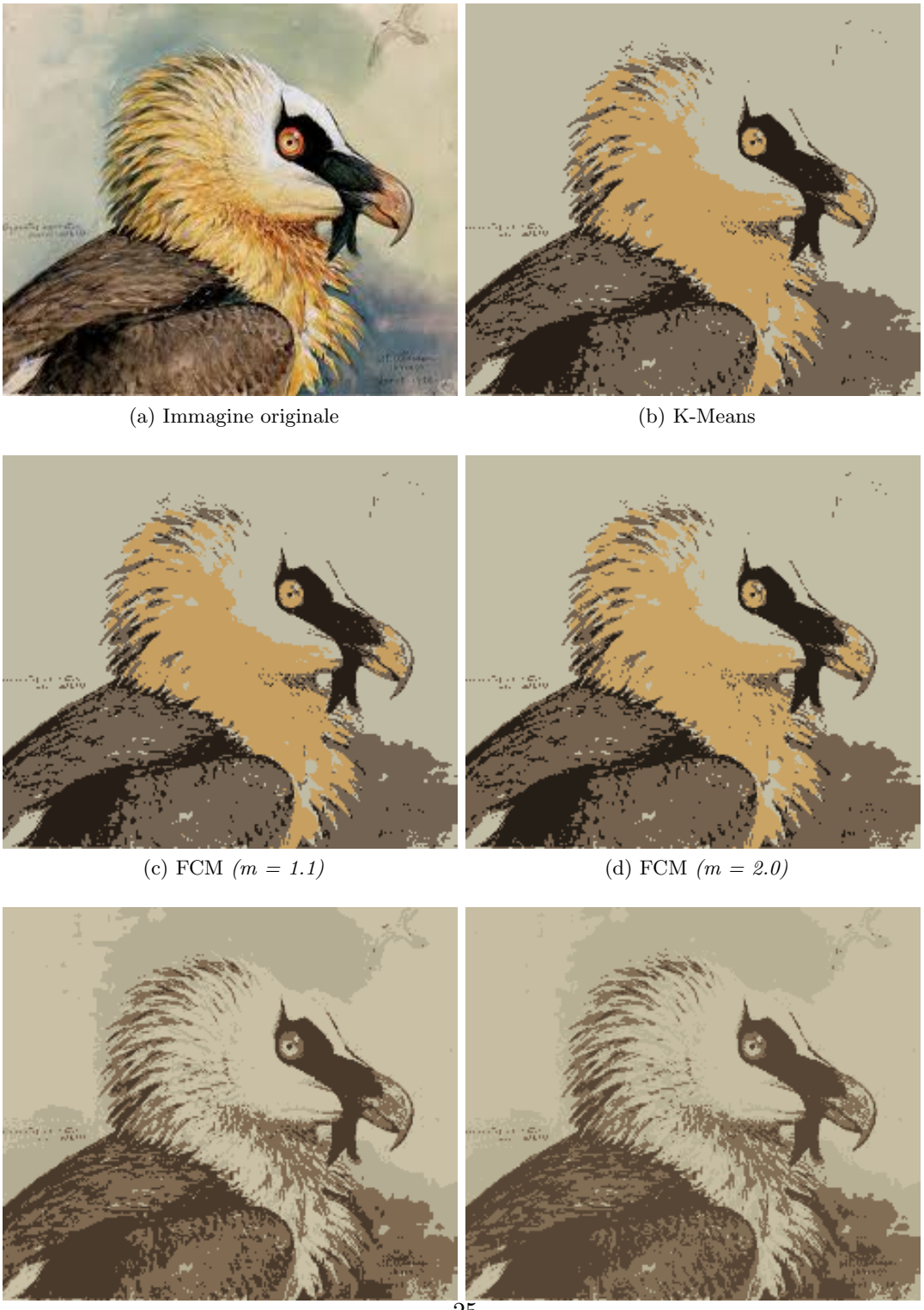
Nel lavorare a questa seconda ristrutturazione e riscrittura del codice, ci si è resi conto che rimanevano alcuni *punti di decisione* circa l'implementazione dell'algoritmo, le cui strade (implementazioni) alternative potevano essere percorse (realizzate) ed un eventuale cambiamento del risultato valutato. Nelle sezioni a seguire si tratterà per l'appunto di queste ulteriori estensioni: un nuovo criterio di inizializzazione, la scelta del criterio di terminazione ed una nuova modalità di visualizzazione.

Di seguito una breve rassegna di esempi di immagini segmentate utilizzando il nuovo plugin al variare del parametro di fuzzyness, e in contrapposizione con la clusterizzazione di K-Means (quando si userà un valore di m pari a 1.1 si cercherà proprio di ottenere una approssimazione di K-Means).

[19] Guardando alle interfacce grafiche dei plugin prodotti, partendo da quello originario, passando, nell'ordine, per K-Means, Fuzzy C-Means e la sua versione spaziale, si nota un aumento notevole di widget e selettori che consentono la parametrizzazione di quasi tutti le variabili di decisione associati agli algoritmi.

[20] La differenza fra K-Means e Fuzzy C-Means, quantomeno nelle nostre implementazioni, è notevole e dovuta al numero di operazioni maggiori da eseguire (il calcolo della matrice U non può essere implicito) e alla maggiore complessità di queste (il numero di esponenziazioni e moltiplicazioni non è di poco conto).

[21] *Java* è notoriamente svantaggiato, seppur di poco in alcune operazioni, rispetto a linguaggi di più basso livello come *C* e *C++*



(a) Immagine originale

(b) K-Means

(c) FCM *(m = 1.1)*

(d) FCM *(m = 2.0)*

(e) FCM *(m = 5.0)*

(f) FCM *(m = 10.0)*

Figura 5: Immagine originale *(a)* elaborata in 4 cluster con Fuzzy C-Means, spazio colore *L*a*b**, inizializzazione K-Means++ e differenti visualizzazioni in *(c)*, *(d)*, *(e)*, *(f)* al variare di m; l'immagine in *(b)* è stata clusterizzata con K-Means in 4 cluster, spazio colore *L*a*b** e inizializzazione K-Means++ a partire da *(a)*.

Criteri di inizializzazione

Nella scelta della inizializzazione di K-Means si era visto come partire da una matrice dei centroidi costruita in maniera random più o meno ragionata. La scelta dei centroidi iniziali viene fatta fra i vari punti nello spazio dei dati disponibili e questo, come visto, può portare ad inconvenienti di tipo numerico.

La stragrande maggioranza delle implementazioni di Fuzzy C-Means non ha a che fare con questi inconvenienti, e la ragione per cui ne risulta essere immune è che adotta un criterio di inizializzazione differente[22]: procede con la generazione casuale della matrice delle membership U, producendo inizialmente ogni suo valore in maniera random uniforme in $[0, 1]$ e successivamente normalizzando ciascun valore per la somma dei valori di membership di tutti i cluster per ogni dato. Si noti che tale modalità rende molto remota la probabilità che un cluster (successivamente calcolato con la regola (5) di update della matrice V) risulti uguale ad uno dei dati in partenza ed allo stesso modo fa arbitrariamente allontanare l'algoritmo da un possibile risultato ottimale (richiedendo dunque, nel caso peggiore, un numero maggiore di iterate).

Per dovere di completezze abbiamo implementato anche questo ulteriore criterio di inizializzazione nel nuovo plugin, le esecuzioni effettuate come test non hanno segnalato alcun tipo di miglioramento nella segmentazioni di immagini campione.

Criteri di terminazione

La molteplicità dei criteri per la terminazione dell'algoritmo iterativo presenti in letteratura ha sollevato la questione circa quale fosse quello da implementare.

Solitamente, ad ogni iterata viene osservato il comportamento dell'algoritmo (la sua convergenza) in relazione ai cambiamenti avvenuti o sulla matrice delle membership U, o sulla matrice dei centroidi dei cluster V o sul valore della funzione obiettivo J rispetto l'iterata precedente, se questi cambiamenti sono inferiori ad una certa piccola soglia ci si ferma. Ma come fare a calcolare il cambiamento per le matrici? e che criterio incorporare nell'algoritmo?

Per cercare una risposta a queste domande si è creato un test[23] per raccogliere dati ottenuti, con varie configurazioni dell'algoritmo eseguito su un set di immagini, e riferiti alle seguenti variabili osservate:

- differenza fra i valori della funzione obiettivo $|J^{(n+1)} - J^{(n)}|$
- norma di Frobenius della differenza della matrice dei centroidi $||V^{(n+1)} - V^{(n)}||_F$

[22] Il nostro tentativo di adattare un criterio di inizializzazione pensato per K-Means, *K-Means++* a Fuzzy C-Means, sebbene abbia richiesto uno sforzo implementativo maggiore ci ricompensa con un numero inferiore di interate il più delel volte.

[23] Si è provveduto a creare la classe TestManager con le responsabilità di eseguire ripetutamente e per varie configurazioni di parametri l'algoritmo e di loggarne i risultati su file per poi poterli analizzare.

- norma max della differenza della matrice dei centroidi $||V^{(n+1)} - V^{(n)}||_{max}$
- norma di Frobenius della differenza della matrice delle membership $||U^{(n+1)} - U^{(n)}||_F$
- norma max della differenza della matrice delle membership $||U^{(n+1)} - U^{(n)}||_{max}$

Dopo aver collezionato un congruo numero di esecuzioni e di dati, questi ultimi sono stati importati nel software di analisi R tramite il quale sono stati plottati gli andamenti delle variabili lungo il numero di iterate (400) eseguite dall'algoritmo e si sono cercate delle regole riassuntive per il loro comportamento[24].

Risulta evidente che il comportamento delle differenze della funzione obiettivo in determinate situazioni (quando ad esempio il numero di cluster fornito all'algoritmo eccede il numero di cluster sottostanti ai dati) è più caotico rispetto alle altre misure osservate e non monotono descrente, inoltre potrebbe arbitrariamente allontanarsi dal valore 0 risultando quindi la misura meno affidabile sulla quale stabilire un valore soglia. In generale la norma di Frobenius si è dimostrata molto più suscettibile alle dimensioni delle matrici sulla quale veniva calcolata piuttosto che la norma max, quando l'immagine da clusterizzare aumentava di dimensioni, cresceva notevolmente il numero dei pixel e di conseguenza il numero delle righe della matrice U (mentre V rimaneva inalterata) e il valore della norma di Frobenius su U risultava anch'esso crescere di conseguenza (rimanendo invece stazionario rispetto alla V). La norma max, invece, ha un valore che prescinde dal numero di elementi nella matrice (questa conclusione è avvalorata dalla formula), ma il suo valore è comunque in parte dipendente all'aumento del numero dei cluster, se aumentano i cluster, il valore medio di una membership diminuirà e la differenza massima possibile fra due membership da una iterata alla successiva sarà più piccolo.

Alla luce delle considerazioni di cui sopra, si è scelto di utilizzare come criterio di default quello che computa la norma max sulla matrice U, ma, considerando che il computo su V risulta essere molto più efficiente e che per alcuni domini, e che variazioni minime sul valore dei centroidi sono spesso ininfluenti in determinati domini[25], e che lo stesso Bezdek in [12] esprimeva perplessità circa la scelta di un criterio su U o V si è infine deciso di offrire la scelta del criterio all'utente come ulteriore estensione parametrica dell'algoritmo (ponendo però un numero massimo limite di iterate dovessero il criterio e la soglia introdotta non far convergere l'algoritmo per uno specifico set di valori in input).

Modalità di visualizzazione

Tutti i metodi di visualizzazione dei cluster implementati nella versione precedente del plugin sono inizialmente inutilizzabili con Fuzzy C-Means e la ragione risiede nel fatto

[24] I dati raccolti dai test in file CSV e i relativi grafici sono allegati al resto della documentazione

[25] Si consideri ad esempio una immagine i cui pixel sono rappresentati da valori RGB interi compresi tra 0 e 255, allora una soglia di 0.0001 posta con il criterio di norma sulla matrice V risulta sproporzionatamente piccola

che tutti necessitano di conoscere, per ogni pixel il cluster di appartenenza in maniera tale da assegnarli una particolare colorazione, Fuzzy C-Means fornisce, invece, al termine dell'esecuzione, la matrice di membership che rappresenta il grado di appartenenza di ogni pixel ad ogni cluster con un valore fuzzy e non crisp. Per poter essere riutilizzati anche in questo plugin diventa necessario *defuzzificare* la matrice delle membership U per stabilire un unico cluster di appartenenza a pixel. Si è scelto di farlo tramite la funzione max, essa resituirà per ogni pixel il cluster con membership maggiore fra tutti.

Una operazione di questo tipo potrebbe far pensare alla inutilità dell'implementare Fuzzy C-Means come metodo di segmentazione, poiché il risultato di appartenenza fuzzy a più di un cluster, peculiarità dell'algoritmo viene brutalmente persa[26] al termine una volta che diviene necessario visualizzare i risultati.

Le matrici in output all'algoritmo sono spesso utilizzate come input per altri sistemi, i quali sono in grado di utilizzare l'informazione fuzzy in esse contenuta. Per rendere maggiormente *fruibile* e più facilmente *visibile* il lavoro svolto dal nostro plugin, è parso necessario realizzare un ulteriore metodo di visualizzazione che riesca a rappresentare direttamente l'output fuzzy senza ricorrere ad una defuzzificazione. La risultate di questo sforzo è una ulteriore estensione del progetto, il metodo di visualizzazione a **Membership Stack**.

Esso prevede la creazione di uno stack di immagini in scala di grigio contenente tante slice quanti sono i cluster in input all'algoritmo e in cui ogni slice rappresenta una fetta della matrice delle membership U poiché ogni suo pixel avrà come valore di luminosità il proprio valore di appartenenza al dato cluster[27].

Con una visualizzazione di questo tipo, esattamente speculare a quella con Stack Binari da noi presentata per l'hard clustering, si può percepire la *variabilità* dell'appartenenza di ogni pixel e la *profondità* della clusterizzazione fuzzy. Pixel con gradi di appartenenza ai cluster molto vicini fra loro avranno la stessa luminosità di grigio in ogni slice e l'immagine totale passerà da una rappresentazione a stack quasi binari per valori di m prossimi a uno a stack sempre più uniformi in colorazione per valori maggiori.Saranno inoltre più facilmente interpretabili quei risultati di immagini clusterizzate con differenti valori del parametro di fuzzyness m ma per le quali le immagini sintetizzate per visualizzare la segmentazione a colori non mostrano grandi differenze.

[26] In una situazione in cui un pixel viene assegnato, al termine della computazione, con eguale grado di incertezza a tutti i sette cluster cercati, i valori delle singole membership differiranno di molto poco fra loro. Applicare una defuzzificazione estraendo il cluster con valore di appartenenza di pochissimo maggiore snaturerà il risultato iniziale e appiattirà tutti i risultati diversi per i valori di membership assegnati, ma che ottengono lo stesso cluster come maggioritario, nella stessa immagine di output.

[27] La corrispondenza fra luminosità del pixel nella slice e valore di membership non è immediata e vale la pena spendere un po' di parole a riguardo, il valore di membership viene inizialmente invertito sottraendolo a 1 in modo tale che valori più scuri rappresentino una appartenenza maggiore al cluster, successivamente il valore ottenuto viene scalato fra 0 e 255 per consentire la visualizzazione in toni di grigio

A riprova di quanto detto, adesso si forniscono i Membership Stack per l'immagine precedentemente clusterizzata con Fuzzy C-Means nella figura 5 (la visualizzazione precedente e quella corrente sono relative agli stessi parametri di input). Si lascia ancora una volta al lettore di questa documentazione il compito di esprimere una valutazione.

(a) FCM $m = 1.1$, cluster 1 (b) FCM $m = 1.1$, cluster 2 (c) FCM $m = 1.1$, cluster 3 (d) FCM $m = 1.1$,cluster 4

(e) FCM $m = 2.0$, cluster 1 (f) FCM $m = 2.0$, cluster 2 (g) FCM $m = 2.0$, cluster 3 (h) FCM $m = 2.0$, cluster 4

(i) FCM $m = 5.0$, cluster 1 (j) FCM $m = 5.0$, cluster 2 (k) FCM $m = 5.0$, cluster 3 (l) FCM $m = 5.0$, cluster 4

(m) FCM $m = 10.0$, cluster 1 (n) FCM $m = 10.0$, cluster 2 (o) FCM $m = 10.0$, cluster 3 (p) FCM $m = 10.0$, cluster 4

Figura 6: Fuzzy Stack da *(a)* a *(p)* dei risultati della clusterizzazione con Fuzzy C-Means della immagine *(a)* comparsa in figura 5

Spatial Fuzzy C-Means (*SFCM*)

Si evince dalla letteratura che l'algoritmo FCM, come anche K-Means, produce risultati non soddisfacenti quando vengono forniti dati affetti da rumore: l'algoritmo assocerà ad ogni dato rumoroso un cluster diverso da quello che ci si aspetterebbe vedendo il cluster assegnato ai dati ad esso *spazialmente* vicini.

Inoltre, con particolare riferimento alle immagini e al task di segmentazione, il risultato del clustering può essere una immagine fortemente frammentata, con zone di colore poco omogenee, come invece ci si aspetterebbe.

Per risolvere questo tipo di problematiche, Chuang, Tzeng, Chen, Wu hanno proposto in citepsfcm una variante del Fuzzy C-Means: **Spatial Fuzzy C-Means**, che tiene conto dell'informazione spaziale associata all'intorno di ogni pixel di una immagine.

SFCM modifica la membership di ciascun pixel rispetto ad un cluster, considerando la membership dei pixel vicini ad esso. Si considera un intorno (*neighborhood*, *NB*), ovvero una finestra quadrata di raggio fissato, centrata nel pixel, per calcolare l'informazione spaziale sottoforma di una funzione h.

$$h_{ij} = \sum_{c \in NB(x_i)} u_{cj} \tag{7}$$

L'informazione contenuta in h_{ij} riguarda la probabilità che il pixel $i-esimo$ appartenga al cluster $j-esimo$ e può essere stimata come la somma delle membership dei vicini del pixel $i-esimo$, rispetto al cluster $j-esimo$. Più pixel del vicinato appartengono allo stesso cluster, maggiore sarà il il valore di h_{ij}.

In SFCM viene modificata la regola di aggiornamento della funzione di membership (6), per ospitare l'informazione spaziale, diventando:

$$u_{ij} = \frac{u_{ij}^p h_{ij}^q}{\sum_{c=1}^{k} u_{ic}^p h_{ic}^q} \tag{8}$$

Per introdurre più peso verso la funzione di membership o più peso verso la funzione spaziale, sono utilizzati due esponenti: p e q, due reali positivi che svolgono un ruolo similare a quello di m in FCM.

Equivalenza fra Spatial Fuzzy C-Means e Fuzzy C-Means

È dimostrabile che Spatial Fuzzy C-Means è equivalente a Fuzzy C-Means per determinati valori di p e q, infatti, considerando l'espressione (8) e assegnando alla variabile p valore 1 e alla variabile q valore 0, si ottiene la regola di aggiornamento (6) di FCM, poichè il valore della funzione spaziale si riduce a 1 e quindi è ininfluente nel prodotto e il valore della memebrship rimane invariato se elevato all'unità e nuovamente normalizzato.

Scelta della funzione spaziale

Una delle possibili parametrizzazioni che è da noi è stata considerata, per l'algoritmo SFCM, è quella della scelta del tipo di funzione spaziale da utilizzare nel passo di update della matrice delle membership.

Prevediamo, infatti, come scelta alternativa alla funzione h_{ij} definita in (7), una funzione spaziale sempre presentata in [3].

Essa è una variante dellaprima che però considera meno il livello di fuzzyness e che causa una maggiore perdita di dettaglio nella segmentazione:

$$h_{ij} = \sum_{c \in NB(x_i)} g_{cj} \tag{9}$$

dove

$$g_{cj} = \begin{cases} 1 & \text{se} u_{cl} < u_{cj} \text{ per } l = 1 \dots k \\ 0 & \text{altrimenti} \end{cases} \tag{10}$$

Computare questo tipo di funzione spaziale vuol dire *defuzzificare* momentaneamente la matrice di membership U ottenendo una matrice di partizioni aventi come valori 1 o 0 in base al fatto che il cluster $j - esimo$ abbia membership maggiore fra tutti gli altri cluster per il pixel $c - esimo$.

SFCM Plugin: Refactoring

La fase di progettazione e ristrutturazione del plugin FCM in SFCM ha visto una nuova ridenominazione delle classi principali del plugin in **SFCM**, **SFCMManager** e **SFCMPlugin** che mantengono le funzionalità e responsabilità assegnate alle corrispettive controparti dei plugin precedenti. Le modifiche da introdurre nella nuova versione, benchè in numero maggiore, sconvolgeranno ancora meno il codice esistente rispetto a quanto fatto nelle due ristrutturazioni precedenti, essendo Spatial Fuzzy C-Means una generalizzazione molto poco forzata di Fuzzy C-Means.

I nuovi parametri introdotti sono relativi ai pesi della funzione di membership e della funzione spaziale, p e q e del raggio della finestra, r, necessaria per il computo della funzione spaziale; essi dovranno essere aggiunti nella interfaccia utente, colelgati alla configurazione dell'algoritmo e incorporati nei metodi che implementano l'algoritmo in SFCM. L'ulteriore parametrizzazione implementata come estensione riguarda la scelta del tipo di funzione spaziale (l'alternativa proposta è stata presentata nella sottosezione precedente).

Concretamente, le modifiche in SFCM hanno coinvolto la riscrittura del metodo che eseguiva direttamente il clustering e l'aggiunta di metodi per l'aggiornamento della funzione di membership in accordo con la (8). La scrittura di un singolo metodo aveva portato ad un algoritmo estremamente inefficiente poiché costretto al ricomputo della funzione spaziale

calcolata su una finestra per *ogni singolo* pixel. All'aumentare del raggio della finestra il tempo di esecuzione si allungava paurosamente. Si è dunque deciso di provvedere ad una ottimizzazione del codice e lo si è fatto percorrendo due vie[28]:

- la prima consiste nell'utilizzare una coda che conservi nei sui elementi le somme per colonne degli elementi contenuti nella finestra centrata sul pixel corrente, muovendosi lungo le colonne sarà sufficiente espellere il primo in coda ed aggiungere un nuovo elemento in coda dopo aver computato unicamente la nuova colonna della finestra; per ogni riga la coda per le prime colonne dovrà comunque essere ricomputata *ex nihilo*. Approssimativamente si computano solo $O(r)$ somme per finestra in luogo di $O(r^2)$.

- la seconda ottimizzazione prevede il computo di una *immagine integrale*[13] a partire da ogni colonna della matrice delle membership U. Questa immagine risiederà in memoria ed avrà dimensioni pari a quelle di U, ma questa memoria aggiuntiva verrà utilizzata per computare la somma di una finestra di arbitrario raggio r in soli quattro accessi a questa matrice.

Per entrambe le soluzioni, volendo computare la versione alternativa della funzione spaziale per come presentata in (9), diviene necessario precomputare in precedenza una matrice G di interi che rappresenti la defuzzificazione di U tramite l'utilizzo dell'operatore di massimo applicato a tutte le membership per ogni pixel.

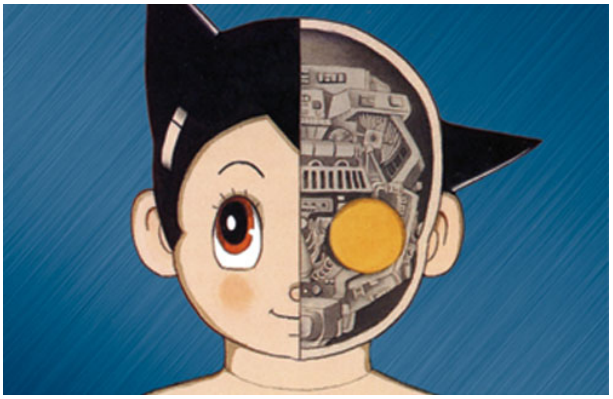
L'implementazione attuale vede la scelta della prima soluzione proposta, anche se apparentemente avere un vantaggio solo per colonne, comunque proporzionale linearmente al raggio della finestra, pare molto più inefficiente di poter effettuare solo quattro accessi in memoria a rescindere da r. La realtà dei fatti ha dimostrato che la costruzione della immagine iniziale si rivela essere particolarmente dispendiosa e il computo di tante immagini integrali, una per ogni cluster, comporta un tempo maggiore che il calcolo per colonne per valori ragionevoli del raggio r[29].

Calcolare la funzione spaziale utilizzando l'espressione alternativa (9) può condurre, in casi particolari, all'azzeramento di una intera colonna di U e quindi ad un errore numerico che propagandosi può arrivare alla generazione di NaN. Per evitare ciò abbiamo posto un controllo che in caso di azzeramento imponga un valore minimo di correzione (10^{-9}) che pare non alterare le corrette esecuzioni dell'algoritmo.

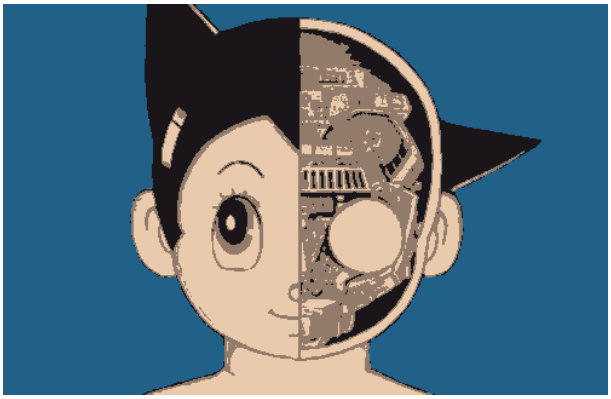
Di seguito una serie di immagini segmentate utilizzando SFCM al variare dei parametri p, q, r e della funzione spaziale.

[28] Solo una delle due, la prima è stata infine collegata al resto del codice e funziona con il plugin consegnato, una parametrizzazione anche di questo punto di scelta ci è sembrato eccessivo. Tuttavia la seconda alternativa è lo stesso presente nel codice presentato in allegato.

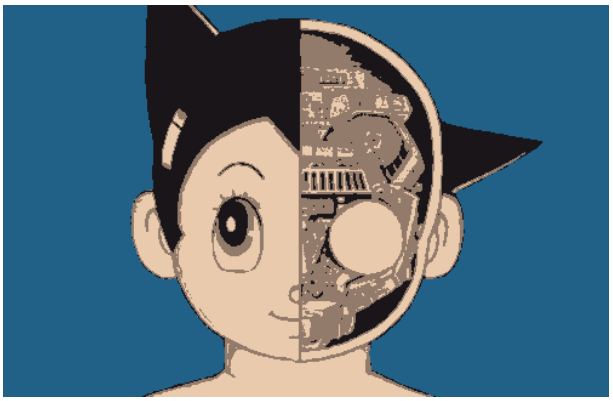
[29] All'aumentare di r il tempo impiegato dal metodo ottimizzato tramite immagine integrale rimane effettivamente costante, mentre la prima implementazione utilizza un tempo crescente. Abbiamo stimato che per $r \leq 20$ sia più conveniente utilizzare la prima proposta di ottimizzazione.



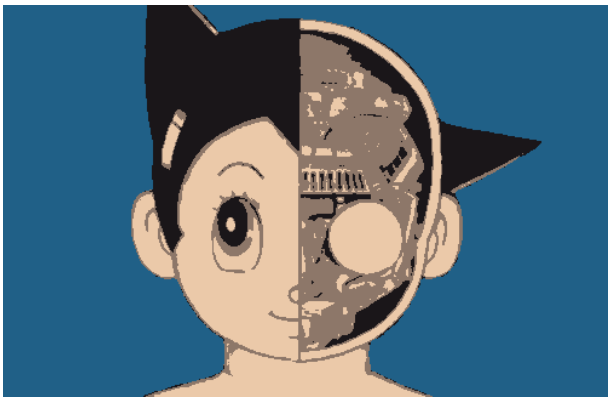
(a) Immagine originale



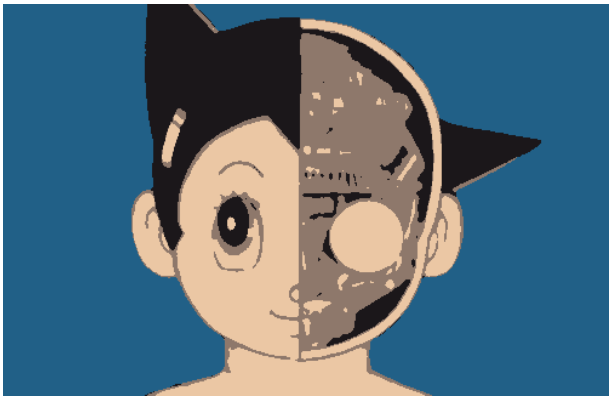
(b) FCM



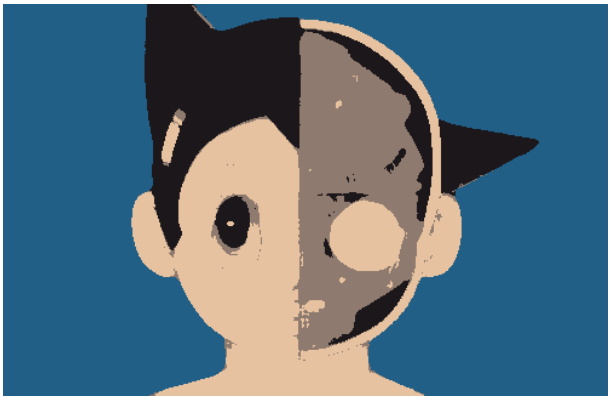
(c) SFCM $p = 1.0$, $q = 0.0$, $r = 0$



(d) SFCM $p = 1.0$, $q = 1.0$, $r = 2$



(e) SFCM $p = 1.0$, $q = 5.0$, $r = 2$



(f) SFCM $p = 1.0$, $q = 5.0$, $r = 5$

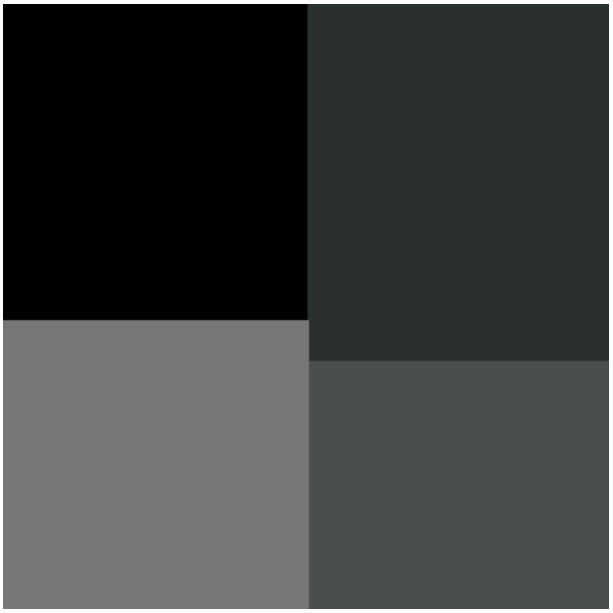
Figura 7: Immagine originale (*500x323* pixel) in *(a)*. Immagine in *(b)* clusterizzata con FCM in 4 cluster, $m = 2.0$, spazio colore *L*a*b**. Le immagini da *(c)* a *(f)* sono state clusterizzate con SFCM in 4 cluster, $m = 2.0$, spazio colore *L*a*b** e h definita nella formula (7), al variare di p, q, r.

(a) Immagine originale

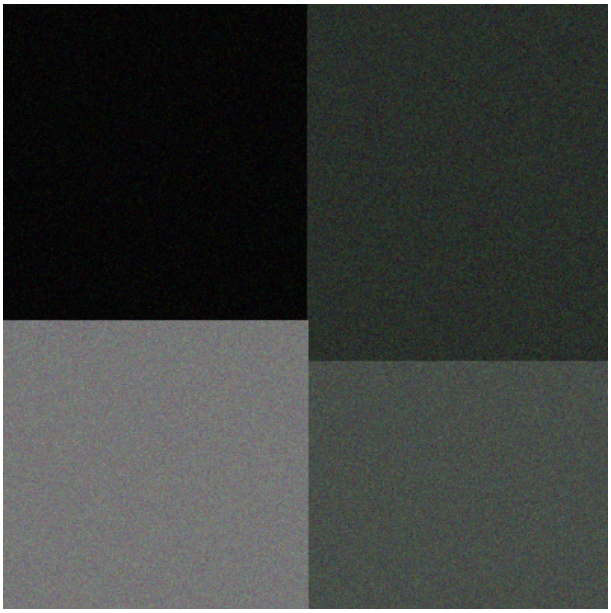
(b) SFCM con funzione spaziale $h(u)$

(c) SFCM con funzione spaziale $h(g)$

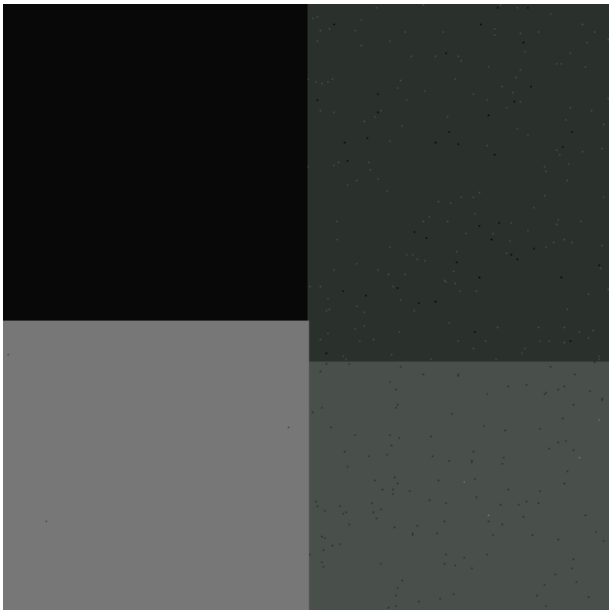
Figura 8: Immagine originale in *(a)*. Immagin in *(b)* e *(c)* clusterizzate con SFCM in 4 cluster, $m = 5.0$, spazio colore $L^*a^*b^*$, $p = 1.0$, $q = 1.0$ ed $r = 2$ e con differenti versioni della funzione spaziale.



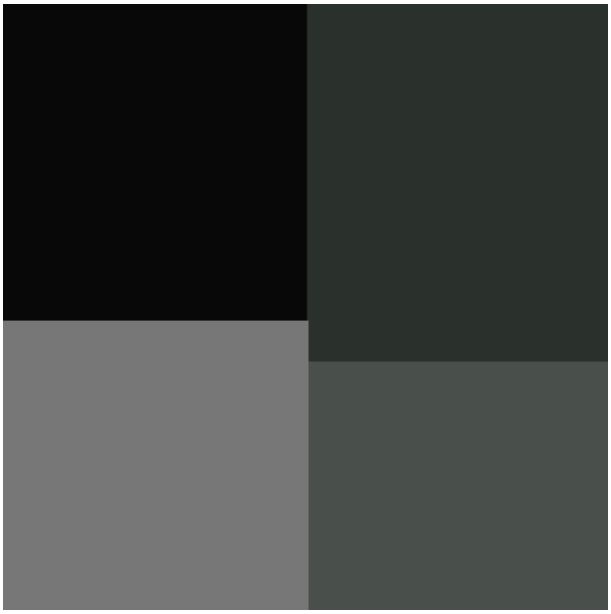
(a) Immagine originale



(b) Immagine con rumore gaussiano ($\sigma = 10$)

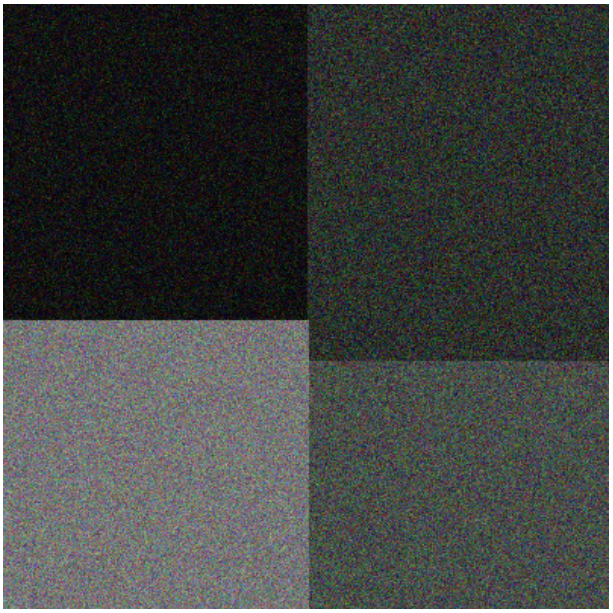


(c) FCM

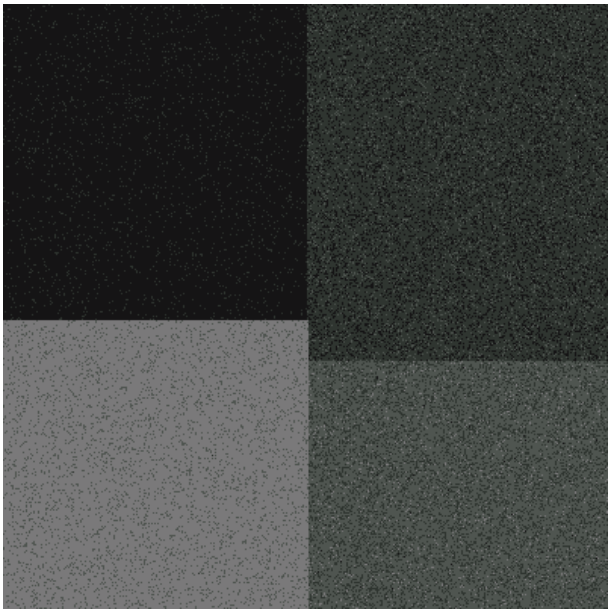


(d) SFCM $p = 1.0$, $q = 2.0$, $r = 2$

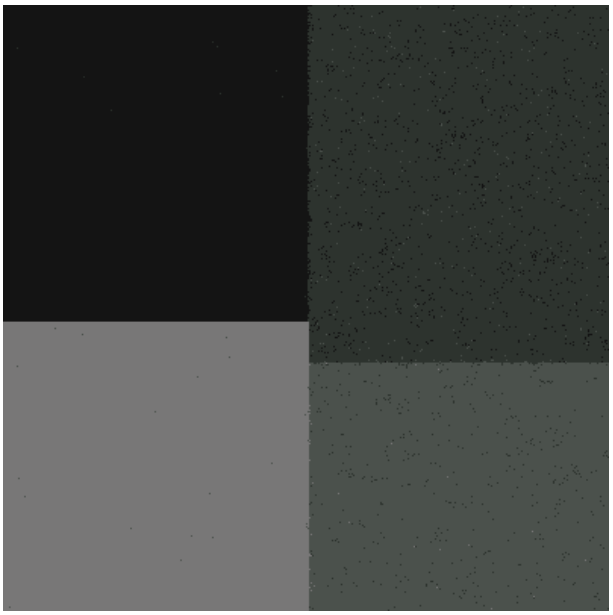
Figura 9: Immagine sintetica originale in *(a)*, a cui è stato aggiunto del rumore gaussiano in *(b)*. Immagine in *(c)* clusterizzata con FCM in 4 cluster, $m = 2.0$. Immagine in *(d)* clusterizzata con SFCM in 4 cluster, $m = 2.0$, h definita nella formula (7), con $p = 1.0$, $q = 2.0$ ed $r = 2$.



(a) Immagine con rumore gaussiano ($\sigma = 25$)



(b) FCM



(c) SFCM $p = 1.0$, $q = 3.0$, $r = 3$

(d) SFCM $p = 1.0$, $q = 6.0$, $r = 6$

Figura 10: Immagine sintetica originale in *(a) della figura 9*, a cui è stato aggiunto del rumore gaussiano in *(a)*. Immagine in *(b)* clusterizzata con FCM in 4 cluster, $m = 2.0$. Immagini in *(c)* e *(d)* clusterizzate con SFCM in 4 cluster, $m = 2.0$, h definita nella formula (7), con valori varianti per p, q, r.

Validazione

Parliamo di *cluster validity* o *validazione dei cluster* quando ci troviamo di fronte al problema di trarre qualche informazione sulla bontà dei risultati ottenuti da un algoritmo di clustering.

Solitamente il processo di validazione considera l'output di un algoritmo di clustering, le matrici U e V, e su di essere si calcolano delle misure, indici o dei coefficenti che sintetizzino i parametri utilizzati.

Ad esempio si possono considerare come misure, la distanza *intracluster* e *intercluster*. Le distanze intracluster ci dicono quanto gli elementi all'interno di un cluster sono coesi o vicini al loro centroide, mentre, una misura di distanza intercluster quanto sono distanti due centroidi.

In letteratura sono state utilizzate diversi tipi di misure per la validazione dei cluster, noi consideriamo in particolare delle misure specifiche per il fuzzy clustering. Le misure che tengono conto della funzione fuzzy di membership sono chiamate in letteratura *fuzzy partition indexes*, quelle che considerano anche la costruzione dei centroidi, *feature structure indexes*.

Fra i fuzzy partition indexes usiamo due coefficienti: V_{pc}, il *partition coefficient* proposto da Bezdek in [14], e $V_{pe}, il partition\ entropy\ coefficient$ utilizzato sempre da Bezdek in [15]; mentre fra i feature structure indexes, invece, usiamo la misura proposta da Xie and Beni in [16].

Ecco i coefficienti V_{pc} e V_{pe} :

$$\mathbf{V_{pc}} = \sum_{i=1}^{N}\sum_{j=1}^{k} u_{ij}^2 \tag{11}$$

$$\mathbf{V_{pe}} = -\frac{\sum_{i=1}^{N}\sum_{j=1}^{k}[u_{ij}log(u_{ij})]}{N} \tag{12}$$

Osserviamo il primo coefficiente V_{pc}, questo considera la somma dei quadrati dei valori di membership. Quindi più aumenta il numero dei cluster, più piccolo diventa il valore dell'indice, non è dunque una misura stabile. La matrice U, a confronto con una altra U' risulta migliore quando V_{pc} è massimo.

Per il secondo coefficiente V_{pe} si calcola l'entropia utilizzando i valori della fuzzy membership, rendendo minima l'entropia, si ottengono cluster migliori. Una nota di carattere numerico in questo caso va espressa poiché $\log u_{ij}$ per numeri vicini allo 0 risulta $-\infty$ di conseguenza si è costretti a sommare un, se pur piccolo[30], valore a u_{ij}.

[30] La correzione numerica applicata alla versione attuale corrisponde al minimo valore reale rappresentabile in Java.

Il problema maggiore di cui soffrono gli indici V_{pc} e V_{pe} è che considerando soltanto la matrice U si perde l'informazione relativa alla struttura della partizione; invecela funzione proposta da Xie and Beni, decide la bontà di un cluster considerando sia la distanza intracluster e intercluster. Vengono preferiti allora clusters con alta distanza fra cluster e minima distanza intracluster.:

$$\mathbf{V_{xb}} = \frac{\sum_{i=1}^{N}\sum_{j=1}^{k} u_{ij}^{m}||x_i - v_j||}{N\left(\min_{i,j}||v_i - v_j||^2\right)} \tag{13}$$

C'è da dire che però anche questo indice soffre un po' per valori di k che si avvicinano a N, cioè decresce se condizionato dall'alto numero dei cluster.Il clustering è maggiormente desiderabila se V_{xb} risulta essere piccolo.

Test di validazione

Si è costruito un test che utilizzasse gli indici sopra presentati per validare l'esecuzione dell'algoritmo SFCM al variare dei numerosi parametri introdotti. Il processo di validazione applicato ad set di immagini campioni, ha considerato come dati di input:

- Un set di interi come parametro del numero di cluster da assegnare all'algoritmo : Number of cluster = {4, 5, 6}
- Un set di reali come parametro di fuzzyness : m = {1.1, 2, 5}
- Un set di reali come parametro di p : P-weight = {1, 2}
- Un set di reali come parametro di q : Q-weight = {0, 1, 2}
- Un set di interi come parametro del raggio della window : radius={1, 2, 5}
- Un set contenente i criteri di inizializzazione : Init mode = {random U, random V, K-Means++}
- Un set contenente le tipologie di funzione spaziale: Spatial function = {Weightiest Cluster, Likeliest Cluster}
- Un set contenente i possibili spazi colore in cui convertire l'immagine: Color Space = {HSB, L*a*b*, XYZ, None}

La gestione degli input e l'esecuzione dell'algoritmo su ogni possibile combinazione, è stata affidata alla classe TestManager, che ha restituito come output, un file contenente i valori associati ai coefficienti V_{pc}, V_{pe}, V_{xb} ottenuti per ogni esecuzione. Il riepilogo dei dati stato fatto con il sofware R.

Nella Tabella (1) vengono raccolti i migliori trenta risultati per l'esecuzione su una immagine a colori raffigurante globuli bianchi. L'asterisco indica il valore migliore nella colonna.

Tabella 1: Risultati del processo di validazione applicato ad una immagine campione

k	m	P-weight	Q-weight	radius	Init mode	Spatial function	Color space	V_{pc}	V_{pe}	V_{xb}
4	1.1	2	1	1	random V (Forgy)	Weightiest Cluster	HSB	0.9966050*	0.008177236*	0.3026898519
4	1.1	2	2	1	random V (Forgy)	Likeliest Cluster	HSB	0.9966207	0.008089993	0.3028015425
4	1.1	2	2	1	random V (Forgy)	Weightiest Cluster	HSB	0.9969872	0.007289166	0.3028019860
4	1.1	2	1	1	K-Means++	Weightiest Cluster	HSB	0.9968741	0.007595373	0.3080133257
4	1.1	2	2	1	K-Means++	Likeliest Cluster	HSB	0.9968253	0.007651103	0.3078799200
4	1.1	2	2	1	K-Means++	Weightiest Cluster	HSB	0.9971872	0.006878929	0.3079944675
4	1.1	2	2	2	K-Means++	Weightiest Cluster	HSB	0.9966061	0.008184146	0.3077786476
4	1.1	2	1	1	random U	Weightiest Cluster	HSB	0.9968745	0.007594315	0.3080140097
4	1.1	2	2	1	random U	Likeliest Cluster	HSB	0.9966208	0.008089758	0.3028011785
4	1.1	2	2	1	random U	Weightiest Cluster	HSB	0.9971876	0.006878086	0.3079953000
5	1.1	2	2	1	K-Means++	Weightiest Cluster	HSB	0.9968222	0.007871771	0.5184937587
6	5	1	0	1	K-Means++	Likeliest Cluster	None	0.2191608	2.386511000	0.0034229486
6	5	1	0	1	random U	Likeliest Cluster	None	0.2192581	2.386202000	0.0034178202*
6	5	1	0	1	random U	Likeliest Cluster	XYZ	0.2350290	2.331548000	0.0036195380
6	5	1	0	1	random V (Forgy)	Likeliest Cluster	XYZ	0.2349800	2.331715000	0.0036271866
6	5	1	0	1	K-Means++	Weightiest Cluster	XYZ	0.2349802	2.331715000	0.0036273725
6	5	1	0	1	random U	Weightiest Cluster	L*a*b*	0.2003750	2.449240000	0.0047828584
6	5	1	0	2	random V (Forgy)	Weightiest Cluster	L*a*b*	0.2003382	2.449392000	0.0048563498
6	5	1	0	5	K-Means++	Likeliest Cluster	L*a*b*	0.2003360	2.449400000	0.0048859043
5	5	1	0	5	K-Means++	Weightiest Cluster	None	0.2596104	2.128710000	0.0050044740
5	5	1	0	5	random U	Likeliest Cluster	None	0.2594690	2.129222000	0.0050300556
5	5	1	0	2	random V (Forgy)	Likeliest Cluster	None	0.2594692	2.129221000	0.0050300878
5	5	1	0	1	K-Means++	Weightiest Cluster	XYZ	0.2760143	2.079533000	0.0058588183
4	1.1	2	0	1	random V (Forgy)	Weightiest Cluster	HSB	0.9955778	0.010536512	0.3024762754
5	1.1	2	2	2	random V (Forgy)	Weightiest Cluster	HSB	0.9954705	0.010784539	0.2902033252
5	1.1	2	0	2	K-Means++	Likeliest Cluster	HSB	0.9954861	0.011154811	0.5181782846
5	1.1	2	2	2	random V (Forgy)	Likeliest Cluster	HSB	0.9952428	0.011279873	0.2901177782
5	1.1	2	1	5	random V (Forgy)	Likeliest Cluster	HSB	0.9947995	0.012344535	0.2898796756
5	1.1	2	0	5	random V (Forgy)	Likeliest Cluster	HSB	0.9944885	0.013170745	0.2905439778
4	1.1	1	2	1	random V (Forgy)	Likeliest Cluster	HSB	0.9938162	0.014893967	0.3028620506
4	1.1	1	2	5	K-Means++	Weightiest Cluster	HSB	0.9930545	0.016623016	0.3023757284
6	1.1	2	0	2	random U	Likeliest Cluster	HSB	0.9928411	0.016990531	0.4496472149
5	1.1	1	2	1	random U	Weightiest Cluster	HSB	0.9928010	0.017375966	0.2899901791
4	1.1	1	1	2	random U	Likeliest Cluster	HSB	0.9923318	0.018418828	0.3025224470
4	1.1	2	1	2	random U	Weightiest Cluster	HSB	0.9964881	0.008477149	0.3078825198
4	1.1	2	2	2	random V (Forgy)	Likeliest Cluster	HSB	0.9963821	0.008662179	0.3027590126
5	1.1	2	1	1	K-Means++	Likeliest Cluster	HSB	0.9961544	0.009403816	0.5183237984
4	1.1	2	1	2	random U	Likeliest Cluster	HSB	0.9960929	0.009344472	0.3026140263
5	1.1	2	2	2	K-Means++	Likeliest Cluster	HSB	0.9959720	0.009857543	0.5191989808
5	1.1	2	1	5	K-Means++	Weightiest Cluster	HSB	0.9958062	0.010316549	0.5197861266
5	1.1	2	2	1	random V (Forgy)	Likeliest Cluster	HSB	0.9956046	0.010455675	0.2903623602
4	1.1	2	0	2	random U	Weightiest Cluster	HSB	0.9955778	0.010536433	0.3024761345
5	1.1	2	0	1	K-Means++	Likeliest Cluster	HSB	0.9954861	0.011154811	0.5181782846
5	1.1	2	1	1	random V (Forgy)	Likeliest Cluster	HSB	0.9952118	0.011427944	0.2904419887
5	1.1	2	1	5	random V (Forgy)	Likeliest Cluster	HSB	0.9947995	0.012344535	0.2898796756
5	1.1	2	0	2	random V (Forgy)	Weightiest Cluster	HSB	0.9944885	0.013170745	0.2905439778
4	1.1	1	2	2	random V (Forgy)	Weightiest Cluster	HSB	0.9933892	0.015884291	0.3028770057
6	1.1	2	0	5	random U	Likeliest Cluster	HSB	0.9928411	0.016990531	0.4496472149

È facile osservare dalla tabella che i risultati ottimi per i coefficienti V_{pc} e V_{pe} hanno lo stesso indice ovvero, entrambi confermano che i cluster prodotti con quei parametri siano i migliori. In tutti questi casi lo spazio colore scelto è stato HSB, e non a caso , poiché una ulteriore, più attenta analisi dell'immagine ha mostrato che il parametro saturazione (S) gioca un ruolo fondamentale nel suddividere le aree dell'immagine riferite ai diversi tipi di globuli bianchi[31]. Clusterizzazioni con valori inferiori, ma di poco sembrano prediligere un raggio, della finestra utilizzata per computare la funzione spaziale, di piccole dimensioni: ancora una volta possiamo ricercare una spiegazione dicendo che raggi di dimensione superiore (ad esempio 5) risultavano in una perdita eccessiva della informazione per le aree distintive delle cellule, che andavano via via omogeneizzandosi troppo; un discorso analogo può essere fatto per il numero di cluster trovato ottimale (4), mentre risulta che per questa immagine la modalità di inizializzazione e la tipologia della funzione spaziale non giocano un ruolo fondamentale. Non si riesce a dire niente di particolare circa i parametri p e q .

Una situazione diversa si presenta, invece, per il coefficiente V_{xb}, il notiamo valori bassi per questo indice hanno sempre preferito $p = 1$ e $q = 0$ che sono i parametri del FCM[32]; per tali configurazioni dei parametri il numero di cluster preferito è maggiore (6) e non vi è neanche una netta preferenza per uno spazio colore in particolare (si potrebbe dire che tutti, tranne HSB, possano equamente portare a buoni risultati). Le configurazioni di parametri trovate sono di più difficile interpretazione e l'indice non fornisce informazioni chiari in proposito della configurazione migliore con gran scarto sulle altre.

Considerazioni maggiormente approfondite si potrebbero fare analizzando i dati provenienti da test effettuati su più immagini.

[31] Ad una segmentazione semantica corrisponde, in questo caso anche una ottima segmentazione secondo gli indici di validazione utilizzati

[32] In tal maniera i parametri r, e scelta della funzione spaziale sono risultati ininfluenti seppure variassero di valore

Riferimenti bibliografici

[1] G. Gan and C. Ma. *Data Clustering - Theory, Algorithms and Applications*. Society for Industrial and Applied Mathematics, 2007.

[2] ImageJ. Image processing and analysis in java. `http://rsbweb.nih.gov/ij/`.

[3] Keh-Shih Chuang, Hong-Long Tzeng, Sharon Chen, Jay Wu, and Tzong-Jer Chen. Fuzzy c-means clustering with spatial information for image segmentation. *Computerized Medical Imaging and Graphics*, 2005.

[4] Jarek Sacha. Imagej plugins: Clustering. `http://ij-plugins.sourceforge.net/`.

[5] MacQueen. Some methods for classification and analysis of multivariate observations. *Proceedings of 5th Berkeley Symposium on Mathematical Statistics and Probability*, 1967.

[6] S. P Lloyd. Least square quantization in pcm. *IEEE Transactions on Information Theory 28 (2): 129137*, 1982.

[7] E. Forgy. Cluster analysis of multivariate data: Effi- ciency vs. interpretability of classification. *Biometrics, vol. 21*, 1965.

[8] M. Emre Celebi. Effective initialization of k-means for color quantization. *Proceedings of the IEEE International Conference on Image Processing (ICIP 2009)*, 2009.

[9] S. Vassilvitskii D. Arthurand. k-means++: The advantages of careful seeding. *Proc of SODA07*, 2007.

[10] J. C. Dunn. A fuzzy relative of the isodata process and its use in detecting compact, well-separated clusters. *Journals of Cybernetics vol 3*, 1973.

[11] James C. Bezdek. Pattern recognition with fuzzy objective function algoritms. *Plenum Press, New York*, 1981.

[12] James C. Bezdek. Efficient implementation of the fuzzy c-means clustering algorithms. *IEEE Transactions on Pattern Analysis and Machine Intelligence. VOL. PAMI-8, NO. 2*, 1986.

[13] P. Viola M. Jones. Robust real-time object detection. *Second International Workshop on Statistical and Computational Theories of Vision Modeling, Learning, Computing, and Sampling*, 2001.

[14] James C. Bezdek. Numerical taxonomy with fuzzy sets. *J. Math. Biol. vol 1*, 1974.

[15] James C. Bezdek. Mathematical models for systematic and taxonomy. *In: proceedings of eight international conference on numerical taxonomy*, 1975.

[16] X. L. Xie and G. Beni. A validity measure for fuzzy clustering. *IEEE Transaction on Pattern Analysis and Machine Intelligence*, 1990.