ALMA MATER STUDIORUM - UNIVERSITÀ DI BOLOGNA
CAMPUS DI CESENA

DIPARTIMENTO DI INFORMATICA – SCIENZA E INGEGNERIA

CORSO DI LAUREA IN INGEGNERIA E SCIENZE INFORMATICHE

TDSFT
(Two-Dimensional Segmentation Fusion Tool): tool open-source per la fusione di segmentazioni bidimensionali create da differenti anatomopatologi

Elaborato in

Programmazione

Relatore

Prof.ssa Antonella Carbonaro

Presentata da

Lorenzo Drudi

Correlatori

Prof. Filippo Piccinini

Prof. Gastone Castellani

Prof. Giovanni Martinelli

Anno Accademico 2022 / 2023

“*Non siamo mai*

completamente formati

ma sempre soggetti ad una

lenta evoluzione coscienziale.”

Proust

PAROLE CHIAVE

Histology

Microscopy

Image Fusion

Open-Source Tool

Standalone Application

ABSTRACT (English version)

The treatment of a tumor consists of numerous steps and activities. Among them, segmentation of the cancer area, that is, the correct identification of its spatial location, is one of the most important and at the same time complex and delicate steps. The difficulty in deriving reliable segmentation stems from the lack of a standard for identifying the edges and surrounding tissues of the tumor area. For this reason, the entire process is affected by considerable subjectivity.

Given a tumor image, different practitioners can associate different segmentations with it, and the diagnoses produced may differ. Moreover, experimental data show that analysis of the same area by the same physician at two separate time points may result in different segmentations being produced.

Given multiple segmentations related to the same tumor, statistical metrics could be exploited to determine the most reliable one, from there identified as *ground truth*. Numerous algorithms have been developed over time for this procedure, but none of them is yet validated. Therefore, research is still active.

In this work, we developed *Two-Dimensional Segmentation Fusion Tool* (TDSFT), an open-source tool also distributed as a standalone application for *MAC*, *Linux*, and *Windows*, which offers a simple and extensible interface where numerous algorithms are proposed to "*mediate*" (*i.e.*, process and fuse) multiple segmentations. *TDSFT* is a tool made with ease of use as a fixed point, to support and help medical specialists during their work. In addition, *TDSFT* is designed to be easily extended with new algorithms thanks to a dedicated graphical interface for configuring new parameters. *TDSFT* can be downloaded at the following link: *https://sourceforge.net/p/tdsft*.

ABSTRACT (Versione italiana)

Il trattamento di un tumore è composto da numerose fasi e attività. Tra queste, sicuramente, la segmentazione della zona malata, ossia la corretta identificazione della sua posizione spaziale, è una tra le più importanti e al tempo stesso complesse e delicate. La difficoltà nel ricavare una segmentazione affidabile deriva dalla mancanza di uno standard per l'individuazione dei bordi e dei tessuti circostanti all'area tumorale. Per questo motivo l'intero processo è affetto da notevole soggettività.

Data una immagine rappresentativa del tumore, è possibile che diversi operatori associno ad essa segmentazioni differenti e le diagnosi prodotte potrebbero differire tra loro. Oltretutto, dati sperimentali mostrano che l'analisi della stessa zona da parte dello stesso medico in due momenti temporali distinti può portare alla realizzazione di segmentazioni diverse.

Date più segmentazioni relative allo stesso tumore, si potrebbero sfruttare metriche statistiche per determinare la più affidabile, da qui identificata come *ground truth*. Per fare questo, nel corso del tempo, sono stati sviluppati numerosi algoritmi ma ad oggi, ancora, non esiste un metodo validato. Per questo motivo, la ricerca nel settore è ancora attiva.

In questo lavoro, abbiamo sviluppato *Two-Dimensional Segmentation Fusion Tool* (TDSFT), un tool open-source realizzato per sostenere e aiutare gli specialisti che offre una interfaccia semplice ed estendibile in cui vengono proposti numerosi algoritmi per "*mediare*" (*i.e.*, elaborare e fondere) molteplici segmentazioni. Inoltre, *TDSFT* è stato progettato per essere facilmente esteso con nuovi algoritmi grazie ad una apposita interfaccia grafica dedicata alla configurazione di nuovi parametri. *TDSFT* è scaricabile al seguente link: *https://sourceforge.net/p/tdsft.*

INDICE

Introduzione

Il percorso di cura di un malato di tumore richiede il contributo di più professionisti che collaborano tra loro secondo tempi e procedure ben definiti ed è composto da diverse fasi e attività. Una di queste è la segmentazione della zona affetta dalla malattia, ossia la corretta identificazione della sua posizione spaziale, che viene effettuata in diverse fasi del processo di cura, a partire dalla diagnosi sino a fasi successive nelle quali la segmentazione (o annotazione) viene utilizzata per analizzare eventuali miglioramenti o peggioramenti della situazione clinica.

La figura medica che solitamente esegue la segmentazione è l'anatomopatologo, ossia un medico specializzato in anatomia patologica il cui ruolo principale è quello di formulare diagnosi, tra le quali le più ricorrenti sono quelle in ambito oncologico.

Il trattamento di un tumore non è un processo standard, esso infatti è studiato e programmato singolarmente per ogni paziente. Così come non esiste un processo standard per il trattamento, non esiste una standardizzazione nemmeno per l'attività di segmentazione. Questo è dovuto al fatto che differenti prestazioni diagnostiche possono produrre immagini con caratteristiche diverse tra loro; tecniche come la *Radiografia*, ad esempio, producono immagini bidimensionali, mentre tecniche come la *Tomografia Computerizzata* o la *Risonanza Magnetica Nucleare*, producono immagini tridimensionali. Inoltre, l'immagine rappresentativa del tumore differisce anche in base alla terapia che lo specialista vuole applicare, infatti, le immagini mediche possono contenere strettamente la zona malata oppure contenere anche i tessuti limitrofi per terapie come la *radioterapia* con la quale, attraverso radiazioni ionizzanti, si vogliono bersagliare anche i tessuti vascolari circostanti.

Sebbene la segmentazione manuale fatta da uno specialista medico sia un processo molto dispendioso in termini di tempo è ancora oggi il metodo utilizzato. Infatti, nonostante settori come *Computer Vision* e *Deep Learning* siano in continua evoluzione, la mancanza di standard sulla delimitazione di un tumore rappresenta ancora un ostacolo da superare, che origina grandi difficoltà per i metodi basati sull'apprendimento automatico, poiché non consente di effettuare un'etichettatura del set di addestramento robusta e consistente. Nel corso del tempo sono stati proposti molti metodi semi-automatici **[1]** o automatici **[2]** ma prima che possano essere utilizzati sono necessari ancora molti miglioramenti.

La mancanza di standard per la delimitazione di un tumore risulta essere una difficoltà anche per l'attività manuale. Infatti, data una immagine rappresentativa di un tumore, è possibile che diversi operatori associno ad essa segmentazioni differenti e le diagnosi prodotte potrebbero differire tra loro. Oltretutto, dati sperimentali mostrano che l'analisi della stessa zona da parte dello stesso medico in due momenti temporali distinti può portare alla realizzazione di segmentazioni diverse.

Date più segmentazioni relative allo stesso tumore è quindi necessario determinare la più affidabile, da qui identificata come *ground truth*. Una delle possibilità è scegliere l'annotazione prodotta dall'anatomopatologo con più esperienza, andando però a ignorare possibili errori commessi, oltre a tutte le scelte effettuate dagli altri specialisti. Un metodo invece che considera tutte le segmentazioni prevede la loro *fusione*. Per fare questo, nel corso del tempo, sono stati sviluppati numerosi algoritmi ma, ad oggi, ancora, non esiste un metodo validato. Per tale motivo la ricerca nel settore è ancora attiva.

Questo progetto di Tesi nasce all'interno del gruppo di ricerca "*Data Science for Health*" (DS4H), composto da ricercatori e professori dell'Università di Bologna (UniBo) e del "*IRCCS Istituto Romagnolo dei Tumori Dino Amadori*" (IRST) di Meldola (FC). Il gruppo

DS4H nasce con lo scopo di coordinare professionisti e risorse sia dal settore informatico, dell'ingegneria dell'informazione e della fisica, che dal settore biomedico, della biologia, della chimica e della medicina.

È in questo contesto che il prof. Filippo Piccinini, lavorando a stretto contatto con esperti quali medici, fisici e anatomopatologi, si accorge della mancanza di uno strumento semplice ed efficace a supporto delle figure professionali del settore, che offrisse la possibilità di unire tra loro molteplici segmentazioni. Questo ha portato alla nascita del progetto di tesi dal titolo "*TDSFT (Two-Dimensional Segmentation Fusion Tool): tool open-source per la fusione di segmentazioni bidimensionali create da differenti anatomopatologi*".

TDSFT è un software open-source con codice sviluppato in *MATLAB* e distribuito anche come applicazione standalone per *MAC*, *Linux* e *Windows*. Al giorno d'oggi, *MATLAB* risulta infatti essere uno degli ambienti informatici di maggior diffusione soprattutto tra ingegneri e ricercatori operanti nel campo sanitario. In particolare, *MATLAB* rappresenta un potente linguaggio di programmazione ottimizzato per il calcolo matematico basato su matrici multidimensionali e, proprio per queste sue caratteristiche, è il linguaggio più comune nel mondo biomedicale. Nonostante sia un ambiente closed-source, è possibile generare applicazioni standalone eseguibili senza la necessità di avere *MATLAB* installato sulla propria macchina e senza dover disporre di nessun tipo di licenza. Infatti, il *MATLAB Runtime*, ossia un insieme di librerie condivise e file necessari ad eseguire codice compilato utilizzando il *MATLAB Compiler*, è disponibile gratuitamente. Le motivazioni precedentemente elencate, e l'ampia diffusione di tale linguaggio fra i ricercatori, hanno portato alla sua scelta come tecnologia alla base di questo progetto.

TDSFT è uno strumento facile da usare progettato per sostenere ed aiutare gli specialisti medici nel loro lavoro. L'applicativo consente infatti di mediare molteplici segmentazioni

mettendo a disposizione dell'utente numerosi algoritmi. Inoltre, essendo un settore in continuo sviluppo e non esistendo ancora un metodo validato per la fusione di segmentazioni bidimensionali, una delle sue principali caratteristiche è l'estendibilità. *TDSFT* consente infatti di aggiungere, in modo semplice e veloce, nuovi algoritmi grazie ad un'apposita interfaccia grafica dedicata alla configurazione di nuovi parametri.

A livello pratico, in questo lavoro di Tesi sono stati completati i seguenti task:

1. Studio dello stato dell'arte per individuare pregi e difetti degli algoritmi per calcolare la "*media*" di curve 2D;
2. Implementazione di 7 diversi algoritmi ed integrazione del codice sorgente di 1 algoritmo sviluppato da un gruppo esterno per un totale di 8 algoritmi disponibili;
3. Implementazione di un'interfaccia *user-friendly* per gestire vari formati di immagini di input e fornire in output la segmentazione "*mediata*" prodotta dall'algoritmo scelto dall'utente;
4. Progettazione ed implementazione di un meccanismo semplice per permettere all'utente di aggiungere eventuali propri algoritmi;
5. Validazione del tool usando dataset di test;
6. Creazione del logo in grafica vettoriale, scrittura del manuale e montaggio del video tutorial;
7. Release del codice sorgente e delle versioni standalone per *MAC*, *Linux* e *Windows*.

1 - Segmentazione in oncologia

La segmentazione è l'attività svolta per delineare i contorni di una zona affetta da tumore. Non è una procedura standard e come l'intero trattamento, anche essa, varia a seconda della situazione clinica del paziente.

1.1 – Immagini mediche tumorali

La segmentazione viene prodotta a partire da un'immagine medica rappresentante il tumore. Questa può essere ottenuta attraverso diversi esami diagnostici e la scelta, anche in questo caso, può variare. Le prestazioni di diagnosi più diffuse sono **[3]**:

- *Radiografia* (RX): tecnica che si basa sulla capacità dei raggi X di impressionare una pellicola fotografica con diverse intensità a seconda dell'assorbimento da essi subito nel passaggio attraverso i diversi tessuti del corpo. Tramite la *RX* si può ottenere una rappresentazione bidimensionale del distretto studiato;
- *Tomografia Computerizzata* (TC): tecnica di imaging anatomico in cui uno stretto fascio di raggi X viene diretto verso il soggetto e ruotato rapidamente intorno al corpo, producendo così segnali di trasmissione. Questi segnali vengono elaborati e ricostruiti per formare immagini trasversali del soggetto in esame. La *TC* produce immagini tridimensionali ad alta risoluzione;
- *Tomografia a emissione di positroni* (PET): tecnica diagnostica di medicina nucleare che prevede che uno scanner rilevi come una piccola quantità di farmaci e agenti fisiologici radio marcati, iniettata nel paziente per via endovenosa, si distribuisce all'interno delle cellule del corpo. Questa prestazione è particolarmente utile per capire se e dove si è sviluppato un tumore, e se si è sviluppato in zone diverse da

quelle già note. Per ottenere informazioni migliori spesso è unita ad altre tecniche quali *TC* (PET/TC) e *risonanza magnetica nucleare* (PET/RMN);

- *Risonanza Magnetica Nucleare* (RMN): esame diagnostico per immagini basato sull'applicazione di un campo magnetico ad alta intensità a tutto il corpo del soggetto e contemporaneamente di onde di radiofrequenza presso il distretto da esaminare. Rispetto ad altre modalità di imaging come la *TC* e la *PET*, la *RMN* ha il vantaggio intrinseco di fornire immagini sfaccettate e di eccellente contrasto oltre che di non utilizzare radiazioni ionizzanti. Inoltre, anche questa tecnica permette di ottenere una rappresentazione tridimensionale del tumore. La *RMN* può essere eseguita in qualsiasi orientamento senza dover ricorrere alla riformattazione dell'immagine in fase di post-elaborazione.

Inoltre, l'immagine medica può essere ottenuta anche tramite *microscopia*, dove tipicamente le immagini acquisite sono bidimensionali (a meno di acquisizioni tramite *microscopi confocali* o a *foglio di luce*). In particolare, ciò avviene nel campo dell'*Istologia*, ramo della biologia che studia la struttura microscopica e ultramicroscopica di tessuti e organi attraverso sezioni comunemente conosciute con il nome di *vetrini istologici*. In questo lavoro di tesi sono state analizzate principalmente immagini di provini istologici acquisite con *microscopi a campo largo*.

La prima caratteristica che differenzia le immagini mediche è quindi il numero di dimensioni; infatti, in base alla prestazione effettuata, è possibile ottenere immagini bidimensionali oppure tridimensionali. La seconda è invece l'estensione dell'area di studio; in determinante situazioni cliniche essa non si limita solamente alla zona affetta da tumore ma comprende anche i tessuti vascolari circostanti. Questo è fondamentale per l'applicazione di alcune

terapie come la *radioterapia*, terapia localizzata che attraverso l'utilizzo di radiazioni ad elevata energia (radiazioni ionizzanti) provoca la necrosi delle cellule tumorali.

I formati di immagine più utilizzati in ambito medico sono **[4]**:

- *Digital Imaging and COmmunications in Medicine* (DICOM): standard che definisce i criteri per la comunicazione, la visualizzazione, l'archiviazione e la stampa di informazioni di tipo biomedico **[5]**;
- *Neuroimaging Informatics Technology Initiative* (NifTI): formato sviluppato per la memorizzazione e l'elaborazione di dati di neuroimmagini comunemente utilizzato per memorizzare immagini cerebrali.

Entrambi questi formati salvano le immagini tridimensionali utilizzando uno *z-stack*, ossia una pila di sezioni bidimensionali ordinate.

1.2 – Segmentazione di una immagine medica 2D

In questa sezione affronteremo l'argomento della segmentazione effettuata su immagini bidimensionali considerando che, nel caso in cui l'immagine sia a tre dimensioni, è sempre possibile selezionare un sottoinsieme di sezioni che compongono lo *z-stack*, segmentarle separatamente, ed infine attraverso la loro interpolazione, produrre la delimitazione tridimensionale del tumore.

Data un'immagine bidimensionale rappresentante un tumore, l'attività di segmentazione consiste nell'annotare in modo binario ogni pixel come sano oppure malato. La difficoltà principale nel ricavare una segmentazione affidabile deriva dalla mancanza di uno standard

per l'individuazione dei bordi della zona malata, che solitamente non ha un contorno ben delineato. Questo porta il processo ad essere affetto da notevole soggettività.

Le metodologie di segmentazione possono essere suddivise in tre gruppi:

- *manuali*: interamente effettuate da medici specializzati;
- *semi-automatiche* **[1]**: nonostante comprendano fasi automatiche, richiedono comunque interazione da parte di uno specialista, ad esempio, in fase di configurazione o di post-elaborazione;
- *automatiche* **[2]**: interamente automatiche senza la necessità di interventi da parte di un medico.

Nonostante settori come *Computer Vision* e *Deep Learning* siano in continua evoluzione, la mancanza di standard sull'individuazione dei bordi non consente un'etichettatura del set di addestramento robusta e consistente. Pertanto, i metodi semi-automatici e automatici trovano difficile applicazione. La segmentazione manuale, nonostante richieda molto tempo, è quindi ancora lo standard e l'annotazione prodotta da un medico esperto è considerata verità di base.

Tra gli strumenti maggiormente utilizzati per la segmentazione manuale troviamo:

- *MedSeg* **[6]**: un tool web gratuito e facile da usare;

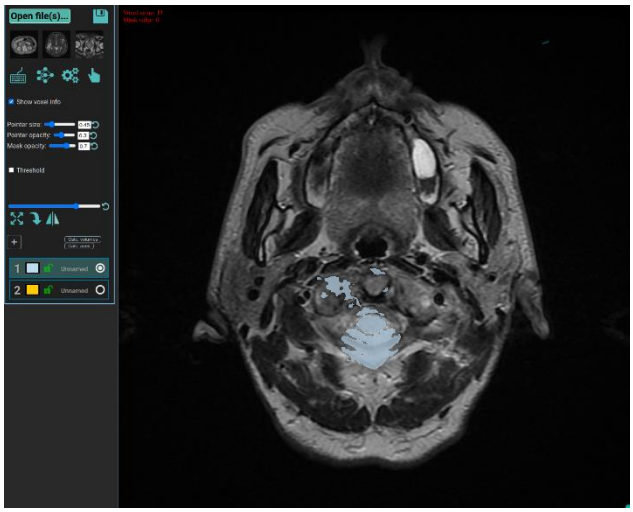

***Figura 1.1**: Finestra principale del tool MedSeg.*

- *ITK-Snap* [7]: un tool sviluppato in *C++*, multipiattaforma e open-source.

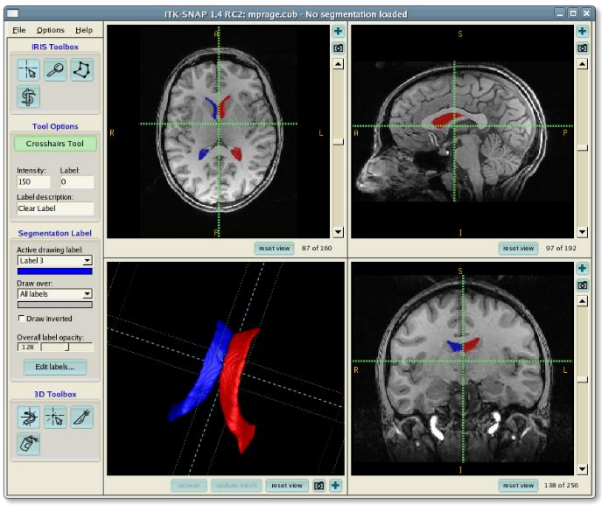

***Figura 1.2**: Finestra principale del tool ITK-Snap.*

La rappresentazione standard di una segmentazione ha sfondo nero e linea di colore bianco con spessore di un pixel. Nel caso quindi di immagini binarie, i pixel di sfondo hanno valore 0 mentre i pixel appartenenti alla segmentazione hanno valore 1.

Segue, in **Figura 1.3** e **Figura 1.4**, un esempio di una segmentazione presa dall'articolo di presentazione del tool *3D-Cell-Annotator* [8]**.**

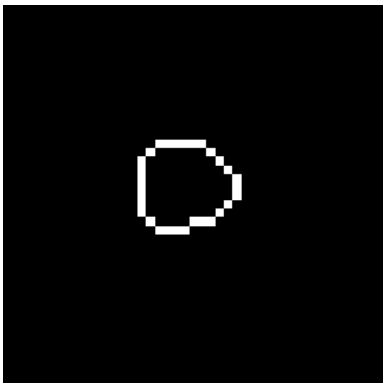

***Figura 1.3**: Esempio di segmentazione rappresentante uno sferoide fotografato tramite microscopia a foglio di luce (LSFM).*

0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	1	1	1	1	1	1	0	0	0	0	0
0	0	1	0	0	0	0	0	0	1	0	0	0	0
0	1	0	0	0	0	0	0	0	0	1	0	0	0
0	1	0	0	0	0	0	0	0	0	0	1	0	0
0	1	0	0	0	0	0	0	0	0	0	0	1	0
0	1	0	0	0	0	0	0	0	0	0	0	1	0
0	1	0	0	0	0	0	0	0	0	0	0	1	0
0	1	0	0	0	0	0	0	0	0	0	1	0	0
0	1	0	0	0	0	0	0	0	0	1	0	0	0
0	0	1	0	0	0	0	1	1	1	0	0	0	0
0	0	0	1	1	1	1	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0

***Figura 1.4**: Rappresentazione binaria della segmentazione in **Figura 1.3**. Dalla rappresentazione in forma matriciale è possibile notare come la segmentazione sia una linea chiusa di spessore di un pixel.*

La ricerca nel settore è ancora molto attiva, un metodo di segmentazione automatico validato consentirebbe infatti di rimuovere la soggettività durante l'attività di segmentazione e, inoltre, consentirebbe ai medici specializzati di risparmiare una grande quantità di tempo.

2 – Tecnologie utilizzate

Nelle seguenti sezioni vengono descritte le principali tecnologie utilizzate per lo sviluppo di *TDSFT*. In particolare, verranno descritti:

- *MATLAB R2022b*: ambiente di sviluppo e linguaggio utilizzato;
- *MATLAB App Designer*: strumento utilizzato per lo sviluppo dell'interfaccia grafica;
- *MATLAB Compiler*: strumento utilizzato per generare le applicazioni standalone.

2.1 – MATLAB 2022b

MATLAB, abbreviazione di *MATrix LABoratory*, è un ambiente di programmazione commerciale ottimizzato per applicazioni scientifiche, elaborazioni numeriche e per la simulazione di sistemi dinamici. Con il termine *MATLAB* si intende anche l'omonimo linguaggio di programmazione creato dalla *MathWorks* (The MathWorks, Inc., Massachusetts, USA). Oggi *MATLAB* è uno degli ambienti di sviluppo scientifici di maggior diffusione nel mondo biomedicale e viene utilizzato in molti corsi universitari e aziende del campo ingegneristico perché, grazie ai tanti *toolbox* (*i.e.*, collezione di programmi e funzioni che permettono la soluzione di problemi mirati a campi specifici), risulta molto versatile e semplice da utilizzare.

In questo progetto è stata utilizzata la release di *MATLAB 2022b*, rilasciata il 20 settembre 2022. Inoltre, è stato utilizzato l'*Image Processing Toolbox* **[9]**.

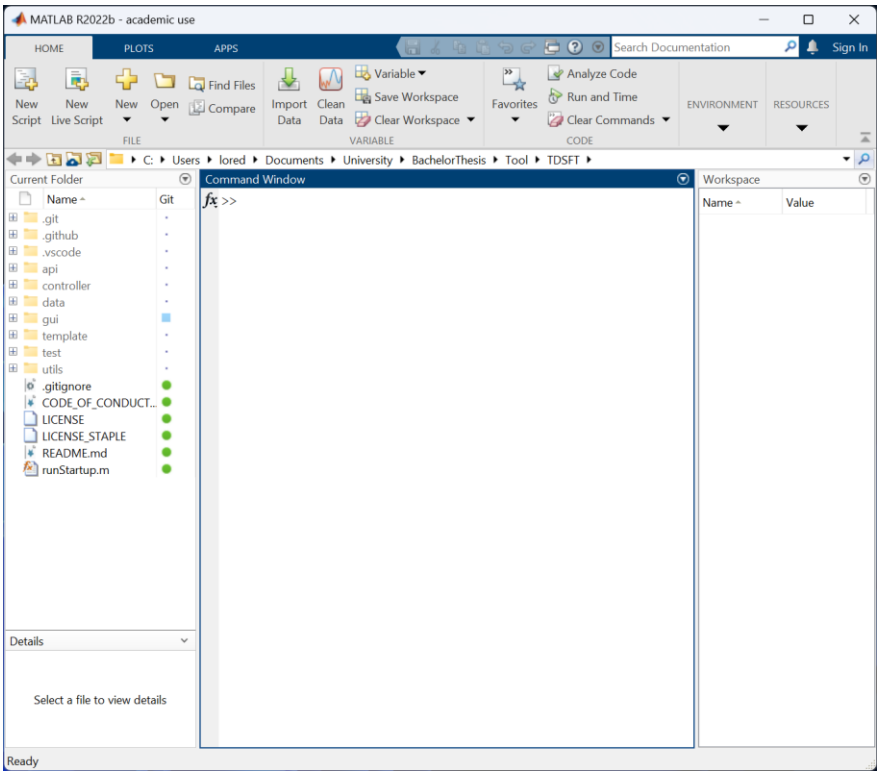

***Figura 2.1**: Finestra principale di MATLAB R2022b.*

2.2 – MATLAB App Designer

App Designer è un ambiente incluso in *MATLAB* che permette di creare applicazioni con *interfacce grafiche utente* (GUI). Questo strumento consente di definire il design in modo agile tramite il trascinamento dei componenti e, tramite l'editor integrato, di programmarne rapidamente il comportamento.

App Designer è stato rilasciato per la prima volta insieme alla release di *MATLAB 2016a* del 21 marzo 2016. Esso è stato sviluppato per andare a sostituire *GUIDE*, ossia il precedente strumento disponibile per lo sviluppo di interfacce utente.

A differenza di *GUIDE*, il quale aveva un approccio procedurale, *App Designer* è stato progettato con un approccio orientato agli oggetti. Questo consente agli utenti di creare codice

modulare, riutilizzabile e rende il processo di sviluppo più organizzato. Inoltre, *App Designer*, fornisce una interfaccia più semplice oltre che una vasta gamma di componenti grafici.

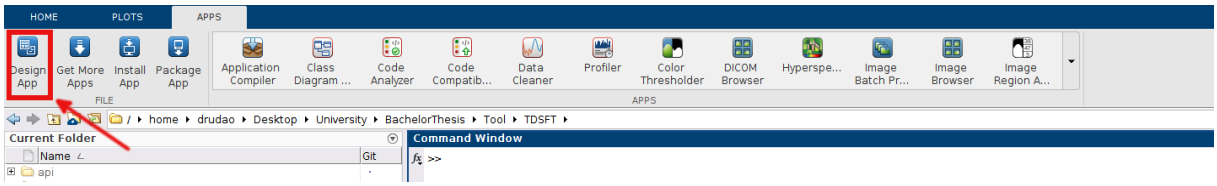

***Figura 2.2**: App Designer si trova nella sezione "APPS" di MATLAB.*

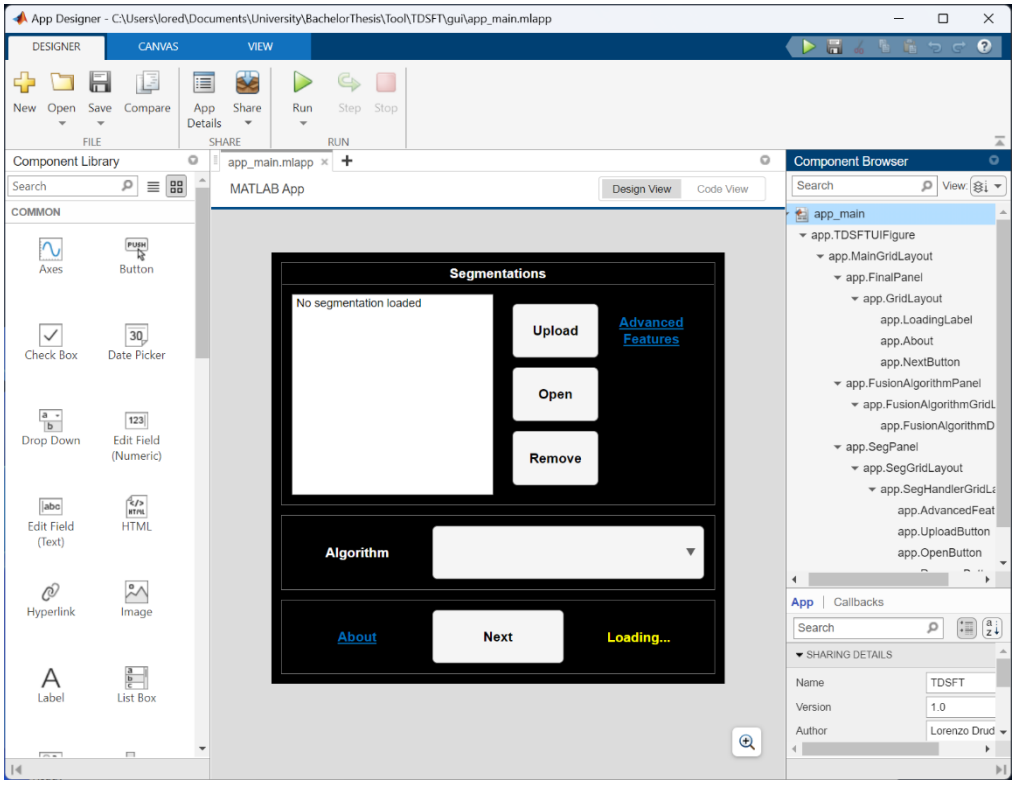

***Figura 2.3**: Interfaccia di App Designer.*

2.3 – MATLAB Compiler

Tramite il *MATLAB Compiler* è possibile generare applicazioni standalone che impacchettano l'applicazione sviluppata in *MATLAB* e che consentono agli utenti di utilizzare tale applicazione senza necessitare di nessun tipo di licenza. Questo è fondamentale per un ambiente closed-source come quello di casa *MathWorks*. Infatti, l'unico requisito per poter

eseguire un'applicazione standalone è che sulla macchina sia presente il *MATLAB Runtime*, ossia un insieme di librerie condivise e file necessari per l'esecuzione.

La versione del *MATLAB Runtime* necessaria per poter eseguire *TDSFT* è la 2022b [10].

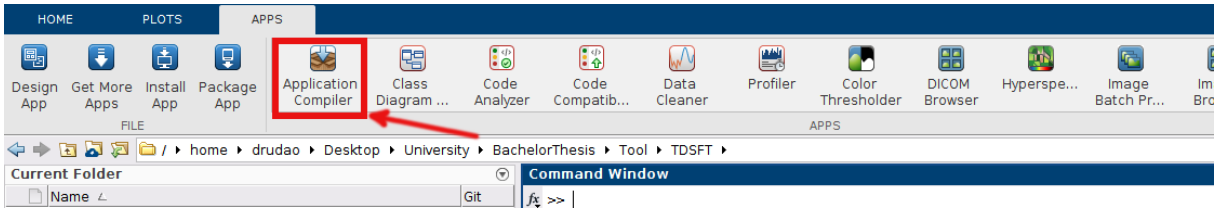

***Figura 2.4**: Il MATLAB Compiler si trova nella sezione "APPS" di MATLAB.*

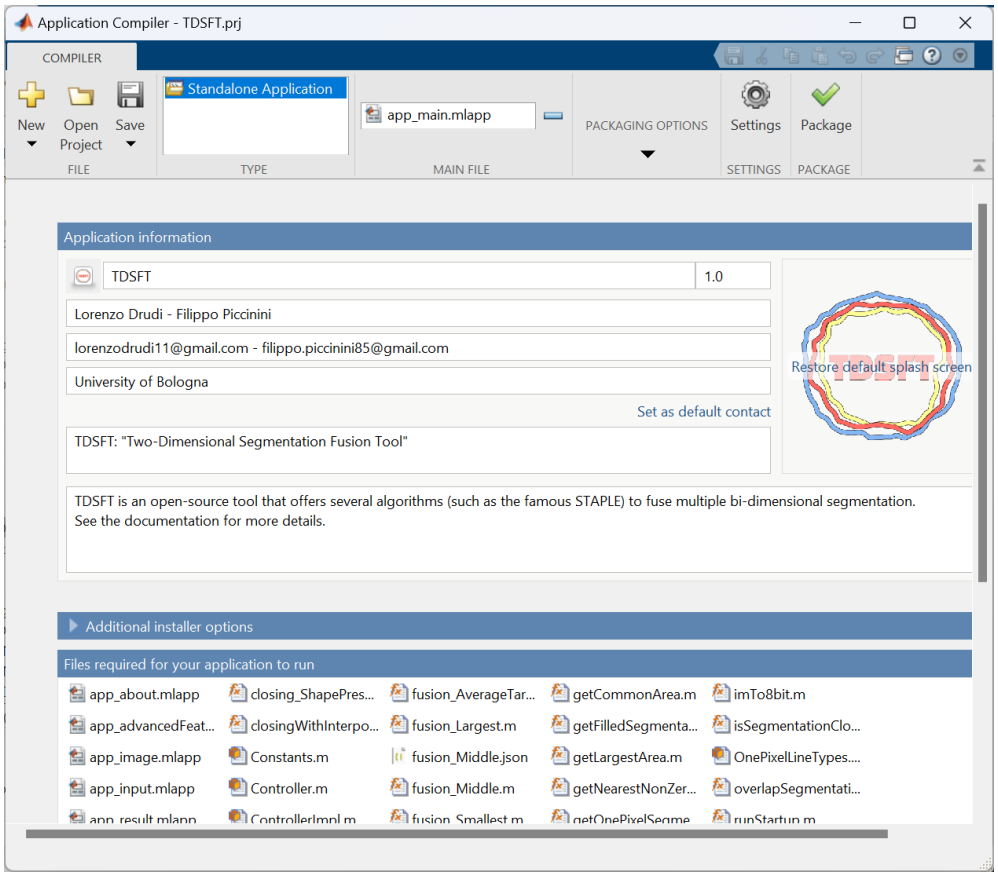

***Figura 2.5**: Interfaccia del MATLAB Compiler.*

3 – Descrizione algoritmi

Come detto precedentemente, l'attività di segmentazione, data la mancanza di standard nel delineare i bordi, è affetta da notevole soggettività. Perciò, data una immagine rappresentativa del tumore, è possibile che diversi operatori associno ad essa annotazioni differenti e le diagnosi prodotte potrebbero differire tra loro.

Date molteplici segmentazioni dello stesso tumore risulta quindi necessario determinare la più affidabile, da qui identificata come *ground truth* (verità di base). Una delle possibilità potrebbe essere lo scegliere l'annotazione prodotta dall'anatomopatologo con più esperienza, andando però a ignorare eventuali errori commessi da esso oltre che tutte le scelte effettuate dagli altri specialisti. Un metodo invece che considera tutte le segmentazioni è unirle tra loro andando a calcolarne la *fusione*.

La media però di curve chiuse, a differenza ad esempio della media tra due punti appartenenti ad un piano, non è un problema definito della geometria. Per questo motivo, nel corso del tempo, gli algoritmi proposti sono stati numerosi. Ad oggi però non esiste ancora un metodo validato standard ufficialmente riconosciuto.

Per le motivazioni precedentemente esposte, in questo progetto sono stati inclusi diversi algoritmi. A seguire, si può trovare una loro descrizione.

3.1 – Segmentazione più grande

Questo algoritmo consente di ottenere la *segmentazione più grande* (Largest), ossia la minima segmentazione che contiene tutte le segmentazioni in input.

Come prima cosa le segmentazioni in input vengono sovrapposte tra loro, cioè vengono sommate le relative matrici, successivamente vengono *riempiti i buchi* (hole-filling) ed infine viene calcolato il perimetro dell'area così ottenuta. La segmentazione più grande è quindi determinata come il margine dell'area totale coperta dalle annotazioni.

***Figura 3.1**: Diagramma di attività dell'algoritmo Largest.*

3.2 – Segmentazione più piccola

Questo algoritmo consente di ottenere la *segmentazione più piccola* (Smallest), ossia il perimetro dell'area comune a tutte le annotazioni in input.

Come prima cosa, per ogni segmentazione binaria, vengono riempiti i buchi e successivamente viene calcolata la loro sovrapposizione. Questo permette di ottenere l'area comune cercando tutti i pixel cui valore è uguale al numero di immagini in input. Ottenuta questa basterà poi calcolarne il perimetro.

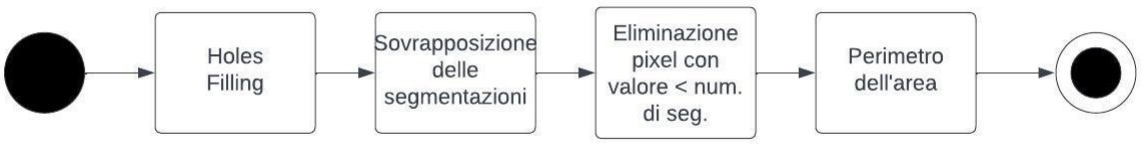

***Figura 3.2**: Diagramma di attività dell'algoritmo Smallest.*

3.3 – Media tra la più piccola e la più grande

Questo algoritmo calcola la *segmentazione media tra la segmentazione più grande e la segmentazione più piccola* (Average Smallest And Largest).

Come prima cosa vengono calcolate la segmentazione più piccola e la segmentazione più grande utilizzando gli algoritmi precedentemente descritti. Successivamente si ottiene l'area compresa tra le due sottraendo il riempimento della più grande al riempimento della più piccola. Infine, per ottenere la segmentazione di un pixel in mezzo a quest'area, viene applicata la *scheletrizzazione* (skeletonization), un'operazione di elaborazione di immagini che, effettuando una serie di passaggi successivi, identifica i bordi dell'oggetto e, nel caso in cui non incida sulla sua connettività, li rimuove. Questo viene ripetuto fino a che non si ottiene una linea di spessore di un pixel.

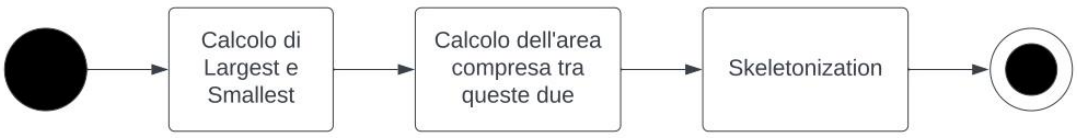

***Figura 3.3**: Diagramma di attività dell'algoritmo Average Smallest And Largest.*

3.4 – Media con segmentazione target

Di questo algoritmo sono state realizzate tre versioni che si differenziano per il modo in cui viene selezionata la segmentazione target, ossia la segmentazione da cui parte la computazione:

1. *Media con la segmentazione più grande* (Average Target Larget): il target è la segmentazione più grande, ottenuta con il metodo precedentemente enunciato;

2. *Media con la segmentazione più piccola* (Average Target Smallest): il target è la segmentazione più piccola, ottenuta anche essa con il metodo precedentemente enunciato;
3. *Media con una delle segmentazioni in input* (Average Target From Input): il target è una delle segmentazioni in input. La scelta è effettuata dal medico in fase di configurazione.

Dopodiché il procedimento è lo stesso per tutti e tre gli algoritmi. Date una serie di segmentazioni in input ed una segmentazione target, l'algoritmo itera su ogni pixel di quest'ultima e, per ognuno di questi, calcola il nuovo pixel della segmentazione media. Il procedimento per ottenere il nuovo pixel si può dividere in due fasi:

1. Dato il pixel della segmentazione target selezionato, viene calcolato il pixel più vicino ad esso di ogni altra segmentazione, ottenendo quindi un insieme di punti;
2. Quindi, dato l'insieme risultante dalla fase 1, il nuovo pixel è ottenuto come segue:
    - nel caso i punti siano due viene calcolato il punto medio del segmento avente questi come estremi;
    - nel caso in cui i punti siano di numero superiore a due ma appartengano alla stessa retta (punti collineari) vengono calcolati i due estremi ed anche in questo caso viene calcolato il punto medio del segmento;
    - negli altri casi invece viene calcolato il baricentro del poligono descritto dall'insieme di punti.

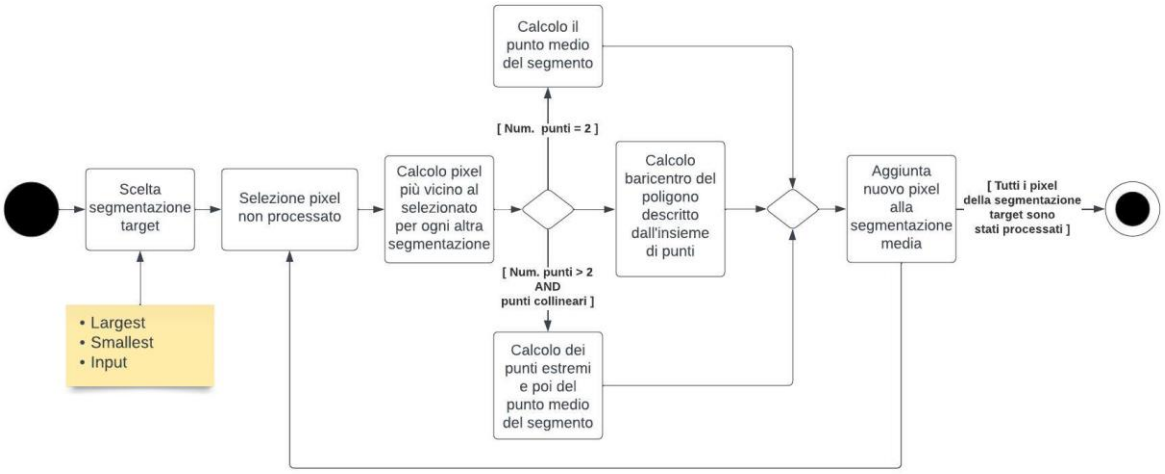

***Figura 3.4**: Diagramma di attività degli algoritmi di media con target.*

3.5 – Segmentazione centrale

Questo algoritmo consente di ottenere la *segmentazione centrale* (Middle), ossia la segmentazione compresa tra le annotazioni in input.

Il primo step è quello di sovrapporre tutte le segmentazioni in input. Successivamente l'algoritmo opera a passi successivi andando ad eliminare, ad ogni iterazione, la segmentazione più grande e la segmentazione più piccola dalla sovrapposizione precedentemente ottenuta. Il numero di iterazioni totali compiute è quindi:

$Tot.\ Iterazioni = \lfloor \frac{Num.seg-1}{2} \rfloor$.

Al termine delle iterazioni gli scenari possibili sono due:

- se il numero di segmentazioni in input era dispari allora al termine delle iterazioni rimarrà solamente una segmentazione, che sarà quindi la segmentazione centrale;
- se il numero di segmentazioni in input era pari, invece, le segmentazioni rimanenti sono due. Per ottenere il risultato il medico dovrà quindi specificare l'algoritmo da

applicare alle restanti annotazioni. Gli algoritmi messi a disposizione per questa fase sono:

- *Largest*;
- *Smallest*;
- *Average Smallest And Largest*.

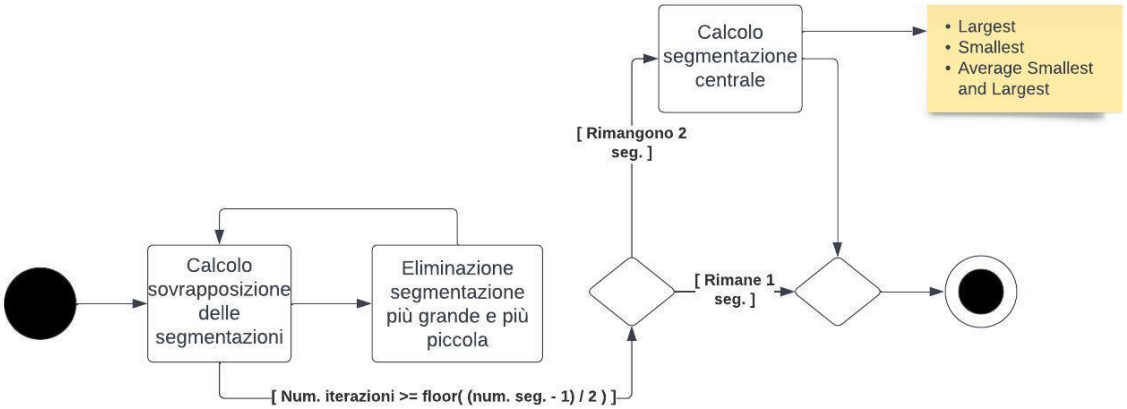

***Figura 3.5**: Diagramma di attività dell'algoritmo Middle.*

3.6 – STAPLE

Simultaneous Truth and Performance Level Estimation (STAPLE), proposto nel 2004 da Warfield *et al.*, è l'algoritmo di "*mediazione*" di segmentazioni binarie ad oggi più famoso e utilizzato **[11].**

STAPLE è un algoritmo di voto ponderato, ogni segmentazione ha quindi un peso associato che, in questo caso, ne riflette l'accuratezza. Inizialmente questo valore è lo stesso per tutte le segmentazioni in input.

Come prima cosa vengono combinate tutte le annotazioni in una singola segmentazione di test, votando semplicemente su ogni pixel. Successivamente *STAPLE* valuta l'accuratezza di ciascuna delle segmentazioni in input rispetto alla segmentazione precedentemente calcolata.

Quindi viene ridisegnata una nuova segmentazione di test, ponderando i voti delle segmentazioni in base alla loro accuratezza. Se una segmentazione risulta essere più accurata delle altre allora avrà peso maggiore. L'algoritmo procede iterando sino a che la variazione della segmentazione di test è al di sotto di un valore di soglia *epsilon* (ξ) o il numero massimo di iterazioni è raggiunto.

Al crescere del numero di segmentazioni la precisione di *STAPLE* aumenta. Inoltre, utilizzando un numero di annotazioni superiore a dieci, il risultato è considerato indipendente dai singoli contributi. Sempre relativamente al numero di immagini è importante dire che, nonostante funzioni anche con solamente due segmentazioni, è consigliato usarne almeno tre per riuscire a stimarne correttamente l'accuratezza.

Sebbene *STAPLE* sia l'algoritmo più utilizzato ha comunque alcuni difetti:

- come tutti gli algoritmi di voto a maggioranza tende a sottostimare i bordi;
- se lo stesso errore viene commesso da più annotatori questo avrà un peso importante e verrà incluso nella segmentazione finale.

L'implementazione utilizzata è presente al seguente link:

*https://www.mathworks.com/matlabcentral/fileexchange/56789-staple-d***.**

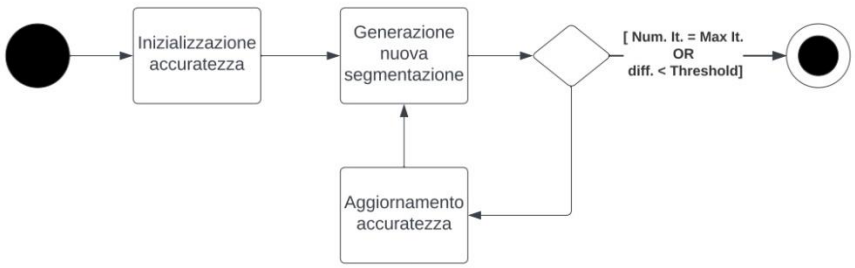

Figura 3.6*: Diagramma di attività dell'algoritmo STAPLE.*

4 – Descrizione TDSFT

TDSFT è un tool open-source, sviluppato in *MATLAB* e distribuito in forma di codice sorgente open-source ed in forma di applicazione standalone per *MAC*, *Linux*, e *Windows*. Offre un'interfaccia semplice ed estendibile in cui vengono proposti numerosi algoritmi per "*mediare*" molteplici segmentazioni.

***Figura 4.1**: Logo di TDSFT.*

TDSFT è scaricabile al seguente link: *https://sourceforge.net/p/tdsft.*

È inoltre possibile visualizzare lo sviluppo completo del progetto al seguente link: *https://github.com/UniBoDS4H/TDSFT***.**

4.1 – Architettura

L'architettura di TDSFT segue il pattern architetturale *Model-View-Controller* (MVC).

Questo pattern è basato sulla separazione dei compiti tra i componenti software che interpretano i tre ruoli principali:

- *model*: incapsula i dati e le loro funzionalità;
- *view*: si occupa dell'interazione con gli utenti e della presentazione dei dati;
- *controller*: riceve i comandi dell'utente dalla view e li attua.

È stata fatta questa scelta per favorire lo sviluppo indipendente delle componenti, il testing e la loro manutenzione.

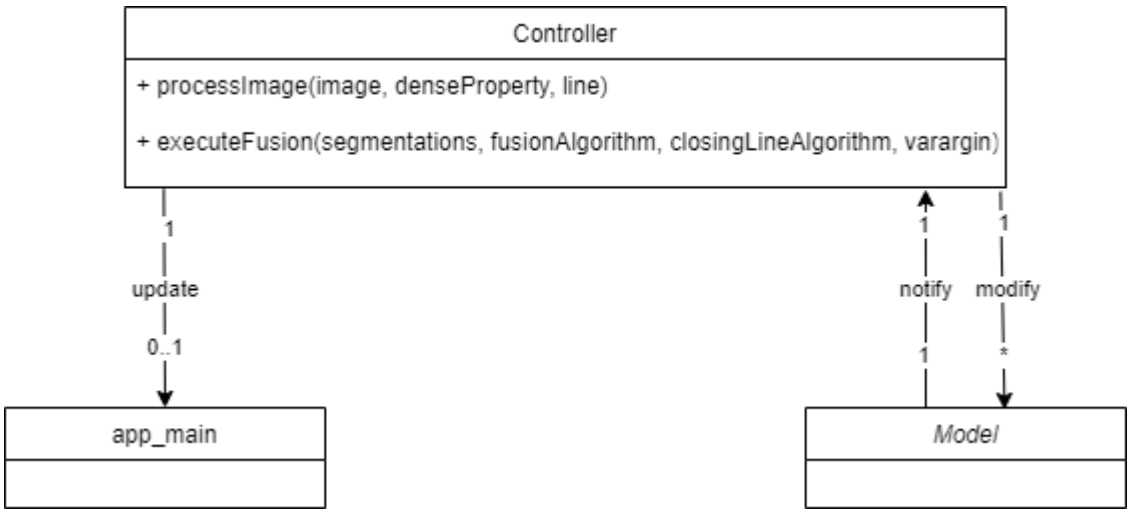

***Figura 4.2**: Diagramma delle classi raffigurante la divisione MVC di TDSFT.*

4.2 – Interfaccia utente

Per lo sviluppo dell'interfaccia utente è stato utilizzato lo strumento di *MATLAB* "*App Designer"* esposto nella **sezione 2.2**.

Durante l'intero processo di sviluppo il punto chiave tenuto a mente è stato la facilità di utilizzo. *TDSFT* è un tool sviluppato infatti per sostenere ed aiutare gli specialisti durante il loro lavoro.

4.2.1 – Finestra principale

TDSFT è progettato come applicazione *multi-finestra*, in app designer chiamate app. La finestra principale è "*app_main*", da essa si può comandare l'intero flusso verso le altre componenti.

Come prima cosa, tramite la finestra principale, l'utente può caricare le proprie segmentazioni utilizzando il bottone "*Upload*". *TDSFT* accetta sia oggetti densi (pieni) che oggetti già segmentati, sia con linea di un pixel che maggiore. È possibile trovare più dettagli sul caricamento delle immagini alla **sezione 4.3**.

È poi possibile selezionare un'immagine per volta per poter interagire con essa; è quindi possibile visualizzarla tramite il bottone "*Open*" e rimuoverla tramite il bottone "*Remove*".

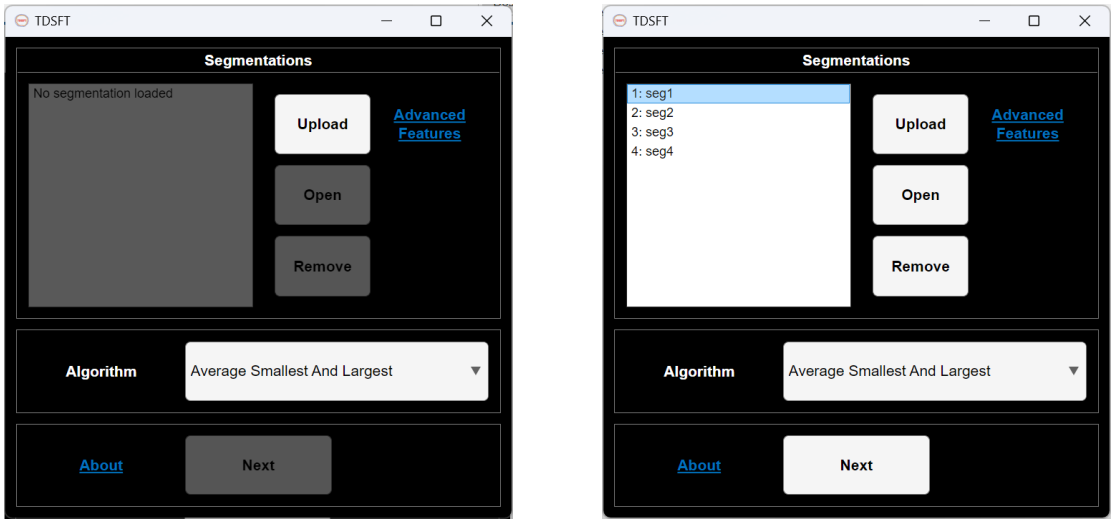

***Figura 4.3**: **In Figura a** è presente il tool senza nessuna segmentazione caricata. In **Figura b** invece sono presenti quattro segmentazioni caricate di cui una selezionata. È possibile notare, infatti, che i bottoni per la rimozione e per la visualizzazione sono attivati.*

***Figura 4.4**: In **Figura a** è presente un esempio di segmentazione con linea di un pixel mentre in **Figura b** è presente un esempio di oggetto denso (pieno).*

4.2.2 – Visualizzazione di una segmentazione

Come detto precedentemente, è possibile aprire ogni immagine caricata in una finestra separata (*app_image*) avente come titolo il nome della immagine stessa. Questa permette di visualizzare sia l'immagine originale convertita in bianco e nero che la segmentazione di un pixel ottenuta.

Data la segmentazione di un pixel, inoltre, è possibile, al fine solamente di una più agevole visualizzazione e tramite uno slider, specificare lo spessore della linea. Questa funzionalità è ottenuta mediante l'utilizzo di una tecnica chiamata *dilatazione* (dilation), operazione morfologica fondamentale che permette di aggiungere pixel ai bordi di un'immagine binaria.

È inoltre disponibile la funzione di zoom in/zoom out utilizzando la rotella del mouse o le gestures del touchpad.

***Figura 4.5**: In **Figura a** è presente la visualizzazione dell'immagine in bianco nero mentre in **Figura b** è presente la sua segmentazione di un pixel.*

4.2.3 – Fusione delle segmentazioni

Caricate almeno due immagini è possibile, tramite il pulsante "*Next*", eseguire la fusione delle segmentazioni utilizzando l'algoritmo selezionato. Quest'ultimo è specificato tramite il *menu a tendina* (drop-down menu) presente nella finestra principale e identificato dall'*etichetta* (label) "*Algorithm*". Per una descrizione degli algoritmi vedere il **Capitolo 3**.

Al termine della computazione verrà visualizzato il risultato in una finestra separata (*app_result*). Anche in questo caso è possibile, come visto precedentemente, applicare una dilatazione per meglio visualizzare il risultato e, in caso si voglia, salvare localmente la segmentazione ottenuta.

Inoltre, è possibile sovrapporre, al risultato ottenuto, le segmentazioni originali tramite la checkbox "*Show Originals*". Quando questa è attivata si visualizzerà in bianco la segmentazione finale mentre in grigio le segmentazioni in input. Inoltre, per evitare una visualizzazione confusionaria, lo slider verrà disabilitato.

Come per la visualizzazione delle immagini di input anche in questo caso è presente la funzione di zoom in/zoom out.

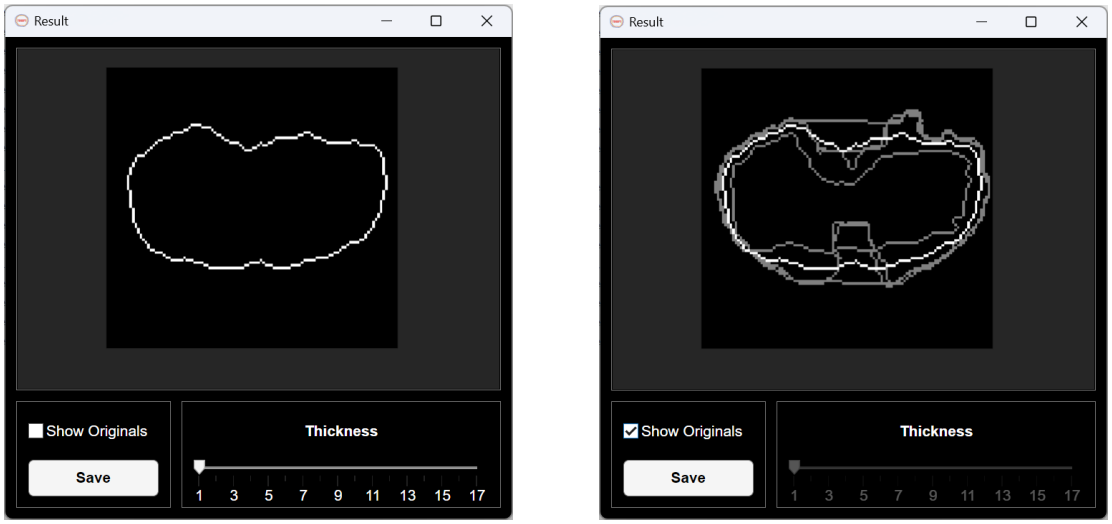

***Figura 4.6**: In **Figura a** è presente la visualizzazione base del risultato ottenuto mentre in **Figura b** a questo vengono sovrapposte anche le segmentazioni di partenza.*

4.2.4 – Algoritmi con parametri dall'utente

Sono presenti, inoltre, algoritmi che richiedono l'inserimento di alcuni parametri da parte dell'utente (*app_input*).

Il primo algoritmo che necessita di un ulteriore parametro è "*Average Target From Input*" (vedi **sezione 3.4**) nel quale è necessario inserire l'indice della segmentazione in input che si vuole utilizzare come target.

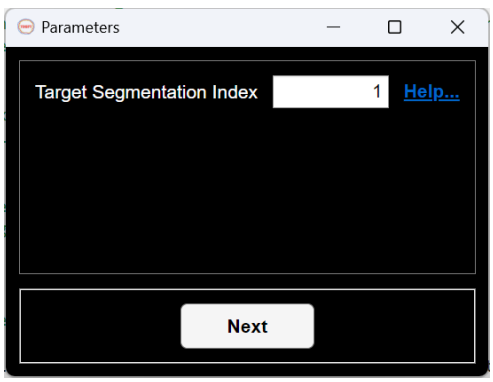

***Figura 4.7**: Finestra di inserimento dei parametri utente dell'algoritmo Average Target from Input.*

Inoltre, nel caso in cui il numero di segmentazioni in input sia pari, l'algoritmo "*Middle"* (vedi **sezione 3.5**) necessita della scelta dell'algoritmo da utilizzare per unire le restanti due annotazioni.

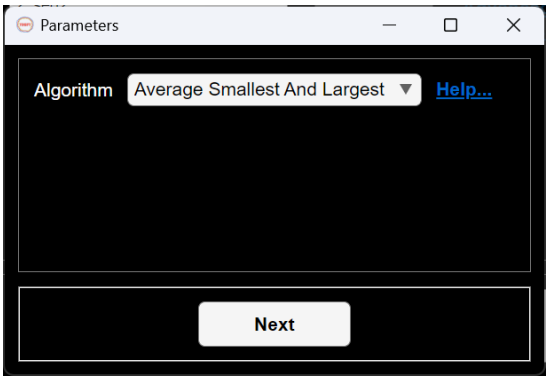

***Figura 4.8**: Finestra di inserimento dei parametri utente dell'algoritmo Middle nel caso in cui il numero di segmentazioni in input sia pari.*

A fianco di ogni parametro è presente un link "*Help…*" che è possibile premere per ottenere maggiori informazioni sull'utilizzo.

Finito l'inserimento dei valori, tramite il pulsate "*Next*", è possibile eseguire la fusione.

4.2.5 – Funzionalità avanzate

Sono inoltre presenti funzionalità avanzate selezionabili premendo sul link "*Advanced Features*" (*app_advancedFeatures*). Una volta selezionati, i parametri scelti, saranno immediatamente disponibili. Inoltre, non è necessario utilizzare sempre gli stessi valori ma è possibile modificarli durante il processo. È ad esempio possibile caricare alcune immagini usando una determinata configurazione e successivamente caricare altre immagini usando parametri differenti.

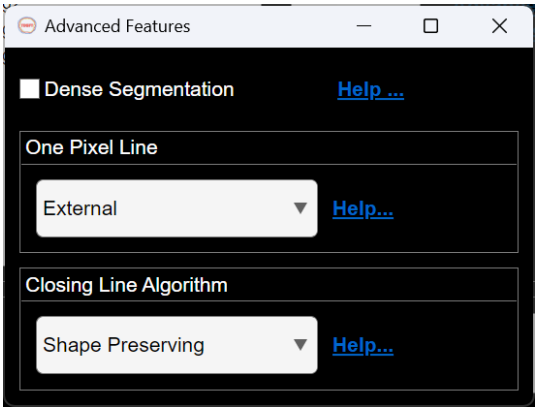

***Figura 4.9**: Finestra delle funzionalità avanzate.*

La checkbox "*Dense Segmentation*" consente di aggiungere, in fase di caricamento, ulteriori test nel caso in cui l'oggetto contenuto nell'immagine caricata sia denso. È consigliato selezionare questa funzione solamente nel caso in cui si necessiti veramente di un controllo di questo tipo. È infatti impossibile differenziare, in maniera assoluta, una segmentazione con linea aperta di spessore di più di un pixel da una segmentazione densa. Entrambi, infatti, sono a tutti gli effetti oggetti densi. Il controllo che *TDSFT* effettua è un test di tipo puramente statistico nel quale viene controllato che il numero di pixel che compongono l'oggetto denso sia almeno il doppio del numero di pixel che compongono il suo perimetro. Nel caso ciò sia vero allora l'oggetto presente nell'immagine è un oggetto denso altrimenti è una segmentazione aperta con linea di più pixel.

Tramite il drop down menu "*One Pixel Line*", è possibile poi selezionare il tipo di linea da usare nel caso in cui venga caricata una segmentazione con linea di spessore maggiore di un pixel. Le opzioni disponibili sono:

- *Linea Esterna* (external): data una linea composta da più pixel la linea esterna è la linea di un pixel più esterna, quindi rivolta verso i bordi dell'immagine. È il valore di default di *TDSFT*;

- *Linea Interna* (internal): data una linea composta da più pixel viene selezionata la linea di un pixel interna, cioè verso il centro dell'immagine;
- *Linea Centrale* (middle): data una linea composta da più pixel viene selezionata la linea intermedia. Nel caso in cui lo spessore sia di soli due pixel la linea usata sarà quella interna. Questa funzione è ottenuta utilizzando l'operazione morfologica skeletonization.

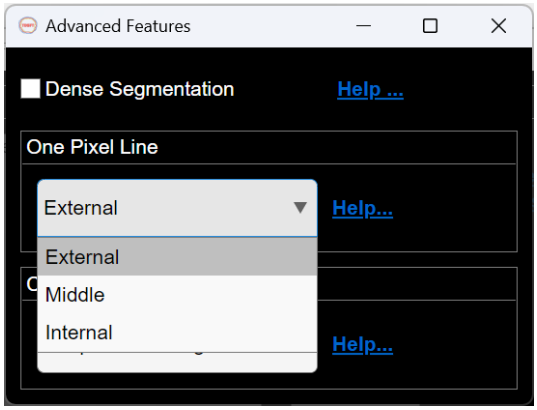

***Figura 4.10**: Funzione "One Pixel Line".*

È importante evidenziare il fatto che, quando è selezionata la checkbox "*Dense Segmentation"*, non è possibile scegliere il tipo di linea. Questo perché se un oggetto è pieno l'unica segmentazione di un pixel possibile è quella che coincide con il suo perimetro. Inoltre, nel caso in cui sia selezionato un tipo diverso da "*external*" e si provi a caricare un oggetto denso, il caricamento darà esito negativo mostrando un messaggio d'errore all'utente.

Infine, tramite il drop down menu "*Closing Line Algorithm*", è possibile selezionare l'algoritmo di chiusura. È infatti possibile che la segmentazione finale non sia già una linea chiusa ma che sia necessario applicare un algoritmo specifico per poter ottenere il risultato finale. Questa situazione è molto comune nel caso in cui venga utilizzato l'algoritmo "*STAPLE*" (**sezione 3.6**) con poche segmentazioni o nel caso in cui vengano utilizzati gli algoritmi della famiglia "*Media con Segmentazione Target*" (**sezione 3.4**).

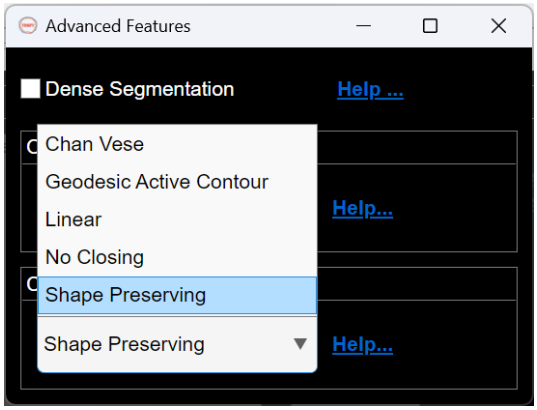

***Figura 4.11**: Funzione "Closing Line Algorithm".*

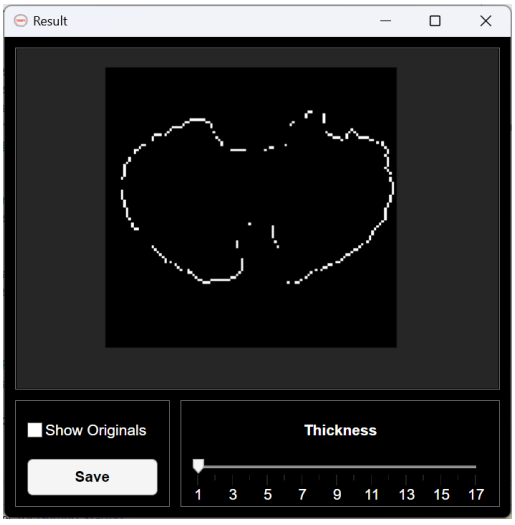

***Figura 4.12**: Esempio di segmentazione finale non chiusa.*

Gli algoritmi di chiusura proposti si dividono in due macro-gruppi:

1. *Algoritmi di contorno attivo* (active contour, or snakes): classe di algoritmi utilizzati per delineare il contorno di un oggetto partendo da una immagine bidimensionale eventualmente rumorosa. Sono basati sul concetto di energia, in cui il contorno è considerato come una superficie elastica che cerca di adattarsi al bordo dell'oggetto desiderato. L'obiettivo di questi algoritmi è di trovare il contorno ottimale che si adatta ai bordi degli oggetti, minimizzando l'energia complessiva. L'energia è definita come una combinazione di termini che misurano l'adesione del contorno ai bordi desiderati, la regolarità della sua forma e l'interazione tra il contorno e l'immagine. Non

risolvono l'intero problema di ricerca dei contorni in quanto richiedono una conoscenza preliminare della forma del bordo desiderato;

2. *Algoritmi di interpolazione*: i metodi appartenenti a questa classe nascono dalla seguente idea: ogni pixel della segmentazione, affinché essa sia chiusa, deve essere connesso ad altri due punti. Perciò, ogni punto dell'annotazione non connesso ad altri due pixel, deve essere connesso al numero di punti vicini necessari per ottenere la segmentazione chiusa. La principale difficoltà riscontrata però deriva proprio dall'individuazione dei punti "vicini": sono infatti presenti molte situazioni in cui non è possibile effettuare questa operazione in maniera semplice ed univoca (**Figura 4.13**). Per questo motivo la scelta è stata quella di convertire tutte le coordinate cartesiane (x, y) dei pixel in coordinate polari, sistema nel quale ogni punto del piano è identificato da una distanza ρ e da un angolo θ. Successivamente le nuove coordinate vengono traslate affinché il punto di riferimento sia il punto centrale della segmentazione più grande (*Largest*, **Sezione 3.1**). Questo permette di ordinare i punti utilizzando le posizioni angolari e quindi determinare i punti vicini tramite questa informazione. Una volta effettuato l'ordinamento dei punti in coordinate polari è poi possibile applicare diversi algoritmi di interpolazione nello stesso spazio per ottenere la segmentazione chiusa. Come punto di riferimento viene utilizzato il centro della segmentazione più grande in quanto si necessita del punto centrale della segmentazione ed il centro dell'immagine non sarebbe corretto in quanto è possibile che le annotazioni non siano perfettamente centrate. Una volta ottenuto l'insieme di punti questo viene convertito in coordinate cartesiane per identificare i pixel facenti parte della segmentazione.

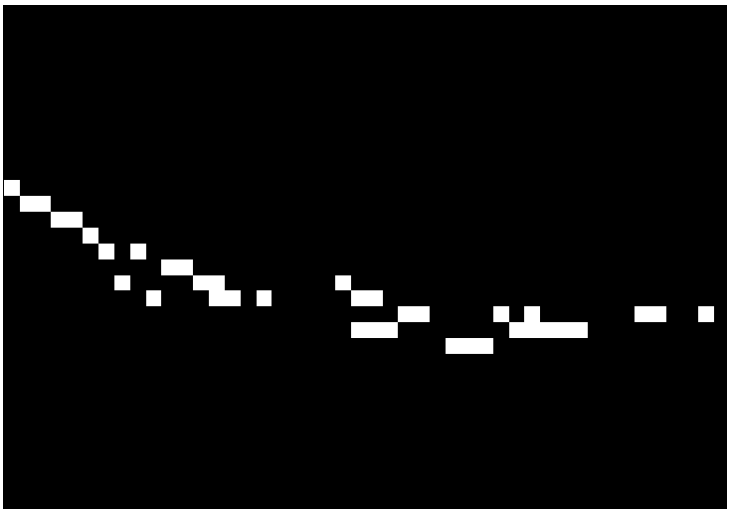

***Figura 4.13**: Esempio nel quale è complesso determinare i punti vicini.*

```
% Get the centroid of the largest segmentation.
largest = fusion_Largest(inputSegmentations);
cn = regionprops(largest, "Centroid").Centroid; % centroid
s = size(fusionResult);

% Find locations of outline pixels.
[idxy, idxx] = find(fusionResult); % in cart coord
[idxth, idxr] = cart2pol(idxx-cn(1), idxy-cn(2)); % in polar coord (theta, rho)

% Sort pixel locations by angular position with reference to largest segmentation center
[idxth, sortmap] = sort(idxth, "ascend");
idxr = idxr(sortmap);

% Query points for interpolation.
nQueryPoints = 100000;
newth = linspace(-pi, pi, nQueryPoints).';

% Use the specified interpolation method to interpolate the missing pixels.
% Specify Nan as the value for query points outside the domain.
newr = interp1(idxth, idxr, newth, method, NaN);

% Remove NaNs, interpolation method produce NaNs for query points outside the domain.
nanp = ~isnan(newr);
newr = newr(nanp);
newth = newth(nanp);

% Construct output image.
[newx, newy] = pol2cart(newth,newr);
res = false(s);

res(sub2ind(s, round(newy + cn(2)), round(newx + cn(1)))) = true;
```

***Codice 4.1**: Funzione base degli algoritmi di chiusura che utilizzando l'Interpolazione (funzione ClosingWithInterpolation).*

In **Codice 4.1** è mostrato il procedimento precedentemente descritto. Nelle prime righe viene ottenuto il baricentro della segmentazione più grande e successivamente le coordinate cartesiane sono convertite in polari utilizzando come riferimento proprio il centroide

precedentemente ottenuto. Fatto ciò, i punti sono ordinati in modo crescente e successivamente è applicato il metodo di interpolazione specificato (vedi i paragrafi successivi) usando la funzione di *MATLAB interp1()*. Infine, vengono prima rimossi i punti fuori dal dominio e poi vengono riconvertite le coordinate in cartesiane effettuando la traslazione inversa a quella effettuata inizialmente per ordinare i punti usando come punto di riferimento il centro della segmentazione più grande.

Tra gli algoritmi del primo gruppo, ossia tra gli algoritmi di *active contour*, sono presenti:

- *Metodo Geodetico* (Geodesic Active Contour) **[12]**: questo approccio consente di collegare i classici metodi di contorno attivo basati sulla minimizzazione dell'energia ai contorni attivi geometrici basati sulla teoria dell'evoluzione delle curve. Si basa su contorni attivi che si evolvono nel tempo in base a misure geometriche intrinseche dell'immagine;
- *Metodo Chan-Vese* **[13]**: modello di contorno attivo che non si basa su una funzione bordo per fermare la curva in evoluzione sul confine desiderato. Con questo metodo è possibile rilevare oggetti i cui confini non sono necessariamente definiti dal gradiente o con confini molto lisci, per i quali i metodi classici di contorno attivo non sono applicabili.

Invece, i metodi di *interpolazione* offerti sono:

- *Interpolazione lineare* (Linear): ogni coppia di punti adiacenti è semplicemente unita da un segmento che può essere calcolato in maniera indipendente dagli altri. Se denotiamo con $(x_{i,}\, y_i)$ e $(x_{i+1,}\, y_{i+1})$ la coppia di punti adiacenti, la funzione interpolante è definita come: $f_i(x) = \frac{x_{i+1} - x}{x_{i+1} - x_i} * y_i + \frac{x - x_i}{x_{i+1} - x_i} * y_{i+1}$;

- *Interpolazione cubica a tratti che preserva la forma* (Shape Preserving) **[14] [15]**: è il metodo di default di *TDSFT*. Viene eseguita l'interpolazione utilizzando un polinomio a tratti cubico *P(x)* con le seguenti proprietà:
    - per ogni sotto-intervallo $x_k \leq x \leq x_{k+1}$, il polinomio *P(x)* è un polinomio cubico di *Hermite* che interpola i punti dati con le derivate specificate nei punti di interpolazione;
    - la derivata prima di *P(x)* è continua mentre la derivata seconda è probabilmente non continua, per cui sono possibili dei salti;
    - il polinomio *P(x)* preserva la forma, per ottenere ciò le pendenze nei punti x_j sono scelte in modo tale che venga preservata la forma dei dati e la monotonicità. Pertanto, sugli intervalli in cui i dati sono monotoni, lo sono anche per *P(x)*, e nei punti in cui i dati hanno un estremo locale, lo hanno anche per *P(x)*.

È possibile infine selezionare "*No Closing*" per visualizzare il risultato senza nessun algoritmo di chiusura applicato.

4.3 – Caricamento delle immagini

Un punto fondamentale di *TDSFT*, oltre agli algoritmi descritti al **Capitolo 3**, è una corretta conversione delle immagini in input nel formato desiderato. Analizzando il codice, i controlli che vengono effettuati su ogni immagine in input sono numerosi.

```
% Convert to 8 bit image.
cImg = imTo8bit(img);

% Convert to grayscale.
grayImg = im2gray(cImg);

% Convert to black and white.
blackAndWhiteImg = imbinarize(grayImg);

% Convert to black background if needed.
% We want the background to be black and the segmentation to be white.
if getBinaryImageBackground(blackAndWhiteImg)
    blackAndWhiteImg = ~blackAndWhiteImg;
end

% If there are more than one object, select the largest one.
[~, numBlobs] = bwlabel(blackAndWhiteImg);
if numBlobs > 1
    blackAndWhiteImg = bwareafilt(blackAndWhiteImg, 1);
end

% Check if the segmentation is an open line.
if ~isSegmentationClosed(blackAndWhiteImg, dense)
    throw(MException("TDSFT:processImage", "Not closed segmentation uploaded"));
end

% Check if the segmentation is empty.
if ~sum(blackAndWhiteImg(:))
    throw(MException("TDSFT:processImage", "Empty segmentation uploaded"));
end

% Check if the segmentation is already of one pixel.
if isequal(blackAndWhiteImg, bwperim(blackAndWhiteImg))
    segmentation = blackAndWhiteImg;
else
    % Get the one pixel segmentation.
    segmentation = getOnePixelSegmentation(blackAndWhiteImg, line);
end
```

***Codice 4.2**: Blocco relativo alla conversione dell'immagine (metodo processImage() della classe ControllerImpl).*

Come prima cosa viene analizzata la *profondità di colore*, ossia il numero di bit utilizzati per rappresentare il colore di ogni singolo pixel. *TDSFT* accetta immagini a 8, 12, 16, 32 e 64 bit che poi converte a 8 bit (**Codice 4.3**). Per fare questo è stata scritta una funzione apposita in quanto il metodo che mette a disposizione *MATLAB* supporta solamente immagini con profondità a 16 bit. Ciò che viene fatto è controllare il valore di ogni pixel e successivamente mapparlo nell'intervallo desiderato [0, 255], ossia il range di valori disponibili per immagini a 8 bit. Da evidenziare che le immagini a 12 bit sono comunque memorizzate come immagini a 16 bit. Per questo motivo viene controllato se il valore massimo assunto dai pixel dell'immagine supera il valore 2^{12-1}, ossia il massimo valore assunto dalle immagini a 12 bit, e solamente nel caso in cui la condizione risulti vera allora l'immagine sarà etichettata come a 16 bit. Inoltre, i formati di immagine supportati sono: *jpg*, *png*, *jpeg*, *bpm*, *tif*, *tiff* e *gif*.

```
if isa(img, ImagesStoringMethods.INT_16.string)
    % 16-bit
    if max(img(:)) > 2^12 % 12-bit images are stored as 16-bit
        cImg  = uint8( 255.*double(img)./(2^16-1) );
    % 12-bit
    else
        cImg  = uint8( 255.*double(img)./(2^12-1) );
    end
% 32-bit
elseif isa(img, ImagesStoringMethods.INT_32.string)
    cImg  = uint8( 255.*double(img)./(2^32-1) );
% 64-bit
elseif isa(img, ImagesStoringMethods.INT_64.string)
    cImg  = uint8( 255.*double(img)./(2^64-1) );
else
    cImg = img;
end
```

***Codice 4.3**: Blocco relativo alla conversione a 8-bit (funzione imTo8bit()).*

Successivamente, l'immagine viene convertita prima ad immagine in *scala di grigi* (grayscale), e successivamente ad immagine binaria, in cui gli unici valori possibili sono quindi 0 oppure 1. Per fare questo sono state utilizzate le funzioni già disponibili in *MATLAB*, *im2gray()* e *imbinarize()*. La prima elimina tonalità e saturazione mantenendo la luminosità mentre la seconda utilizza il metodo di sogliatura automatica *Otsu* **[16]** per ottenere la sua rappresentazione logica.

Ottenuta l'immagine binaria, viene poi controllato che lo sfondo sia di colore nero e l'oggetto di colore bianco, altrimenti i colori vengono invertiti. Questo è ottenuto tramite la funzione *getBinaryImageBackground()* (**Codice 4.4**) che calcola, per ogni lato dell'immagine, la somma dei valori dei pixel, che poi unisce tramite un *OR* logico. Nel caso in cui tutte le somme siano zero, allora anche l'operazione logica avrà questo risultato e ciò implicherà che lo sfondo dell'immagine sia nero, altrimenti i colori dovranno essere invertiti.

```
[h, w] = size(img);

% Get the sum of the first and last row and column.
firstRow = sum( img(1, :) );
lastRow = sum( img(h, :) );
firstCol = sum( img(:, 1) );
lastCol = sum( img(:, w) );

% Since objects can't touch the border, if they are all 0, the background is black.
bg = firstRow || lastRow || firstCol || lastCol;
```

Codice 4.4: *Funzione getBinaryImageBackground().*

Successivamente, nel caso in cui sia presente più di un oggetto, viene mantenuto quello avente area maggiore. Per fare questo sono state utilizzate le funzione built-in *bwlabel()* e *bwareafilt()*, appartenenti al "*Image Processing Toolbox"*. Infine, viene controllato che la segmentazione sia effettivamente chiusa (vedi anche la **sezione 4.2.5**) e che l'immagine contenga un oggetto, quindi che non sia vuota.

Il controllo della segmentazione chiusa è uno dei passaggi più importanti in quanto, con linee aperte, gli algoritmi non funzionerebbero correttamente. Esso è effettuato tramite la funzione *isSegmentationClosed()* (**Codice 4.5**). Come prima cosa viene controllata la presenza di buchi da riempire, e nel caso in cui essi siano presenti, allora significa che l'immagine contiene effettivamente una segmentazione chiusa. Se invece ciò non si verifica le opzioni sono tre:

- l'immagine contiene una segmentazione aperta con linea di un pixel;
- l'immagine contiene un oggetto denso;
- l'immagine contiene una segmentazione aperta ma con linea di spessore di più di un pixel.

Il primo caso è facilmente riconoscibile, infatti se il perimetro e l'oggetto stesso coincidono significa che la segmentazione è aperta. Per i restanti due casi, invece, è necessario effettuare ulteriori controlli ma, come detto anche in **sezione 4.2.5**, ciò non è semplice. Dato che una linea di più pixel si può vedere esattamente come un oggetto denso, di default entrambi sono considerati corretti. Altrimenti è possibile utilizzare l'ulteriore controllo reso disponibile dalla funzionalità avanzata "*Dense Segmentation*".

```
imgFill = imfill(img, "holes");

% If the image can be filled it means the object in the image is not den
% and there is a hole to fill so the segmentation is closed.
if ~isequal(img, imgFill)
    check = true;
else
    % Here there are two options:
    % 1) The object in the image is a dense object;
    % 2) The object in the image is an open line of more than one pixel.

    % Check if it is an open segmentation of 1 pixel.
    perim = bwperim(img);
    check = ~isequal(perim, img);

    if ~check || ~flag
        return;
    end

    % If dense option is true do a statistical test to check to
    % recognize a dense object from an open line of more than one pixel.
    fillCells = nnz(imgFill);
    perimCells = nnz(perim);

    check = fillCells > 2 * perimCells;
end
```

***Codice 4.5**: Blocco relativo al controllo della segmentazione chiusa (funzione isSegmentationClosed()).*

Come ultimo step dell'intero processo di conversione viene calcolata la segmentazione di un pixel usando il tipo di linea specificato (per maggiori dettagli sui tipi disponibili vedere la **sezione 4.2.5** al paragrafo "*One Pixel Line*"). La funzione implementata per questo compito è *getOnePixelSegmentation*() (**Codice 4.6**). Analizzando il codice è importante dire che, nel caso in cui l'oggetto sia denso e il tipo di linea selezionato sia diverso da "*external*", allora viene lanciata un'eccezione. Come illustrato anche precedentemente, infatti, un oggetto denso può utilizzare solamente la linea di tipo esterno. Per il calcolo della linea centrale inoltre è stata utilizzata l'operazione di scheletrizzazione.

```
% Fill the holes.
segFill = imfill(seg, "holes");

% If the segmentation is dense, only the external line is allowed.
if isequal(seg, segFill) && line ~= "external"
    throw(MException("TDSFT:processImage", ...
        "Dense object with line type different from external."));
end

% Get the one-pixel segmentation using the specified line type.
switch line
    case "internal"
        intArea = segFill - seg;
        opSeg = bwperim(intArea);
    case "middle"
        opSeg = bwskel(seg);
        opSeg = imfill(opSeg, "holes");
        opSeg = bwperim(opSeg);
    % Default is external line.
    otherwise
        opSeg = bwperim(segFill);
end
```

***Codice 4.6**: Blocco relativo al calcolo della segmentazione di un pixel (funzione getOnePixelSegmentation()).*

4.4 – Estendibilità

Come detto precedentemente, l'estendibilità è una caratteristica fondamentale di *TDSFT*. Infatti, non esistendo uno standard nel settore ed essendo la ricerca molto attiva, *TDSFT* è stato progettato per poter permettere all'utente di aggiungere i propri algoritmi. A seguire è presente una descrizione di come è stato sviluppato il tool al fine di essere estendibile.

Diverse sono le scelte che hanno permesso la realizzazione di questa caratteristica. Come prima cosa tutti gli algoritmi sono sviluppati come funzioni di *MATLAB* che seguono lo stesso template, questo è presente all'interno della cartella *template* con il nome "*fusion_fusionAlgorithmTemplate.m"*. Le funzioni sono tutte contenute all'interno della cartella "*api/fusionAlgorithms*" ed i nomi dei file iniziano tutti con la radice "*fusion_*". A tempo di esecuzione vengono caricati come algoritmi tutti i file che rispettano queste regole

```
function fusionResult = fusion_fusionAlgorithmTemplate(segmentations)
    % Insert here your code
end
```

***Codice 4.7**: Template degli algoritmi di fusione.*

Tutti gli algoritmi prendono in input come primo parametro un cell-array di nome "*segmentations*" in cui è presente una cella per ogni segmentazione (1 × n. seg.) ed ognuna di queste contiene una matrice righe × colonne che descrive la segmentazione stessa.

```
function fusionResult = fusion_fusionAlgorithmTemplate(segmentations, arg1, arg2)
    % Insert here your code
end
```

***Codice 4.8**: Esempio di algoritmo di fusione con due argomenti aggiuntivi arg1 e arg2.*

Nel caso in cui siano necessari ulteriori parametri inseriti dall'utente (**Codice 4.8**) è possibile predisporre, in modo semplice, una interfaccia grafica dedicata. Per fare questo, prima di tutto, è necessario creare, all'interno della cartella *"api/fusionAlgorithms/inputs"*, un file con estensione *JSON* con nome uguale all'algoritmo per il quale si vuole predisporre l'interfaccia. Questo file conterrà le descrizioni dei componenti grafici necessari per raccogliere i parametri. I componenti messi a disposizione sono:

- *Drop down menu* (DropDown);
- *Checkbox* (Check);
- *Casella di testo* (Text);
- *Selezionatore di una segmentazione di input* (InputSegmentationsSelector);
- *Casella di testo numerica* (Number).

```json
{
    "inputs": [
        {
            "name": "Test DropDown",
            "type": "DropDown",
            "value": [
                "Value 1",
                "Value 2"
            ],
            "help": "This is a help text for the dropdown"
        },
        {
            "name": "Test Check",
            "type": "Check",
            "help": "This is a help text for the check"
        },
        {
            "name": "Test Text",
            "type": "Text",
            "help": "This is a help text for the text"
        },
        {
            "name": "Target Segmentation Index",
            "type": "InputSegmentationsSelector",
            "help": "This is a help text for the text"
        },
        {
            "name": "Test Numeric Text",
            "type": "Number",
            "value": 1,
            "limits": [1, 5],
            "help": "This is a help text for the text"
        }
    ]
}
```

***Codice 4.9**: Elenco ed esempio di tutti i componenti messi a disposizione.*

Come si può osservare in **Codice 4.9** il file *JSON* contiene un array di nome "*inputs*" ed ogni oggetto in esso contenuto è un componente necessario a raccogliere un determinato parametro. All'interno della cartella *template* è presente il file "*ComponentsOverview.json"* che contiene tutti i componenti disponibili. Le proprietà comuni a tutti sono:

- "*name*": il nome del componente;
- "*type*": il tipo di componente;
- "*help*": il testo mostrato nella finestra di dialogo quando l'utente preme il link "Help…" (vedi **sezione 4.2.4**).

Inoltre:

- il componente di tipo "*DropDown*" contiene la proprietà "*value*", in cui è presente l'elenco di tutti i valori mostrati nel menu a tendina;
- il componente di tipo "*Number*" contiene le proprietà "*value*" e "limits", che indicano rispettivamente il valore di default della casella di testo e gli estremi del range di valori in cui è possibile scegliere il numero.

Tutti i parametri sono passati all'algoritmo seguendo l'ordine con il quale sono inseriti i rispettivi componenti all'interno del file *JSON*.

Questa funzionalità non è disponibile solamente per gli algoritmi di fusione, bensì è possibile inserire anche algoritmi di chiusura personalizzati. In questo caso tutte le funzioni si trovano all'interno della cartella "*api/closingAlgorithms*" ed i nomi dei file iniziano con la radice "*closing_*". Anche questo file di template è presente nella cartella già citata precedentemente con il nome "*closing_closingLineAlgorithmTemplate.m*".

```
function res = closing_closingLineAlgorithmTemplate(fusionResult, inputSegmentations)
    % Insert here your code
end
```

***Codice 4.10**: Template degli algoritmi di chiusura.*

In questo caso non è possibile passare parametri aggiuntivi oltre quelli già messi a disposizione dal template. In particolare, i parametri disponibili sono:

- *fusionResult*: matrice righe × colonne che descrive la segmentazione che si vuole chiudere;
- *inputSegmentations*: cell-array contenente tutte le annotazioni di input.

È però da specificare che questa funzionalità è disponibile solamente se il tool viene eseguito dentro l'ambiente di *MATLAB*. Questo è dato dal fatto che una applicazione standalone può

utilizzare solamente le funzioni aggiunte in fase di compilazione. Per aggiungere i propri metodi è quindi necessario utilizzare i sorgenti di *TDSFT*. Le opzioni per chi vuole utilizzare questa feature sono quindi due: creare una nuova standalone con i propri file utilizzando il *MATLAB Compiler* oppure eseguire il tool direttamente dentro l'ambiente di *MATLAB*. Per quest'ultima modalità, al fine di una corretta esecuzione, è necessario trovarsi all'interno della cartella principale del progetto.

5 – Risultati sperimentali

Per testare e presentare *TDSFT* sono stati compiuti studi sperimentali utilizzando un dataset preso dall'articolo di presentazione del tool *3D-Cell-Annotator* **[8]** contenente un insieme di sferoidi fotografati tramite *microscopia a foglio di luce* (Light-Sheet Fluorescence Microscope, LSFM). In particolare, il dataset contiene 52 sferoidi tridimensionali e, per ottenere immagini bidimensionali da elaborare, è stato seguito il processo esposto in **sezione 1.2**. In questo specifico caso sono state selezionate casualmente tre sezioni dello *z-stack* e, in ognuna di queste, è stato selezionato casualmente uno sferoide.

Per ogni sferoide il dataset utilizzato contiene sette segmentazioni. Due di queste sono state ottenute tramite processo manuale da due annotatori diversi: il primo di questi è un esperto del settore con molti anni di pratica mentre il secondo annotatore è un ricercatore con meno anni di esperienza. Sono poi presenti quattro segmentazioni ottenute con diversi tool di segmentazione automatica:

- *MINS* **[17]** (versione 1.3): è l'acronimo di *Modular Interactive Nuclear Segmentation* ed è uno strumento di segmentazione basato su *MATLAB/C++* sviluppato per il conteggio delle cellule e per la misurazione dell'intensità di fluorescenza di dati di immagini 2D e 3D;
- *Pagita* **[18]** (versione 2.2): è un algoritmo automatico per segmentare e classificare simultaneamente cellule e nuclei in immagini 3D e 4D;
- *XPIWITGUI* **[19]** (versione 1.0): è l'acronimo di *XML Pipeline Wizard for ITK*, si tratta di un'applicazione wrapper basata su *XML* per l'*Insight Toolkit* che combina le prestazioni di un'implementazione *C++* pura con una interfaccia grafica facile da usare;

- *OpenSegSPIM* **[20]** (versione 1.1): è un tool open-source e user-friendly sviluppato in *MATLAB* per l'analisi quantitativa automatica 3D per dati confocali, multi-fotone e *LSFM*.

Infine, è presente la segmentazione ottenuta con *3D-Cell-Annotator* **[8]**, uno strumento open-source per la segmentazione di singole cellule in immagini 3D di microscopia. La segmentazione prodotta dall'annotatore esperto è stata utilizzata come *ground truth* mentre le restanti sei sono state utilizzate come input per *TDSFT*.

Come indice statistico per valutare i risultati ottenuti è stato utilizzato l'*Indice di Jaccard*, un indice statistico usato per confrontare la similarità e la diversità di insiemi campionari. Questo indice è definito come la dimensione dell'intersezione diviso la dimensione dell'unione $J(A,B) = \frac{|A \cap B|}{|A \cup B|}$, per questo motivo è anche chiamato *Intersection Over Union* (IoU). Applicato alle segmentazioni binarie, l'intersezione è composta da tutti i pixel comuni alle due annotazioni mentre l'unione è composta dall'unione dei pixel delle due immagini. L'indice può assumere un valore compreso tra 0 e 1, dove 1 indica la perfetta similarità. Come detto precedentemente l'annotazione A è stata mantenuta costante, e corrisponde all'annotatore esperto, mentre come segmentazioni B sono stati utilizzati, a turno, i risultati ottenuti con tutti gli algoritmi contenuti in *TDSFT* (**Tabella 1**).

È importante precisare che, durante l'intera esecuzione, è stata utilizzata la configurazione di default di *TDSFT*, ossia linea esterna e metodo di chiusura "*Shape-Preserving*". Inoltre, per quanto riguarda l'algoritmo "*Average Target From Input"* (vedi **Sezione 3.4**), è stata utilizzata come target la segmentazione prodotta da *3D-Cell-Annotator*.

	Spheroid 1	Spheroid 2	Spheroid 3	Avg
Average Smallest And Largest	0.876777	0.944828	0.767196	0.862934
Average Target From Input (1)	0.804545	0.879195	0.763158	0.815633
Average Target Largest	0.831776	0.901316	0.817708	0.850267
Average Target Smallest	0.79717	0.873333	0.611702	0.760735
Largest	0.878151	0.904459	0.760163	0.847591
Middle - Avg	0.861905	0.868966	0.846876	0.859249
Middle - Largest	0.851307	0.868966	0.846939	0.855737
Middle - Smallest	0.819048	0.786207	0.821429	0.808895
Smallest	0.52381	0.710345	0.255319	0.496491
STAPLE	0.882629	0.917808	0.855721	0.885386

***Tabella 1**: Indici di Jaccard relativi alle esecuzioni sui tre sferoidi precedentemente citati.*

	Algorithms	Results
1	Staple	0.885386
2	Average Smallest And Largest	0.862934
3	Middle - Average	0.859249
4	Middle - Largest	0.855737
5	Average Target Largest	0.850267
6	Largest	0.847591
7	Average Target From Input (1)	0.815633
8	Middle - Smallest	0.808895
9	Average Target Smallest	0.760735
10	Smallest	0.496491

***Tabella 2**: Classifica rispetto ai risultati medi mostrati in Tabella 1.*

In **Tabella 1** si possono osservare i risultati ottenuti tramite l'applicazione degli algoritmi proposti da *TDSFT* alle segmentazioni degli sferoidi precedentemente citate. In particolare, la colonna "*Avg*" mostra l'Indice di Jaccard medio tra i tre risultati ottenuti. Successivamente, gli algoritmi sono ordinati in modo decrescente proprio in relazione alle prestazioni medie; questo ordinamento è mostrato in **Tabella 2**.

STAPLE (vedi **Sezione 3.6**) conferma di essere l'algoritmo che permette di ottenere i risultati più affidabili. In particolare, questo algoritmo è particolarmente consigliato quando il numero

di segmentazioni da mediare è superiore a cinque ricordando che, nel caso in cui il numero sia maggiore di dieci, il risultato è considerato indipendente dai singoli contributi. *STAPLE* è però sconsigliato con solamente due segmentazioni, in questo caso infatti non è possibile calcolare correttamente l'accuratezza.

Sempre nelle prime posizioni sono poi presenti "*Average Smallest And Largest"* (vedi **Sezione 3.3**) e "*Middle – Average"* (vedi **Sezione 3.5**), ossia il metodo che calcola la *segmentazione centrale* utilizzando come algoritmo per le rimanenti due segmentazioni "*Average Smallest And Largest*". Riguardo a questi risultati è necessario dire che, nonostante abbia ottenuto un risultato inferiore, è maggiormente consigliato "*Middle – Average*". Il risultato dell'algoritmo che calcola la segmentazione intermedia tra la più grande e la più piccola è infatti completamente dipendente dalla forma di queste due; per cui l'affidabilità di questo risultato può variare molto al variare dei dati su cui è applicato. Da questo è possibile dedurre che segmentazioni molto piccole o molto grandi possono sbilanciare molto l'annotazione finale ottenuta dal processo di fusione.

Per quanto riguarda i risultati di "*Middle – Largest*" (vedi **Sezione 3.5**) e di "*Average Target Largest*" (vedi **Sezione 3.4**) l'osservazione da fare è la medesima fatta per l'algoritmo classificatosi in seconda posizione. Questi due metodi sono dipendenti dalla forma della segmentazione più grande ("*Largest*", **Sezione 3.1**) e l'avere ottenuto buoni risultati significa quindi che la segmentazione più grande stessa non si discosta troppo dalla ground truth. Questo è dimostrato anche dai risultati ottenuti con l'algoritmo "*Largest*".

6 – Conclusioni e sviluppi futuri

Questo progetto di Tesi ha portato allo sviluppo del tool open-source *TDSFT* scaricabile gratuitamente al seguente link: *https://sourceforge.net/p/tdsft*.

TDSFT nasce per colmare la mancanza evidenziata da alcuni specialisti dell'*Istituto IRCCS IRST di Meldola* (FC) di un software a supporto dell'attività di segmentazione di un tumore che consentisse di mediare tra loro più segmentazioni bidimensionali.

La segmentazione è l'attività con la quale viene identificata la corretta posizione spaziale di un tumore ed è fondamentale in diverse fasi del trattamento, a partire dalla diagnosi sino a fasi successive nelle quali viene utilizzata per analizzare eventuali miglioramenti o peggioramenti. La zona malata però non è delimitata da margini ben definiti e, la mancanza di uno standard nella loro identificazione, porta questo passaggio ad essere affetto da notevole soggettività.

L'origine delle richieste formulate dai medici e biologi dell'*IRST di Meldola* e la conseguente nascita di questo progetto di Tesi deriva quindi dalla possibilità che diversi specialisti associno ad una stessa immagine rappresentativa di un tumore annotazioni differenti. Pertanto, date più segmentazioni dello stesso tumore, è necessario identificare la più affidabile.

TDSFT consente di unire tra loro più annotazioni bidimensionali tramite numerosi algoritmi messi a disposizione dell'utente. Attualmente sono stati implementati sette diversi algoritmi (senza contare le forme nate da sotto parametri come la fusione imposta in caso di numeri pari di curve). Inoltre, non esistendo ancora un metodo validato, il software è stato progettato per permettere l'aggiunta in modo semplice di ulteriori algoritmi attraverso ad un'apposita interfaccia grafica dedicata alla configurazione di nuovi parametri.

Le caratteristiche principali di *TDSFT* sono:

1. *Semplicità di utilizzo*: è stato sviluppato per supportare i medici durante il loro lavoro;
2. *Estendibilità*: essendo la ricerca nel settore molto attiva e non esistendo attualmente un metodo validato è fondamentale dare all'utente la possibilità di aggiungere eventuali propri algoritmi;
3. *Open-source*: nonostante sia sviluppato nell'ambiente closed-source di *MATLAB* è distribuito come applicazione standalone per *MAC*, *Linux* e *Windows* e non richiede quindi nessun tipo di licenza per essere utilizzato.

I possibili sviluppi futuri sono molteplici. Prima di tutto il tool potrebbe essere esteso per supportare direttamente immagini tridimensionali. Questo lo renderebbe uno strumento completamente compatibile con ogni formato di immagine oggi utilizzato in ambito medico ed andrebbe a soddisfare totalmente i bisogni evidenziati dagli specialisti.

Inoltre, un ulteriore sviluppo futuro, potrebbe essere quello di realizzare una versione parallelizzata degli algoritmi proposti. Questo consentirebbe di migliorare l'esperienza utente rendendo i tempi di esecuzione migliori. Si potrebbe quindi esplorare il *Parallel Computing Toolbox* di *MATLAB* **[21]**, un toolbox che consente di eseguire calcoli paralleli su processori multicore e su GPU.

Infine, per presentare alla comunità *TDSFT* ed il lavoro svolto in questo progetto di Tesi, stiamo preparando un articolo scientifico dal titolo: "*TDSFT (Two Dimensional Segmentation Fusion Tool): an extensible and open-source tool for combining different bidimensional annotations.*", con autori: "*Filippo Piccinini, Lorenzo Drudi, Jae-Chul Pyun, Misu Lee, Bongseop Kwak, Giovanni Martinelli, Antonella Carbonaro, Gastone Castellani*", che a breve verrà sottomesso ad una rivista scientifica di categoria *Q1*.

Bibliografia

[1] Parmar C, Rios Velazquez E, Leijenaar R, Jermoumi M, Carvalho S, *et al*. (2014). "Robust Radiomics Feature Quantification Using Semiautomatic Volumetric Segmentation". PLOS ONE 9(7): e102107.

[2] Ricky R. Savjani, Michael Lauria, Supratik Bose, Jie Deng, Ye Yuan, Vincent Andrearczyk (2022). "Automated Tumor Segmentation in Radiotherapy, Seminars in Radiation Oncology". Volume 32, Issue 4, Pages 319-329, ISSN 1053-4296.

[3] Beyer, T., Bidaut, L., Dickson, J. *et al*. "What scans we will read: imaging instrumentation trends in clinical oncology". Cancer Imaging 20, 38 (2020).

[4] Larobina M., Murino L. "Medical Image File Formats". J Digit Imaging 27, 200–206 (2014).

[5] https://www.dicomstandard.org/current

[6] https://www.medseg.ai/

[7] Paul A. Yushkevich, Joseph Piven, Heather Cody Hazlett, Rachel Gimpel Smith, Sean Ho, James C. Gee, and Guido Gerig. "User-guided 3D active contour segmentation of anatomical structures: Significantly improved efficiency and reliability". Neuroimage 2006 Jul 1;31(3):1116-28, http://www.itksnap.org/.

[8] Ervin A Tasnadi and others. "3D-Cell-Annotator: an open-source active surface tool for single-cell segmentation in 3D microscopy images". Bioinformatics, Volume 36, Issue 9, May 2020, Pages 2948–2949.

[9] https://it.mathworks.com/products/image.html

[10] https://it.mathworks.com/products/compiler/matlab-runtime.html

[11] Warfield, S. K., Zou, K. H., & Wells, W. M. (2004). "Simultaneous truth and performance level estimation (STAPLE): an algorithm for the validation of image segmentation". IEEE transactions on medical imaging, 23(7), 903–921.

[12] V. Caselles, R. Kimmel, G. Sapiro. “Geodesic active contours. International Journal of Computer Vision”. Volume 22, Issue 1, pp. 61-79, 1997.

[13] T. F. Chan and L. A. Vese. “Active contours without edges”. IEEE Transactions Image Processing, vol. 10, no. 2, pp. 266-277, Feb. 2001, doi: 10.1109/83.902291.

[14] https://it.mathworks.com/help/matlab/ref/pchip.html#bvjbz1m-2

[15] Fritsch F. N. and R. E. Carlson. “Monotone Piecewise Cubic Interpolation”. SIAM Journal on Numerical Analysis. Vol. 17, 1980, pp.238–246.

[16] Otsu N. “A Threshold Selection Method from Gray-Level Histograms”. IEEE Transactions on Systems, Man, and Cybernetics. Vol. 9, No. 1, 1979, pp. 62–66.

[17] Xinghua Lou, Minjung Kang, Panagiotis Xenopoulos, Silvia Muñoz-Descalzo, Anna-Katerina Hadjantonakis. “A Rapid and Efficient 2D/3D Nuclear Segmentation Method for Analysis of Early Mouse Embryo and Stem Cell Image Data”. Stem Cell Reports, Volume 2, Issue 3, 2014, Pages 382-397.

[18] Gul-Mohammed J, Arganda-Carreras I, Andrey P, Galy V, Boudier T. “A generic classification-based method for segmentation of nuclei in 3D images of early embryos”. BMC Bioinformatics.

[19] Bartschat Andreas, Hübner Eduard, Reischl Markus, Mikut Ralf, Stegmaier Johannes. (2015). “XPIWIT-an XML pipeline wrapper for the Insight Toolkit”. Bioinformatics.

[20] Laurent Gole, Kok Haur Ong, Thomas Boudier, Weimiao Yu, Sohail Ahmed. “OpenSegSPIM: a user-friendly segmentation tool for SPIM data”. Bioinformatics 2016.

[21] https://it.mathworks.com/products/parallel-computing.html#scale-up-matlab-applications

Ringraziamenti

Vorrei, per prima cosa, ringraziare la Prof.ssa Antonella Carbonaro dell'Università di Bologna per avermi seguito nel ruolo di Relatrice per questa Tesi, offrendomi la possibilità, in questi sette mesi, di realizzare un progetto multidisciplinare nel quale ho potuto unire informatica e medicina.

Ringrazio il Prof. Filippo Piccinini, correlatore di questo progetto, per avermi introdotto al mondo della microscopia e dell'oncologia, per la sua costanza nell'assistere e monitorare il mio contributo, aiutandomi nell'applicare le conoscenze apprese negli ultimi tre anni a un settore inizialmente a me nuovo.

Ringrazio anche i Professori Giovanni Martinelli dell'IRCCS IRST di Meldola e Gastone Castellani dell'Università di Bologna, anche loro correlatori di questa Tesi, per avermi permesso di realizzare al meglio il mio lavoro.

Ringrazio i miei genitori per i valori e gli insegnamenti trasmessi, per le occasioni di confronto e per avermi permesso di costruire liberamente il mio percorso accompagnandomi e sostenendomi nelle scelte fatte ma lasciando che fossi io a costruire il mio futuro blocco per blocco.

Ringrazio mio fratello Nicolò e tutti i miei amici per esserci sempre stati ed avermi strappato un sorriso quando ne avevo bisogno. Grazie per esserci stati anche quando, durante i periodi di studio più intensi, mi chiudevo davanti al PC ed uscivo di rado.

Ringrazio infine tutti i colleghi e i compagni di studio, i momenti passati insieme mi hanno permesso di vivere con più leggerezza questo percorso ed i consigli che mi avete dato sono stati fondamentali e mi hanno permesso di migliorare ancor di più.