

ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA
CAMPUS DI CESENA

DIPARTIMENTO DI INFORMATICA – SCIENZA E INGEGNERIA
Corso di Laurea in Ingegneria e Scienze Informatiche

Segmentazione e analisi di immagini di radici

Elaborato in:
Visione artificiale

Relatore:
Prof.
Alessandra Lumini

Presentata da:
Alessandro Brasini

Sessione IV
Anno Accademico 2021-2022

Introduzione

Lo studio del fenotipo delle piante oggigiorno gioca un ruolo cruciale nella selezione varietale per produrre varietà che si adattino meglio ad ambienti provvisti di risorse limitate e per sviluppare un'agricoltura a basso impatto. L'analisi delle caratteristiche del fenotipo, però, risulta un lavoro tedioso e che richiede molto tempo; per questo motivo realizzare strumenti che permettano l'analisi delle caratteristiche del fenotipo in modo automatico permetterebbe di velocizzare drasticamente i tempi di ricerca. Tra le caratteristiche analizzabili vi è il sistema, o architettura, dell'apparato radicale, il cui studio permetterebbe di selezionare varietà in grado di resistere a stress ambientali come la siccità. In questa tesi presenteremo un metodo di estrazione e analisi di alcune misure in modo automatizzato del sistema radicale di piante di orzo a diversi stadi di crescita e fatte crescere in un ambiente controllato a partire da un dataset contenenti immagini ad alta risoluzione.

Il problema è stato affrontato prima producendo una segmentazione delle radici, combinando l'uso di una rete neurale profonda con tecniche classiche di visione artificiale, e, successivamente, analizzando lo scheletro di quest'ultima per ricostruire il sistema radicale di ciascuna pianta.

Il contributo originale è un metodo di primo ritaglio e ridimensionamento delle immagini, post-processing basato su tecniche classiche di visione, i cui parametri sono stati fine-tuned sul dataset, in modo da rifinire la maschera di segmentazione prodotta dalla rete e un algoritmo di ricostruzione del sistema radicale. Il metodo proposto risulta efficace su immagini in cui l'illuminazione è costante e con piante il cui apparato radicale non è eccessivamente

complesso a livello di intersezioni tra radici di altre piante. Il prototipo realizzato offre risultati incoraggianti per quanto riguarda la ricostruzione del sistema radicale e, più che come strumento di analisi di precisione, risulta, al momento, più adeguato come uno strumento per effettuare stime preliminari. Il lavoro è suddiviso in tre capitoli: nel primo si parlerà in modo più approfondito del contesto del problema e dell'analisi di alcuni metodi esistenti per risolverlo, nel secondo si analizzerà in dettaglio il metodo proposto mentre, nel terzo, si discuterà dei risultati ottenuti e dei problemi e limiti riscontrati nel metodo.

Indice

Introduzione	i
1 Fenotipizzazione delle piante	1
1.1 Architettura del sistema radicale	1
1.2 Il problema affrontato	2
1.2.1 Dataset fornito	2
1.3 Metodi esistenti	6
1.3.1 ChronoRoot	6
1.3.2 RootNav-2.0	6
2 Metodo proposto	11
2.1 Estrazione area di lavoro	11
2.1.1 Estrazione maschera vetrino	14
2.1.2 Calcolo intersezioni	19
2.1.3 Rotazione e ritaglio	21
2.2 Estrazione linea semi	22
2.3 Segmentazione	27
2.3.1 Metodi non addestrati	27
2.3.2 Metodi addestrati	30
2.4 Post process	35
2.4.1 Raffinamento maschera	38
2.4.2 Localizzazione semi	40
2.5 Estrazione	42
2.5.1 Skeletonizzazione e prune	43

2.5.2	Creazione grafo	50
2.5.3	Estrazione sistema radicale	56
2.5.4	Salvataggio	69
3	Risultati sperimentali	73
	Conclusioni	79
	Bibliografia	81
	Ringraziamenti	83

Elenco delle figure

1.1	Vetrino con bande laterali	4
1.2	Esempi di immagini con e senza il suffisso 'R'	4
1.3	Alcune immagini provenienti dal dataset. Come si può notare sono molto eterogenee tra di loro	5
1.4	Diagramma di funzionamento di RootNav-2.0	7
1.5	Esempio di immagine segmentata dalla rete di RootNav-2.0. In arancione la maschera di segmentazione mentre i punti verdi indicano la posizione delle punte delle radici	8
1.6	Algoritmo di estrazione del sistema radicale di RootNav-2.0 applicato ad alcune immagini	9
2.1	Schema ritaglio e ridimensionamento	12
2.2	Possibili configurazioni di vetrini	14
2.3	Maschera finale	15
2.4	Schema delle operazioni per ricavare la maschera dell'intero vetrino	15
2.5	Schema delle operazioni per ricavare la maschera che permette di rimuovere gli eventuali supporti laterali	17
2.6	Visualizzazione delle rette, in verde, ed intersezioni tra di esse, in rosso	19
2.7	Rappresentazione di una retta secondo la forma di Hesse. ρ è la distanza dall'origine (0,0) e θ è l'angolo di rotazione della retta.	20

2.8	Immagine ritagliata con linea dei semi inclinata di circa 2°	22
2.9	Schema individuazione linea	23
2.10	Confronto tra l'immagine non equalizzata e quella equalizzata.	24
2.11	Immagine ruotata secondo l'inclinazione della linea dei semi. In verde la linea trovata.	26
2.12	Immagine segmentata con carta poco porosa o poco evidenti	28
2.13	Immagine segmentata con carta porosa o evidenti	29
2.14	Esempio di immagine dal dataset etichettato	30
2.15	Esempio di immagine etichettata manualmente	31
2.16	Diagramma della rete HSNet	32
2.17	Immagini a 512x512 px segmentate a diversi stadi di crescita	33
2.18	Immagini a 1024x1024 px segmentate a diversi stadi di crescita	34
2.19	Schema operazioni di post process	35
2.20	Histogram matching	36
2.21	Confronto tra la maschera rifinita e quella non	37
2.22	Maschera rifinita	38
2.23	Schema raffinamento maschera al di sotto della linea dei semi	39
2.24	Schema localizzazione semi	41
2.25	Schema estrazione	42
2.26	Scheletro con protrusioni	43
2.27	Kernel per individuare tutti i possibili i nodi. I primi cinque vengono fatti ruotare, ognuno, di altri 90°, 180° e 270°.	44
2.28	Confronto tra prune morfologico e quello realizzato da noi	45
2.29	A sinistra le coordinate dei vicini di un punto p a coordinate (y, x). A destra il sistema di riferimento locale all'intorno.	48
2.30	Regole di eliminazione vicini validi. La 'V' indica un vicino valido di 'P', le 'X' indicano eventuali vicini eliminati.	48
2.31	Possibili configurazioni di vicini validi: a sinistra tutti i punti di foreground (F) vicini a P, mentre a destra solo quelli vali- di: le 'V' corrispondono ai vicini validi di 'P', le 'X' ai vicini eliminati, mentre gli spazi vuoti a punti di background.	49

2.32 Scheletonizzazione della maschera	50
2.33 Esempio di suddivisione di un dettaglio dello scheletro in percorsi e sottopercorsi. Nota: il grafo e lo scheletro non sono in scala	52
2.34 Rappresentazione di come viene calcolato l'angolo, ovvero il peso, degli archi. Per semplicità, si considera l'asse delle <i>y</i> che cresce verso l'alto.	53
2.35 Ricerca del <i>forking node</i> . In giallo il centroide della bounding box dei semi trovata nella sezione 2.4.2 e in rosso il punto sullo scheletro più vicino a questa. Si percorre lo scheletro verso il basso fino al primo nodo (in magenta): questo nodo verrà etichettato come nodo sorgente della pianta. La stessa operazione la si ripete per tutte le bounding box trovate.	54
2.36 Rappresentazione di un dettaglio del grafo e quello completo. I nodi rossi rappresentano i nodi sorgenti da cui partono le radici mentre quelli arancioni i nodi terminali. Nota: le immagini sono state specchiate per avere angoli coerenti rispetto alle immagini.	56
2.37 Esempio di attraversamento degli archi. Il punto magenta è il nodo di partenza, le frecce verdi indicano gli archi percorsi mentre quelle rosse gli archi confrontati ma scartati.	57
2.38 Esempio di inizializzazione ed esplorazione di una nuova radice	61
2.39 Esempio di esplorazione con radici sovrapposte.	62
2.40 Possibile diagramma delle classi. La classe <code>DiGraph</code> rappresenta un grafo orientato	63
2.41 Rappresentazione visuale della struttura del formato RSML. . .	70
3.1 Immagine con illuminazione non costante	74
3.2 Esempio di porzione d'immagine in cui gli spessori della maschera di segmentazione non sono precisi	75

3.3 Esempio grafico del problema che si può verificare nell'algoritmo. Nell'immagine di sinistra è rappresentata lo stato delle radici alla fine dell'iterazione i , in quella di destra l'iterazione successiva ($i + 1$)	75
3.4 Esempio del metodo applicato ad una immagine. Nota: i germogli non sono visualizzati	76
3.5 Immagini con sovrapposto lo scheletro: l'apparato radicale di ciascuna pianta è evidenziato con un colore diverso, eccetto i gambi che sono evidenziati con il colore verde	77

Capitolo 1

Fenotipizzazione delle piante

In genetica, con il termine fenotipo si intende l'insieme delle caratteristiche morfologiche e funzionali di un organismo determinate dall'interazione fra la sua costituzione genetica e l'ambiente¹. In ambito botanico, le caratteristiche osservabili della pianta possono essere, la forma, il colore, le dimensioni e le proprietà del sistema radicale. Comprendere e misurare in modo accurato questi processi per tutto il ciclo di vita della pianta in un ambiente in continua mutazione gioca quindi un ruolo cruciale per sviluppare varietà che si adattino meglio ad ambienti provvisti di risorse limitate e per sviluppare un'agricoltura a basso impatto² [1]. Per questo motivo realizzare strumenti in grado di analizzare queste caratteristiche in modo semiautomatico o automatico, piuttosto che manuale, permette di accorciare i tempi di analisi e, di conseguenza, accelerare il processo di selezione varietale.

1.1 Architettura del sistema radicale

Con architettura del sistema, o apparato, radicale si intende l'insieme e la struttura delle radici primarie e secondarie di cui è composta una pianta. Le radici hanno la duplice funzione di fissare la pianta al terreno, assorbimento

¹<https://www.treccani.it/enciclopedia/fenotipo>

²<https://www.fao.org/family-farming/detail/en/c/1115210/>

di acqua e sali minerali e la conduzione di questi fino al gambo, o stelo, e di immagazzinare scorte alimentari. Lungo la radice si sviluppa una regione pilifera o assorbente, detta così perché molte delle sue cellule superficiali si estroflettono, fino a formare dei peli, i quali si adattano strettamente alle minutissime particelle di terreno al fine di aumentare la superficie assorbente. È proprio per mezzo di questi peli, detti peli radicali, che la radice riesce ad assorbire l'acqua e le sostanze discioltevi. Il sistema radicale può essere a fittone, in cui una radice principale si estende in profondità nel terreno, o a fascicolazione, in cui le radici si ramificano in molte radici più piccole³.

L'architettura del sistema radicale è importante per la salute e la produttività delle piante, poiché influisce sull'assorbimento di acqua e nutrienti e sulla capacità della pianta di resistere a stress ambientali come la siccità [2].

1.2 Il problema affrontato

In questa tesi ci occuperemo di estrarre in modo automatico il sistema radicale da immagini di piante di orzo fatte crescere in un ambiente controllato. Una delle maggiori sfide che presenta è la corretta segmentazione e ricostruzione del sistema radicale delle piante presenti nelle immagini del dataset fornитoci. Come si discuterà anche in 1.2.1, le immagini presentano diverse caratteristiche che incrementano il livello di difficoltà di questo lavoro: nella segmentazione, per esempio, l'illuminazione non uniforme e il basso contrasto, mentre, nella ricostruzione dell'apparato radicale, l'intersezione delle radici di una pianta con quelle di un'altra e le radici che, da separate, si "uniscono", sovrapponendosi, in una sola.

1.2.1 Dataset fornito

Il dataset fornito è diviso in quattro sessioni, ognuna divisa, in media, in altre quattro sotto-sessioni contenente un numero di fotografie che varia dalle 20 alle 70 ad una risoluzione di 6000x4000px. L'esposizione delle fotografie

³<https://www.treccani.it/enciclopedia/radice#Botanica>

risulta variabile tra le sessioni: sia visivamente, che confrontando i metadati, risultano avere una esposizione maggiore nelle sessioni nella numero '1' e '2' mentre una minore nella numero '3' e '4'. In ciascuna fotografia è presente una custodia per CD aperta, o vetrino, su sfondo nero e, dalla sessione numero '3', vengono inseriti due pezzi di plastica ai lati del vetrino, o "bande laterali" (figura 1.1). Sopra il vetrino è posta una carta assorbente che può avere una texture più o meno evidente (figura 1.2b e 1.2a) e, sopra a quest'ultima, possono essere presenti fino a tre piante. Sulla carta è disegnata una linea, che può essere non perfettamente parallela al lato inferiore e superiore del vetrino, la quale ha il ruolo di dividere approssimativamente le radici dal seme e germoglio. Sulla carta molto, inoltre, compaiono molto frequentemente delle macchie di colore giallo. Ciascuna fotografia è identificata in modo univoco da un numero che è riportato sulla carta del vetrino e sul nome del file. Possono comparire due immagini con lo stesso numero ma, una di queste, avrà aggiunto il suffisso 'R' (es. '23' e '23R'): rappresentano le stesse piante ma, quella con il suffisso, rappresenta le piante fatte crescere inclinate rispetto a quella senza (figura 1.2b). Le piante presenti nelle immagini con il suffisso presentano delle radici più intrecciate tra di loro: queste risulteranno essere le più difficili da analizzare correttamente. In figura 1.3 alcune immagini provenienti dal dataset.

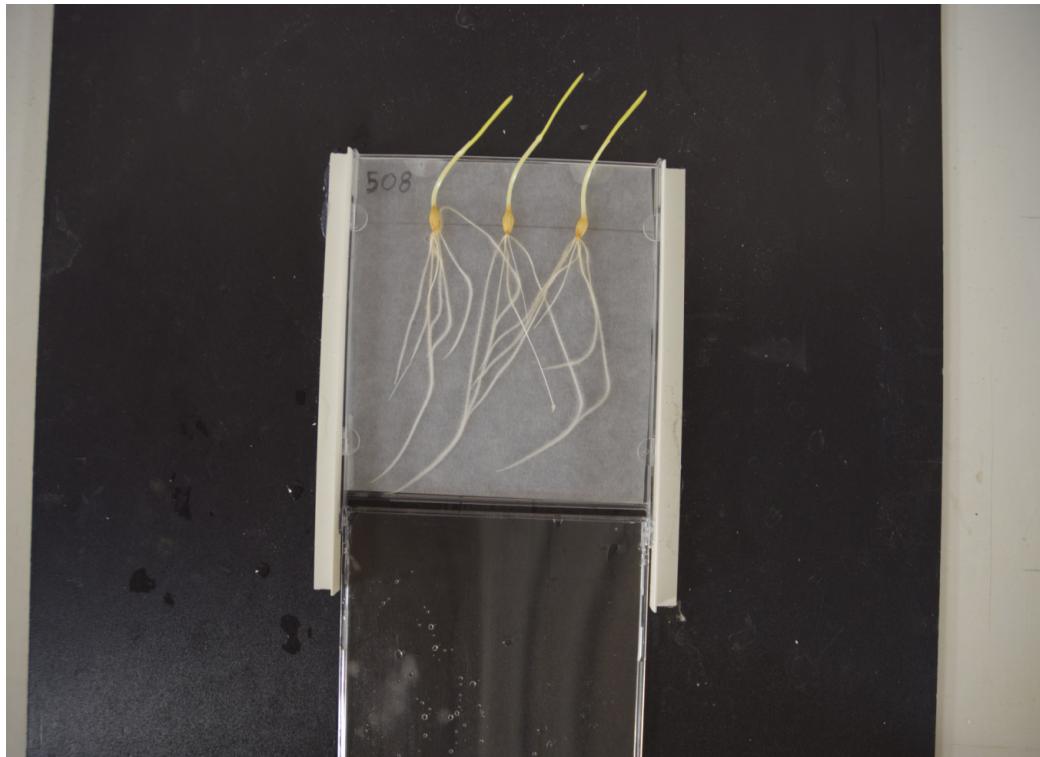
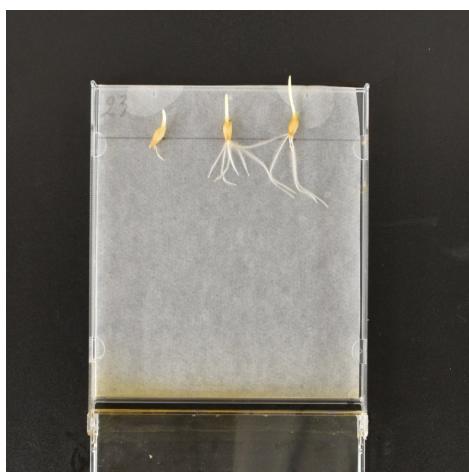


Figura 1.1: Vetrino con bande laterali



(a) Senza il suffisso



(b) Con il suffisso

Figura 1.2: Esempi di immagini con e senza il suffisso 'R'



Figura 1.3: Alcune immagini provenienti dal dataset. Come si può notare sono molto eterogenee tra di loro

1.3 Metodi esistenti

In questa sezione discuteremo di alcuni metodi esistenti per l'estrazione completamente automatizzata del sistema radicale delle piante. I metodi analizzati fanno tutti uso di reti neurali profonde per la segmentazione delle immagini.

1.3.1 ChronoRoot

ChronoRoot [3] si avvale del video della crescita delle piante, fatte crescere su di un vetrino, per effettuare l'estrazione del loro sistema radicale. Il flusso di lavoro può essere riassunto nei seguenti punti:

1. acquisizione video della crescita delle piante
2. segmentazione delle piante per ogni frame del video
3. per ogni frame segmentato, individuazione della regione d'interesse di ogni pianta
4. skeletonizzazione delle maschere di segmentazione con eventuale ricostruzione delle radici non correttamente segmentate
5. costruzione del grafo a partire dalle skeletonizzazioni
6. estrazione del sistema radicale tramite una ricerca in profondità

Il punto di forza di questo metodo è la capacità di ricostruire fedelmente il sistema radicale grazie al video fornito, cosa di cui noi purtroppo non disponiamo.

1.3.2 RootNav-2.0

RootNav-2.0 [4], a differenza di ChronoRoot, effettua l'estrazione del sistema radicale a partire da immagini di dimensione 1024x1024 px. Questo

metodo passa le immagini di input ad una rete convoluzionale, la quale, effettua sia la segmentazione che la localizzazione della posizione dei semi e delle punte delle radici tramite una *heat map regression*. Per ogni seme trova tutte le *shortest path*, mediante l'algoritmo A^* , fino alle punte delle radici localizzate: in questo modo riescono a ricostruire il sistema radicale di ciascuna pianta. Infine, viene salvato il sistema radicale in formato RSML [5]. In figura 1.4 lo schema riassuntivo del funzionamento generale del metodo.

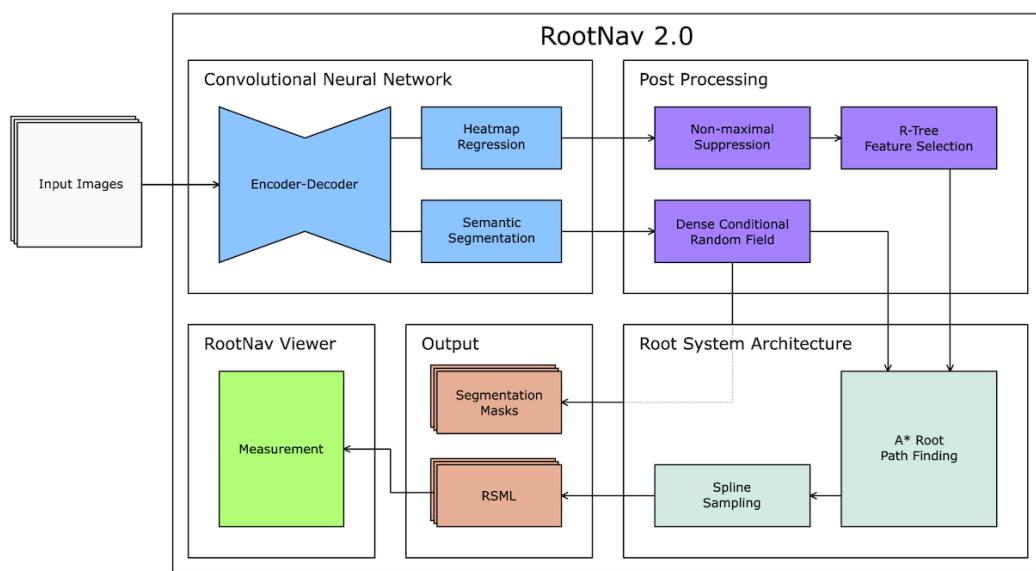


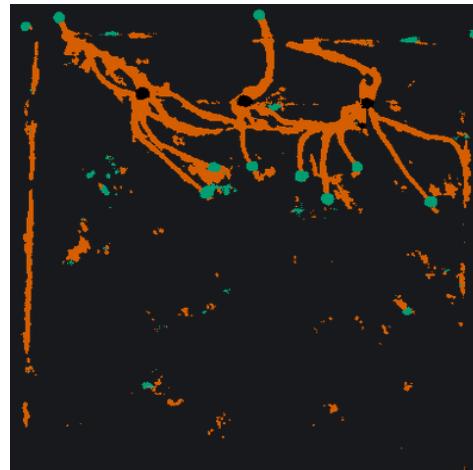
Figura 1.4: Diagramma di funzionamento di RootNav-2.0

La rete permette di addestrare nuovi dataset a partire dai pesi ottenuti dagli autori, addestrato per 500000 iterazioni un loro dataset (descritto nella sezione 2.3.2.1), utilizzando la tecnica di *transfer learning*: questo permette di velocizzare la fase di addestramento di nuovi dataset "riprendendo" dall'ultima iterazione fatta. Per questo motivo è stato addestrata una nuova rete, tramite la funzionalità fornita, su alcune immagini annotate a mano provenienti dal nostro dataset (descritto in 1.2.1), annotando in più la posizione dei semi e delle punte delle radici. La rete è stata addestrata per relativamente altre poche iterazioni, in quanto le metriche che restituiva la rete non miglioravano con l'avanzare del numero di iterazioni fatte. Come si

può vedere in figura 1.5, la segmentazione sul nostro dataset risulta piuttosto imprecisa.



(a) Originale



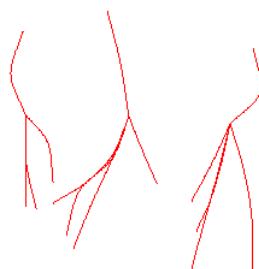
(b) Maschera prodotta

Figura 1.5: Esempio di immagine segmentata dalla rete di RootNav-2.0. In arancione la maschera di segmentazione mentre i punti verdi indicano la posizione delle punte delle radici

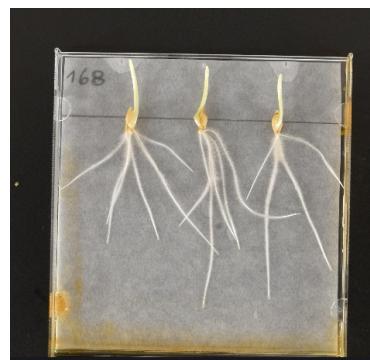
Infine, si è provato l'algoritmo di ricostruzione del sistema radicale utilizzando come maschere di segmentazione alcune segmentate a mano da noi e inserendo manualmente le coordinate dei semi e delle punte delle radici identificate. Come però si può verificare dalla figura 1.6, l'algoritmo non rileva correttamente le radici quando si incrociano con quelle di altre piante e non gestisce il caso in cui le punte delle radici tendono a combaciare, cosa in cui nel nostro dataset compaiono molto frequentemente entrambi i casi. In generale RootNav-2.0 risulta però fare un eccellente lavoro su immagini in cui sono presenti piante le cui radici non si incrociano con altre piante, quindi ben distanziate tra di loro, e le cui punte corrispondono al numero esatto di radici presenti nella pianta (una punta = una radice).



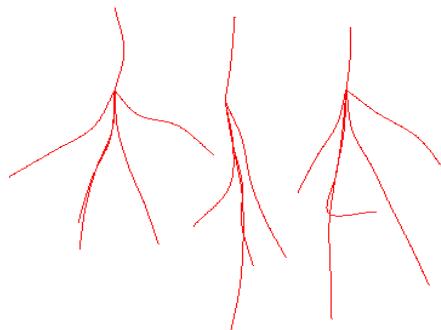
(a) Originale



(b) Risultato estrazione



(c) Originale



(d) Risultato estrazione

Figura 1.6: Algoritmo di estrazione del sistema radicale di RootNav-2.0 applicato ad alcune immagini

Capitolo 2

Metodo proposto

In questo capitolo illustreremo in dettaglio il metodo proposto. Ad alto livello si eseguono sequenzialmente le seguenti operazioni:

1. pre-elaborazione delle immagini (estrazione area di lavoro e linea dei semi)
2. segmentazione
3. post-elaborazione della maschera di segmentazione
4. estrazione caratteristiche

Per ogni operazione sarà dedicato un sottocapitolo in cui si parlerà nel dettaglio ed uno schema riassuntivo delle ulteriori operazioni applicate. Per tutto il documento si utilizzerà un sistema di coordinate cartesiano dove le coordinate x crescono verso destra e le coordinate y verso il basso. Un punto verrà indicato nel formato (y, x) e le coordinate saranno considerate come valori interi.

2.1 Estrazione area di lavoro

Come riportato in 1.2.1, il dataset fornito include immagini ad una risoluzione elevata, mentre il processo di segmentazione delle immagini risulta

inefficiente per immagini troppo grandi. Per questo problema sarà utilizzato un segmentatore che accetta immagini di dimensione 512x512px. Per questo motivo è necessaria una prima fase di ritaglio e ridimensionamento dell'immagine. Nella figura 2.1 è riportato uno schema generale del flusso di operazioni eseguite.

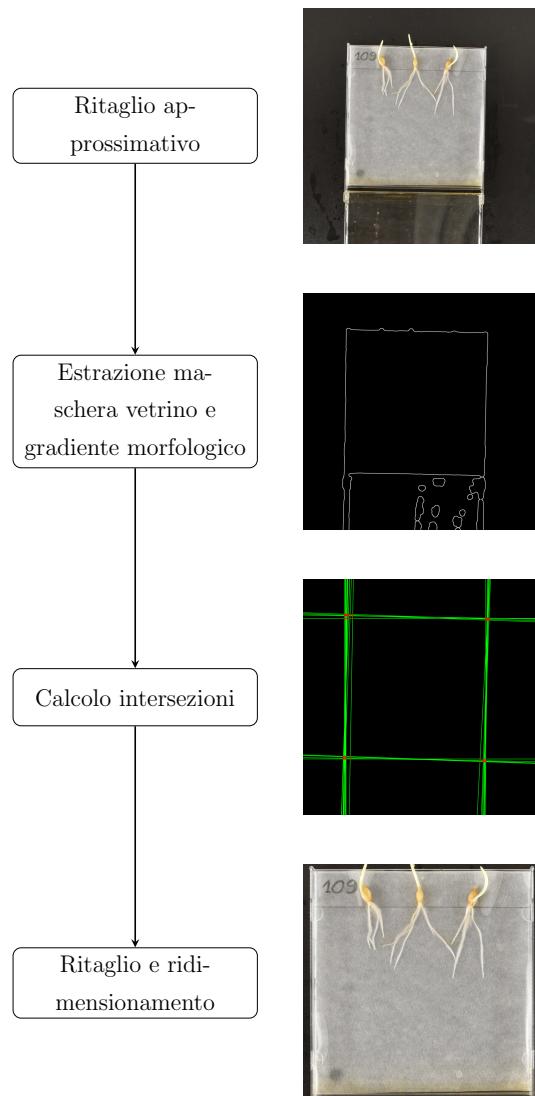


Figura 2.1: Schema ritaglio e ridimensionamento

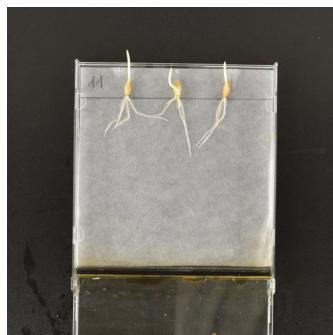
Analizzando le immagini si è notato che tutti i vetrini sono sempre disposti approssimativamente in una certa regione dell'immagine; per questo motivo si effettua un primo ritaglio in modo da ridurre il costo computazionale delle successive elaborazioni. La regione individuata si estende da circa il punto $p1 = (288, 1000)$ al punto $p2 = (3316, 4256)$.

Successivamente si procede all'estrazione della maschera del vetrino su cui poggiano le radici; il metodo verrà discusso in 2.1.1.

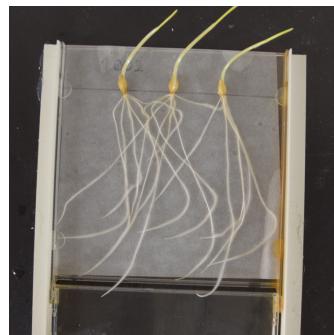
Alla maschera trovata viene applicato un gradiente morfologico con kernel circolare di dimensione $k = (3, 3)$, ricavando in questo modo un contorno di spessore pari a $3px$. Utilizzando questa immagine vengono trovate le linee rette che passano per i punti del contorno, ovvero i lati del vetrino, vengono calcolati i punti d'intersezione e, infine, si trova il quadrato di area minima che li racchiude. Il processo di individuazione e calcolo delle intersezioni verrà discusso in 2.1.2. Infine si procede al calcolo dell'inclinazione per ritagliare e ruotare l'immagine come verrà discusso in 2.1.3.

2.1.1 Estrazione maschera vetrino

All'interno del dataset compaiono due tipi di configurazioni di vetrini: senza supporti laterali (figura 2.2a) e con supporti (figura 2.2b). Questo ha fatto in modo che non fosse possibile applicare una semplice binarizzazione, in quanto incorporerebbe anche le bande laterali.



(a) Senza supporti laterali



(b) Con supporti laterali

Figura 2.2: Possibili configurazioni di vetrini

Per mitigare il problema, si eseguono due elaborazioni parallele alla regione individuata precedentemente:

- estrazione maschera dell'intero vetrino (figura 2.4)
- estrazione maschera senza supporti laterali (figura 2.5)

Le due maschere trovate verranno poi fuse assieme mediante un'operazione di bitwise AND (o AND bit a bit): questo permetterà di trovare la maschera che evidenzia solamente l'area del vetrino senza i supporti laterali. Infine, per rifornire la maschera, si applica una apertura morfologica circolare di dimensione $ker = (50, 50)$. In figura 2.3 la maschera finale.

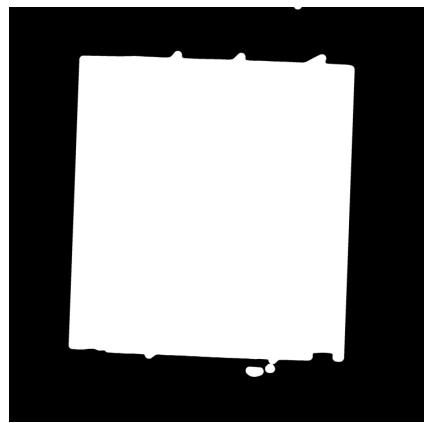


Figura 2.3: Maschera finale

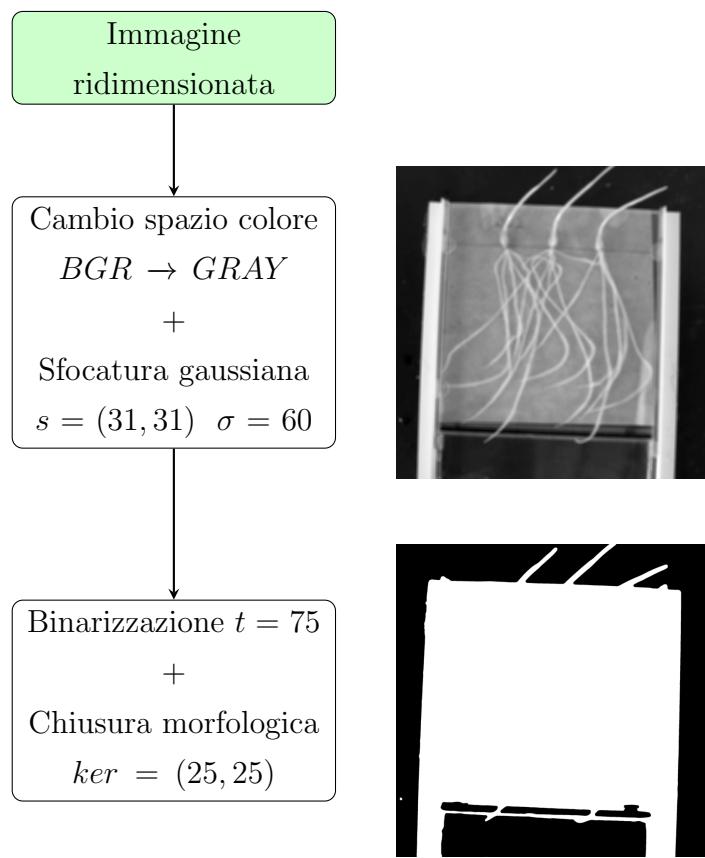


Figura 2.4: Schema delle operazioni per ricavare la maschera dell'intero velino

Come illustrato nella figura 2.4, si procede ad eseguire un cambio spazio di colore da *BGR* (Blue, Green, Red) a *GRAY* (o toni di grigio) ed una sfocatura gaussiana con dimensione del filtro $s = (31, 31)$ e $\sigma = 60$: questo permette di avere una binarizzazione più uniforme all'interno della regione. Si esegue poi una binarizzazione con soglia $t = 75$ e una chiusura morfologica con kernel circolare di dimensione $ker = (25, 25)$ per chiudere eventuali piccoli buchi residui.

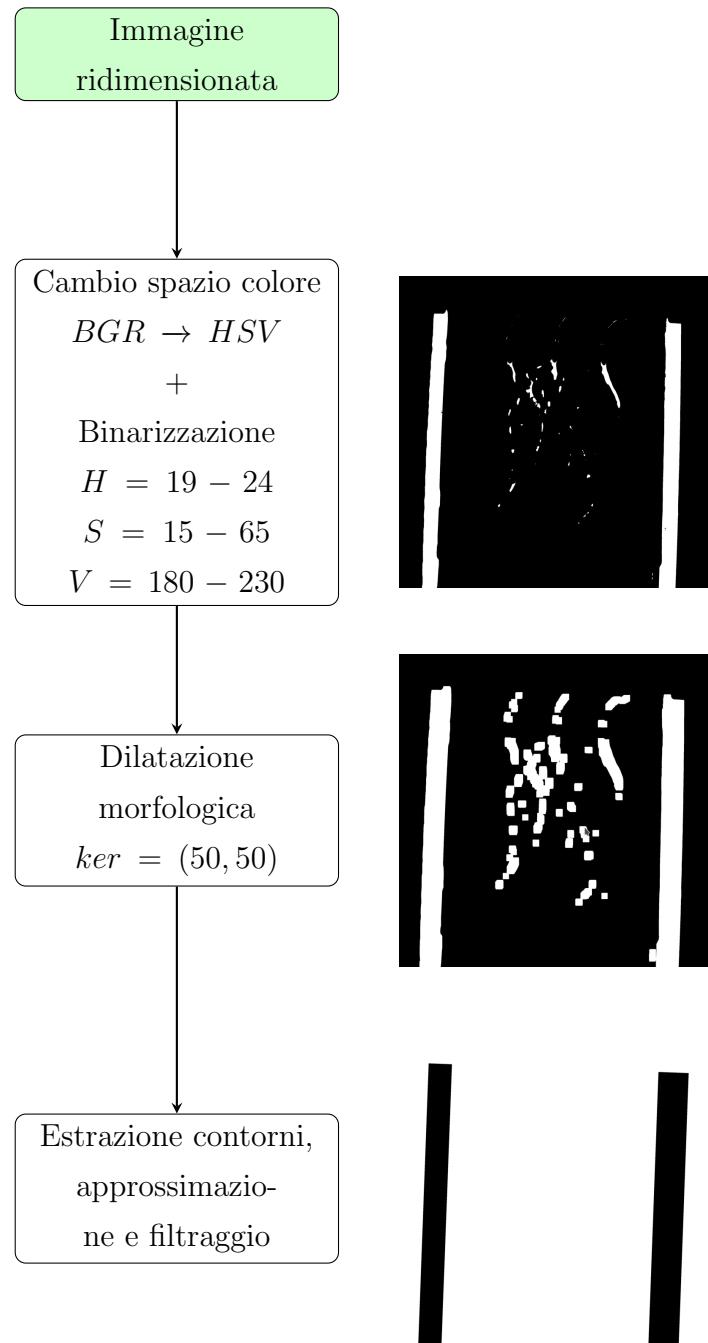


Figura 2.5: Schema delle operazioni per ricavare la maschera che permette di rimuovere gli eventuali supporti laterali

Come illustrato in figura 2.5, si procede ad eseguire un cambio spazio di colore da *BGR* a *HSV* (Hue, Saturation, Value) e a fare una binarizzazione sul range dei seguenti valori:

- Hue: 19 – 24
- Saturation: 15 – 65
- Value: 180 – 230

Questi intervalli di valori, di ciascun canale *HSV*, corrispondono indicativamente all'intervallo di colori in cui ricadono i supporti. Per cercare di compensare l'approssimazione degli intervalli, si esegue una dilatazione morfologica con kernel rettangolare di dimensione $ker = (50, 50)$. In seguito si ricavano i contorni di tutti i pixel contigui tra di loro e, per ognuno, lo si approssima con un rettangolo di area minima in grado di contenere il contorno al suo interno. Successivamente, per ogni rettangolo, si mantengono quelli che hanno una altezza uguale o superiore a $1800px$, che corrisponde indicativamente all'altezza dei supporti, e larghezza minore dell'altezza. Infine, si crea un'immagine completamente bianca (corrispondente al valore 255) e, i rettangoli individuati come supporti, vengono colorati di nero (corrispondente al valore 0). Questa immagine sarà la maschera.

2.1.2 Calcolo intersezioni

Per trovare i lati del vetrino, viene applicata la trasformata di Hough [6] utilizzando come valore di threshold dell'accumulatore $t = 100$: questo vuol dire che verranno filtrate solamente le rette che hanno come valore di pixel "accumulati" un numero maggiore o uguale a 100. Il valore non è stato scelto appositamente molto alto, nonostante un lato del vetrino sia all'incirca $1500px$, in quanto i bordi tendono ad essere leggermente irregolari: scegliendo un valore di threshold vicino a questo valore, la maggior parte delle volte le linee non venivano correttamente individuate. Una volta trovate le rette si procede a calcolare le intersezioni tra di esse (figura 2.6).

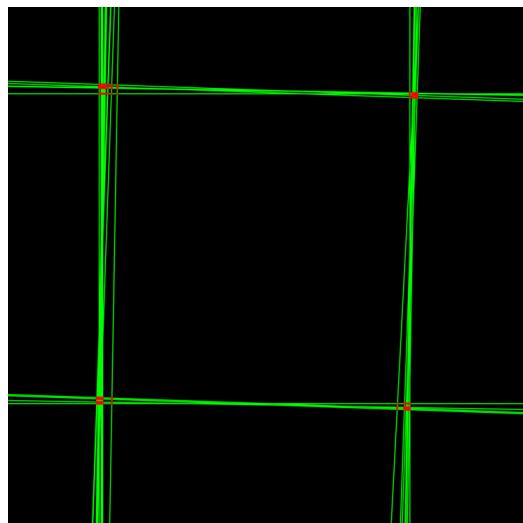


Figura 2.6: Visualizzazione delle rette, in verde, ed intersezioni tra di esse, in rosso

La trasformata di Hough rappresenta le rette secondo la forma di Hesse (figura 2.7).

L'equazione di una retta è scritta come:

$$y = \left(-\frac{\cos \theta}{\sin \theta} \right) x + \left(\frac{\rho}{\sin \theta} \right) \quad (2.1)$$

riarrangiando i termini:

$$\rho = x \cos \theta + y \sin \theta \quad (2.2)$$

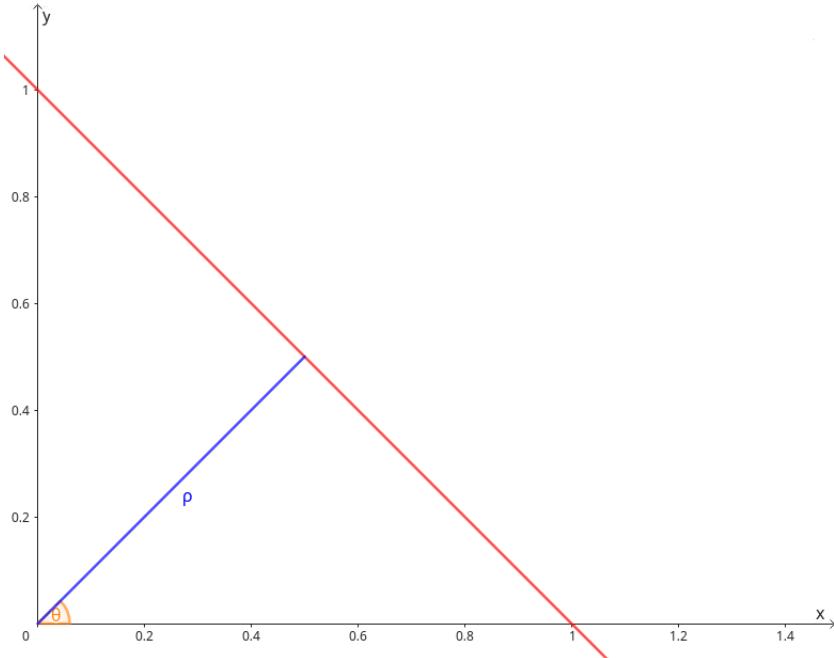


Figura 2.7: Rappresentazione di una retta secondo la forma di Hesse.
 ρ è la distanza dall'origine $(0,0)$ e θ è l'angolo di rotazione della retta.

Per trovare le intersezioni tra due rette è sufficiente risolvere il sistema di equazioni:

$$\begin{cases} \rho_1 = x \cdot \cos \theta_1 + y \cdot \sin \theta_1 \\ \rho_2 = x \cdot \cos \theta_2 + y \cdot \sin \theta_2 \end{cases} \quad (2.3)$$

Non ci interessa trovare le intersezioni tra tutte le coppie di rette, bensì solo quelle perpendicolari tra di loro. Per questo motivo, prima vengono partizionate in due gruppi utilizzando l'algoritmo K-Means e come criterio di raggruppamento l'angolo θ . Così facendo otterremo, nel nostro caso, un gruppo che avrà un angolo $\theta \sim 0^\circ$, che rappresentano le rette orizzontali, e l'altro $\theta \sim 90^\circ$, che rappresentano le rette verticali. Per ogni retta di un gruppo, infine, si trovano le intersezioni per ogni retta dell'altro gruppo.

2.1.3 Rotazione e ritaglio

Dalle coordinate del quadrato trovato si trova l'inclinazione θ e posizione del centroide c ; questi serviranno per costruire la matrice di rotazione definita come segue:

$$M = \begin{bmatrix} \alpha & \beta & (1 - \alpha) \cdot c.x - \beta \cdot c.y \\ -\beta & \alpha & \beta \cdot c.x + (1 - \alpha) \cdot c.y \end{bmatrix} \quad (2.4)$$

dove:

$$\alpha = \text{scala} \cdot \cos \theta,$$

$$\beta = \text{scala} \cdot \sin \theta$$

Nel nostro caso $\text{scala} = 1.0$ in quanto in questo step non applichiamo trasformazioni di ingrandimento/rimpicciolimento. Nota: θ definito positivo indica una rotazione in senso antiorario.

La matrice trovata viene applicata all'immagine tramite una trasformazione affine che permette di correggere l'eventuale inclinazione del vetrino. Successivamente si ritaglia la regione rettangolare che si estende dal punto $p1 = (min_y, min_x)$ al punto $p2 = (max_y, max_x)$, dove min_y e max_y corrispondono, rispettivamente, al valore minimo e massimo di y tra i valori delle coordinate del quadrato ricavato precedentemente. Similmente lo si fa per min_x e max_x . Infine si effettua il ridimensionamento dell'immagine a 1024x1024. Le dimensioni sono superiori a quelle accettate dal segmentatore, per poter avere una migliore qualità negli step successivi.

2.2 Estrazione linea semi

In questa sezione parleremo di come viene individuata la linea che divide i semi dalle radici e correggere l'inclinazione dell'immagine in base a questa. Nella precedente sezione abbiamo corretto l'inclinazione dell'immagine secondo l'orientazione del vetrino ma, come possiamo vedere dalla figura 2.8, la linea è visibilmente inclinata: questo comporta delle complicazioni nelle operazioni che verranno eseguite nella post-elaborazione della maschera e nell'individuazione dei semi. Nella figura 2.9 è riportato uno schema delle operazioni eseguite.



Figura 2.8: Immagine ritagliata con linea dei semi inclinata di circa 2°.

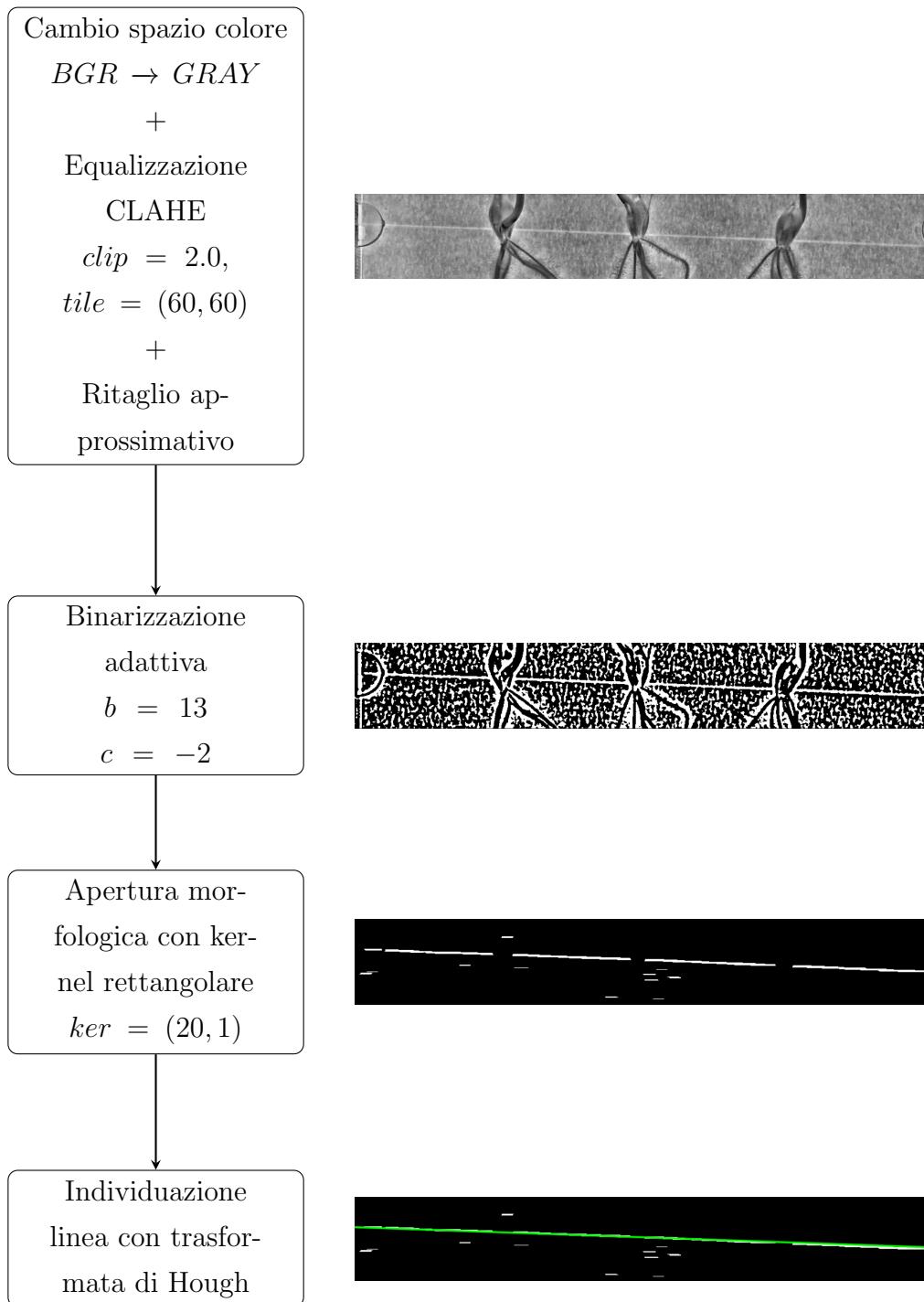


Figura 2.9: Schema individuazione linea

Partendo dall’immagine ritagliata, si esegue una conversione di spazio colore da *BGR* a toni di grigio e si trova il suo negativo facendo un bitwise NOT (o NOT bit a bit) della stessa. Si è riscontrato, però, che tra le immagini l’intensità della matita può variare e, in alcune, è appena visibile: per questo motivo è necessaria una equalizzazione dell’istogramma dell’immagine. La tecnica che ha dato i risultati migliori risulta quella **CLAHE** (Contrast Limited Adaptive Histogram Equalization) [7] con clip limit $clip = 2.0$ e dimensione dei tile (o blocchi) $tile = (60, 60)$. In figura 2.10 è mostrata la differenza tra l’immagine equalizzata e quella non: come si può notare nell’immagine equalizzata, la linea è più evidente.

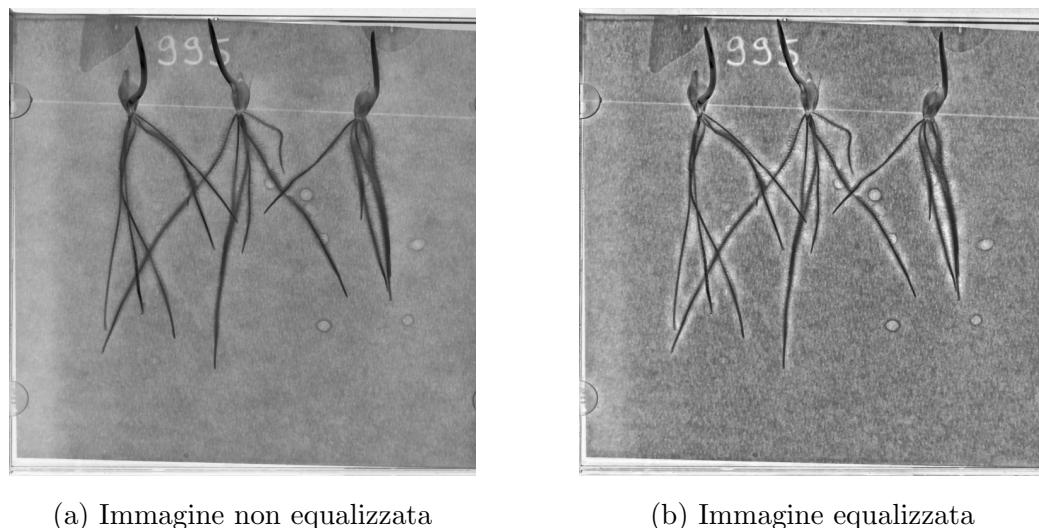


Figura 2.10: Confronto tra l’immagine non equalizzata e quella equalizzata.

Successivamente viene ritagliata la regione in cui solitamente si trova la linea, che va, indicativamente, dal punto $p1_{ROI} = (148, 0)$ a $p2_{ROI} = (300, 1023)$, e viene applicato una leggera sfocatura con dimensione $s = (5, 5)$. Viene eseguita una binarizzazione adattiva con dimensione dei blocchi $b = 13$, valore di threshold scelto come valore medio del blocco e costante $c = -2$. La binarizzazione adattiva consiste nel suddividere l’immagine in blocchi quadrati di una certa dimensione (nel nostro caso $13px$) e, su quel blocco, si

esegue una binarizzazione con un valore di threshold scelto, nel nostro caso, come valore medio dei pixel all'interno del blocco a cui si aggiunge il valore della costante c (nel nostro caso -2). Si è scelto questo tipo di binarizzazione, rispetto ad una globale, per il fatto che la linea non sempre risultava visibile. Una volta binarizzata l'immagine, viene applicata una apertura morfologica con kernel rettangolare di dimensione $ker = (20, 1)$: questo permette di ricavare gruppi di pixel che formano un elemento rettangolare lungo $20px$ e alto $1px$. Trovati gli elementi, si trova la linea con la trasformata di Hough mantenendo solo quella con valore di accumulatore più alto. Come spiegato nella sezione 2.1.2, la trasformata di Hough ritorna i valori ρ (distanza dall'origine) e θ (angolo di rotazione della retta); per trovare il valore della coordinata y della linea dei semi, in modo che sia dritta, è sufficiente risolvere l'equazione della retta 2.1 imponendo $x = 0$, in quanto non ci interessa sapere il valore della coordinata x , e $\theta = 90^\circ$, considerando una retta parallela all'asse x : si ottiene quindi $y = \rho$. La retta che descrive la linea dei semi dritta avrà come estremi i punti $p1 = (y + p1_{ROI}, 0)$ e $p2 = (y + p1_{ROI}, 1023)$ a cui dobbiamo sommare a y la coordinata y del punto $p1_{ROI}$. Per raddrizzare l'immagine, invece, è sufficiente applicare una trasformazione affine con matrice M definita come in 2.4, dove:

$$c = (h/2, w/2),$$

$$\theta = \theta - 90^\circ$$

In questo caso h e w , rispettivamente l'altezza e la larghezza dell'immagine, sono entrambi uguali a 1024.

In figura 2.11 è possibile visualizzare l'immagine ruotata.

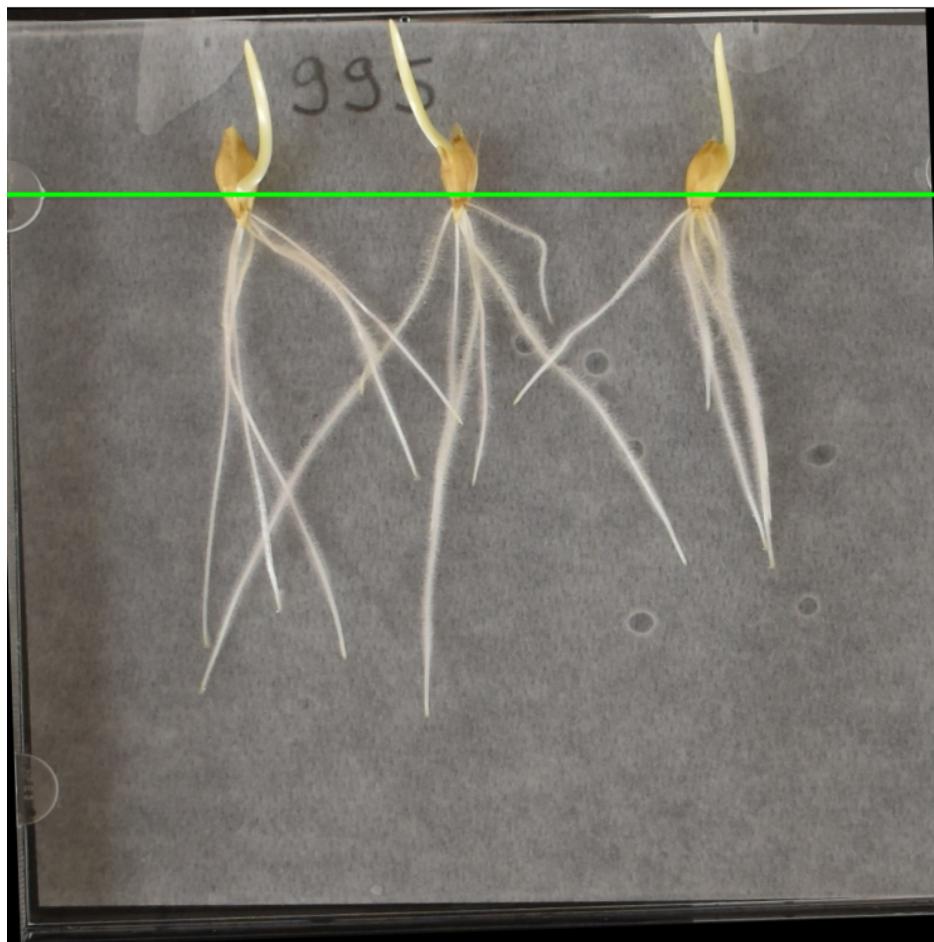


Figura 2.11: Immagine ruotata secondo l'inclinazione della linea dei semi. In verde la linea trovata.

2.3 Segmentazione

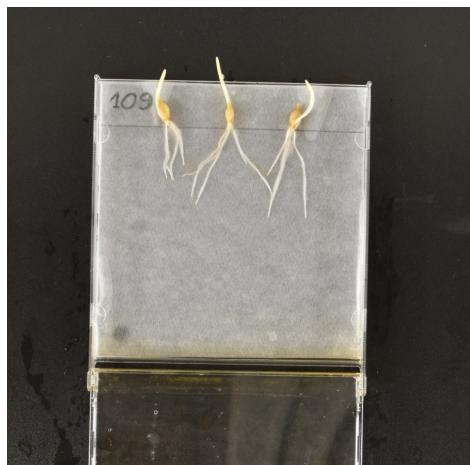
Con il termine segmentazione si intende l'estrazione dell'oggetto di interesse (la radice in questo caso) dallo sfondo. Il problema della segmentazione consiste nella predizione della classe a cui appartiene ciascun pixel di una immagine come foreground (radici/regioni di interesse) o background (sfondo). Il risultato della segmentazione è una immagine binaria in cui il colore nero è associato allo sfondo mentre il bianco alle radici. La segmentazione di immagini può essere realizzata utilizzando metodi addestrati e non. Di seguito si discuterà dei risultati che i due metodi hanno prodotto.

2.3.1 Metodi non addestrati

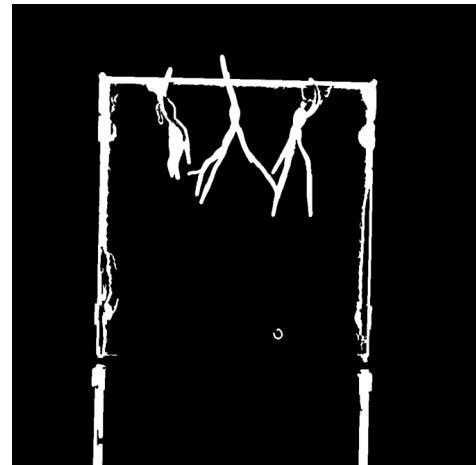
Inizialmente si è provato a segmentare le radici usando dei metodi non addestrati (binarizzazione, lavorare su diversi spazi colore ecc.) in quanto più veloci e non necessitano di un dataset etichettato, cosa di cui non disponiamo, su cui fare la fase di training. Il flusso di operazioni più promettenti è stato il seguente:

1. correzione gamma con $\gamma = 0.3$
2. applicazione per ogni pixel della funzione $g(y, x) = f(y, x) * 3.0 - 200$, dove $f(y, x)$ descrive il valore del pixel alle rispettive coordinate (y, x) .
3. sfocatura gaussiana $s = (5, 5)$, $\sigma = 10$
4. cambio spazio colore $BGR \rightarrow HLS$
5. applicazione del filtro derivativo di Sobel sul canale della saturazione
6. calcolo del magnitudo e binarizzazione con il metodo di Otsu [8]
7. chiusura morfologica con kernel circolare di dimensione $ker = (3, 3)$
8. chiusura contorni con area minore di $1000px$
9. rimozione componenti connesse con area minore di $150px$

L'obiettivo principale, quindi, era quello di individuare i bordi delle radici e riempirli. In figura 2.12 è riportato un tentativo con discreti risultati. Questo metodo però si è rilevato inefficace nel momento in cui la carta sottostante alle radici viene sostituita con una in cui i pori sono più evidenti: questo fa in modo che il filtro derivativo abbia un'alta risposta anche in prossimità dei pori. In figura 2.13 è riportato un esempio. Sono stati fatti anche tentativi di segmentazione utilizzando il colore ma, data la non omogeneità del dataset (illuminazione non costante, macchie sulla carta dello stesso colore delle radici), anche questo è risultato inefficace e, per questo, scartato.

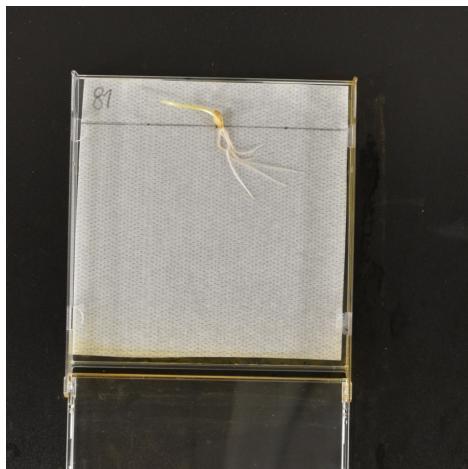


(a) Originale

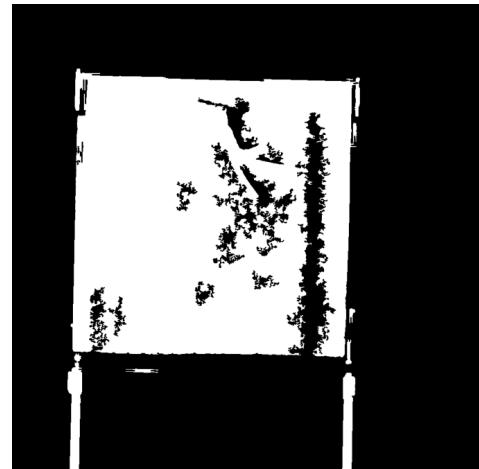


(b) Segmentata

Figura 2.12: Immagine segmentata con carta poco porosa o poco evidenti



(a) Originale



(b) Segmentata

Figura 2.13: Immagine segmentata con carta porosa o evidenti

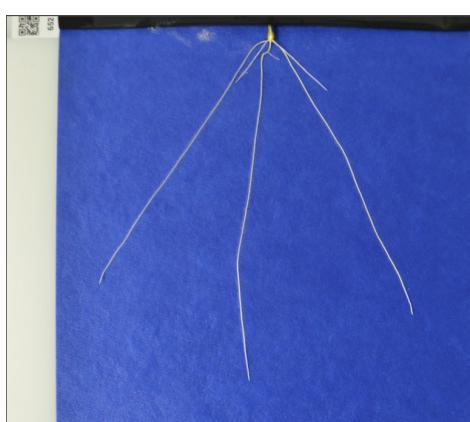
Date queste problematiche, e l'aver trovato nel frattempo un dataset etichettato che assomiglia a quello fornito, si è deciso di non procedere oltre con metodi non addestrati.

2.3.2 Metodi addestrati

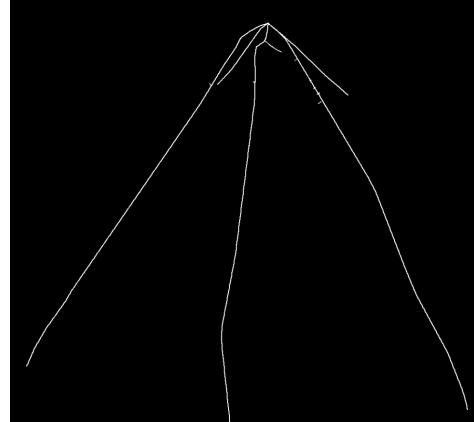
A seguito di una ricerca più approfondita, si è trovato in rete un dataset di grano etichettato che assomiglia molto a quello fornito. Questo ci ha permesso di poter provare dei metodi addestrati, in particolare è stata presa la decisione di provare direttamente delle reti neurali profonde in quanto, quelle più recenti, riescono a fornire risultati soddisfacenti anche con dataset relativamente piccoli.

2.3.2.1 Dataset di training

Il dataset trovato [9] si tratta di quello che viene allegato con RootNav-2.0: contiene 3630 immagini a colori di *Triticum aestivum*, o grano tenero, ad una risoluzione di 1900x2000 px. Il dataset in questione risulta abbastanza adatto in quanto il grano presenta un apparato radicale molto simile a quello di orzo, ovvero fascicolato [10] [11]. Il ground truth delle immagini è salvato in formato RSML, perciò, prima di passarle alla rete, è necessario renderizzarle in modo da produrre una immagine binaria: lo sfondo è rappresentato dal colore nero e in bianco le radici (figura 2.14).



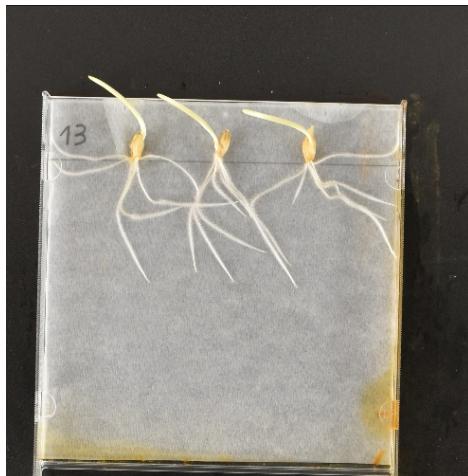
(a) Originale



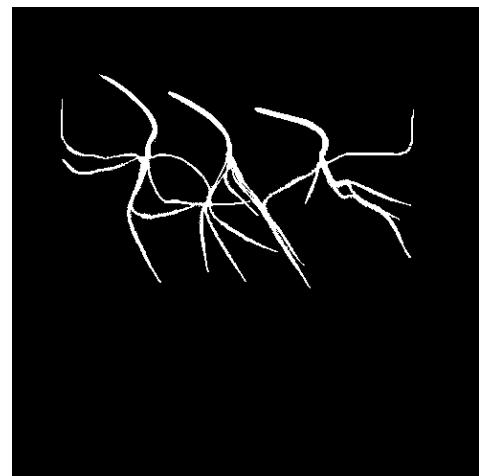
(b) Ground truth

Figura 2.14: Esempio di immagine dal dataset etichettato

In aggiunta sono state segmentate 15 immagini provenienti dal nostro dataset usando il software "Friendly Ground Truth" [12]. In figura 2.15 una immagine segmentata manualmente.



(a) Originale



(b) Etichettata

Figura 2.15: Esempio di immagine etichettata manualmente

2.3.2.2 Modello usato

La segmentazione delle immagini è stata realizzata addestrando la rete HSNet [13], realizzata per la segmentazione di polipi intestinali. Gli autori propongono una rete ibrida che combina una rete convoluzionale e una basata su transformer: con la prima si mira ad estrarre caratteristiche locali, ovvero relative ad una piccola porzione di immagine, come i bordi, mentre con la seconda caratteristiche a livello globale, ovvero relative all'intera immagine, come la forma dell'oggetto. In figura 2.16 il diagramma della rete.

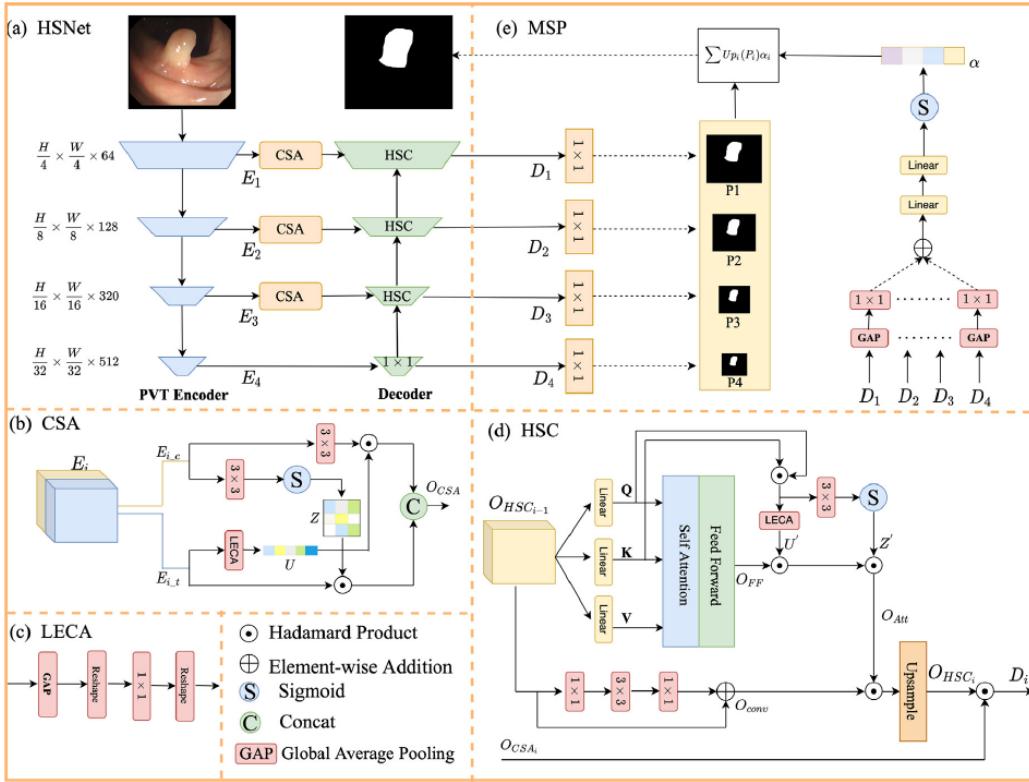
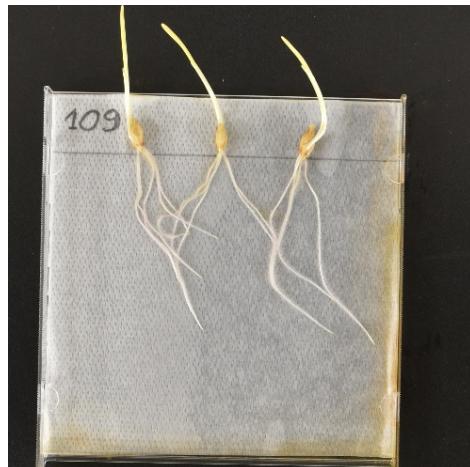


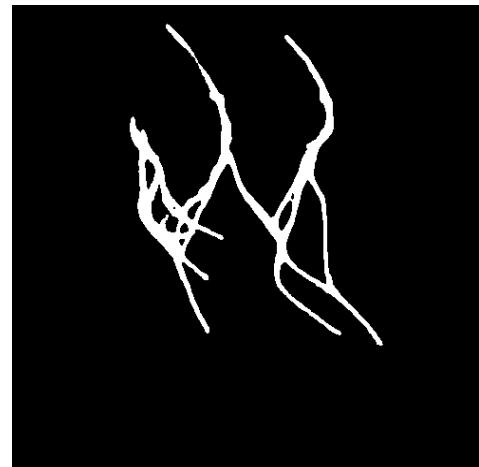
Figura 2.16: Diagramma della rete HSNet

L'addestramento è stato fatto sul dataset descritto precedentemente e sulle poche immagini etichettate manualmente provenienti dal nostro. La rete accetta immagini di dimensione 512x512 px; Per una prima verifica della qualità della segmentazione, si sono passate alla rete le immagini ritagliate grossolanamente e alle risoluzione accettata. Come però si può vedere dalla

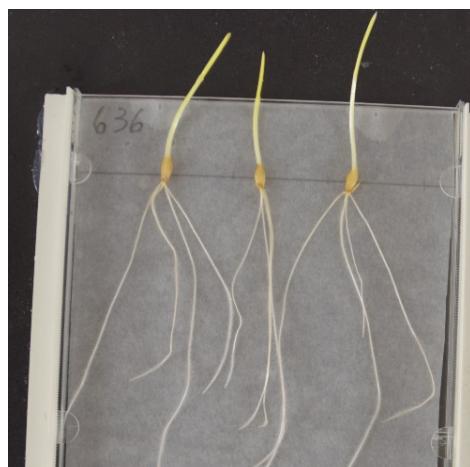
figura 2.17, nonostante il risultato della segmentazione sia buono, alcune radici vengono spezzate.



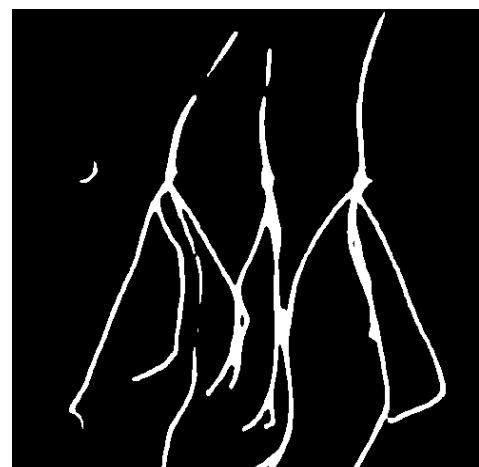
(a) Originale



(b) Maschera prodotta



(c) Originale



(d) Maschera prodotta

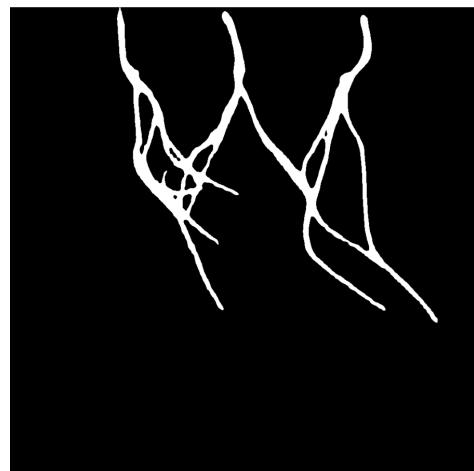
Figura 2.17: Immagini a 512x512 px segmentate a diversi stadi di crescita

Per provare a migliorare la segmentazione si è deciso, prima di tutto, di ritagliare le immagini come proposto nella sottosezione 2.1 e di passare le immagini alla rete ad una risoluzione di 1024x1024 px. Visto che la rete accetta immagini a 512x512 px, si divide l'immagine in quattro sotto-immagini, le si segmentano individualmente e, infine, si costruisce la maschera "fonden-

do” le quattro maschere prodotte. Un problema di questo approccio è la mancanza delle informazioni provenienti dalle altre immagini, che potrebbe comportare ad una degradazione della segmentazione, però, come si può vedere dalla figura 2.18, la qualità delle maschere prodotte è migliorata.



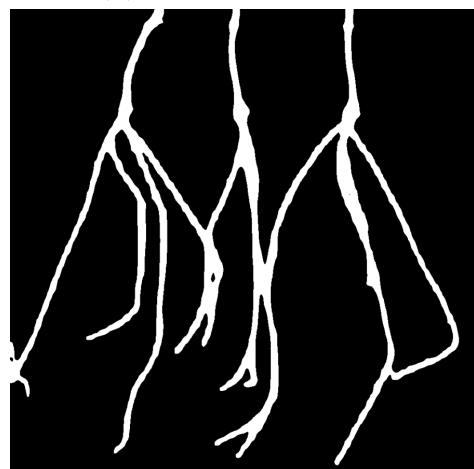
(a) Originale



(b) Maschera prodotta



(c) Originale



(d) Maschera prodotta

Figura 2.18: Immagini a 1024x1024 px segmentate a diversi stadi di crescita

2.4 Post process

Le maschere prodotte dalla rete non sono sempre accurate, per questo motivo è necessario effettuare un post process al fine di migliorarne la qualità. In figura 2.19 è riportato uno schema riassuntivo delle operazioni.

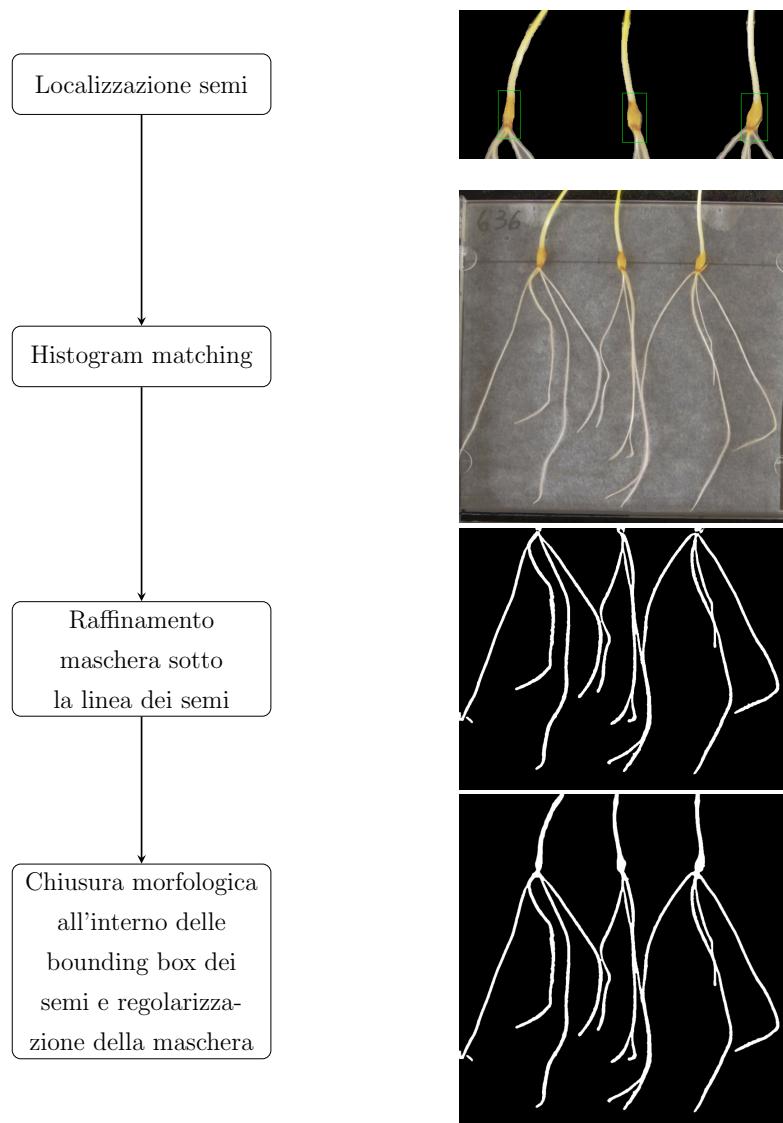


Figura 2.19: Schema operazioni di post process

Si localizzano i semi e si trova, per ognuno, la relativa bounding box (2.4.2). Si esegue un *histogram matching* [14] a partire da una immagine presa come riferimento: questo permette di avere immagini il più possibili uniformi tra di loro a livello di esposizione. Come immagine di riferimento si è scelta quella riportata in figura 2.20c. Si divide poi l'immagine in due parti, stessa cosa lo si fa per la maschera, utilizzando come "linea divisoria" la linea dei semi trovata in 2.2 e si procede a migliorare la segmentazione della maschera al di sotto della linea come verrà descritto in 2.4.1. La parte al di sopra della linea viene mantenuta invariata in quanto risulta essere sempre segmentata bene. Infine si procede a riunire la maschera rifinita con la parte superiore della maschera e si esegue un chiusura morfologica con kernel circolare di dimensione $ker = (11, 11)$ all'interno delle regioni delimitate dalle bounding box dei semi (figura 2.21a). Quest'ultima operazione serve per fare in modo che le radici siano "attaccate" ai semi, infatti, come si vede nella prima immagine di figura 2.19 in prossimità dei semi, possono essere presenti dei buchi. Infine si applica un filtro di smooth mediano con dimensione $d = 5$ all'intera maschera prodotta con lo scopo di smussare bordi appuntiti e si riempiono i buchi con area minore di 30 px. In figura 2.21b la maschera rifinita.

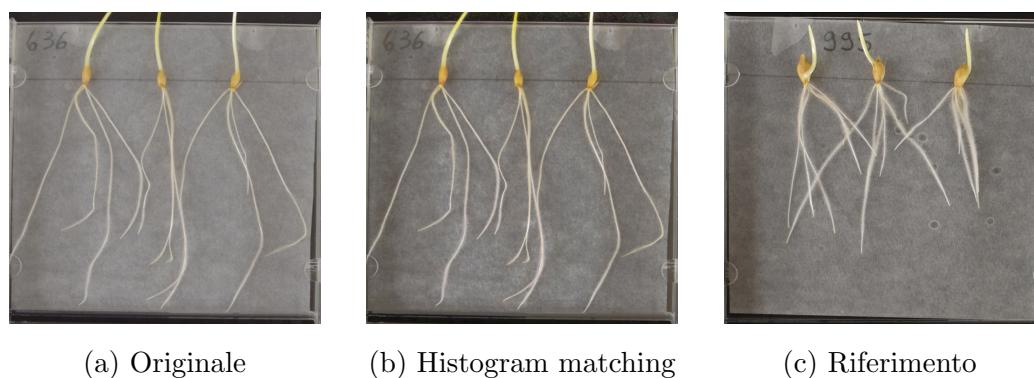
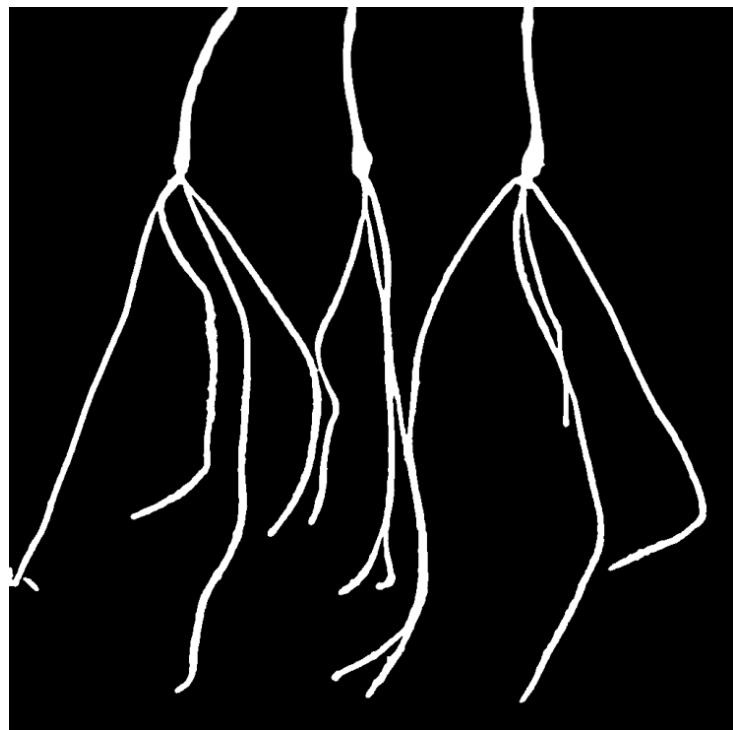
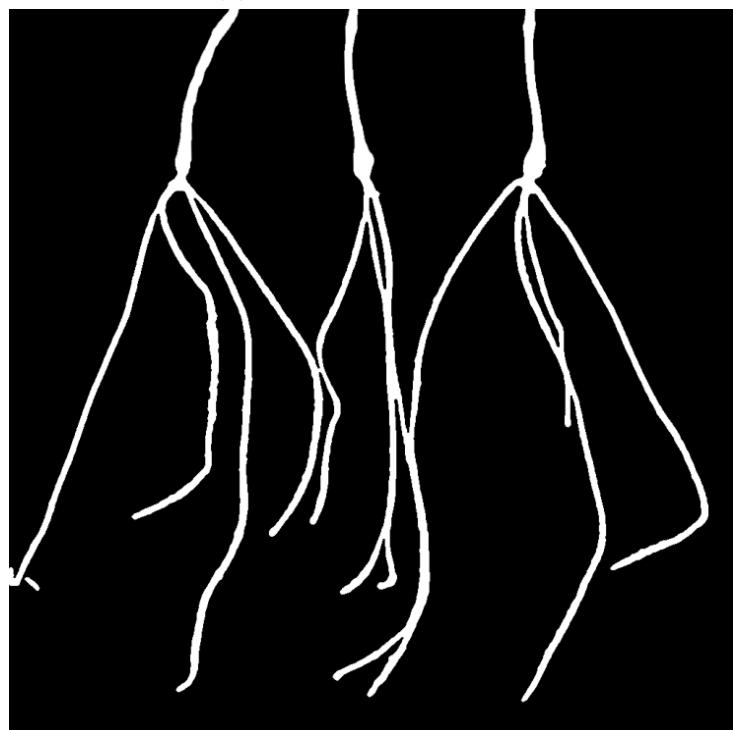


Figura 2.20: Histogram matching



(a) Maschera non rifinita



(b) Maschera rifinita

Figura 2.21: Confronto tra la maschera rifinita e quella non

2.4.1 Raffinamento maschera

Partendo dall'immagine e maschera ritagliati come descritto in precedenza, mediante un'operazione di bitwise AND, si ricava solo la parte dell'immagine esposta dalla maschera. All'immagine "mascherata" viene applicata una operazione di regolazione automatica della luminosità e contrasto [15] con clip dell'istogramma $clip = 37\%$: questa operazione risulta essere di aiuto nel evidenziare maggiormente le radici, eliminando eventuali residui di carta e peli radicali di leggera intensità. Si procede con attenuare il rumore prodotto dalla eventuale presenza di peli radicali mediante un downscale e upscale piramidale e si esegue una equalizzazione CLAHE con $clip = 40$ e dimensione dei blocchi $tile = (20, 20)$. Per concludere si applica una binarizzazione adattiva con $b = 21$ e $c = 0$ (figura 2.22).

In figura 2.23 è riportato uno schema riassuntivo

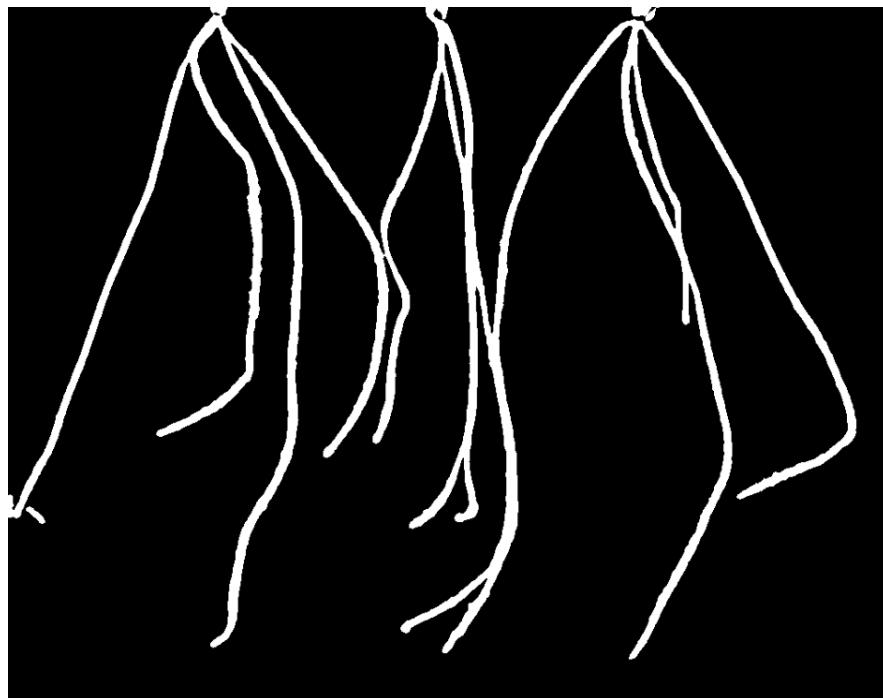


Figura 2.22: Maschera rifinita

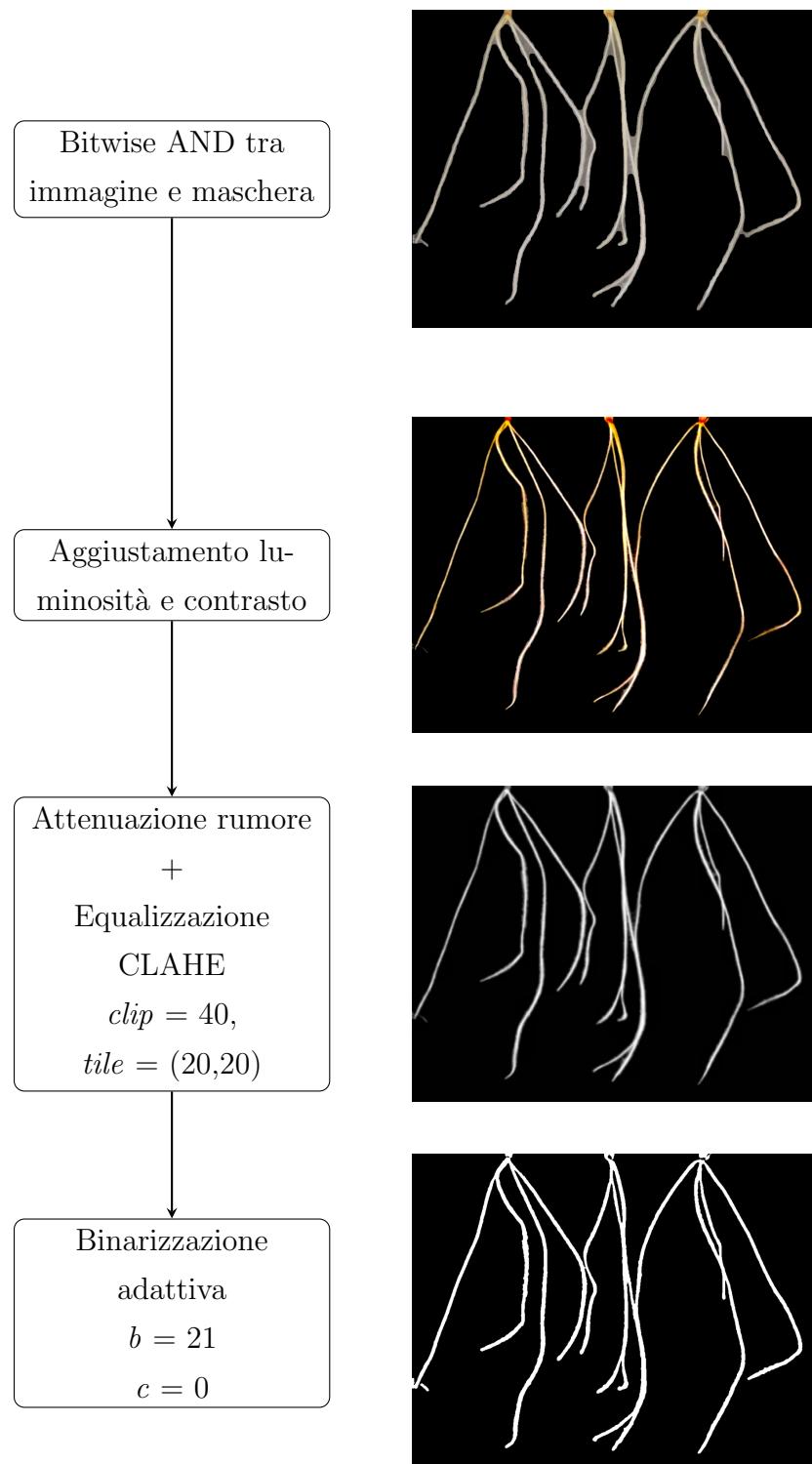


Figura 2.23: Schema raffinamento maschera al di sotto della linea dei semi

2.4.2 Localizzazione semi

Per costruzione i semi sono posizionati lungo la linea dei semi, ma non sempre presentano un corretto allineamento. Si inizia col ritagliare la regione in cui si trovano solitamente i semi: la zona individuata va dal punto $(0, 0)$ al punto $(300, 1023)$, un'area più grande rispetto a quella delimitata della linea dei semi per il fatto che i semi non sono sempre allineati correttamente al di sopra di quest'ultima. Con un cambio di spazio colore da BGR a HSV si binarizza l'immagine sul range di valori corrispondenti al colore dei semi:

- H = 12 - 25
- S = 100 - 255
- V = 100 - 255

Con una apertura morfologica con kernel circolare di dimensione $ker = (9, 9)$ si eliminano eventuali piccoli punti e, con uno di dimensione $ker = (21, 21)$, si effettua una dilatazione morfologica. Infine si estraggono le bounding box dei semi filtrando solamente quelle che hanno una grandezza e altezza maggiore di 40px in modo da rimuovere eventuali falsi positivi. In figura 2.24 è riportato uno schema riassuntivo.

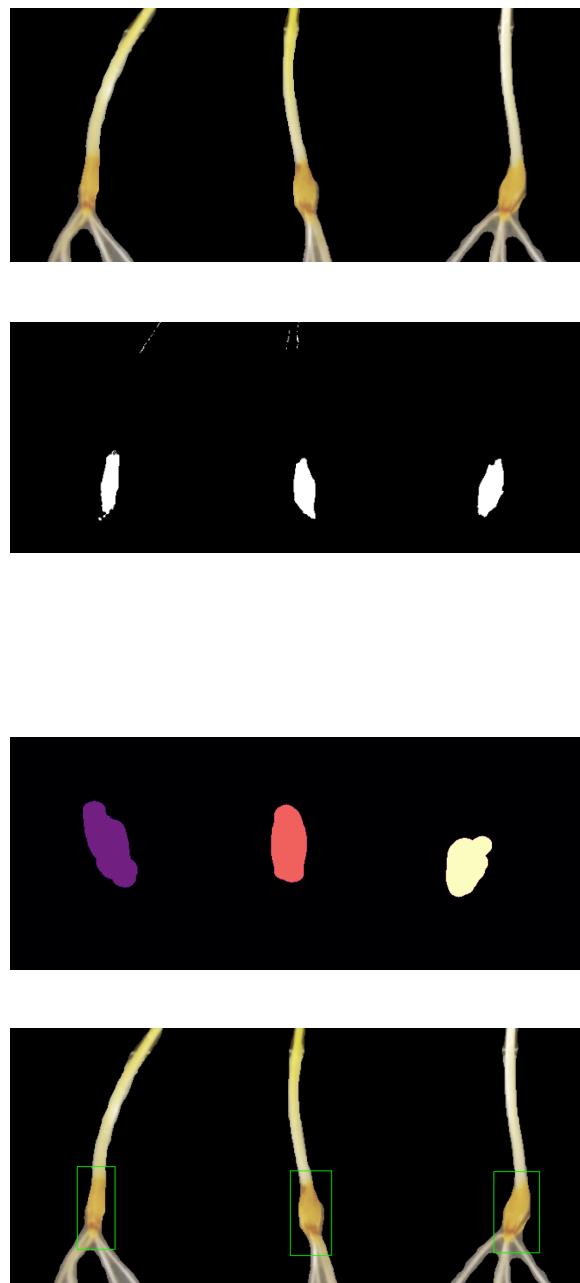
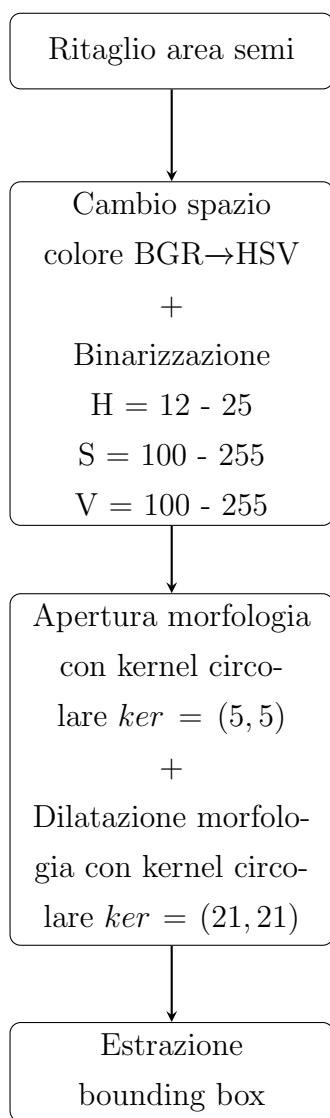


Figura 2.24: Schema localizzazione semi

2.5 Estrazione

In questa sezione discuteremo il metodo proposto per l'estrazione automatica del sistema radicale di ciascuna pianta, il quale verrà poi analizzato per estrarre varie caratteristiche quali *convex hull*, numero e spessore medio delle radici, radice più lunga ecc. In figura 2.25 lo schema generale del metodo.

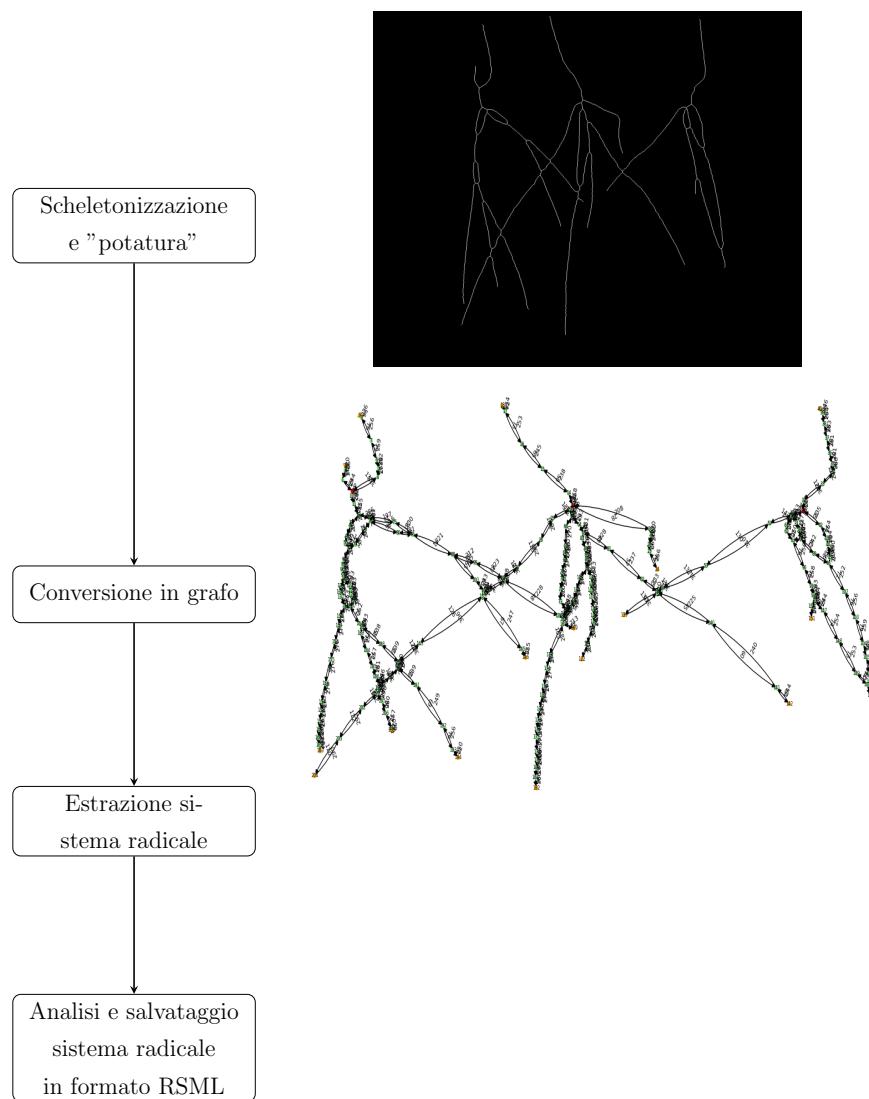


Figura 2.25: Schema estrazione

2.5.1 Skeletonizzazione e prune

Come metodo di skeletonizzazione della maschera si è scelto di utilizzare quello di tipo `medial axis` [16] il quale permette di ricavare, nell’implementazione utilizzata, anche la trasformata distanza, utile per stimare lo spessore delle radici. Lo scheletro trovato si presenta come una matrice booleana dove il valore `True` corrisponde ad un elemento dello scheletro (foreground), `False` altrimenti (background). La trasformata distanza, invece, come una matrice di floating point. L’operazione di skeletonizzazione può produrre delle corte protrusioni, o *branch*, dovute a delle irregolarità presenti nella maschera (figura 2.26), per questo motivo è necessaria una fase di *prune* o ”potatura” in modo che queste non vengano riconosciute come elementi terminali. Per indicare un punto terminale si userà sia il termine *tip* che *endpoint*.



Figura 2.26: Scheletro con protrusioni

Inizialmente si è provato un prune di tipo morfologico [17] ma non produceva i risultati attesi (figura 2.28b). Si è deciso quindi di creare la seguente

procedura 1: trovare tutti i nodi e, da ognuno, "camminare", o *walk*, fino agli endpoint che può raggiungere. Per ogni cammino, il quale consiste in una lista ordinata di punti, se la sua lunghezza è inferiore ad una certa soglia (nel nostro caso 10) ai punti corrispondenti nella matrice viene assegnato il valore **False**, effettuando così una "rimozione" della protrusione. I nodi si ricavano attraverso una operazione di morfologia di tipo **Hit-Or-Miss** con i kernel riportati in figura 2.27. I valori dei kernel, usati in questa implementazione, possono assumere i seguenti valori:

- 1: nella posizione indicata ci deve essere un punto di foreground
- -1: nella posizione indicata ci deve essere un punto di background
- 0: nella posizione indicata ci deve essere o un punto di foreground o di background

0	1	0
1	1	1
0	0	0

1	0	1
0	1	0
1	0	0

1	0	1
0	1	0
0	1	0

0	1	0
1	1	0
0	0	1

0	0	1
1	1	1
0	1	0

0	1	0
1	1	1
0	1	0

1	0	1
0	1	0
1	0	1

Figura 2.27: Kernel per individuare tutti i possibili i nodi. I primi cinque vengono fatti ruotare, ognuno, di altri 90° , 180° e 270° .

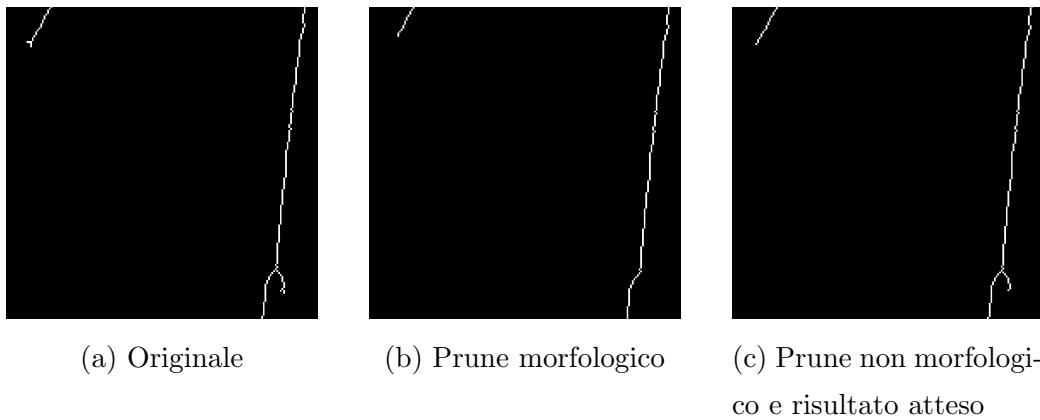


Figura 2.28: Confronto tra prune morfologico e quello realizzato da noi

Algorithm 1 Algoritmo di prune

```

1: function PRUNE(skel, branch_threshold)
2:   pruned  $\leftarrow$  copy of skel
3:   nodes_idx  $\leftarrow$  get nodes coordinates from skel
4:   for node in nodes_idx do
5:     neighbors  $\leftarrow$  valid_neighbors of node in pruned
6:     for n in neighbors do
7:       endpoint_type, path  $\leftarrow$  walk(pruned, node, n)
8:       points  $\leftarrow$  points from path excluding node
9:       if endpoint_type is PointType.TIP and
10:          length(points)  $\leq$  branch_threshold then
11:            for point in points do
12:              pruned[point]  $\leftarrow$  False
13:            end for
14:          end if
15:        end for
16:      end for
17:    return pruned
18: end function

```

Algorithm 2 Funzione per estrarre il cammino dato il nodo/punto e un suo vicino

```

1: function WALK(skel, sender_point, start_point)
2:   prev_point  $\leftarrow$  sender_point
3:   cur_point  $\leftarrow$  start_point
4:   initialize array path with sender_point
5:   res  $\leftarrow$  PointType.NODE
6:   while True do
7:     n  $\leftarrow$  valid_neighbors of cur_point in skel
8:     if prev_point  $\in$  n then
9:       remove prev_point from n
10:      end if
11:      add cur_point to path
12:      if length(n)  $>$  1 then                                 $\triangleright$  The point is a node
13:        break
14:      end if
15:      if length(n) = 1 then                                 $\triangleright$  The point is a tip
16:        res  $\leftarrow$  PointType.TIP
17:        break
18:      end if
19:      prev_point  $\leftarrow$  cur_point
20:      cur_point  $\leftarrow$  n[0]
21:    end while
22:    return res, new Path(path)
23: end function

```

Nella funzione 2 come valore di ritorno compare l’oggetto `Path`: questo verrà discusso più avanti nella sezione 2.5.2, per il momento ci interessa solamente che contiene la proprietà `points` che ritorna i punti che contiene il cammino. Nelle funzioni 1 e 2 si utilizza la funzione `valid_neighbors`: questa restituisce le coordinate dei vicini validi di un punto p in coordinate (y, x) all’interno dello scheletro. La funzione prima calcola tutte le coordinate dei punti vicini al punto, sia quelli che appartengono allo scheletro che quelli non (figura 2.29), poi filtra solo quelli appartenenti effettivamente allo scheletro, togliendo il punto p .

Infine, da quelli validi, si esegue un filtro con i seguenti criteri locali all’intorno:

- se in `N` c’è un punto valido, eventuali punti validi in `NW` e `NE` vengono scartati
- se in `W` c’è un punto valido, eventuali punti validi in `NW` e `SW` vengono scartati
- se in `S` c’è un punto valido, eventuali punti validi in `SW` e `SE` vengono scartati
- se in `E` c’è un punto valido, eventuali punti validi in `NE` e `SE` vengono scartati

Questo è necessario per avere un comportamento corretto nel percorrere i punti dello scheletro. In figura 2.30 è riportata un visualizzazione grafica delle regole mentre in figura 2.31, alcuni esempi.

In figura 2.32 un esempio dello scheletro prodotto.

(y-1, x-1)	(y-1, x)	(y-1, x+1)	NW	N	NE
(y, x-1)	(y, x)	(y, x+1)	W	C	E
(y+1, x-1)	(y+1, x)	(y+1, x+1)	SW	S	SE

Figura 2.29: A sinistra le coordinate dei vicini di un punto p a coordinate (y, x) . A destra il sistema di riferimento locale all'intorno.

X	V	X	X			
	P			P		
			X			

Figura 2.30: Regole di eliminazione vicini validi. La 'V' indica un vicino valido di 'P', le 'X' indicano eventuali vicini eliminati.

F		
	P	
	F	F

V		
	P	
	V	X

F		F
F	P	
F		

X		V
V	P	
X		

F	F	F
F	P	
F		

X	V	X
V	P	
X		

F		F
	P	
F		F

V		V
	P	
V		V

F	F	F
F	P	F
F	F	F

X	V	X
V	P	V
X	V	X

Figura 2.31: Possibili configurazioni di vicini validi: a sinistra tutti i punti di foreground (F) vicini a P, mentre a destra solo quelli validi: le 'V' corrispondono ai vicini validi di 'P', le 'X' ai vicini eliminati, mentre gli spazi vuoti a punti di background.

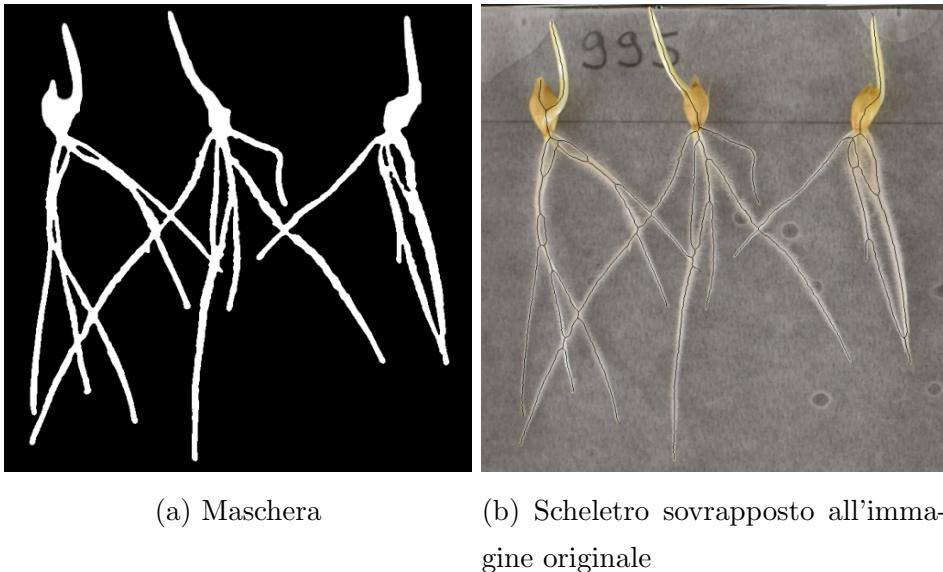


Figura 2.32: Skeletonizzazione della maschera

2.5.2 Creazione grafo

In questa sezione affronteremo il problema della conversione dello scheletro, trovato nella sezione precedente, in un grafo orientato in modo da avere una maggiore facilità di esplorazione della struttura.

Come prima cosa, per ogni nodo dello scheletro si individuano i percorsi fino ai nodi vicini o agli endpoint; questi poi vengono suddivisi in sottopercorsi in modo che l'errore di una retta di regressione, fatta passare per i punti dei sottopercorsi, sia minore o uguale a una certa soglia (nel nostro caso 1.0). In questo modo si approssimano elementi curvi dello scheletro in linee spezzate. In figura 2.33 è riportato un esempio.

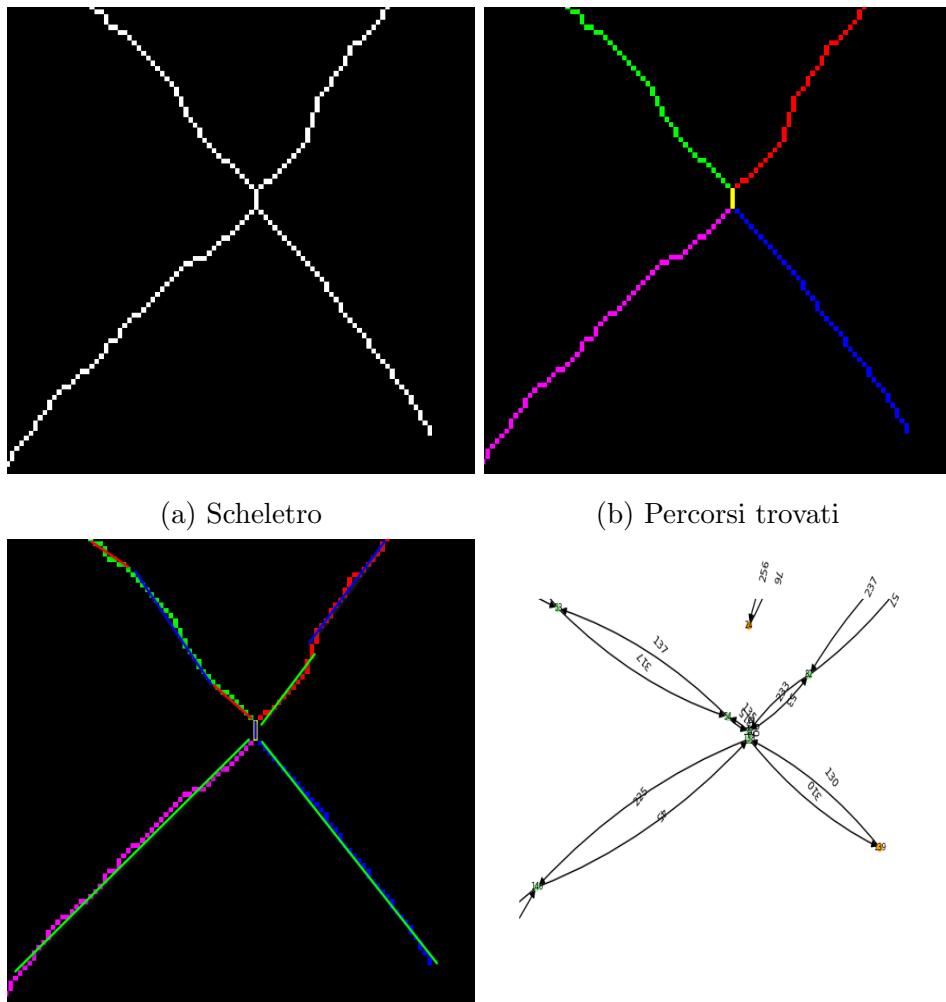
Gli endpoint dei sottopercorsi vengono poi trasformati in nodi del grafo mentre gli archi rappresentano le loro congiunzioni. Nei nodi del grafo vengono memorizzate le coordinate (y, x) corrispondenti alla posizione dell'endpoint e la tipologia, che può essere nodo, tip (endpoint) o sorgente. L'individuazione del nodo sorgente, o *forking node*, è spiegato in figura 2.35. Negli archi, invece, vengono memorizzati la lista dei punti del sottopercorso associato e relativa lunghezza (numero di punti). Ogni arco è bidirezionale e il peso as-

sociato è l'angolo tra i due nodi (figura 2.34); questa scelta è stata fatta per semplificare l'algoritmo di individuazione delle radici.

In 3 è proposto lo pseudocodice per la creazione del grafo.

Ritorniamo all'oggetto `Path`, la cui spiegazione era stata lasciata in sospeso nel precedente sottocapitolo: l'oggetto rappresenta il percorso e contiene la funzione `obtain_subpaths`, la quale suddivide il percorso in sottopercorsi secondo il metodo discusso.

In figura 2.36 un dettaglio di una porzione del grafo e la rappresentazione finale dello scheletro come grafo.



(c) Suddivisione approssimativa dei percorsi in sottopercorsi. Ogni linea rappresenta una retta di regressione e, i punti coperti da questa, i sottopercorsi

d) Corrispondente porzione di grafo

Figura 2.33: Esempio di suddivisione di un dettaglio dello scheletro in percorsi e sottopercorsi. Nota: il grafo e lo scheletro non sono in scala

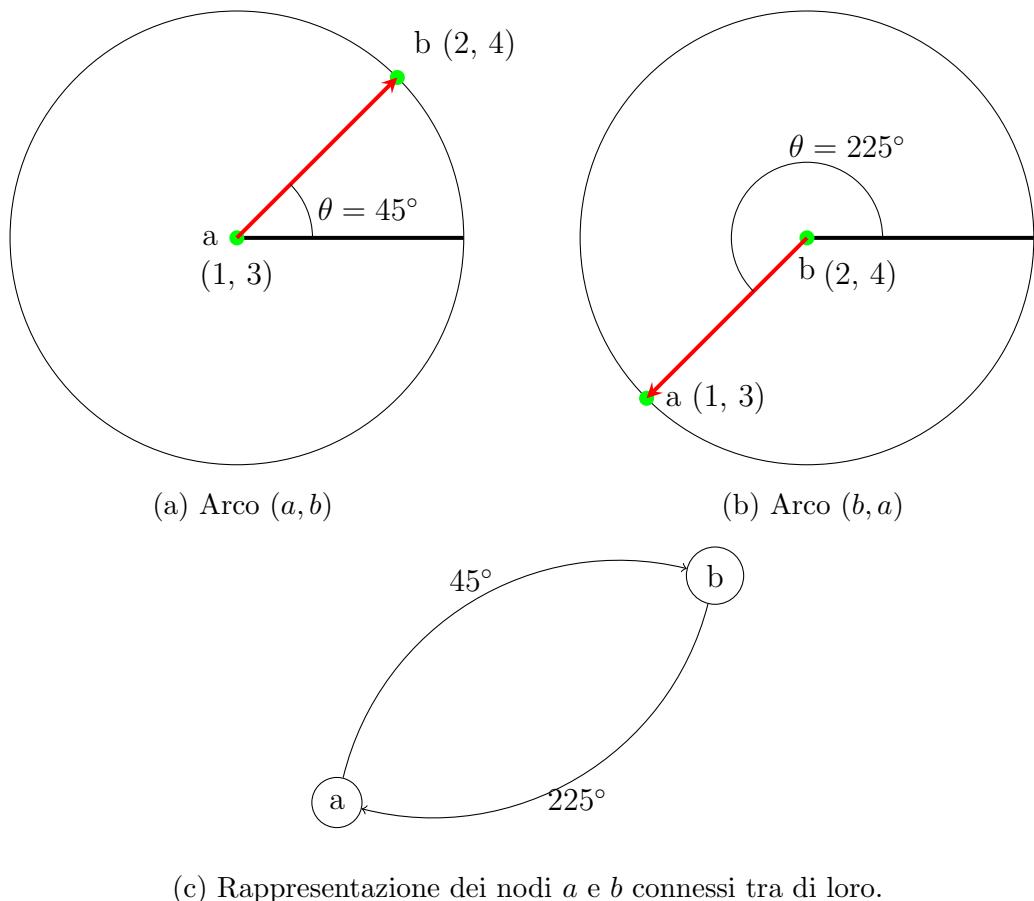


Figura 2.34: Rappresentazione di come viene calcolato l'angolo, ovvero il peso, degli archi. Per semplicità, si considera l'asse delle y che cresce verso l'alto.

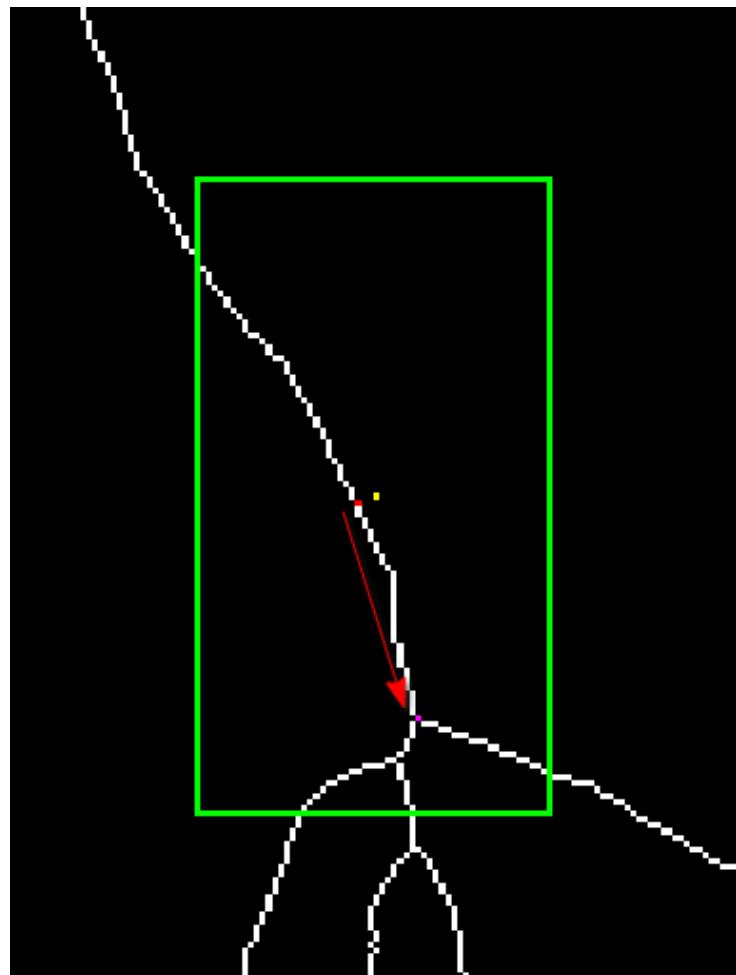


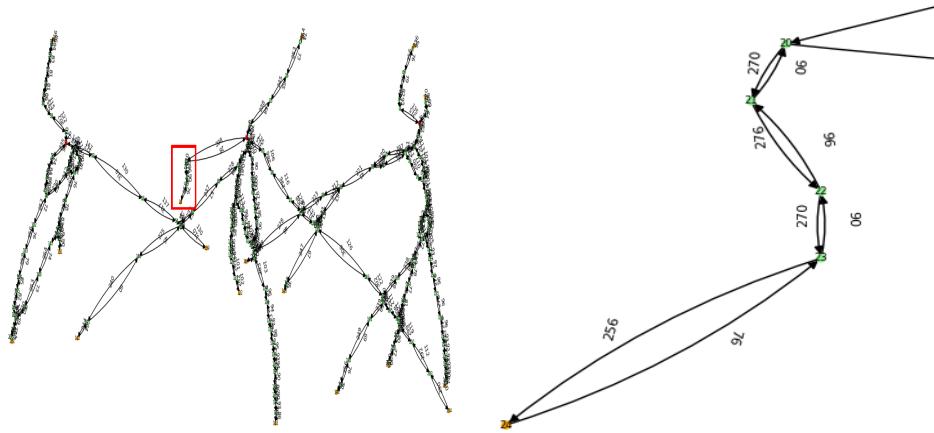
Figura 2.35: Ricerca del *forking node*. In giallo il centroide della bounding box dei semi trovata nella sezione 2.4.2 e in rosso il punto sullo scheletro più vicino a questa. Si percorre lo scheletro verso il basso fino al primo nodo (in magenta): questo nodo verrà etichettato come nodo sorgente della pianta. La stessa operazione la si ripete per tutte le bounding box trovate.

Algorithm 3 Funzione che crea il grafo date le bounding box dei semi e lo scheletro

```

1: function CREATEGRAPH(seeds_bounding_box_centroids, skel)
2:   queue  $\leftarrow$  empty queue
3:   G  $\leftarrow$  empty directed graph
4:   visited_neighbors  $\leftarrow$  empty set
5:   skel  $\leftarrow$  skeleton
6:   nodes_idx  $\leftarrow$  get nodes coordinates from skel
7:   for node in nodes_idx do
8:     neighbors  $\leftarrow$  valid_neighbors of node in skel
9:     neighbors  $\leftarrow$  neighbors excluding those contained in visited_neighbors
10:    for n in neighbors do
11:      endpoint_type, path  $\leftarrow$  walk(skel, node, n)
12:      points  $\leftarrow$  points from path excluding n
13:      add penultimate_point from path to visited_neighbors
14:      subpaths  $\leftarrow$  obtain_subpaths from path
15:      for subpath in subpaths do
16:        add nodes and edges to G given subpath
17:      end for
18:      if endpoint_type is PointType.TIP then
19:        G.nodes[path.endpoint]['node_type']  $\leftarrow$  PointType.TIP
20:      end if
21:    end for
22:  end for
23:  for centroid in seeds_bounding_box_centroids do
24:    forking_node  $\leftarrow$  find forking node from centroid in skel
25:    G.nodes[forking_node]['node_type']  $\leftarrow$  PointType.SOURCE
26:  end for
27:  convert nodes label to attribute pos
28:  set nodes label to integer value
29:  return G
30: end function

```



(a) Rappresentazione finale del grafo (b) Dettaglio ingrandito del grafo

Figura 2.36: Rappresentazione di un dettaglio del grafo e quello completo. I nodi rossi rappresentano i nodi sorgenti da cui partono le radici mentre quelli arancioni i nodi terminali. Nota: le immagini sono state specchiate per avere angoli coerenti rispetto alle immagini.

2.5.3 Estrazione sistema radicale

In questa sezione si descriverà l'algoritmo per l'estrazione del sistema radicale di ciascuna pianta individuata.

Osservando le radici si è osservato che tendono ad avere un andamento regolare, ovvero che non presentano dei bruschi cambiamenti di angolazione, bensì il cambiamento risulta graduale. Partendo da questa osservazione si è pensato di adottare un sistema di "navigazione" basato principalmente su una regola euristica: attraversare gli archi la cui differenza di peso (angolo) tra gli archi vicini è minore (figura 2.37) fino ad arrivare un nodo terminale.

Come però si può sempre vedere dalla figura 2.37, alcuni archi sono più lunghi e altri più corti; prendiamo in considerazione gli archi segnalati dalle frecce verdi (partendo sempre dal nodo magenta): il primo arco è piuttosto esteso, indicando che quella radice sta seguendo attualmente quella traiettoria, mentre il secondo è molto corto (presumibilmente un sottopercorso composto da tre-quattro punti) con un cambiamento di traiettoria relativa-

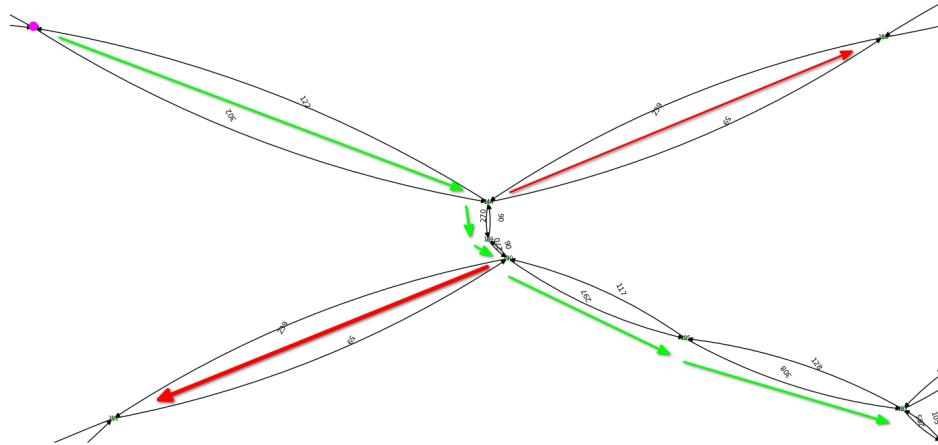


Figura 2.37: Esempio di attraversamento degli archi. Il punto magenta è il nodo di partenza, le frecce verdi indicano gli archi percorsi mentre quelle rosse gli archi confrontati ma scartati.

mente brusco se confrontato al primo, indicando che molto probabilmente starà attraversando una zona leggermente curva. Confrontando quindi l'arco corrente direttamente con i suoi vicini si rischierebbe di percorrere archi che appartengono ad altre radici. È necessario quindi un modo di regolarizzare l'andamento della radice, valutando con un peso maggiore l'angolo degli archi con una lunghezza maggiore rispetto a quelli con lunghezza minore. Per ovviare a questo problema si è scelto di utilizzare un filtro di tipo *Exponential Moving Average* (EMA): un filtro infinite-impulse response (IIR) il quale mette maggiormente in evidenza i dati recenti, riducendo l'importanza dei dati più vecchi in modo esponenziale [18]. L'equazione è definita come segue:

$$y[i] = \alpha \cdot x[i] + (1 - \alpha) \cdot y[i - 1] \quad (2.5)$$

dove:

- y è il valore dell'output
 - x è il valore dell'input
 - i indica l'indice dell'elemento

- α indica quanto è forte l'effetto di smooth. È un valore compreso nell'intervallo $[0, 1]$

Visto che il calcolo del nuovo valore dipende solo dal valore precedente di y , nel nostro caso l'equazione la si può riscrivere come:

$$\text{angle_ema} = \alpha * \text{angle_to_smooth} + (1 - \alpha) * \text{angle_ema} \quad (2.6)$$

dove angle_ema è inizializzato con il peso del primo arco della radice.

Più $\alpha \rightarrow 0$, maggiore sarà l'effetto di smoothing; viceversa più $\alpha \rightarrow 1$, minore sarà l'effetto. Il valore di α viene scelto ogni volta che si percorre un nuovo arco secondo il seguente criterio:

- se la lunghezza dell'arco è maggiore di 25 (punti), allora $\alpha = 0.8$
- se la lunghezza dell'arco è minore di 25, allora α dipende dalla lunghezza ed è mappato linearmente tra 0.01 (la lunghezza è uguale a 1) e 0.76 (la lunghezza è uguale a 24)

La funzione di mappatura è definita come segue:

$$\alpha = \frac{(x - \text{in_min}) \cdot (\text{out_max} - \text{out_min})}{(\text{in_max} - \text{in_min}) + \text{out_min}} \quad (2.7)$$

dove:

- $\text{in_min} = 1$
- $\text{in_max} = 24$
- $\text{out_min} = 0.01$
- $\text{out_max} = 0.76$

L'idea quindi è quella di aggiungere, alla struttura dati della radice, una variabile che contiene il valore attuale dell'angolo regolarizzato (angle_ema) la quale viene aggiornata, secondo le specifiche definite sopra, ogni volta che la radice attraversa un arco.

L'algoritmo di estrazione del sistema radicale delle piante, invece, lo si può riassumere nei seguenti punti:

1. per ogni nodo sorgente si inizializza la struttura dati della pianta con relativo nodo
2. per ogni pianta, partendo dalla sorgente, si trova il gambo
3. per ogni pianta si trovano gli archi adiacenti al seme e, per ognuno, si inizializza la radice che parte da quell'arco e la si esplora
4. le piante, a turno, inizializzano una nuova radice e la fanno esplorare
5. quando tutte le piante non hanno più radici da esplorare, l'algoritmo termina

Per trovare il gambo, o *stem*, si trova il nodo vicino a quello sorgente che è il più in alto e, dall'arco (*sorgente, vicino_trovato*), si esplora il gambo come se fosse una radice. Per la ricerca del gambo si potrebbero adottare un metodo più intelligente come esplorare, per ogni arco adiacente al nodo sorgente, una radice e trovare quella che ha la maggior parte dei nodi al di sopra del nodo sorgente in modo da filtrare eventuali radici che crescono al di sopra del nodo sorgente (si presuppone che i gandi crescano più in alto rispetto alle radici). Ad ogni modo il metodo banale implementato fornisce comunque dei risultati soddisfacenti.

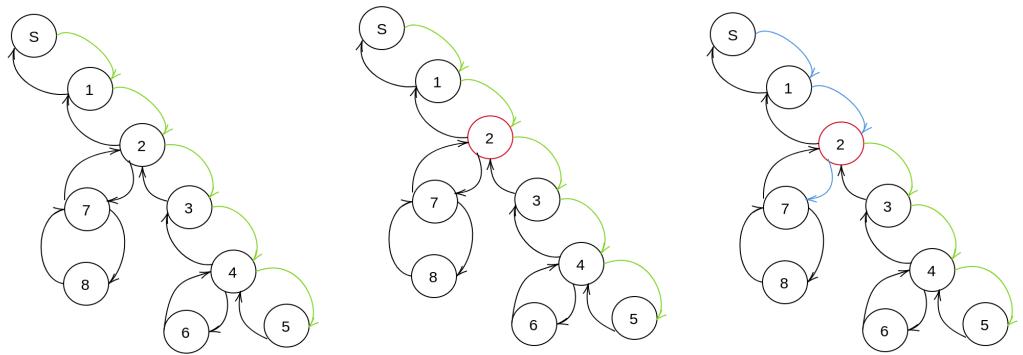
Le operazioni che svolgono a turno le piante consistono nel:

1. inizializzare una nuova radice
 - (a) si trova il nodo con degli archi adiacenti non ancora esplorati tra le radici che contiene la pianta (figura 2.38b)
 - (b) trovato il nodo, si inizializza una nuova radice e si aggiungono gli stessi archi percorsi dalla radice trovata fino a quel nodo, aggiungendo infine, alla nuova radice, uno degli archi adiacenti non esplorati (figura 2.38c): l'esplorazione quindi continuerà da quell'arco (figura 2.38d)

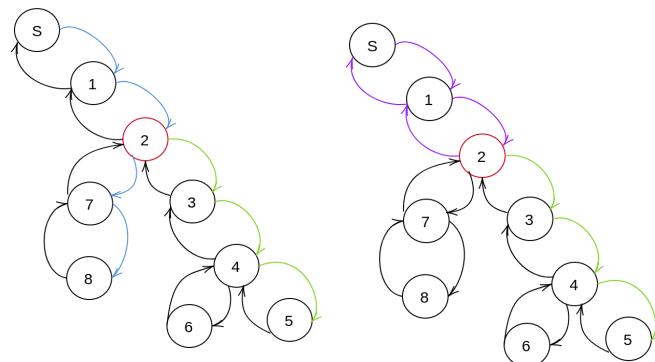
2. esplorare la radice

L'idea, quindi, è quella di prima esplorare tutte le radici che partono dagli archi direttamente adiacenti al seme della pianta, in quanto si presuppone che da quegli archi partano radici appartenenti per forza alla pianta. Una volta esplorate tutte le radici che partono da quegli archi, si fa l'inizializzazione delle nuove radici come descritto sopra per ovviare al problema in cui le radici si sovrappongano tra di loro come illustrato nell'esempio di figura 2.39.

L'esplorazione della radice consiste nel metodo descritto più in alto dando, in più, la priorità agli archi ancora non percorsi/attraversati, eccetto il caso in cui la differenza tra *angle_ema* e il peso dell'arco che si vuole percorrere è maggiore di 55° . Ogni volta che la radice esplora un arco, questo viene aggiunto ad una sua lista interna di archi esplorati in modo da ricostruire il percorso fatto. Un arco è considerato percorso se o l'arco stesso è stato percorso o l'arco in senso contrario è già stato percorso; questa regola prevenire la possibilità che, nell'inizializzazione di una radice a partire da un nodo, questa parta dall'ultimo arco e inizi a percorrere la stessa radice ma in senso contrario (figura 2.38e). Facendo esplorare alle piante una radice alla volta si cerca di evitare che non vengano inizializzate, ed esplorate, radici che non fanno parte della pianta. In 4 è riportato un possibile pseudocodice dell'algoritmo, mentre in 2.40 un basico diagramma delle classi.

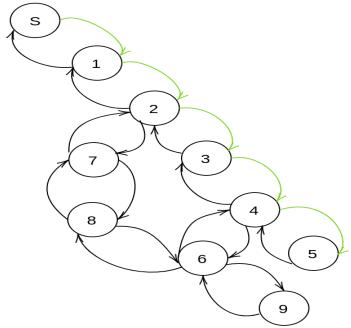


- (a) In verde il percorso di una radice completamente esplorata
 (b) Il nodo rosso è il nodo più in alto trovato
 (c) In blu il percorso della nuova radice inizializzata

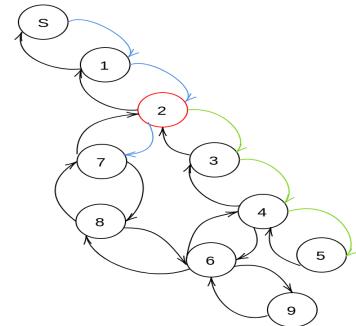


- (d) In blu il percorso della nuova radice completamente esplorata
 (e) In viola il percorso della radice esplorata male

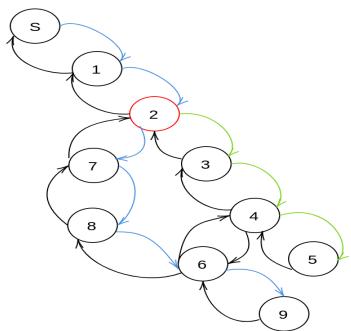
Figura 2.38: Esempio di inizializzazione ed esplorazione di una nuova radice



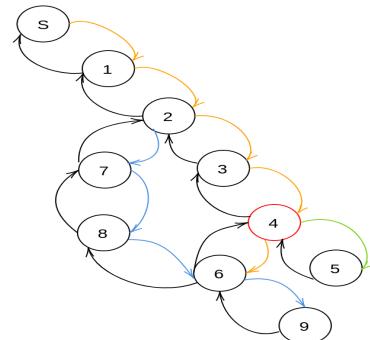
(a) Prima radice già esplorata nell'inizializzazione della pianta



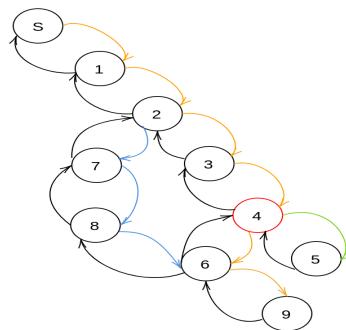
(b) Inizializzazione della seconda radice a partire dal nodo 2



(c) Esplorazione della seconda radice



(d) Inizializzazione della terza radice a partire dal nodo 4



(e) Esplorazione della terza radice

Figura 2.39: Esempio di esplorazione con radici sovrapposte.

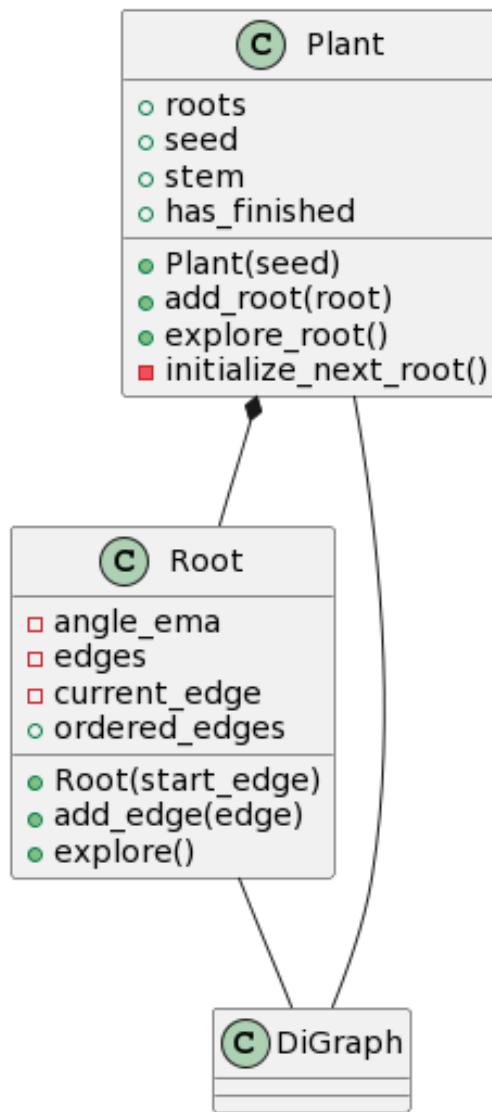


Figura 2.40: Possibile diagramma delle classi. La classe **DiGraph** rappresenta un grafo orientato

Algorithm 4 Funzione per l'estrazione dato il grafo G

```
1: function EXTRACT( $G$ )
2:    $plants \leftarrow$  empty array
3:    $seeds \leftarrow$  get nodes who are the sources from  $G$ 
4:   set  $G$  visible to Root and Plant classes
5:   for  $seed$  in  $seeds$  do
6:      $plant \leftarrow$  initialize new Plant with  $seed$ 
7:     add  $plant$  to  $plants$ 
8:   end for
9:   while True do
10:    if all plants in  $plants$  have finished to explore then
11:      break
12:    end if
13:    for  $plant$  in  $plants$  do
14:      if  $plant.has\_finished$  then
15:         $plant.explore\_root()$ 
16:      end if
17:    end for
18:  end while
19:  return  $plants$ 
20: end function
```

Algorithm 5 Classe *Plant* parte 1

```
1: roots ← empty array
2: graph ← None
3: has_finished ← False
4: stem ← None

5: function INIT(seed)
6:   seed_node_neighbors ← neighbors node of seed
7:   stem_start_node ← find stem starting node from seed_node_neighbors
8:   remove stem_start_node from seed_node_neighbors
9:   stem ← initialize new Root with edge (seed, stem_start_node)
10:  stem.explore()
11:  for neighbor in seed_node_neighbors do
12:    root ← initialize new Root with edge (seed, neighbor)
13:    root.explore()
14:    add root to roots
15:  end for
16: end function

17: function EXPLORE_ROOT( )
18:   if has_finished then
19:     return
20:   end if
21:   root ← initialize_next_root()
22:   if root is None then
23:     has_finished ← True
24:     return
25:   end if
26:   add root to roots
27:   root.explore()
28: end function
```

Algorithm 6 Classe *Plant* parte 2

```

29: function INITIALIZE_NEXT_ROOT( )
30:   new_root  $\leftarrow$  None
31:   tmp  $\leftarrow$  empty array
32:   for root in roots do
33:     highest_node_in_root  $\leftarrow$  find highest node that have at least
34:                           one non walked neighbor edge in root
35:     if highest_node_in_root is None then
36:       return None
37:     end if
38:     neighbor_edge  $\leftarrow$  get a non walked neighbor edge
39:                           from highest_node_in_root
40:     if highest_node_in_root is not None then
41:       add to tmp tuple of root and highest_node_in_root
42:     end if
43:   end for
44:   tmp_root  $\leftarrow$  find highest node in tmp
45:   found_root, node  $\leftarrow$  unpack tuple tmp_root
46:   new_root  $\leftarrow$  initialize new Root from found_root edges until node
47:   return new_root
48: end function

```

Algorithm 7 Classe *Root* parte 1

```
1: edges  $\leftarrow$  empty set
2: current_edge  $\leftarrow$  None
3: ordered_edges  $\leftarrow$  empty array
4: start_edge  $\leftarrow$  None
5: angle_ema  $\leftarrow$  0

6: function INIT(edge)
7:   start_edge  $\leftarrow$  edge
8:   add_edge(edge)
9:   angle_ema  $\leftarrow$  weight of edge
10: end function

11: function ADD_EDGE(edge)
12:   add edge to edges
13:   add edge to ordered_edges
14:   G[edge]['walked']  $\leftarrow$  True
15:   angle_ema  $\leftarrow$  calculate new smoothed angle given edge weight
16:   current_edge  $\leftarrow$  edge
17: end function
```

Algorithm 8 Classe *Root* parte 2

```

18: function EXPLORE( )
19:   while True do
20:     previous_node, current_node  $\leftarrow$  current_edge
21:     if G.nodes[current_node]['node.type'] is PointType.TIP then
22:       break
23:     end if
24:     neighbors  $\leftarrow$  neighbors node of current_node
25:     if length(neighbors) = 1 then
26:       add_edge((current_node, neighbors[0]))
27:       continue
28:     end if
29:     neighbouring_edges  $\leftarrow$  neighbouring edges of current_node with
30:                           relative absolute difference between their
31:                           weight and angle_ema
32:     minimum_edge  $\leftarrow$  edge from neighbouring_edges
33:                           with lowest difference
34:     non_walked_edges  $\leftarrow$  non walked edges from neighbouring_edges
35:     if length(non_walked_edges) > 0 then
36:       tmp_edge  $\leftarrow$  minimum edge from non_walked_edges
37:                           based on the angle_ema and weight difference
38:       if tmp_edge angle difference < 55 then
39:         minimum_edge  $\leftarrow$  tmp_edge
40:       end if
41:     end if
42:     add_edge(minimum_edge)
43:   end while
44: end function

```

2.5.4 Salvataggio

In questa sezione parleremo del salvataggio del sistema radicale delle piante presenti all'interno dell'immagine.

Per salvare il sistema radicale delle piante si è usato il formato standard **RSML**, acronimo di *Root System Markup Language* [5]: un formato basato sullo standard XML studiato per salvare l'architettura del sistema radicale delle piante secondo uno standard unificato, in modo da permettere una maggiore facilità di condivisione e interoperabilità tra i diversi programmi di analisi disponibili. Il formato RSML è suddiviso negli elementi *metadata* e *scene*, o scena. All'interno dell'elemento *metadata* vengono memorizzate le informazioni tecniche e sperimentali relativi alla scena, mentre la scena stessa contiene la rappresentazione del sistema radicale di ciascuna pianta. Brevemente, la scena rappresenta una immagine dentro alla quale possono essere presenti una o più piante, le quali possono contenere una o più radici. Per emulare questo tipo di struttura si utilizza un tipo di struttura di tipo nidificata, dove l'elemento scena conterrà una o più elementi di tipo pianta e, quest'ultimo, conterrà uno o più elementi di tipo radice. Gli elementi di tipo radice, invece, contengono un elemento di tipo geometrico il quale definisce, come suggerisce il nome, la geometria della radice per mezzo di una polilinea. Sulla polilinea si possono associare informazioni locali aggiuntive come, ad esempio, il diametro della radice in corrispondenza di specifici punti; queste informazioni vengono inserite all'interno degli elementi di tipo funzione che, come suggerisce il nome, mappano un punto sulla polilinea ad un corrispondente valore. Le informazioni globali all'intera scena o pianta o radice, come ad esempio la lunghezza di una radice, invece, vengono associate mediante elementi di tipo proprietà.

In questo lavoro generiamo un file RSML con le seguenti informazioni:

- piante con relative radici e posizione del seme
- diametro delle radici
- angolo di crescita del gambo

In figura 2.41 una rappresentazione della struttura del formato RSML.

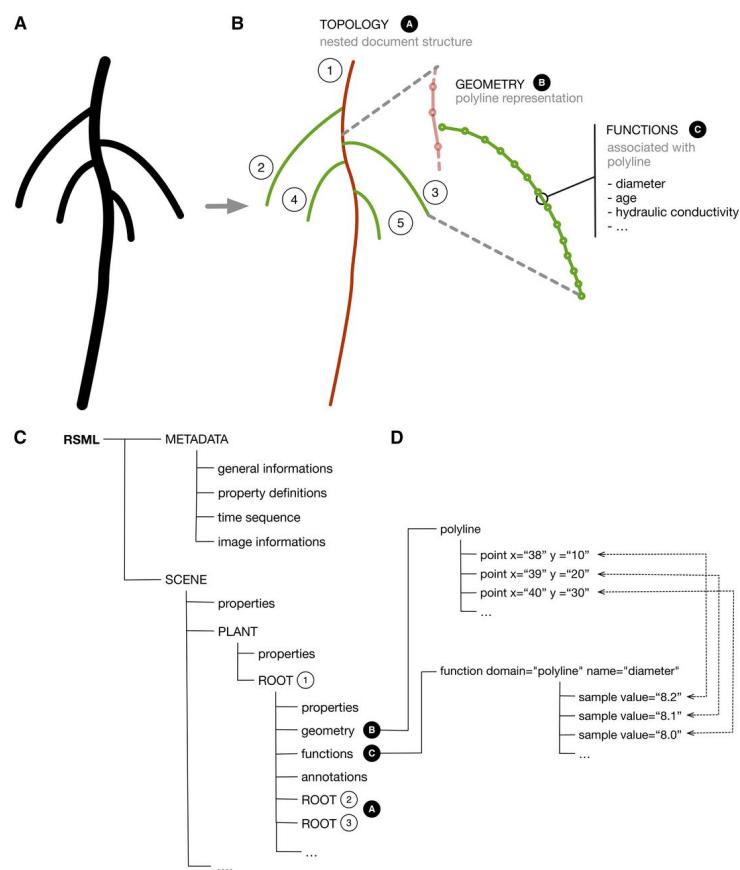


Figura 2.41: Rappresentazione visuale della struttura del formato RSML.

A partire dalla struttura dati delle piante viene generato anche un file aggiuntivo in formato **JSON** in cui, invece, sono memorizzate, per ciascuna pianta, le seguenti misure:

- numero di radici

- angolo di crescita del gambo
- spessore medio di ciascuna radice e spessore medio di tutte le radici
- area coperta dalle radici

Nel listato 2.1 un esempio del file json prodotto. Per ottenere gli spessori di una radice si ottengono, prima, tutti i gli archi che ha attraversato la radice, da questi si ottengono i punti associati e, infine, si prendono i valori della matrice delle distanze, ottenuta con la skeletonizzazione medial axis, in corrispondenza delle coordinate dei punti.

Listing 2.1: Esempio delle misure in formato JSON

```

1 {
2   "plants": [
3     {
4       "plant_id": 1,
5       "max_root_lenght": 484,
6       "roots_number": 6,
7       "stem_angle": 74.98,
8       "roots_thickness": {
9         "1": 6.23,
10        "2": 12.56,
11        "3": 7.09,
12        "4": 7.13,
13        "5": 6.21,
14        "6": 6.54
15      },
16      "average_roots_thickness": 6.7,
17      "convex_hull": 99238.5
18    },
19    {
20      "plant_id": 2,
21      "max_root_lenght": 566,
22      "roots_number": 6,

```

```
23     "stem_angle": 72.29,
24     "roots_thickness": {
25         "1": 5.32,
26         "2": 5.45,
27         "3": 5.63,
28         "4": 5.56,
29         "5": 5.95,
30         "6": 5.87
31     },
32     "average_roots_thickness": 5.66,
33     "convex_hull": 165799.0
34 },
35 {
36     "plant_id": 3,
37     "max_root_lenght": 230,
38     "roots_number": 5,
39     "stem_angle": 79.41,
40     "roots_thickness": {
41         "1": 7.02,
42         "2": 5.35,
43         "3": 5.38,
44         "4": 6.34,
45         "5": 5.69
46     },
47     "average_roots_thickness": 6.04,
48     "convex_hull": 54484.5
49 }
50 ]
51 }
```

Capitolo 3

Risultati sperimentali

L'efficacia del metodo proposto è stata valutata sul dataset descritto in sezione 1.2.1 attraverso un'analisi visiva dei risultati ottenuti. Purtroppo non avendo a disposizione un ground truth per il database non è stato possibile misurare le performance con indicatori oggettivi. Pertanto i risultati sono stati valutati solo visivamente per mezzo di immagini di debug, per valutare la qualità del post-process delle segmentazioni, e animazioni, per valutare la bontà dell'algoritmo di estrazione del sistema radicale. Uno dei problemi principali del metodo proposto rimane la corretta segmentazione della radice dallo sfondo: effettuando il post-process della maschera di segmentazione ottenuta dalla rete neurale, si è osservato in generale un notevole miglioramento della maschera di segmentazione nella maggior parte delle immagini del dataset. Permangono problemi invece nei casi in cui l'illuminazione non è costante, come si può vedere in figura 3.1

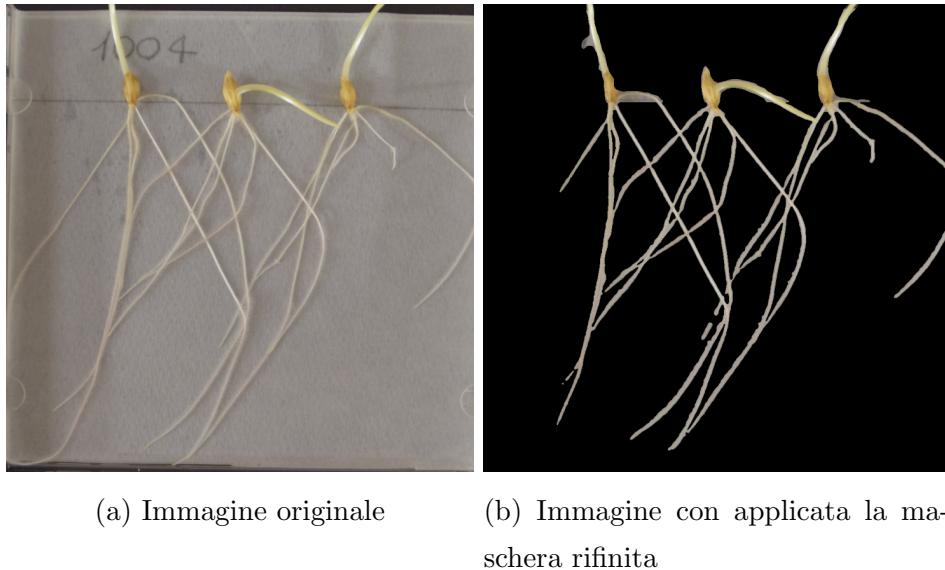


Figura 3.1: Immagine con illuminazione non costante

Un altro problema riscontrato riguarda il corretto spessore della maschera: nella maggior parte delle maschere le radici risultano avere uno spessore più grande di circa 6px, rendendo di conseguenza questa misura piuttosto imprecisa. Questo fenomeno si osserva in particolare dove ci sono peli radicali molto evidenti che, nella binarizzazione adattiva, vengono "inclusi" nella radice. In figura 3.2 un esempio.

Per quanto riguarda l'algoritmo di costruzione del grafo, invece, l'euristico di attraversamento risulta piuttosto accurato mentre, l'inizializzazione delle radici a partire dal nodo, presenta dei problemi quando sono presenti nell'immagine piante con molte radici e intrecciate tra di loro. Il problema è illustrato in figura 3.3. A parte questi tipi di problemi, con specifiche immagini, il metodo sembra funzionare piuttosto bene, soprattutto in immagini in cui le piante non si intrecciano molto tra di loro. In figura 3.4 un esempio del risultato renderizzato a partire dal file RSML generato e in figura 3.5 altri esempi. Il prototipo realizzato non sempre è in grado di estrarre misure precise, ma può essere considerato un ottimo strumento per effettuare delle stime preliminari.

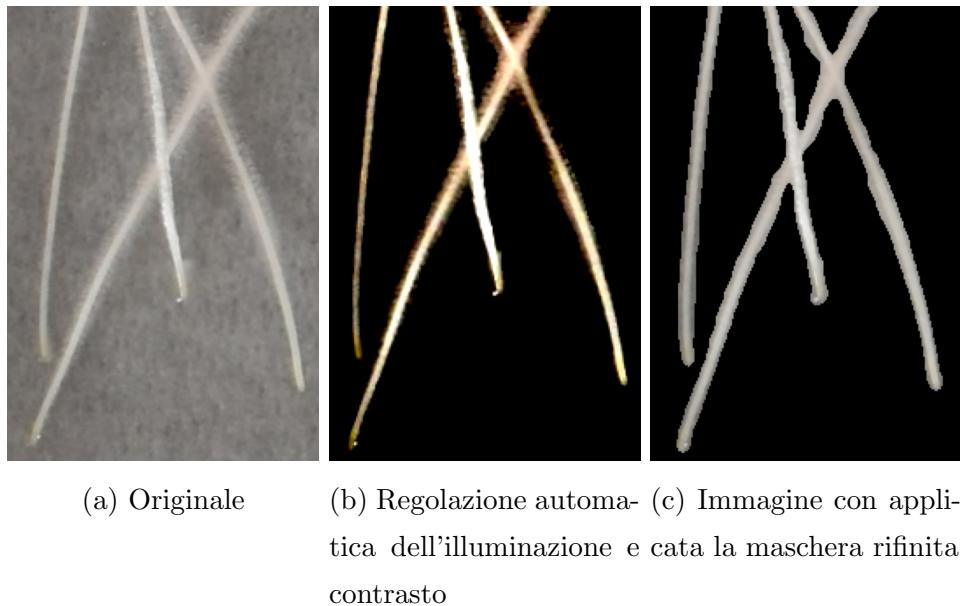


Figura 3.2: Esempio di porzione d'immagine in cui gli spessori della maschera di segmentazione non sono precisi

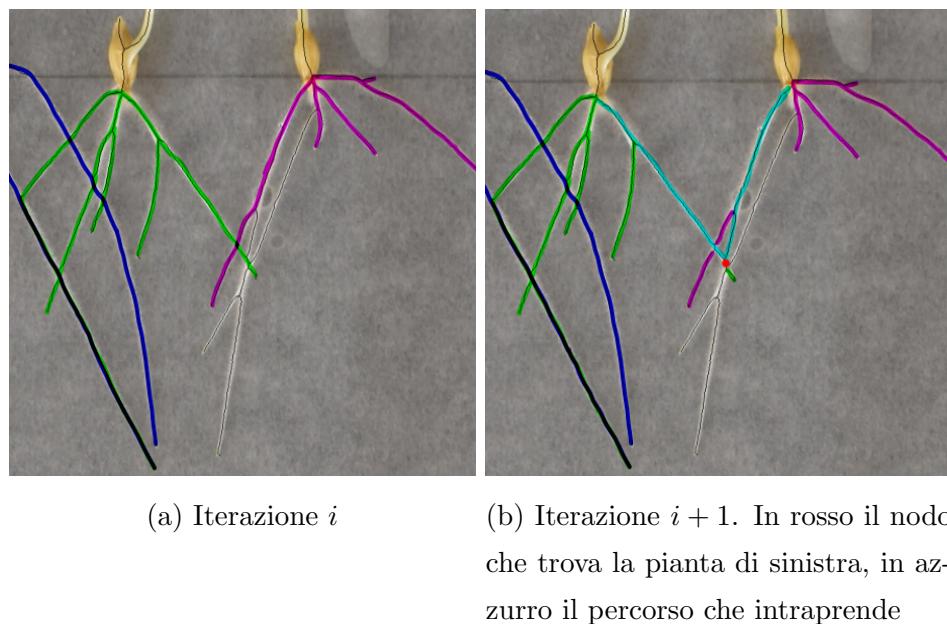
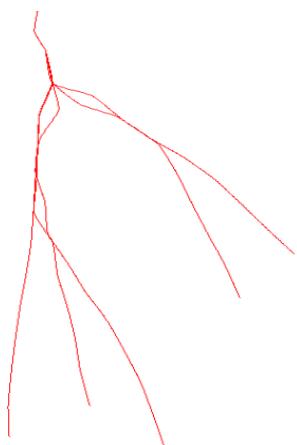


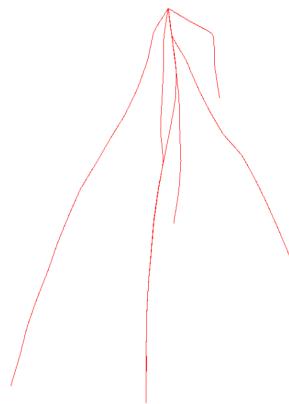
Figura 3.3: Esempio grafico del problema che si può verificare nell'algoritmo. Nell'immagine di sinistra è rappresentata lo stato delle radici alla fine dell'iterazione i , in quella di destra l'iterazione successiva ($i + 1$).



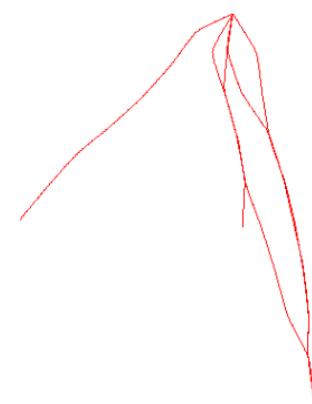
(a) Originale



(b) Prima pianta



(c) Seconda pianta



(d) Terza pianta

Figura 3.4: Esempio del metodo applicato ad una immagine. Nota: i germogli non sono visualizzati



Figura 3.5: Immagini con sovrapposto lo scheletro: l'apparato radicale di ciascuna pianta è evidenziato con un colore diverso, eccetto i gambi che sono evidenziati con il colore verde

Conclusioni

In questa tesi abbiamo realizzato un metodo completamente automatizzato per l'estrazione del sistema radicale di piante d'orzo provenienti da un specifico dataset contenente immagini ad elevata risoluzione. Per raggiungere questo obiettivo è stato necessario progettare:

- un algoritmo di ritaglio e ridimensionamento delle immagini
- un metodo di segmentazione delle immagini mediante una rete neurale profonda
- il post-processing delle maschere di segmentazione prodotte dalla rete
- la skeletonizzazione delle maschere raffinate per ottenere una struttura navigabile
- la trasformazione dello scheletro in grafo per una maggiore facilità di elaborazione
- un algoritmo di estrazione del sistema radicale

Il metodo proposto è molto lontano da essere considerato robusto ma, per il momento, produce risultati soddisfacenti con immagini la cui illuminazione è regolare, senza zone di ombra, e con piante il cui apparato radicale non è eccessivamente complesso a livello di intersezioni tra radici di altre piante. Il codice del prototipo che implementa il metodo proposto è disponibile al repository ¹.

¹<https://github.com/alebrassi/Tesi> Nota: allo stato attuale non è stato applicato nessun tipo di refactor al codice, quindi potrebbe risultare difficile da leggere e comprendere

Tra gli sviluppi futuri vi è, come prima cosa, risolvere il problema dell'algoritmo di estrazione del sistema radicale citato nel capitolo 3: purtroppo, tra difficoltà di debug dell'algoritmo stesso e dell'eterogeneità delle immagini, ci si è accorti di questo problema ad uno stadio abbastanza avanzato del progetto. Una possibile risoluzione potrebbe essere quella di, al posto di inizializzare una nuova radice per ogni pianta a partire dal nodo più alto tra le radici, cercare sempre il nodo più alto ma, questa volta tra tutte le radici di tutte le piante: nel caso in cui il nodo trovato appartenga a più radici, si può assegnare alla radice che ha un angolo di inserimento minore In secondo luogo, cercare di rimuovere la parte di post-processing delle maschere di segmentazione: questo può essere ottenuto mediante l'addestramento della rete su immagini provenienti dal nostro dataset segmentate manualmente da un annotatore esperto. In caso in cui le segmentazioni non siano comunque molto soddisfacenti, si potrebbe provare ad eseguire una seconda segmentazione dell'immagine utilizzando la tecnica *watershed* basata su marker [19], dove i marker corrisponderebbero ai punti dello scheletro ottenuto dalla maschera di segmentazione rifinita: si è osservato che lo scheletro prodotto risulta ben allineato alle radici, rendendo questa tecnica una buona candidata per risolvere il problema dello spessore delle radici. Inoltre sarebbe interessante provare ad utilizzare il filtro di Kalman [20] per ottenere una migliore predizione delle traiettorie delle radici.

Bibliografia

- [1] Roland Pieruschka and Uli Schurr. Plant phenotyping: Past, present, and future. *Plant Phenomics*, 2019, 2019. doi: 10.34133/2019/7507131. URL <https://spj.science.org/doi/abs/10.34133/2019/7507131>.
- [2] Jonathan P. Lynch. Steep, cheap and deep: an ideotype to optimize water and N acquisition by maize root systems. *Annals of Botany*, 112 (2):347–357, 01 2013. ISSN 0305-7364. doi: 10.1093/aob/mcs293. URL <https://doi.org/10.1093/aob/mcs293>.
- [3] Nicolás Gaggion, Federico Ariel, Vladimir Daric, Éric Lambert, Simon Legendre, Thomas Roulé, Alejandra Camoirano, Diego H Milone, Martin Crespi, Thomas Blein, and Enzo Ferrante. ChronoRoot: High-throughput phenotyping by deep segmentation networks reveals novel temporal parameters of plant root system architecture. *GigaScience*, 10 (7), 07 2021. ISSN 2047-217X. doi: 10.1093/gigascience/giab052. URL <https://doi.org/10.1093/gigascience/giab052>. giab052.
- [4] Robail Yasrab, Jonathan A Atkinson, Darren M Wells, Andrew P French, Tony P Pridmore, and Michael P Pound. RootNav 2.0: Deep learning for automatic navigation of complex plant root architectures. *GigaScience*, 8(11), 11 2019. ISSN 2047-217X. doi: 10.1093/gigascience/giz123. URL <https://doi.org/10.1093/gigascience/giz123>. giz123.
- [5] Guillaume Lobet, Michael P. Pound, Julien Diener, Christophe Pradal, Xavier Draye, Christophe Godin, Mathieu Javaux, Daniel Leit-

- ner, Félicien Meunier, Philippe Nacry, Tony P. Pridmore, and Andrea Schnepf. Root System Markup Language: Toward a Unified Root Architecture Description Language . *Plant Physiology*, 167(3):617–627, 01 2015. ISSN 0032-0889. doi: 10.1104/pp.114.253625. URL <https://doi.org/10.1104/pp.114.253625>.
- [6] Richard O Duda and Peter E Hart. Use of the hough transformation to detect lines and curves in pictures. *Communications of the ACM*, 15(1):11–15, 1972.
- [7] Stephen M. Pizer, E. Philip Amburn, John D. Austin, Robert Cromartie, Ari Geselowitz, Trey Greer, Bart ter Haar Romeny, John B. Zimmerman, and Karel Zuiderveld. Adaptive histogram equalization and its variations. *Computer Vision, Graphics, and Image Processing*, 39(3):355–368, 1987. ISSN 0734-189X. doi: [https://doi.org/10.1016/S0734-189X\(87\)80186-X](https://doi.org/10.1016/S0734-189X(87)80186-X). URL <https://www.sciencedirect.com/science/article/pii/S0734189X8780186X>.
- [8] Nobuyuki Otsu. A threshold selection method from gray-level histograms. *IEEE Transactions on Systems, Man, and Cybernetics*, 9(1):62–66, 1979. doi: 10.1109/TSMC.1979.4310076.
- [9] University of Nottingham. Annotated Crop Image Database. <https://plantimages.nottingham.ac.uk/>, 2022.
- [10] Wikipedia. *Hordeum vulgare* — wikipedia, l’enciclopedia libera, 2022. URL http://it.wikipedia.org/w/index.php?title=Hordeum_vulgare&oldid=127780994. [Online; in data 16-febbraio-2023].
- [11] Wikipedia. *Triticum aestivum* — wikipedia, l’enciclopedia libera, 2022. URL http://it.wikipedia.org/w/index.php?title=Triticum_aestivum&oldid=129341136. [Online; in data 16-febbraio-2023].

- [12] Kyle Seidenthal. Friendly Ground Truth, 5 2021. URL https://github.com/p2irc/friendly_ground_truth.
- [13] Wenchao Zhang, Chong Fu, Yu Zheng, Fangyuan Zhang, Yanli Zhao, and Chiu-Wing Sham. Hsnet: A hybrid semantic network for polyp segmentation. *Computers in Biology and Medicine*, 150:106173, 2022.
- [14] Paul Bourke. Histogram matching, 2011. URL <http://paulbourke.net/miscellaneous/equalisation/>.
- [15] <https://stackoverflow.com/users/11162165/nathancy>. Automatic contrast and brightness adjustment of a color photo of a sheet of paper, 2019. URL <https://stackoverflow.com/a/56909036>.
- [16] Ashley Walker Robert Fisher, Simon Perkins and Erik Wolfart. Morphology - skeletonization / medial axis transform, 2000. URL <https://homepages.inf.ed.ac.uk/rbf/HIPR2/skeleton.htm>.
- [17] Ashley Walker Robert Fisher, Simon Perkins and Erik Wolfart. Morphology - thinning, 2000. URL <https://homepages.inf.ed.ac.uk/rbf/HIPR2/thin.htm#2>.
- [18] Geoffrey Hunter. Exponential moving average (ema) filters, 2014. URL <https://blog.mbedded.ninja/programming/signal-processing/digital-filters/exponential-moving-average-ema-filter/>.
- [19] F. Meyer. Color image segmentation. 1992.
- [20] Rudolf E. Kálmán. A new approach to linear filtering and prediction problems” transaction of the asme journal of basic. 1960.

Ringraziamenti

Ringrazio i miei genitori per il costante sostegno e per avermi aiutato a superare i momenti più bui. Ringrazio i miei amici e compagni di università conosciuti in questi anni, che mi hanno aiutato e con i quali ho condiviso tanti bei momenti. Ringrazio la Prof. Lumini, la mia relatrice, per l'estrema disponibilità e per avermi supportato, e sopportato, durante lo sviluppo di questa tesi molto interessante.