UNIVERSITÀ POLITECNICA DELLE MARCHE

FACOLTÀ DI INGEGNERIA

Corso di Laurea Triennale in
Ingegneria Informatica e dell'Automazione

Stima del grado di maturazione di olive mediante immagini multispettrali VIS - NIR coregistrate

Estimation of ripening stage of olives from multi-spectral VIS NIR co-registered images

Relatore:
Dott. Mancini Adriano

Laureando:
Andreotti Vittorio

Anno Accademico 2019-2020

Indice

Capitolo 1

Introduzione

Viviamo in un mondo in cui l'evoluzione tecnologica ed informatica incide ormai su ogni tipologia di lavoro ed attività. Si tratta, di conseguenza, di un radicale cambiamento che non può essere ignorato, ma sfruttato per ottenere privilegi e uno sviluppo sostenibile. Il focus in questo lavoro è rivolto al forte legame della tecnologia con il settore agricolo che, già nel 2019, secondo una ricerca realizzata dall'**Osservatorio AgriFood** [1] del *Politecnico di Milano* in collaborazione con l'*Università degli studi di Brescia*, aveva registrato una crescita nell'uso del digitale pari al 270% rispetto al 2017. Nello specifico si evidenzia il rapporto tra le strategie tradizionali e le innovazioni della **Agricoltura 4.0** nell'ambito della raccolta, analisi e impiego dei dati al servizio della filiera agroalimentare [2]. Una delle conseguenze della suddetta evoluzione tecnologica ed informatica è, infatti, la capacità di reperire una quantità innumerevole di dati che può essere analizzata ed impiegata per migliorare la produttività e sostenibilità del settore. L'Agricoltura di precisione nasce infatti con l'idea di sfruttare al meglio le risorse a disposizione per massimizzare la produzione e ridurre l'impatto ambientale ed economico, fattori diventati cruciali nell'ultimo decennio. Mentre l'80% delle terre coltivabili è già sfruttato, l'esponenziale crescita nell'ultimo secolo della popolazione mondiale ha portato anche ad un aumento considerevole del fabbisogno alimentare e quello della risorsa più preziosa sul pianeta: l'acqua. L'obiettivo di questo settore è quindi quello di sfruttare al meglio le risorse a disposizione, dall'utilizzo intelligente di prodotti agricoli, ad un impiego sostenibile delle terre coltivabili. Sulla base di queste premesse, diventa di fondamentale importanza individuare il momento più opportuno per la raccolta dei frutti. A tale scopo, il seguente elaborato ha l'obiettivo di evidenziare quali siano i benefici nell'utilizzo di camere multispettrali rispetto alle telecamere tradizionali nella stima del grado di maturazione di olive. Già sfruttate in innumerevoli campi come la medicina, questa tipologia di telecamere è in grado di percepire caratteristiche non catturabili da un normale sensore o dall'occhio umano.

1.1 Obiettivi

Lo scopo del progetto è lo sviluppo di algoritmi per l'elaborazione di immagini multispettrali e non, muovendo i primi passi verso la classificazione del grado di maturazione delle olive con immagini di questo tipo.
Il dataset, utilizzato come base di partenza e contenente le rilevazioni, è stato fornito dall'*Università Politecnica delle Marche*.

1.1.1 Prima fase

Creazione di un dataset ottenuto processando il materiale fornito, ruotando, ridimensionando ed equalizzando le immagini.

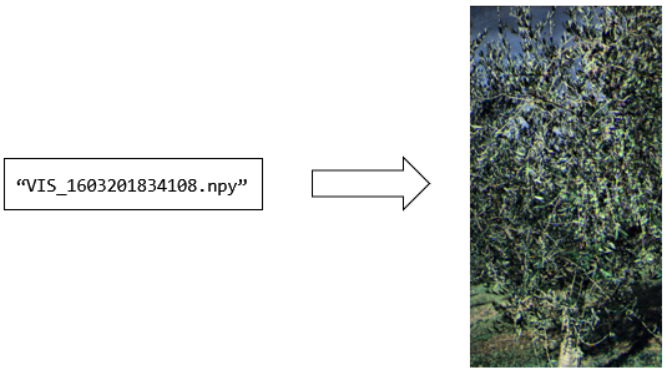


Figura 1.1 – Esempio di risultati ottenuti dall'elaborazione delle rilevazioni del dataset

1.1.2 Seconda fase

Coregistrazione delle immagini, punto cardine del progetto, e creazione di *label*, utili nella stima del *ripening*.

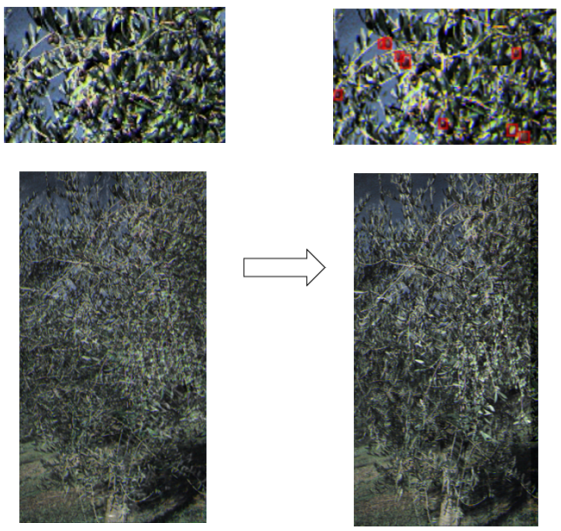


Figura 1.2 – Esempio di risultati ottenuti dalla coregistrazione e dal processo di labeling

1.1.3 Terza fase

Nell'ultima fase è mostrato un primo approccio per la classificazione della maturazione, ovvero il riconoscimento di particolari comportamenti delle olive in gruppi detti **cluster** mediante tecniche non supervisionate.

1.2 Struttura della tesi

Per rendere quanto più possibile chiaro il lavoro svolto, si espone di seguito la struttura del presente elaborato:

- **Capitolo 2:** tratta dal punto di vista teorico tutte le tecniche utilizzate e le tecnologie sfruttate per lo sviluppo. Fornisce inoltre le istruzioni per poter scaricare il materiale e il codice per poter replicare quanto fatto.
- **Capitolo 3:** illustra l'elaborazione iniziale delle immagini, includendo la scelta dei migliori canali, l'esportazione in formato `.png` in cartelle strutturate e l'equalizzazione.
- **Capitolo 4:** descrive la creazione di un *dataset* per la stima del grado di maturazione.
- **Capitolo 5:** capitolo riservato alla stima del *ripening*.
- **Capitolo 6:** presentazione dei risultati ottenuti e discussione degli sviluppi futuri.

Capitolo 2

Strumenti e metodi

Il seguente capitolo affronta gli strumenti e le tecniche di elaborazione di immagini che hanno permesso la realizzazione del progetto.

2.1 Spettro elettromagnetico

Ciò che accade quando guardiamo un oggetto è che il nostro occhio percepisce la luce riflessa e genera impulsi elettrici che vengono inviati al cervello per elaborare quanto abbiamo visto. In realtà, ciò che vediamo è solo una piccola parte dell'intera radiazione elettromagnetica riflessa dall'oggetto: l'occhio umano è in grado infatti di percepire radiazioni con una lunghezza d'onda che va da circa 390nm a 700nm [3], ma in realtà lo spettro elettromagnetico è molto più ampio.

Oggetto di studio di questo elaborato è la radiazione che ha lunghezza d'onda compresa tra i 400nm e i 1000nm circa, ovvero riguardante sia lo spettro visibile (VIS) che la radiazione del vicino infrarosso (NIR).

Nello spettro del visibile, i colori che percepiamo di un oggetto sono il risultato di risposte diverse alla luce incidente sulla sua superficie [4]. Allo stesso modo, la radiazione infrarossa viene assorbita e riflessa in modo differente dagli oggetti, in base alla diversa composizione dei materiali e alla loro struttura chimico-fisica.

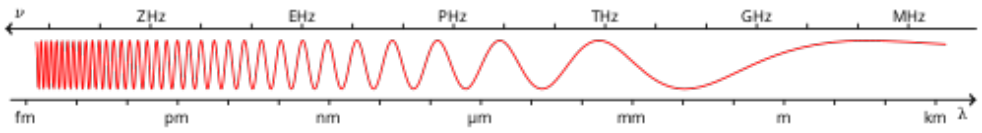


Figura 2.1 – Spettro elettromagnetico

2.1.1 VIS e NIR

In spettroscopia[1], la parte dello spettro che riguarda la luce visibile è denominata **VIS** (*Visual Spectrum*), mentre la porzione che concerne la radiazione infrarossa prende il nome di **NIR** (*Near infrared spectrum*).

Precisare la differenza tra questi due concetti è importante poiché d'ora in avanti se ne farà un largo uso.

2.1.2 Camera spettrale

Il ruolo della camera multi/iper-spettrale diventa cruciale quando, dato un oggetto, se ne vogliono evidenziare caratteristiche che altrimenti non potrebbero essere percepite dall'occhio umano. Quest'ultimo infatti è in grado di captare esclusivamente la radiazione elettromagnetica con frequenze all'interno di una banda specifica, mentre tali dispositivi sono in grado invece di catturare frequenze molto più ampie, divise per comodità in bande (il numero dipende dal sensore utilizzato).

Per la realizzazione del seguente progetto sono stati utilizzati due dispositivi (realizzati dalla IMEC[2]):

- **IMEC CMV2K-SSM4x4 VIS**: sensore che cattura la radiazione con una lunghezza d'onda compresa tra *470nm - 630nm* e suddivide lo spettro in *16* bande [5].

0	1	2	3
4	5	6	7
8	9	10	11
12	13	14	15

Figura 2.2 – Posizione dei filtri in un *mosaic pattern* per un sensore VIS.

- **IMEC CMV2K-SSM5x5 NIR**: sensore che cattura la radiazione con una lunghezza d'onda compresa tra *650nm - 950nm* e suddivide lo spettro in *25* bande [5].

0	1	2	3	4
5	6	7	8	9
10	11	12	13	14
15	16	17	18	19
20	21	22	23	24

Figura 2.3 – Posizione dei filtri in un *mosaic pattern* per un sensore NIR.

Entrambi i sensori presentano un filtro di tipo **Mosaic**, che non ricopre l'intera superficie, lasciandone scoperta una porzione. La parte coperta dal filtro è denominata *Active Area*.

[1] Parte della fisica che si occupa dello studio dello spettro elettromagnetico.

[2] Organizzazione di ricerca e sviluppo internazionale, attiva nel campo delle tecnologie digitali.

Questa tipologia di filtro è costruita in modo che abbia n righe e m colonne per ogni gruppo di $n \times m$ sotto-pixel, con un pattern che si ripete w volte per la lunghezza e h volte l'altezza dell'*active area*. La misura $n \times m$ fornisce il numero di bande catturate dal sensore.

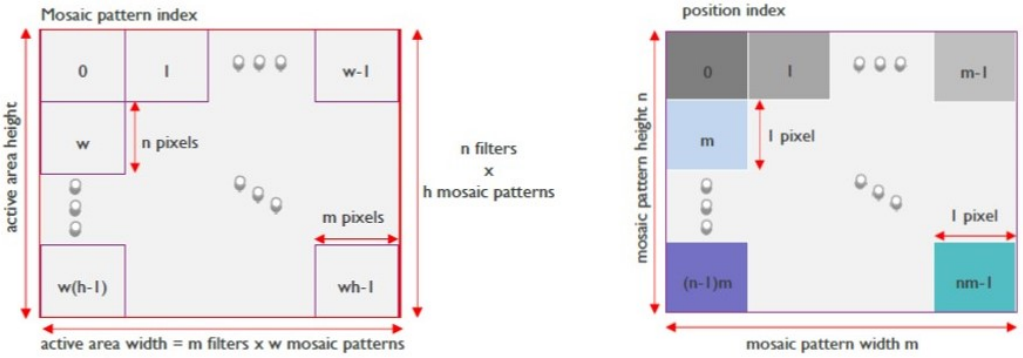


Figura 2.4 – Organizzazione del *filter mosaic*. A sinistra l'organizzazione dei pattern nell'*active area*, a destra l'organizzazione di un singolo pattern. [5]

2.2 Analisi ed equalizzazione istogramma

Nello scattare una fotografia digitale ad un soggetto, il dispositivo che viene impiegato è in grado, attraverso un sensore fotosensibile, di catturare la luce riflessa dall'oggetto. Per ottenere l'immagine digitale, il segnale luminoso viene trasformato in segnale elettrico e, tramite un convertitore A/D[3], si converte in formato digitale [6]. L'altezza e la larghezza, espressi in pixel, sono memorizzate nell'*header* del file, da cui tali informazioni vengono estratte per una corretta visualizzazione dell'immagine. A tale scopo, viene impiegata una griglia detta *raster* (si parla in questo caso di immagini **bitmap**, in modo che i pixel formino una matrice $M \times N$, dove M è il numero dei pixel in riga e N il numero di colonne. Per rappresentare il colore di ogni pixel viene utilizzato un particolare *modello di colore* [7], solitamente **RGB** quando si tratta di computer grafica (per la stampa vengono utilizzati altri modelli colore, più prestanti nelle tonalità blu-verdi, come il **CMYK**) [8].

2.2.1 Modello colore

Per modello di colore si intende un modello matematico che, utilizzando tre o quattro componenti cromatiche, è in grado di rappresentare dei colori tramite numeri, nello specifico tali numeri indicano la quantità di ognuna delle componenti utilizzate per la realizzazione di un determinato colore.

Tornando all'RGB, questo è dato da una combinazione di tre componenti: Red, Green e Blu. È un modello additivo (a differenza del **CMYK** che è invece sottrattivo), dove ogni colore si ottiene tramite somma delle varie componenti.

[3]Convertitore Analogico-Digitale

Rappresentando ogni colore con 8 bit, per ognuno di essi si hanno $2^8 = 256$ valori tonali, con la possibilità di rappresentare fino a $256 \times 256 \times 256 = 16,7$ milioni di colori diversi.

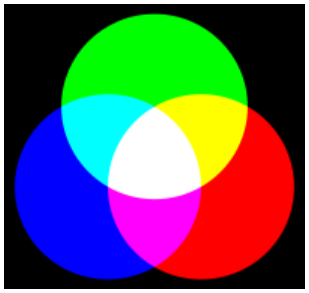


Figura 2.5 – Modello colore RGB [9]

Figura 2.6 – Modello colore CMYK [10]

2.2.2 Istogramma

La precedente introduzione alle immagini digitali consente al lettore di comprendere meglio quanto trattato nella seguente sezione.

Partendo da un'immagine, un istogramma consente di ottenere informazioni che solitamente non sono percepibili ad occhio nudo, come l'esposizione e la gamma tonale, indicatori della qualità della fotografia scattata. In particolare, esistono vari tipologie di istogramma:

- **Istogramma RGB**: rappresenta il valore dei pixel sovrapponendo i tre canali di colore (RGB);
- **Istogramma della luminosità**: rappresenta la distribuzione dei pixel in base alla loro luminosità;
- **Istogramma dei colori**: rappresenta su un unico grafico le informazioni relative a tutti i canali.

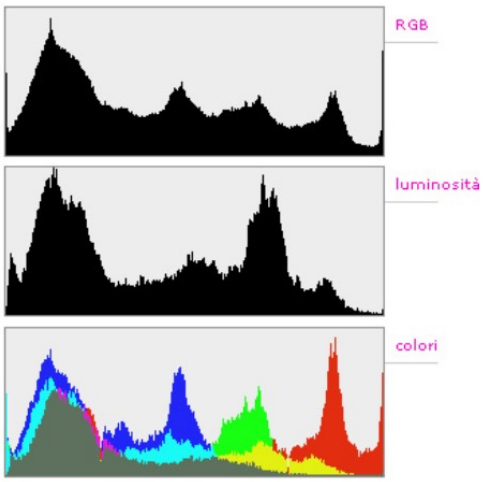


Figura 2.7 – Esempio dei 3 tipi di istogrammi [11]

Soffermando l'attenzione sull'istogramma della luminosità, questo è un utile strumento per apprendere se l'immagine risulta sovraesposta o sottoesposta. Se sull'asse delle x sono presentati i valori tonali dei pixel, sull'asse delle y è presentata la loro distribuzione, per cui un'immagine ***sottoesposta*** avrà picchi alti nella parte sinistra, un'immagine ***sovraesposta*** presenterà picchi alti nella parte di destra e ancora un'immagine risulterà ***correttamente esposta*** se presenterà picchi nella parte centrale dell'istogramma.

Figura 2.8 – Istogramma di un'immagine sottoesposta [11]

Figura 2.9 – Istogramma di un'immagine sovraesposta [11]

Figura 2.10 – Istogramma di un'immagine correttamente esposta [11]

2.2.3 Registrazione

La **registrazione di immagini** è una tecnica di elaborazione in grado di allineare geometricamente una coppia di immagini, operando con rotazioni, ridimensionamenti e inclinazioni [12]. Il suo utilizzo è necessario in molteplici contesti, il ché garantisce la possibilità di confrontare i dati ottenuti:

- **Differenti punti di vista**: le immagini inquadrano la medesima scena, ma sono acquisite da differenti angolazioni.
- **Istanti temporali diversi**: la stessa scena è catturata in istanti temporali diversi, talvolta scanditi da intervalli regolari.
- **Dispositivi diversi**: la stessa scena è acquisita da dispositivi diversi [13].

Le applicazioni sono molteplici e vanno da impieghi militari e medici, fino alla **computer vision**.

Il problema fondamentale è trovare la miglior trasformazione geometrica che possa allineare l'immagine detta **moving** a quella detta **fixed**. Per trovarla esistono numerosi algoritmi che si dividono in due grandi categorie, dove la scelta migliore dipende dal singolo caso, a seconda degli input a disposizione.

- **Feature-based**:
  Tutte le tecniche *feature-based* cercano di trovare caratteristiche conformi tra le immagini che vengono sottoposte, lavorando su ogni pixel. Gli algoritmi che ne fanno parte sono:
  - **FAST, MinEigen e Harris:** applicano la *corner detection*[4] tra le scene. Non supportano scale differenti e inclinazioni tra un'immagine ed un'altra. L'*Harris* che ha un tasso di efficienza maggiore rispetto agli altri.
  - **Brisk:** simile alle precedenti, ma supporta i ridimensionamenti e le rotazioni.
  - **Surf:** a differenza delle precedenti, applica la *blob detection*[5], supportando ridimensionamenti e rotazioni [14].
- **Intensity-based**:
  La tecnica di registratura *intensity-based* confronta le immagini nel dominio spaziale, focalizzandosi sull'ottenere per la **moving** la stessa intensità della **fixed**. A seconda dei casi, gli algoritmi sono:
  - **Monomodal intensity:** utile in tutti i casi in cui le immagini da registrare sono state ottenute dallo stesso sensore e presentano valori simili di luminosità e contrasto.
  - **Multimodal intensity:** se le immagini provengono da dispositivi diversi, questa potrebbe essere la tecnica migliore, considerata la sua maggiore efficacia lavorando su *detection* con differenti esposizioni.
  - **Phase correlation:** la migliore tra le precedenti, ma lavora nel dominio della frequenza. Funziona molto bene con immagini di diversa luminosità, ma è anche la più dispendiosa in termini computazionali [14].

Alla base di ognuno dei precedenti, la **registrazione** viene implementata secondo il seguente schema:

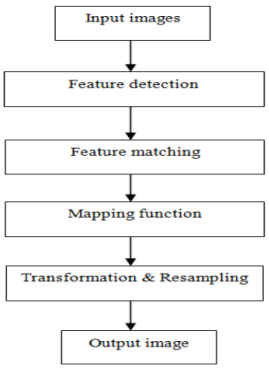


Figura 2.11 – Passi implementativi della registrazione di immagini [13]

[4]Tecnica di estrazione di caratteristiche di oggetti basata sul riconoscimento di angoli e forme.

[5]Tecnica di estrazione di caratteristiche di oggetti basata su luminosità o colori.

2.3 Clustering

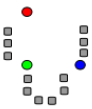
Per poter individuare comportamenti simili tra dati appartenenti alla stessa categoria, uno dei metodi a cui si può ricorrere è il **clustering**.
Adottando tale approccio i dati vengono suddivisi in un certo numero di **cluster**, dove ognuno identifica una determinata *"caratteristica"*. Il numero dipende dall'algoritmo adottato e le applicazioni riguardano tutti quei contesti che richiedono lo studio di **grandi quantità di dati**, come l'analisi di immagini, la bio-informatica e il machine-learning.
Gli **algoritmi di clustering** conosciuti sono oltre **100** e si dividono in due categorie:

- **Hard clustering**: i dati sono divisi in un certo numero di *cluster*, con la possibilità per ogni dato di appartenere ad un solo gruppo.
- **Soft clustering**: ogni dato può appartenere a più cluster, essendo più "vicino" ad uno rispetto che ad un altro.

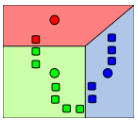
2.3.1 *k-means*

Il ***k-means*** appartiene al primo gruppo, nello specifico è tra i più conosciuti fra gli algoritmi della categoria **centroid models**. Ne fanno parte tutte quelle tecniche iterative che hanno lo scopo di trovare un elemento, detto **centroide**, che rappresenta il punto *"focale"* del cluster.
Riguardo il ***k-means***, esistono numerose varianti, di seguito ne verranno illustrate due, quella di **default** e la ***k-means++***.
Per il metodo di **default**, scelto un numero $\boldsymbol{k}$ di cluster in cui *raggruppare* le $\boldsymbol{n}$ osservazioni, si procede in modo iterativo come segue:

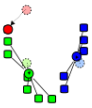
1. I $\boldsymbol{k}$ centroidi vengono scelti inizialmente in modo casuale.



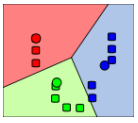
2. Per ogni centroide viene creato un cluster. Per scegliere quale dato appartiene ad un determinato gruppo esistono ancora una volta numerosi metodi.
   Un calcolo standard viene effettuato sfruttando la **distanza euclidea**.



3. Si trova la nuova collocazione dei centroidi calcolando la media per ogni *cluster* tra tutti i punti.



4. Si ripetono gli ultimi due passi fino a trovare la soluzione ottima, ovvero fin quando le nuove iterazioni non producono miglioramenti sensibili [15].



La variante ***k-means++*** sceglie i valori dei $\boldsymbol{k}$ centroidi successivi al primo in modo più efficiente rispetto al precedente:

1. Il primo centroide è scelto in modo casuale e denotato con c_1.
2. Si calcola la distanza tra ogni osservazione x_m e c_1, denotandola con $d(x_m, c_1)$.

3. Si trova il nuovo centroide c_2 in modo casuale tra le misurazioni, usando la distribuzione di probabilità $\frac{d^2(x_\mathrm{m}, c_1)}{\sum_{j=1}^{n} d^2(x_\mathrm{j}, c_1)}$.

4. Si calcola nuovamente la distanza tra ogni punto e i due centroidi, assegnando il punto al centroide più vicino.

5. Si procede in modo iterativo dal passo 3 al 4 fin quando, scelti ***k*** *clusters*, non si trovano ***k*** *centroidi [15].*

2.3.2 *DBSCAN*

Altro algoritmo frequentemente utilizzato per effettuare il clustering è il **DBSCAN**, acronimo di *Density-Based Spatial Clustering of Applications with Noise*. Come suggerisce anche il nome, appartiene alla categoria **density models**, di cui fanno parte tutti gli algoritmi che collegano i dati sfruttando densità spaziali sufficientemente alte. Solitamente viene implementato in tutte quelle applicazioni in cui non si conosce a priori il numero di cluster in cui raggruppare i dati. Per questo motivo, scelti i due parametri ε[6] e *minPts*[7], *DBSCAN* è in grado di trovare un certo numero di cluster, che può essere variato modificando in modo opportuno tali parametri.
I concetti alla base del suo funzionamento sono:

- **Density Reachability:** un punto è definito *raggiungibile* da un altro se la distanza che li separa è al massimo ε.
- **Density Connectivity:** dei punti si dicono connessi se partendo da un punto p, si riesce a raggiungere q visitando n punti p_i, con p_i raggiungibile da p_{i-1}.
- **Core point:** punto che ha almeno *minPts* a distanza ε.
- **Border point:** punto che ha almeno un *core point* a distanza ε.
- **Noise point:** punto che ha meno di *minPts* a distanza ε.

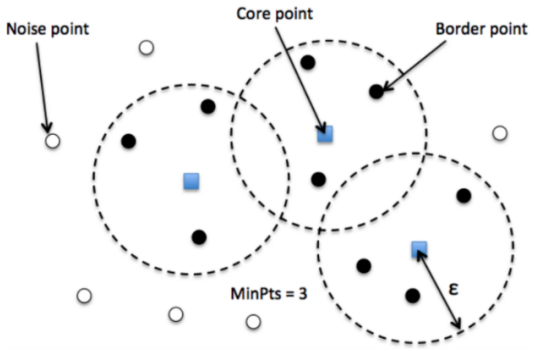


Figura 2.12 – Esempio che illustra la tipologia di punti del *DBSCAN* [16]

[6]Distanza usata per trovare i punti nell'intorno di raggio ε del punto.
[7]Numero minimo di punti per considerare densa una regione.

L'algoritmo procede nel seguente modo:

1. Viene scelto un punto *casuale* tra gli n disponibili nel dataset e per ognuno di questi si calcola la sua distanza dal punto. Se vengono trovati un numero di vicini maggiori di *minPts*, questi entrano a far parte di un nuovo cluster, altrimenti il punto viene classificato come **noise point** e, se si troverà a meno di una distanza ε nelle successive esplorazioni, potrebbe comunque entrare a far parte di un cluster.

2. Si visitano tutti i punti appartenenti al cluster trovato e per ognuno di essi vengono ricercati quelli appartenenti ad un loro intorno I_ε.

3. L'algoritmo termina quando tutti i punti sono stati visitati.

Si tenga in considerazione che, come accade per il *k-means*, esistono molti modi per determinare la distanza che consente di classificare un punto come vicino di un altro. Ancora una volta, la **distanza euclidea** è la scelta di default.
Come si può capire, le prestazioni del *DBSCAN* dipendono fortemente da come vengono scelti i parametri. Un punto di partenza potrebbe essere:

- scegliere $minPts \geq D + 1$ oppure $minPts = D \cdot 2$, con D pari alla dimensione del dataset. In ogni caso $minPts \geq 3$ per far sì che sia un *DBSCAN*[8].

- scegliere ε sfruttando un grafico ***k-distance***[9]. L'ascissa del punto in cui il grafico mostra l'inizio di una curvatura è il valore da prendere inizialmente come ε. Una ragionevole scelta è $k = minPts - 1$. Si tenga conto che un valore di ε troppo piccolo comporterebbe che una parte dei dati non vengano inclusi in nessun cluster, mentre un valore troppo grande li raggrupperebbe nello stesso cluster. La scelta oculata di ε garantisce buone performance dell'algoritmo [17].

2.3.3 Differenze tra *k-means* e *DBSCAN*

Per riassumere, di seguito verranno elencati per ogni algoritmo sia i punti di forza che i punti deboli:

- **k-means:** utile se si conosce a priori la "forma" dei dati e il numero di cluster in cui si intende classificarli. È poco dispendioso dal punto di vista computazionale, ma scegliendo un punto casuale come centroide, i risultati potrebbero non essere consistenti.

- **DBSCAN:** utile se non si conosce il numero di cluster e i dati hanno "forme" particolari. Classificando alcuni punti come **noise point**, questi non vengono "forzati" in nessun cluster. Tra gli svantaggi, sicuramente la scelta dei parametri che diventa complicata se la dimensione del dataset è elevata.

[8] con $minPts = 2$ l'algoritmo diventa **Gerarchico**.

[9] Grafico in cui è mostrato l'andamento della distanza di ogni punto dal suo k-esimo più vicino.

2.4 Strumenti software

2.4.1 MATLAB

MATLAB è un linguaggio di programmazione e un potente ambiente di sviluppo multi-piattaforma che mette a disposizione strumenti per l'elaborazione matriciale, il calcolo numerico e la creazione di algoritmi per l'analisi dei dati. È ampiamente diffuso per lo sviluppo di progetti di ricerca e aziendali. La versione utilizzata per lo sviluppo è la **R2020a** [18].

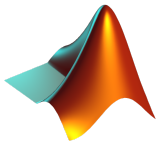


Figura 2.13 – Logo di MATLAB [19]

2.4.1.1 Toolbox

MATLAB fornisce una serie di **Toolbox** che consentono di ampliare le possibilità di sviluppo in base alle necessità.

Image Processing Toolbox:
Image Processing Toolbox è un applicativo molto utile nel processamento, analisi e visualizzazione delle immagini [20]. In particolare, si è reso fondamentale il suo utilizzo nella fase di **registrazione**.

Statistics and Machine Learning Toolbox:
Statistics and Machine Learning Toolbox fornisce particolari strumenti per descrivere, analizzare e modellare i dati per applicazioni di intelligenza artificiale [21]. Tra le tante funzioni di cui dispone, ci si è serviti in particolare del **clustering** nella fase di **stima del grado di maturazione**.

2.4.2 Labelbox

Labelbox [22] è una piattaforma di ***data labeling***, molto utile in applicazioni di machine learning. Consente di creare delle etichette e di assegnarle, mediante forme di vario tipo, ad oggetti presenti nelle immagini. Nel progetto è stato utilizzato per poter tracciare sulle immagini di piante d'olivo i *bounding box*, utili nella fase di analisi della riflettanza delle olive.

Figura 2.14 – Logo di Labelbox [23]

2.4.3 JupiterLab

JupiterLab è un interfaccia utente web-based appositamente creata per *Project Jupiter*, un ambiente di sviluppo il cui scopo è quello di raggruppare in un'unica soluzione oltre 40 linguaggi. Il nome deriva dall'unione dei 3 *core programming languages*: Julia, Python e R [24]. *JupiterLab* consente di creare *Jupiter notebook*, terminali, editor di testo e altri componenti [25]. Un *notebook* è un documento (formattato in JSON) che permette di scrivere codice e annotazioni in blocchi, che vengono eseguite dal *JupiterLab* in modo distinto.
Grazie all'esecuzione del *lab* sulle macchine dell'*Università Politecnica delle Marche*, è stato possibile lavorare in ***R*** alla pulizia delle bande rumorose, come verrà mostrato più in dettaglio nel capitolo 5.

Figura 2.15 – Logo di Jupiter [26]

2.4.4 R

R è un linguaggio e un ambiente di sviluppo per l'elaborazione statistica di dati. Come *MATLAB* è supportato da quasi tutti i Sistemi Operativi. Può essere considerata un'implementazione open-source di *S*, da cui eredita l'essere un linguaggio orientato agli oggetti. Deve la sua forza all'interfacciamento con diversi moduli che ne consentono una grande estensione delle funzionalità, che vanno dalla comunicazione con database all'esecuzione di codice *C*, *C++* e *Fortran* per i tasks più complessi.
Possiede diversi strumenti per:

- manipolazione facile e veloce dei dati, con possibilità di salvarli;
- calcolo matriciale e vettoriale;
- visualizzazione grafica dei risultati ottenuti [27].

Tutte queste caratteristiche hanno concesso di analizzare alcuni risultati e ottenere delle approssimazioni utili nella fase di **clustering**.

Figura 2.16 – Logo di R [27]

2.4.4.1 Package

Di seguito sono elencati i package utilizzati per lo sviluppo in R.

forecast:
Libreria che fornisce metodi e strumenti per l'analisi di serie storiche univariate e per la visualizzazione dei risultati [28].

mgcv:
mgcv consente di lavorare con modelli detti **GAM** (*Generalized Additive Model*), che usano funzioni come l'interpolazione con splines per la predizioni in modelli di regressione [29].

2.4.5 PuTTy

PuTTy è un client SSH e Telnet che consente di controllare da remoto un sistema informatico come se ci si trovasse davanti al suo terminale. È un software open-source e supportato da Windows, Linux e sistemi Unix-like [30].
Nello sviluppo è servito per lavorare da remoto su un *JupiterLab* eseguito sulla rete dell'*Università Politecnica delle Marche*, con il collegamento che è stato garantito dal protocollo *SSH*.

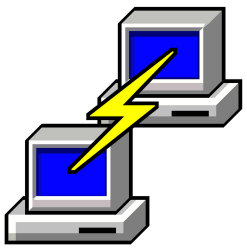


Figura 2.17 – Logo di PuTTy [30]

2.4.6 GitHub

GitHub è un software per la condivisione del codice sorgente veloce e affidabile, che nasce con l'obiettivo di consentire a grandi team di sviluppo di lavorare in modo efficiente [31]. Il suo utilizzo è risultato fondamentale nella fase di **versioning** del codice poiché ha permesso di creare diversi *branches*, uno per ogni macro-sezione di sviluppo.

Il codice sorgente del progetto può essere scaricato dal *repository* **`https://github.com/vittorioandreotti/AI4Ripening`**.

GitHub

Figura 2.18 – Logo di GitHub [32]

Capitolo 3

Elaborazioni delle immagini

Questo capitolo riflette sulle tecniche adottate e sugli algoritmi utilizzati per elaborare le immagini. Partendo da un *dataset*, verranno mostrate le procedure adoperate per ottenere le immagini suddivise in cartelle e il processo di equalizzazione, necessario per la preparazione di un *dataset* utile nel passaggio successivo.

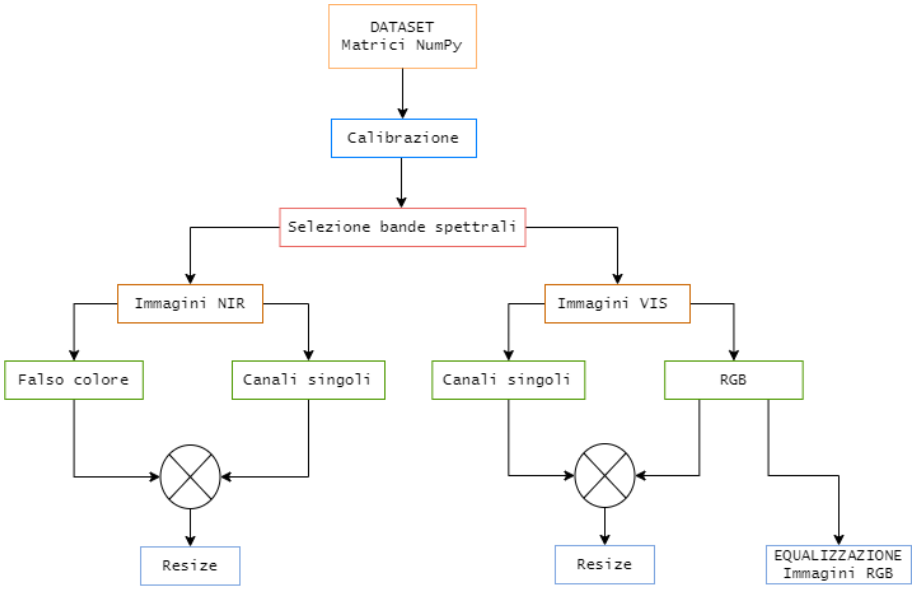


Figura 3.1 – Workflow delle elaborazioni illustrate nel presente capitolo

3.1 Suddivisione delle rilevazioni nei singoli canali

Partendo dal dataset fornito all'inizio del progetto, è necessario leggere il contenuto di ogni file e suddividerlo in canali, cosa che permetterà le successive elaborazioni. La suddivisione in canali si riflette come suddivisione in cartelle.

Dopo aver scelto la cartella contenente tutte le rilevazioni (in formato *.npy*) e su quale delle due tipologie lavorare (**VIS** o **NIR**), lo script *MATLAB* si occupa di creare una cartella denominata **exportVIS** o **exportNIS** (a seconda della scelta).

3.1.1 Calibrazione e normalizzazione

Le rilevazioni necessitano di calibrazioni per eliminare eventuali rumori di fondo e convertire il digital number in riflettanza. I file delle calibrazioni, contenuti nella cartella del *dataset*, sono state ottenute nel seguente modo:

- **Black**: il sensore è stato coperto dal *copri-obiettivo*.
- **White**: è stata utilizzata una tavoletta di calibrazione (Spectralon [33]) con una riflettanza al 40%.

La normalizzazione è stata eseguita dividendo per 255 il valore di riflettanza di ogni pixel, così che fosse espressa in centesimi, con valori compresi tra 0 e 1.

Un'altra elaborazione che è necessario citare riguarda la struttura del *filter mosaic* di cui si è già discusso. I file **`M5.m`** e **`M4.m`** (rispettivamente per **NIR** e **VIS**) contengono i valori che consentono di estrapolare, dai 2048×1088 pixel, solo i pixel corrispondenti ad una determinata banda spettrale.

3.1.2 Lettura del file

I file delle rilevazioni sono salvati nel seguente formato:

$$<MODE^{1}>\_<time\_in\_formato\_linuxepoch^{2}>.npy$$

Da notare l'estensione: i file sono esportazioni del pacchetto **NumPy** [3] di **Python** e vengono forniti già in questo formato. Lo script che si occupa della lettura e del parsing carica su una variabile il path che identifica l'immagine, il quale viene poi passato ad una funzione che si occupa dell'elaborazione. La chiamata alla funzione restituisce una struttura contenente tutti i canali e un falso colore. Riguardo tale struttura, grazie ad un ciclo **for**, viene suddivisa e memorizzata nelle cartelle, dopo opportune rotazioni. Di seguito lo script:

```
for i = 1:length(MOD_dir)
    if MOD_dir(i).isdir == true
        continue;
    end
    path_o = sprintf('%s/%s', rootpath,filename);
    [Im16,Im,I,Iq] = calibImg(path_o,Ib,Iw,M,ws,channel, alpha);
    filename_in = sprintf('%s/%s/%02d/%02d_%s.png',rootpath,export, 0, 0,
    filename);
    Iq_rot270 = rot90(Iq, 3);
    imwrite(Iq_rot270,filename_in);
    Ir = uint8(zeros(size(I)));
    for j = 1:k
        Ir = normalizeImg(I(:,:,j));
        Ir_res = imresize(Ir, alpha, 'bilinear');
        filename_in = sprintf('%s/%s/%02d/%02d_%s.png',rootpath, export, j,j,
    filename);
        Ir_rot270 = rot90(Ir_res, 3);
        imwrite(Ir_rot270,filename_in);
    end
end
```

[1] Per *MODE* si intende **VIS** o **NIR**.

[2] Formato per indicare istanti temporali. Il numero corrisponde ai secondi trascorsi dal 01/01/1970.

[3] Libreria software che offre supporto per poter lavorare a strutture dati quali matrici e vettori [34].

Per l'immagine in falso colore, questa è stata ottenuta passando il parametro `channel` alla funzione `calibImg`:

- **VIS**: è bastato prendere i canali corrispondenti alle lunghezze d'onda dell'RGB **`-> [15,8,1]`** .
- **NIR**: sono state generate tutte le possibili combinazioni ($24 \times 24 \times 24 = 13824$) e tra queste, dopo un'attenta analisi, ne è stata scelta la migliore **`-> [8, 8, 13]`**.

Del parsing se ne occupa la funzione `import_npy`. Questa legge i primi byte di header, per estrarre poi la struttura composta da 2048×1088 valori (che corrisponde alla risoluzione del sensore).

```
function I = import_npy(path)
I = uint16(zeros(1088,2048));
fp = fopen(path,'r');
c = fread(fp, 8);
c = fread(fp, 2);
c = fread(fp, 118);
data = fread(fp,2048*1088*2);
fclose(fp);
for i = 1:(2048*1088)
    v = uint16(data((i-1)*2+1) + data((i-1)*2 + 2) * 256);
    r = floor( (i - 1) / 2048) + 1;
    c = mod(i - 1, 2048) + 1;
    I(r,c) = v;
end
```

Listing 3.1 – Estratto di import_npy.m

Lanciando l'esecuzione degli script sia in modalità **NIR** che **VIS** si ottengono i risultati mostrati di seguito.

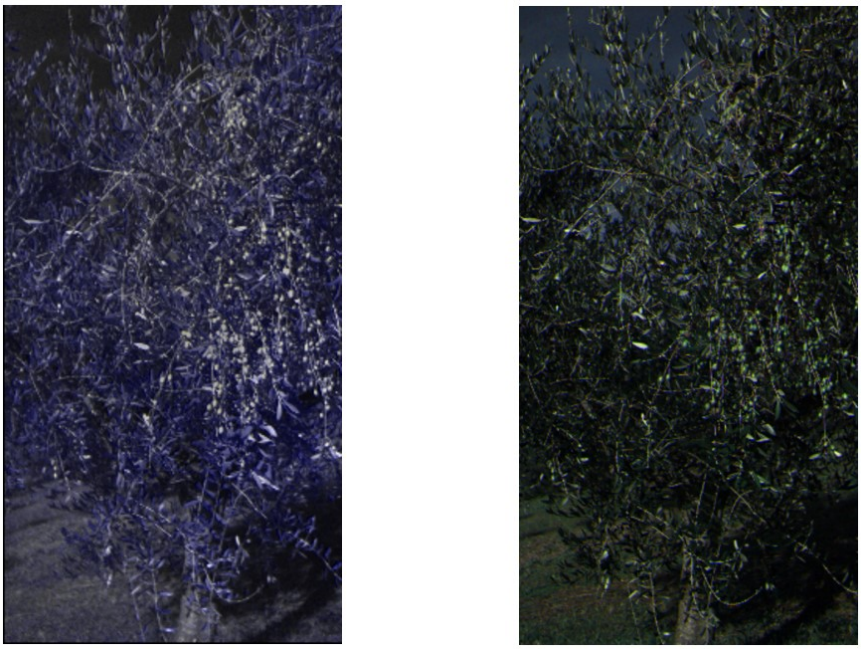


Figura 3.2 – Esempio di elaborazione **NIR** e **VIS** in falso colore.

3.1.3 Struttura delle cartelle

Come detto, ottenute tutte le elaborazioni, si è verificata la necessità di salvarle in cartelle che avessero una struttura chiara. Poiché il numero varia a seconda della modalità scelta, si è optato per un ciclo iterativo che effettuasse un controllo su una variabile appositamente allocata in fase di scelta dell'utente.

```
for r = 0:k
    destDir = sprintf('%s/%s/%02d',rootpath, export, r);
    if not(isfolder(destDir))
       mkdir(destDir);
    end
end
```

Listing 3.2 – Estratto di batchProcessImg.m

Da notare il controllo su `destDir`: se la cartella è già presente sul disco, non viene creata e il ciclo termina senza effettuare operazioni.
Lo stesso "stratagemma" è stato adottato all'occorrenza per la creazioni di altre *directory*.

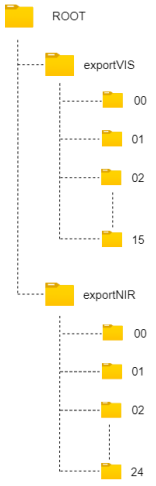


Figura 3.3 – Struttura delle cartelle.

3.1.4 Resize

L'esportazione produce immagini con dimensione 410×218 pixel per le **VIS** e 512×272 per le **NIR**. Un ridimensionamento si è reso necessario considerata la bassa risoluzione.

MATLAB dispone della funzione `imresize` [35] che, prendendo come input un fattore di scala k, produce in output un'immagine con risoluzione $M*k \times N*k$. Nel caso in cui $k > 1$, i pixel mancanti sono ottenuti per interpolazione.

Tra i vari metodi messi a disposizione dalla funzione, è stato scelto quello di *interpolazione bilineare*: il pixel è calcolato come media dei valori di riflettanza assunti dalla sottomatrice 2×2 più vicina [35].

Sono state condotte varie prove al fine di ottenere esportazioni che fossero il miglior compromesso tra dimensione del file e risoluzione. Alla fine, tale valore, si è dimostrato essere $k = 3,5$, portando le immagini **VIS** ad avere una risoluzione finale di 1792×952 pixel e una grandezza di 789kB per ogni canale.

Riguardo le **NIR**, si è preferito che queste avessero la stessa dimensione delle **VIS**, per consentire ai processi che verranno illustrati più avanti di essere il più efficaci possibile. Il fattore di scala scelto è $k = 4,37$.

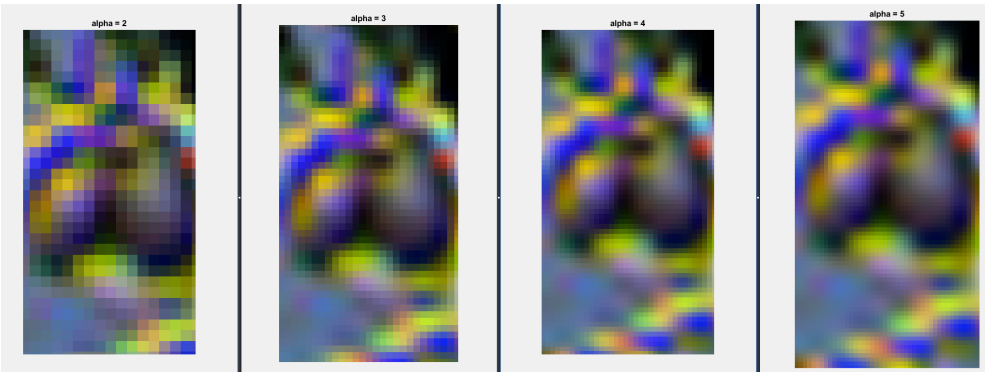


Figura 3.4 – Scelta del fattore di scala

3.2 Equalizzazione istogramma

La precedente elaborazione produce in output immagini VIS che dovranno essere *labellizzate* su **Labelbox**.

Come si nota in Figura 3.2 però, riconoscere le olive risulta complicato. Si è scelto di adottare la tecnica di *equalizzazione dell'istogramma*, che permette di distribuire i valori di riflettanza dei pixel su un intervallo più ampio, così che il risultato mostri un miglior bilanciamento delle tonalità. Lavorando con strutture $M \times N \times 3$, la soluzione adottata consente di suddividere la struttura in 3 e per ognuna effettuare l'equalizzazione.

MATLAB fornisce la funzione `adapthisteq` che, adoperando su immagini **HSV**[4], applica l'algoritmo *CLAHE* che lavora su una piccola porzione di quest'ultima [36].

La seguente sezione di codice mostra la conversione di ogni canale in HSV, la sua equalizzazione e nuovamente la ricomposizione dell'immagine in 3 canali.

```
for i=1:steps
    if VIS_export_dir(i).isdir == true
        continue;
    end
    file = VIS_export_dir(i).name;
    path_compl=strcat(path, file);
    path_in=sprintf('%s/%s',destDir, file);
    RGB = imread(path_compl);
    HSV = rgb2hsv (RGB);
    R = HSV(:,:,1);
    G = HSV(:,:,2);
    B = HSV(:,:,3);
    R = adapthisteq(R);
    G = adapthisteq(G);
    B = adapthisteq(B);
    RGB_ = hsv2rgb (HSV);
    imwrite(RGB_, path_in);
end
```

Listing 3.3 – Estratto di histEq.m

[4]**Hue Saturation Value** è un modello colore rappresentato da 3 variabili: tonalità, saturazione e luminosità.

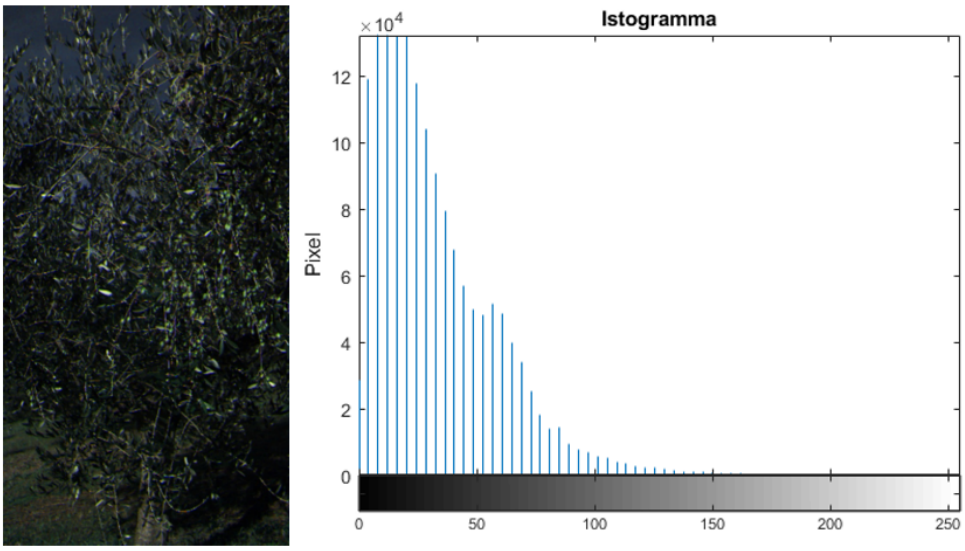


Figura 3.5 – Immagine e istogramma di un singolo canale originali

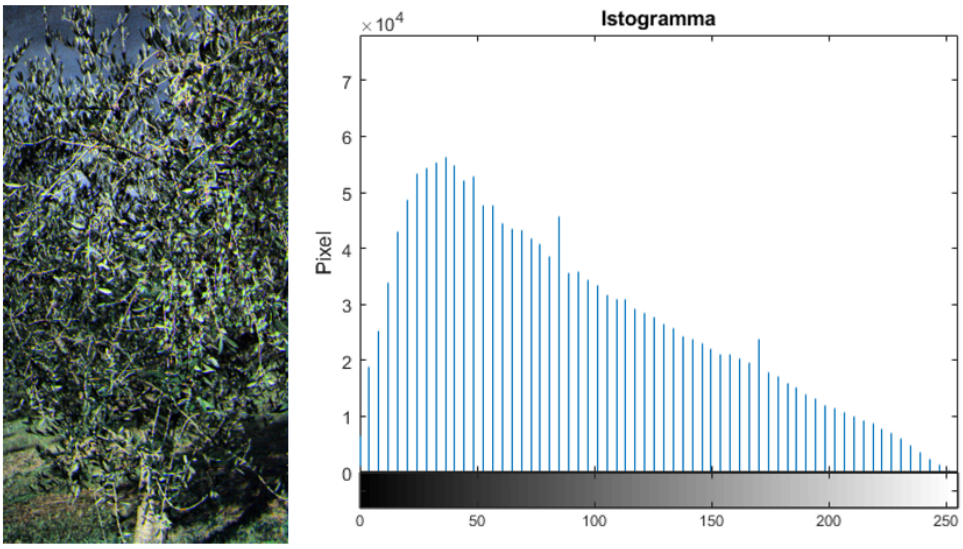


Figura 3.6 – Immagine e istogramma di un singolo canale dopo l'equalizzazione

Le immagini equalizzate sono state salvate nella rispettiva cartella **Histeq** in **00**.

Capitolo 4

Labeling

Per creare il dataset che consentirà i primi passi verso l'addestramento di una rete neurale è stato utilizzato la nota piattaforma di *labeling* **Labelbox**.

Questo capitolo infatti illustrerà i passaggi che hanno consentito di ottenere un'esportazione in formato **JSON** di tutte le *label* e il processo di **parsing**, necessario per ottenere una struttura dati manipolabile.

4.1 Labeling

Le **56 immagini VIS** ottenute al passo precedente, più altre immagini di test ottenute da un altro *dataset* (per un totale di 71) sono state caricate online dopo aver creato un nuovo progetto.

La piattaforma, nella sezione **Settings**, consente di creare ciò che vengono chiamati **Objects**, classi che possono essere attribuite agli oggetti sottoposti a labeling. Considerato lo scopo del progetto si è resa necessaria la creazione della sola classe **Oliva**.

Riguardo la forma degli Objects, il tool consente una discreta possibilità di scelta:

- **Plygon**: poligono formato da un numero di punti scelto dall'utente.
- **Bounding box**: rettangolo.
- **Polyline**: linea continua formata da più segmenti.
- **Point**: punto.
- **Entity**.
- **Segmentation**: forma disegnata a mano libera.

Il **Bounding box**, per quanto sia impreciso per contornare un'oliva rispetto ad una forma disegnata a mano libera, offre un buon compromesso tra tempo e qualità del risultato.

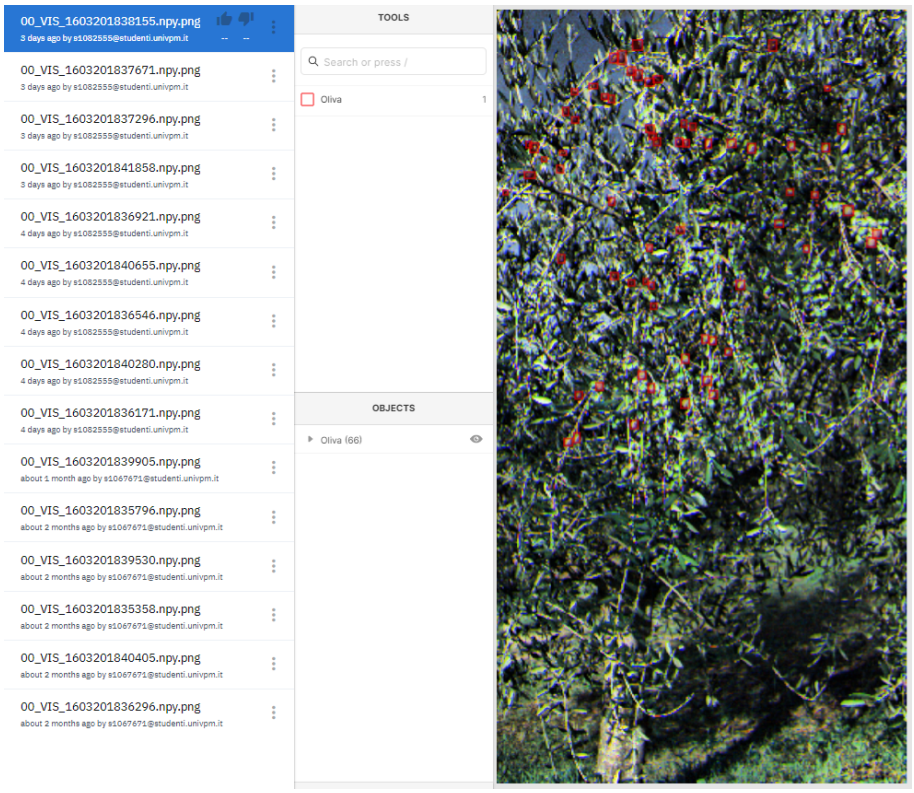


Figura 4.1 – Processo di labelizzazione del dataset su Labelbox

Con un dataset di 71 immagini sono stati ricavati ben **2547 bounding box**. Riguardo i criteri che hanno portato a scegliere un'oliva rispetto ad un'altra, vista la scarsa risoluzione dell'immagine e l'elevata quantità di frutti, sono state considerate solamente quelle che fossero ben separate dallo sfondo. Questo sarà utile in futuro per consentire alla rete neurale di non far confusione e ottenere buoni risultati durante l'addestramento.

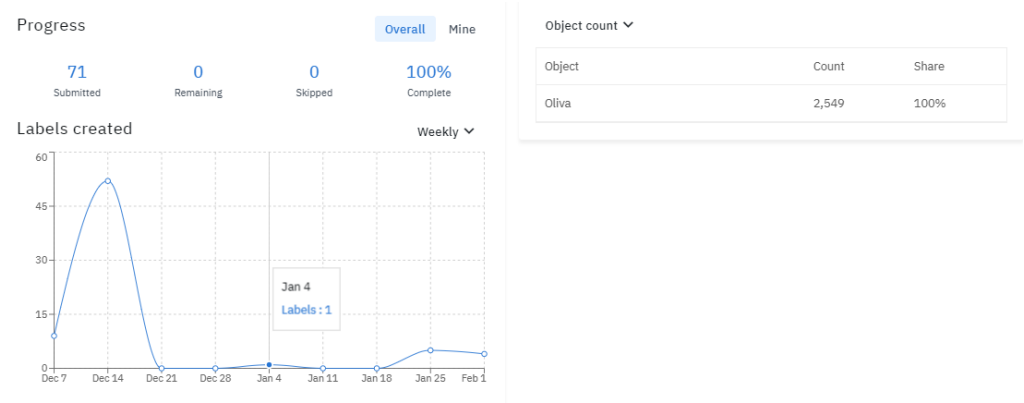


Figura 4.2 – Risultati del labeling

4.2 Esportazione in formato JSON

Labelbox consente all'utente di esportare il suo lavoro in formato *CSV* oppure in formato *JSON*. Nonostante il **CSV** offra alcuni vantaggi in termine di spazio, non è la scelta migliore quando la struttura è piuttosto complessa come in questo caso, dato che utilizza il carattere *comma* per separare i valori. Il **JSON** invece offre la possibilità di rappresentare strutture gerarchicamente complesse e un'elevata flessibilità, mostrando i suoi limiti quando si parla di spazio occupato [37]. Tutto questo ha fatto pendere l'ago della bilancia su quest'ultimo, portando il formato **JSON** ad essere scelto per rappresentare l'esportazione.

Un estratto viene mostrato di seguito:

```
[
  {
    ID: ckina80ce00003a68ixacih9t,
    DataRow ID: ckima35v00ets0rdw3w8ibzwr,
    Labeled Data: ...,
    Label: {
      objects: [
          {
           featureId: ckin9j1u606520y99eoni8zq8,
           schemaId: cki908j7706ko0ybvbp7z3t16,
           title: Oliva,
           value: oliva,
           color: f70005,
           bbox: {
             top: 93,
             left: 292,
             height: 20,
             width: 16
           },
           instanceURI: ...
          }
          {
          },
         ],
        classifications: []
    },
    Created By: ...,
    Project Name: AI4Ripening,
    Created At: 2020-12-13T15:27:35.000Z,
    Updated At: 2020-12-13T15:27:57.000Z,
    Seconds to Label: 1712.541,
    External ID: 00_VIS_1603201836671.npy.png,
    Agreement: -1,
    Benchmark Agreement: -1,
    Benchmark ID: null,
    Dataset Name: VIS_Equalizzato_Resize (3.5),
    Reviews: [],
    View Label: ...
  }
]
```

Come si può notare, ogni immagine differisce da un'altra per un `ID` e che contiene una chiave `Label`.

Ad ogni chiave è associato un oggetto `objects` che contiene, a sua volta, un numero di elementi tanti quanti sono i **bounding box**. Ognuno di essi presenta informazioni quali il nome dell'oggetto (che corrisponde ad *Oliva* per tutti) e le coordinate.

Riguardo le coordinate, queste rappresentano:

- **`top`** e **`left`**: coordinate dell'angolo in alto a sinistra, partendo dal medesimo angolo per l'immagine;
- **`height`**: altezza in pixel del **bounding box**;
- **`width`**: larghezza in pixel del **bounding box**.

4.3 Parsing

Il **parsing** del *JSON* ha permesso di caricare tutta la struttura su *MATLAB*, fondamentale per poter ottenere statistiche sulla riflettanza delle olive.

La funzione che ha consentito il processo è **`jsondecode`** [38]: aprendo il file e ottenendo da esso un `character vector` da passare a `jsondecode` si ottiene un struttura formata da 71 elementi e 16 campi.

```
1 text = fileread(filename);
2 JS = jsondecode(text);
```

Listing 4.1 – Estratto di `bbox.m`

Fields	ID	DataRowID	LabeledData	Label	CreatedBy	ProjectName	CreatedAt	UpdatedAt	SecondsToLabel	ExternalID	Agreement	BenchmarkAgreement	BenchmarkID	DatasetName	Reviews	ViewLabel
1	'ckina80...	'ckima35v00ets...	'https://storage...	1x1 struct	[]	'AI4Ripening'	'2020-12-13...	'2020-12-13T...	1.7125e+03	'00_VIS_1603...	-1	-1	[]	'VIS_Equalizzato_...	[]	'https://edito...
2	'ckinax7f...	'ckima35v20exc...	'https://storage...	1x1 struct	[]	'AI4Ripening'	'2020-12-13...	'2020-12-13T...	967.1860	'00_VIS_1603...	[]	-1	[]	'VIS_Equalizzato_...	[]	'https://edito...
3	'ckincg2...	'ckima35v00eu4...	'https://storage...	1x1 struct	[]	'AI4Ripening'	'2020-12-13...	'2020-12-13T...	1.6246e+03	'00_VIS_1603...	[]	-1	[]	'VIS_Equalizzato_...	[]	'https://edito...
4	'ckindnx...	'ckima35v30exo...	'https://storage...	1x1 struct	[]	'AI4Ripening'	'2020-12-13...	'2020-12-13T...	607.5740	'00_VIS_1603...	[]	-1	[]	'VIS_Equalizzato_...	[]	'https://edito...
5	'ckinellz7...	'ckima35v00eu...	'https://storage...	1x1 struct	[]	'AI4Ripening'	'2020-12-13...	'2020-12-13T...	1.1452e+03	'00_VIS_1603...	[]	-1	[]	'VIS_Equalizzato_...	[]	'https://edito...
6	'ckinf3p...	'ckima35v00eus...	'https://storage...	1x1 struct	[]	'AI4Ripening'	'2020-12-13...	'2020-12-13T...	674.5650	'00_VIS_1603...	-1	-1	[]	'VIS_Equalizzato_...	[]	'https://edito...
7	'ckinfqd...	'ckima35uy0erk...	'https://storage...	1x1 struct	[]	'AI4Ripening'	'2020-12-13...	'2020-12-13T...	646.6040	'00_VIS_1603...	[]	-1	[]	'VIS_Equalizzato_...	[]	'https://edito...
8	'ckingdu...	'ckima35v00ev4...	'https://storage...	1x1 struct	[]	'AI4Ripening'	'2020-12-13...	'2020-12-13T...	744.3300	'00_VIS_1603...	-1	-1	[]	'VIS_Equalizzato_...	[]	'https://edito...
9	'ckingzqt...	'ckima35uz0erw...	'https://storage...	1x1 struct	[]	'AI4Ripening'	'2020-12-13...	'2020-12-13T...	788.1340	'00_VIS_1603...	[]	-1	[]	'VIS_Equalizzato_...	[]	'https://edito...
10	'ckiot5m...	'ckima35v00evg...	'https://storage...	1x1 struct	[]	'AI4Ripening'	'2020-12-14...	'2020-12-14T...	529.7270	'00_VIS_1603...	[]	-1	[]	'VIS_Equalizzato_...	[]	'https://edito...
11	'ckiotait...	'ckima35uz0es8...	'https://storage...	1x1 struct	[]	'AI4Ripening'	'2020-12-14...	'2020-12-14T...	229.0500	'00_VIS_1603...	[]	-1	[]	'VIS_Equalizzato_...	[]	'https://edito...
12	'ckiotszw...	'ckima35v00evs...	'https://storage...	1x1 struct	[]	'AI4Ripening'	'2020-12-14...	'2020-12-14T...	585.8330	'00_VIS_1603...	[]	-1	[]	'VIS_Equalizzato_...	[]	'https://edito...
13	'ckiovc8...	'ckima35uz0esk...	'https://storage...	1x1 struct	[]	'AI4Ripening'	'2020-12-14...	'2020-12-14T...	981.8820	'00_VIS_1603...	[]	-1	[]	'VIS_Equalizzato_...	[]	'https://edito...
14	'ckiovt2n...	'ckima35v00ew...	'https://storage...	1x1 struct	[]	'AI4Ripening'	'2020-12-14...	'2020-12-14T...	667.9070	'00_VIS_1603...	[]	-1	[]	'VIS_Equalizzato_...	[]	'https://edito...
15	'ckiowf7...	'ckima35uz0es...	'https://storage...	1x1 struct	[]	'AI4Ripening'	'2020-12-14...	'2020-12-14T...	720.3510	'00_VIS_1603...	-1	-1	[]	'VIS_Equalizzato_...	[]	'https://edito...
16	'ckiox05...	'ckima35v10ew...	'https://storage...	1x1 struct	[]	'AI4Ripening'	'2020-12-14...	'2020-12-14T...	804.6090	'00_VIS_1603...	[]	-1	[]	'VIS_Equalizzato_...	[]	'https://edito...
17	'ckioxk1...	'ckima35v00et8...	'https://storage...	1x1 struct	[]	'AI4Ripening'	'2020-12-14...	'2020-12-14T...	624.8100	'00_VIS_1603...	[]	-1	[]	'VIS_Equalizzato_...	[]	'https://edito...
18	'ckioxw2...	'ckima35v10ew...	'https://storage...	1x1 struct	[]	'AI4Ripening'	'2020-12-14...	'2020-12-14T...	463.2540	'00_VIS_1603...	-1	-1	[]	'VIS_Equalizzato_...	[]	'https://edito...
19	'ckip07a...	'ckima35v00etk...	'https://storage...	1x1 struct	[]	'AI4Ripening'	'2020-12-14...	'2020-12-14T...	658.1240	'00_VIS_1603...	-1	-1	[]	'VIS_Equalizzato_...	[]	'https://edito...
20	'ckip0vu...	'ckima35v10ex4...	'https://storage...	1x1 struct	[]	'AI4Ripening'	'2020-12-14...	'2020-12-14T...	945.4420	'00_VIS_1603...	[]	-1	[]	'VIS_Equalizzato_...	[]	'https://edito...
21	'ckiqi0dn...	'ckima35v00etw...	'https://storage...	1x1 struct	[]	'AI4Ripening'	'2020-12-15...	'2020-12-15T...	515.6290	'00_VIS_1603...	[]	-1	[]	'VIS_Equalizzato_...	[]	'https://edito...
22	'ckiqi9hs...	'ckima35v20exg...	'https://storage...	1x1 struct	[]	'AI4Ripening'	'2020-12-15...	'2020-12-15T...	401.2480	'00_VIS_1603...	[]	-1	[]	'VIS_Equalizzato_...	[]	'https://edito...
23	'ckiqipz0...	'ckima35v00eu8...	'https://storage...	1x1 struct	[]	'AI4Ripening'	'2020-12-15...	'2020-12-15T...	625.7100	'00_VIS_1603...	-1	-1	[]	'VIS_Equalizzato_...	[]	'https://edito...
24	'ckiqjcr8...	'ckima35v00euk...	'https://storage...	1x1 struct	[]	'AI4Ripening'	'2020-12-15...	'2020-12-15T...	416.3110	'00_VIS_1603...	[]	-1	[]	'VIS_Equalizzato_...	[]	'https://edito...
25	'ckiqjztj...	'ckima35v00eu...	'https://storage...	1x1 struct	[]	'AI4Ripening'	'2020-12-15...	'2020-12-15T...	751.5450	'00_VIS_1603...	-1	-1	[]	'VIS_Equalizzato_...	[]	'https://edito...

Figura 4.3 – Estratto di `JP`

Come detto in precedenza, ogni `Label` possiede un oggetto `objects`.

Fields	featureId	schemaId	title	value	color	bbox	instanceURI
1	'ckin9j1u606520y99eoni8zq8'	'cki908j7706ko0ybvbp7z3t16'	'Oliva'	'oliva'	'#f70005'	*1x1 struct*	'https://api.lab...
2	'ckin9jmsb057n0y7e099ifsjr'	'cki908j7706ko0ybvbp7z3t16'	'Oliva'	'oliva'	'#f70005'	*1x1 struct*	'https://api.lab...
3	'ckin9kimj067r0y99chash4pl'	'cki908j7706ko0ybvbp7z3t16'	'Oliva'	'oliva'	'#f70005'	*1x1 struct*	'https://api.lab...
4	'ckin9knte059u0y7e97ssdssf'	'cki908j7706ko0ybvbp7z3t16'	'Oliva'	'oliva'	'#f70005'	*1x1 struct*	'https://api.lab...
5	'ckin9kx8s068w0y99eeoc9hq0'	'cki908j7706ko0ybvbp7z3t16'	'Oliva'	'oliva'	'#f70005'	*1x1 struct*	'https://api.lab...
6	'ckin9ohiv06g40y99h1d49ujf'	'cki908j7706ko0ybvbp7z3t16'	'Oliva'	'oliva'	'#f70005'	*1x1 struct*	'https://api.lab...

Figura 4.4 – Estratto di `Label.objects` dell'immagine "00_VIS_1603201836671.npy.png"

Ogni `objects` possiede un numero di elementi pari al numero di *bounding boxes* presenti nell'immagine. Tra i campi quello più interessante è `bbox` che contiene le coordinate del *bounding box*.

Field	Value
top	93
left	292
height	20
width	16

Figura 4.5 – Struttura di `Label.objects.bbox(1)`

Per comprendere meglio la struttura di `JP`, di seguito viene proposto uno schema in cui si sono tenuti in considerazione solo i campi più rilevanti.

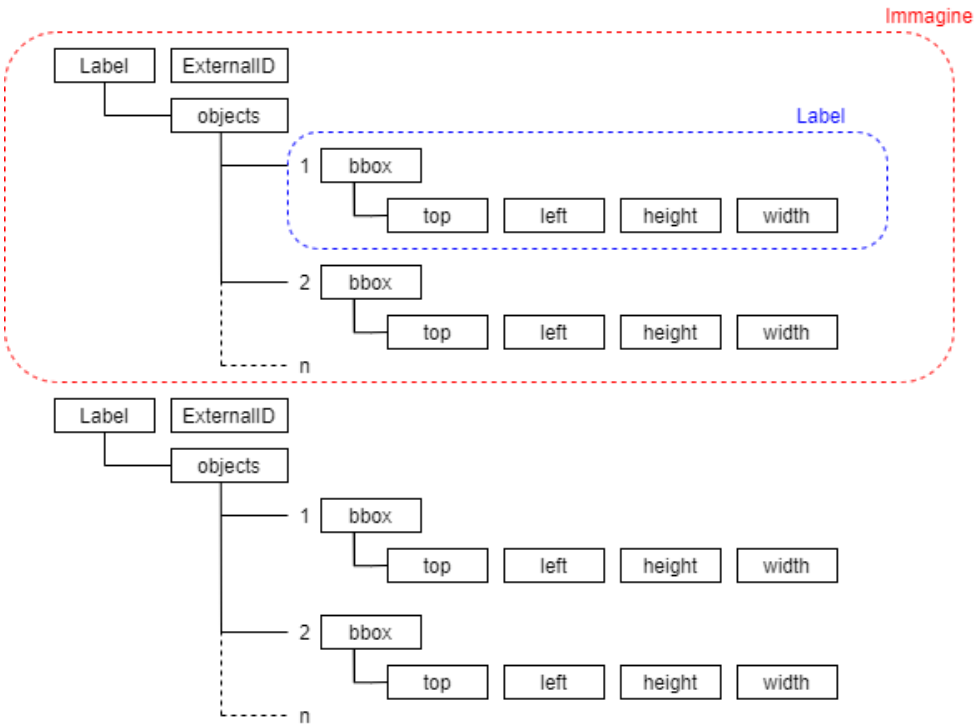


Figura 4.6 – Struttura di `JP`

Capitolo 5

Stima del grado di maturazione

Il presente capitolo ha lo scopo di illustrare i primi passi verso la stima del grado di maturazione, partendo dalla **registrazione** delle immagini, passando per il **calcolo delle statistiche di riflettanza** dei *bounding box*, concludendo con il **clustering** delle statistiche ottenute al passo precedente.

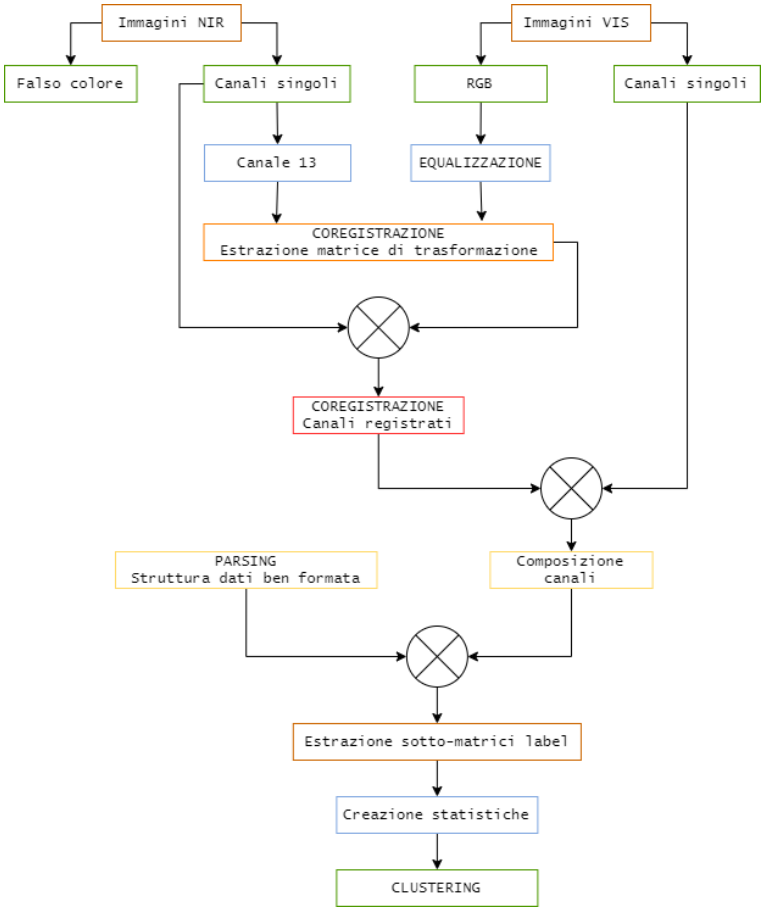


Figura 5.1 – Workflow delle elaborazioni illustrate nel presente capitolo.

5.1 Registrazione immagini

Le immagini **NIR** e **VIS**, al momento della rilevazione, sono state ottenute da camere differenti. Questo ha causato un disallineamento tra ogni coppia, dovuto alle diverse angolazioni dei sensori multispettrali. Le operazioni successive richiedono un riallineamento, così che ogni oliva di ogni canale abbia le stesse coordinate in pixel.

La scelta della tecnica da adottare è ricaduta sulla **registrazione di immagini**.

5.1.1 Primi approcci

L'idea di base è partire da un'immagine **NIR** e una **VIS** che siano, in termini temporali, il meno distanti possibile. Ottenuto il miglior risultato, il processo elaborativo è esteso all'intero dataset.

MATLAB dispone di un utile tool, il **Registration Estimator** [39], che può essere aperto o tramite `command window`[1] o attraverso il tab degli strumenti, nella sezione `APPS`.
Caricate le immagini, facendo attenzione alla distinzione di *moving* e *fixed*, il tool effettua una conversione in scala di grigi. Il **Registration Estimator** propone molti approcci.

- **Feature based: SURF**
  Per il primo tentativo è stata scelta la tecnica *SURF*, con immagini selezionate dalla cartella 00 (in falso colore). Dopo alcune prove, cambiando in modo opportuno i parametri *number of detected features* e *quality of matched features*, applicando una trasformazione *rigida* e *affine* si è giunti ad un risultato abbastanza soddisfacente.

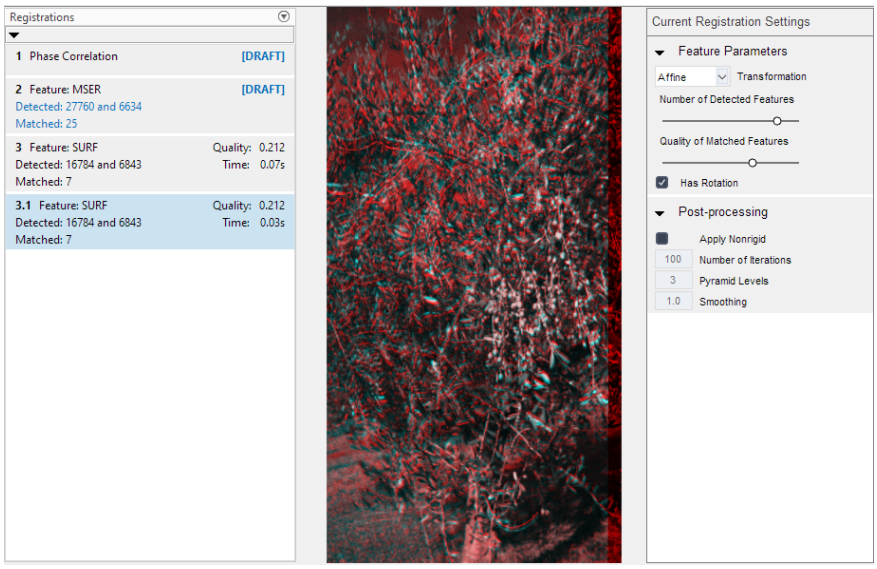


Figura 5.2 – Risultato approccio SURF.

[1] `registrationEstimator(J,I)` con J *moving* e I *fixed* [39].

Esportando la funzione e la relativa matrice di trasformazione per automatizzare il processo anche per le altre immagini (passo che verrà illustrato più avanti), il metodo dimostra le sue debolezze. Per alcune di esse risulta impossibile trovare almeno due *feature*, per altre invece, anche se il numero risulta sufficiente per applicare la trasformazione, il risultato è da scartare in quanto inutilizzabile. Infatti, delle **56 immagini**, 54 sono state coregistrate, tra cui solamente **46** sono corrette

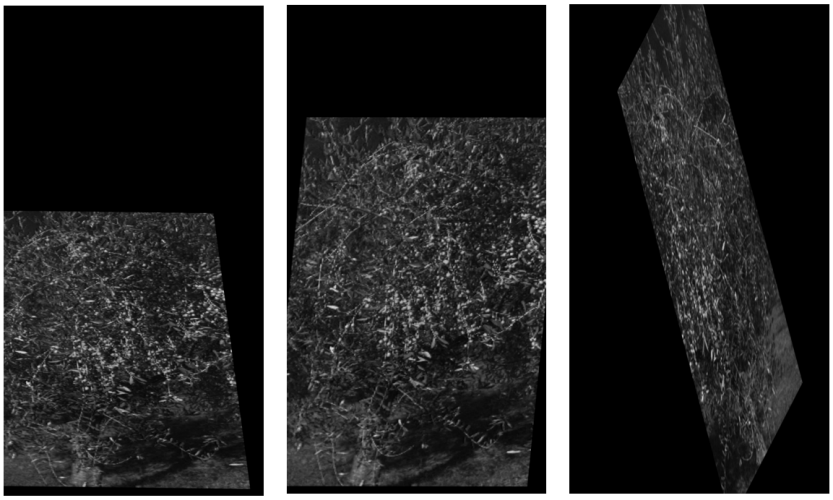


Figura 5.3 – Esempio di risultati inutilizzabili

- **Point mapping**
  La **Point mapping** [40] consente individuare a "mano" le *feature* utili ad estrarre la matrice di trasformazione. Dalla `command window`, passando alla funzione `cpselect` la coppia di immagini, *MATLAB* lancia il tool **Control Point Selection**.
  Terminata la selezione dei punti, chiudendo la finestra, vengono caricate due variabili nel `workspace`, che serviranno per applicare la trasformazione.
  La funzione `fitgeotrans` [41], specificando come input le coordinate dei `moving points`, dei `fixed points` e la tipologia di trasformazione, restituisce in output un `transformation object` che consente di registrare le immagini.

```
J = imread (<path_immagine_NIR>);
I = imread (<path_immagine_VIS>);
[mp, fp] = cpselect(J, I, 'Wait', true);
tform = fitgeotrans(mp,fp,'projective');
Rfixed = imref2d(size(I));
registered = imwarp(J,tform,'OutputView',Rfixed);
imshowpair(I, registered, 'blend');
```

Listing 5.1 – Point mapping

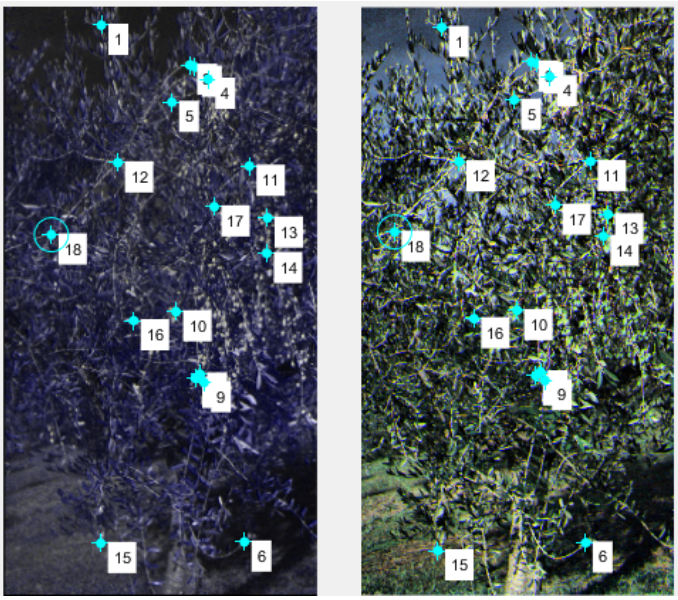


Figura 5.4 – Esempio del tool **Control Point Selection.**

I risultati ottenuti con questo metodo sono **eccellenti**, ma la matrice di trasformazione è unica per ogni coppia di immagini e automatizzare il processo significherebbe applicare ad ognuna di queste lo stesso `transformation object`, portando a risultati inconcludenti su tutto il dataset, meno che per una sola coppia.

Figura 5.5 – Risultato ottimale della funzione `imshowpair()`

- **Intensity-based: Multimodal intensity**
  L'ultimo test è stato eseguito con una tecnica che si è dimostrata essere molto dispendiosa dal punto di vista computazionale, ma molto efficace nel risultato. Eseguendo sempre il tool **Registration Estimator** e sfruttando questa volta la `multimodal intensity` con un $initialradius = 0.00125$ e una trasformazione *rigid*, si ottengono i seguenti risultati:

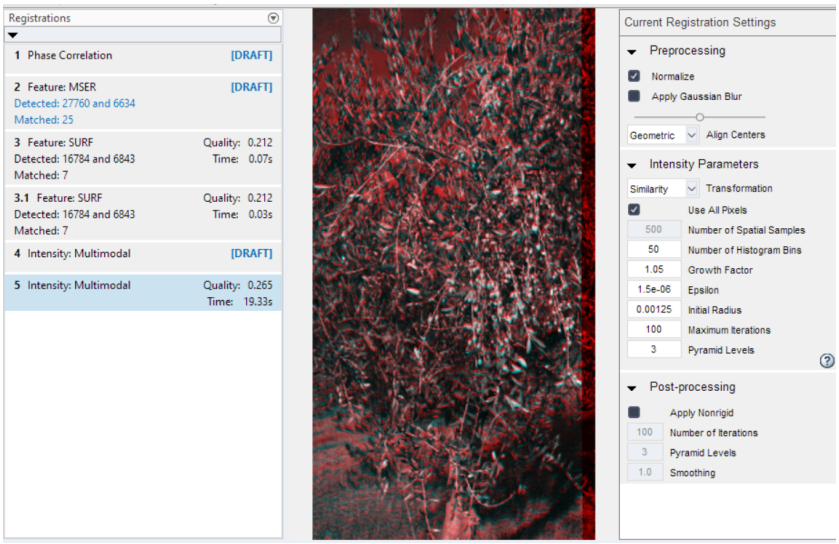


Figura 5.6 – Registration estimator app con approccio Multimodal intensity

Come si nota dalla Figura 5.6, il tempo si esecuzione è stato di circa 19 secondi.
Soddisfatti dal risultato, si è proceduto all'estrazione della funzione elaborativa `registerImages` che potesse automatizzare l'intero processo.

5.1.2 Registrazione dell'intero dataset

Trovata la migliore trasformazione, il passo successivo consiste nel **registrare tutte le immagini** che compongono il dataset. La sfida iniziale è stata determinare la coppia di immagini **NIR** e **VIS** che fossero state scattate quasi nello stesso istante, per essere sicuri che la scena rappresentata fosse la stessa.
La creazione di un algoritmo di *scanning* che, considerato il nome dell'immagine **VIS**, estraendo il *linux_epoch* e confrontandolo con quello della **NIR**, calcolandone lo scarto, ha portato ad ottimi risultati.

```
for i = 1:steps
    if VIS_dir(i).isdir == true
        continue;
    end
    VIS_tm = str2double(VIS_dir(i).name(8:end-8));
    Scrt = VIS_tm;
    for j = 1:length(NIR_dir)
        if NIR_dir(j).isdir == true
            continue;
        end
        NIR_tm = str2double(NIR_dir(j).name(8:end-8));
        scrt = abs(VIS_tm - NIR_tm);
        if scrt < Scrt
            temp = NIR_tm;
            Scrt = scrt;
        else
            continue;
        end
    end
```

Listing 5.2 – Estratto di regIm.m

Per ogni coppia, la registrazione è stata eseguita passando le immagini alla funzione `registerImages`. Ciò che restituisce la funzione è l'immagine registrata e una struttura contenente la matrice di trasformazione da applicare agli altri canali.
Tra le **NIR**, si è scelto di prendere il canale 13 poiché presentava un contrasto migliore rispetto agli altri. In modo iterativo sono state calcolate le registrazioni per tutti i restanti canali e salvate nelle rispettive cartelle.

```
VIS = sprintf('%s/00_VIS_%d.npy.png',pathVIS, VIS_tm);
NIR = sprintf('%s/%02d_NIR_%d.npy.png',pathNIR, 13, temp);
imgVIS = imread(VIS);
imgNIR = imread(NIR);
[movreg, p] = registerImages(imgNIR, imgVIS);
for k = 0 : 24
    NIR = sprintf('%s/exportNIR/%02d/%02d_NIR_%d.npy.png',rootpath, k, k, temp
    );
    imgNIR = imread(NIR);
    filename = sprintf('%s/exportNIR/%02d/Reg/%02d_NIR_%d.npy.png',rootpath, k
    , k, VIS_tm);
    movreg = imwarp (imgNIR, p.movRefObj, p.tform, 'OutputView', p.fixRefObj,
    'SmoothEdges', p.val);
    imwrite(movreg, filename);
end
```

Listing 5.3 – Estratto di regIm.m

Questo passaggio ha permesso di evidenziare i punti deboli del primo approccio, portando a provare altri metodi, fino a trovare nel **multimodal intensity** la soluzione migliore.
Confrontando le immagini registrate con le *fixed* infatti, si nota quanto queste siano ben allineate.

5.2 Statistiche riflettanza

Estrapolare per ogni oliva del dataset, valori sulla riflettanza come **minimo**, **massimo**, **media**, **moda** e **mediana** offre strumenti potenti per poter creare alcune statistiche, così da cercare nei vari canali comportamenti analoghi. Di fatto, questo è il primo passo verso la stima del grado di maturazione.

5.2.1 Composizione dei canali

L'idea di base è quella di costruire una struttura tridimensionale, formata da $1792 \times 952 \times 39$ valori. Le prime due grandezze rappresentano il numero di pixel di ogni canale, l'ultimo invece è la somma dei 15 canali **VIS** e dei 24 canali **NIR**.
L'importanza di registrare le immagini ora appare evidente: la struttura sarà costituita da immagini ben allineate. Il seguente codice mostra come è stata effettuata l'operazione, con alcuni controlli che permettono di avvisare l'utente nel caso in cui l'immagine non esista.

```
for i = 1:length(VIS_dir)
    dir_nameVIS = str2double(VIS_dir(i).name);
    if isnan(dir_nameVIS) || dir_nameVIS == 0
        continue;
    end
    VIS = sprintf('%s/%02d/%02d_VIS_%d.npy.png', pathVIS, dir_nameVIS,
    dir_nameVIS, name);
    imgVIS = imread(VIS);
    allch(:,:, dir_nameVIS) = imgVIS;
end

for i = 1:length(NIR_dir)
    dir_nameNIR = str2double(NIR_dir(i).name);
    if isnan(dir_nameNIR) || dir_nameNIR == 0
        continue;
    end
    NIR = sprintf('%s/%02d/Reg/%02d_NIR_%d.npy.png', pathNIR, dir_nameNIR,
    dir_nameNIR, name);
    try
        imgNIR = imread(NIR);
        allch(:,:,dir_nameNIR+15) = imgNIR;
    catch
        err = sprintf('Immagine NIR_%d non trovata', name);
        errordlg(err, 'ERRORE');
        break;
    end
end
```

Listing 5.4 – Estratto di `compositeChannel.m`

5.2.2 Estrazione delle sotto-matrici

Ottenute le precedenti strutture (una per ogni immagine), il passo successivo è stato "bucarle" per ricavare delle sotto-matrici $M \times N \times 39$ sulle quali calcolare le statistiche, dove $M \times N$ indica la grandezza del **bounding box**
Partendo dalla `struct` ottenuta dal **parsing** del dataset di **Labelbox**, un confronto del campo `externalID` con il nome dell'immagine ha permesso di selezionare le coordinate dei soli **bounding box** per quell'immagine, che sono poi servite ad estrarre dalla struttura tridimensionale le sotto-matrici.

```
for i = 1:length(JS)
   externalID = str2double(JS(i).ExternalID(8:end-8));
    if externalID ~= img_name
        continue;
    else
        bboxes = JS(i).Label.objects;
    end
end
```

Listing 5.5 – Estratto di bbox.m per il confronto tra il nome e l'ID

```
for i = 1:length (bboxes)
    coord_bbox = bboxes(i).bbox;
    x_start = coord_bbox.left;
    x_stop = x_start + coord_bbox.width;
    y_start = coord_bbox.top;
    y_stop = y_start + coord_bbox.height;
    bbox = allch(y_start : y_stop, x_start : x_stop, :);
    cell_bbox {i,1} = bbox;
end
```

Listing 5.6 – Estratto di bbox.m per l'estrazione delle sotto-matrici

Per essere sicuri che quanto fatto fosse corretto, i **bounding box** sono stati disegnati su un'immagine test in *MATLAB*, sfruttando `rectangle` [42], e confrontati con quanto fatto su **Labelbox**.

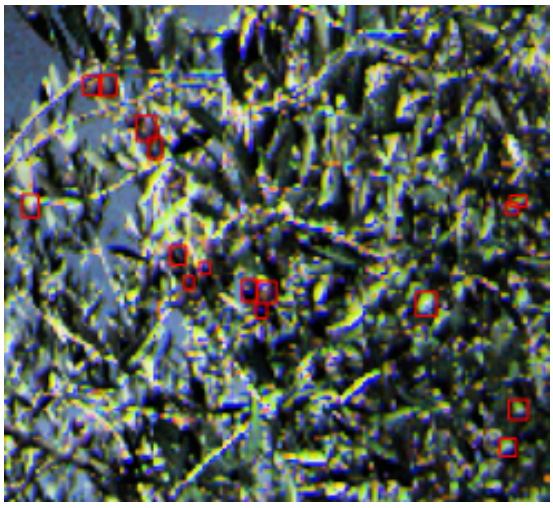


Figura 5.7 – Verifica allineamento **bounding box** con `rectangle()`

5.2.3 Calcolo statistiche

Per il calcolo, si è tenuto conto che i valori di riflettanza fossero compresi tra 0 e 255, perciò una normalizzazione ha portato tali valori ad essere compresi tra 0 e 1.
Altro aspetto da considerare era la dimensione di ogni sotto-matrice che rappresentava il bounding box: linearizzare, compiendo una trasposizione delle righe, ha fatto sì che si passasse da una struttura tridimensionale ad una bidimensionale, di grandezza $(M * N) \times 39$.
Per comodità, tutte le statistiche di ogni **bounding box** sono state inserite in una `struct`, la quale a sua volta è contenuta in una struttura in cui sono racchiuse le statistiche di ogni *bb* di ogni immagine.

```
function struct_stats = stats_bbox (cell_bbox)
    for i = 1:length(cell_bbox)
        bb = double(cell_bbox{i,1});
        bb_norm = bb./255;
        s = size(bb_norm);
        bb_r = reshape(bb_norm, s(1)*s(2), 39);
        min_bb = min(bb_r);
        max_bb = max(bb_r);
        mean_bb = mean (bb_r);
        mode_bb = mode (bb_r);
        median_bb = median (bb_r);
        struct_stats(i).min = min_bb;
        struct_stats(i).max = max_bb;
        struct_stats(i).mean = mean_bb;
        struct_stats(i).mode = mode_bb;
        struct_stats(i).median = median_bb;
    end
```

Listing 5.7 – Estratto di `stats_bbox.m` per il calcolo delle statistiche

5.3 Clustering

Per stabilire il grado di maturazione delle olive, prima di addestrare una rete neurale a lavorare con delle immagini multispettrali, è interessante capire quali risultati si riuscirebbero ad ottenere sfruttando un approccio **unsupervised**[2] rispetto ad uno **supervised**[3]. Grazie a degli algoritmi di **clustering**, è stato possibile analizzare le statistiche effettuate sui **bounding box**, il ché ha evidenziato "comportamenti" ben differenziabili delle olive.

5.3.1 Eliminazione bande rumorose

Partendo dai risultati ottenuti con l'estrazione delle statistiche, il primo passo è stato valutare l'andamento dei valori in un grafico. Questo dovrebbe consentire di far notare le prime differenze tra i canali con lunghezze d'onda non percepibili e quelli VIS.

[2]Tecnica di apprendimento automatico che consiste nel fornire al sistema input non classificati.

[3]Tecnica di apprendimento automatico in cui vengono fornite le classi degli oggetti utilizzati come input.

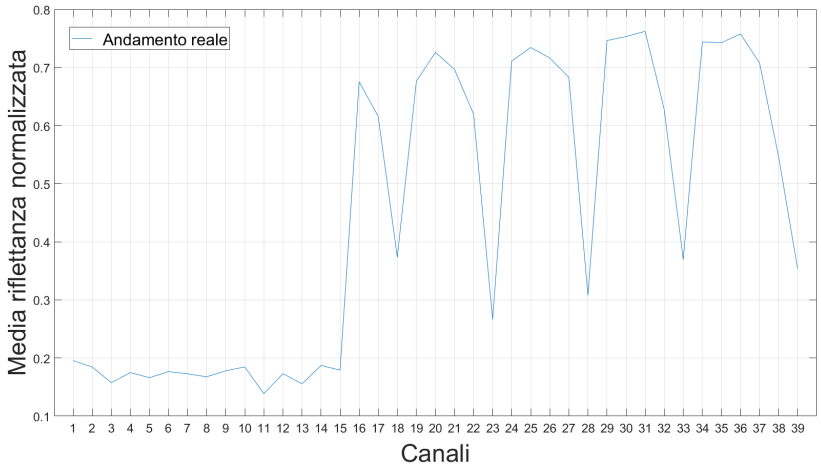


Figura 5.8 – Plot dei valori medi di riflettanza per il primo **bounding box** dell'immagine `'00_VIS_1603201834108.npy.png`.

Ricordando che i canali da 0 a 15 riguardano lo spettro del visibile e da 16 a 39 il range infrarosso, si nota come alcuni di questi risultino assimilabili a del "rumore".
Provando con varie tecniche di interpolazione per ottenere un andamento più lineare della funzione, è chiaro che tali valori ne disturbino l'andamento. Per la tecnica di **interpolazione con spline**[4], questa risente **fortemente** del disturbo, mentre l'**interpolante polinomiale di decimo grado**[5], dimostra come l'effetto sia minore, ma comunque presente.
Il comportamento è analogo per gli altri **bounding box**, anche di altre immagini.

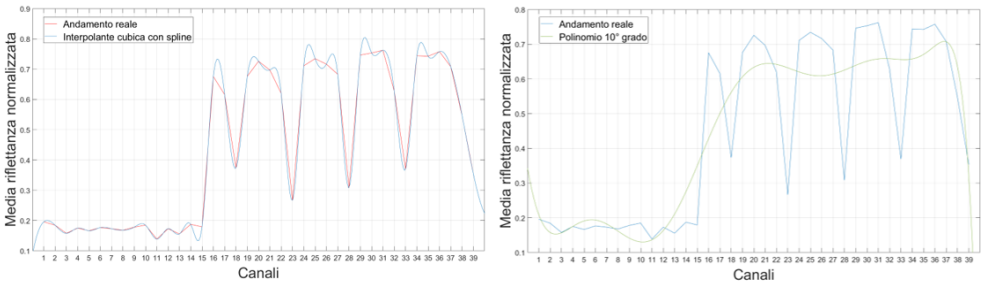


Figura 5.9 – Test con funzioni interpolanti.

Si è scelto dunque di non considerare l'insieme dei canali **18, 23, 28, 33, 39**, che nel **NIR** corrispondono a **3, 8, 13, 18, 24**.

[4]Tecnica che consente di approssimare una funzione ad un insieme di polinomi, così che il valore calcolato per due polinomi vicini in un punto scelto come nodo, sia uguale.

[5]L'interpolazione polinomiale è una tecnica che consente di approssimare una funzione, passante per dei punti scelti, ad un polinomio di grado n.

5.3.2 Ricostruzione bande e previsione nuovi valori

Eliminati tali canali, per ricostruire i valori mancanti è stato fruttato il **package forecast** di **R**.
Per caricare i file delle statistiche sul laboratorio messo a disposizione dall'*UNIVPM*, queste sono state prelevate da *MATLAB* e salvate in formato `.txt`, un file per ogni immagine, per un totale di **54**.

```
for j = 1:length(all_stats)
    for i = 1:length(all_stats(j).stats)
        X(:,i) = all_stats(j).stats(i).mode;
    end
    filein = sprintf('x%d_mode.txt', j);
    dlmwrite(filein, X);
    clear X;
end
```

Listing 5.8 – Estratto di `totxt.m`

Forecast consente di analizzare e modellare serie temporali, in particolare, la funzione che ha permesso di eseguire la previsione è stata `tsclean`, che calcola i valori mancanti effettuando un'interpolazione lineare.
Per i valori già esistenti, è stata creata invece una previsione tramite `predict()`, sostituendoli a quelli precedenti. I nuovi dati sono stati poi salvati in file con estensione `.csv`.

```
for(i in 1:54) {
    file <- paste('./olives/Mean/x', i,'_mean.txt', sep = '')
    getwd()
    v <- read.csv(file, header = FALSE)
    length(colnames(v))
    tempi <- c(1:17,19:22,24:27,29:32,34:37)
    r <- v[tempi,1:length(colnames(v))]

    clean2 <- function(x){ tsclean(x, replace.missing = FALSE, lambda = NULL)
    }

    dati <- sapply(r, FUN = clean2)
    d <- data.frame(dati)

    f1 <- function(x) { return(predict(gam(as.vector(x) ~ s(tempi, bs = c(cs),
     k = -1),na.action = na.exclude),data.frame(tempi))) }

    dati.tempi.fit <- apply( d, MARGIN = 2, f1)

    filew <- paste('./olives/Mean/x', i, '_mean_filtered.csv', sep = '')
    write.csv(data.frame(tempi,dati.tempi.fit),filew, row.names = FALSE)
}
```

Listing 5.9 – Estratto dell'elaborazione in R sulla media

"Plottando" i risultati ottenuti con quelli reali si può notare come l'andamento della funzione sia più conforme ai dati iniziali, approssimando con un grado di accuratezza più alto rispetto i test precedenti.

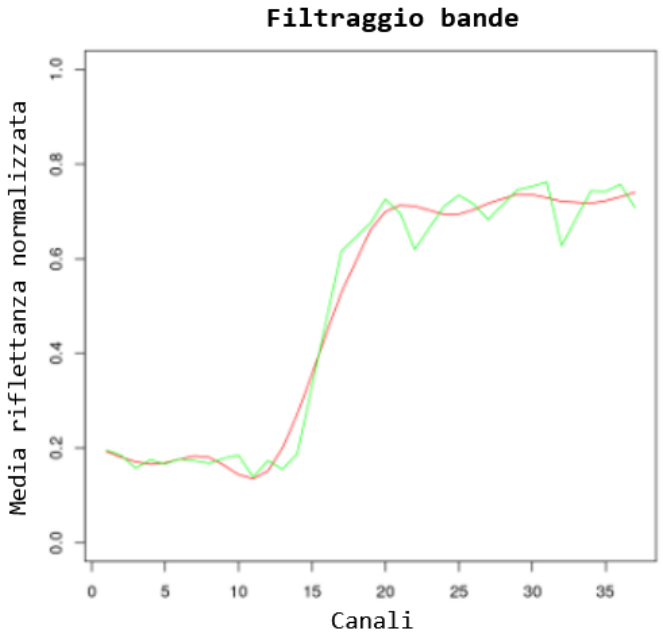


Figura 5.10 – Risultato dell'elaborazione in R sulla media.

5.3.3 Clustering

I risultati ottenuti possono essere importati su *MATLAB* sotto forma matriciale con un semplice script, che consente inoltre di creare un *dataset* da passare alle funzioni di `clustering`.
Il dataset è di dimensioni 2026×33, dove le righe rappresentano i **bounding box** e le colonne i **33 canali** *"filtrati"*.

```
All_bb = [];
for i = 1:54
    filein = sprintf('./Analysis/x%d_filtered.csv', i);
    T = csvread(filein, 1, 1);
    All_bb = [All_bb, T];
end
All_bb_tr = transpose(All_bb);
```

Listing 5.10 – Estratto di k_means.m e dbscan.m sulla media

Si tenga conto che per ottenere il dataset per la moda e mediana è necessario modificare la variabile `filein`, aggiungendo rispettivamente `_mode` e `_median` prima di `_filetered.csv`.
Gli algoritmi di clustering adottati sono principalmente due:

- ***k-means:***
  MATLAB possiede la funzione `k-means` [43] che di default applica l'algoritmo ***k-means++*** e sfrutta la **distanza euclidea**. Prendendo come input il numero di cluster $\boldsymbol{k}$ in cui si vogliono classificare le osservazioni e un dataset costituito da $M \times N$ elementi, restituisce una matrice $M \times 1$, dove l'unica colonna rappresenta a quale *cluster* è stato assegnata l'osservazione. L'output risulta fondamentale per *plottare* i risultati e verificare come i dati sono stati suddivisi nei vari **clusters**.

Tenuto conto del lavoro svolto dal collega *Thaliath Amal Benson*, si è scelto di testare un numero variabile di cluster, dal **2** al **5**. La variabilità è dovuta al fatto che per alcuni gradi di maturazione é difficile distinguerne uno dall'altro [44].

```
k = %numero di cluster;
cidxk = kmeans(All_bb_tr,k,'dist','sqeuclidean');

bboxsymb = {'r','g','c','b','y'};
for i = 1:k
    clust = find(cidxk==i);
    a = bboxsymb{i};
    plot([1:17,19:22,24:27,29:32,34:37], All_bb_tr(clust, :), a);
    hold on;
end
```

Listing 5.11 – Estratto di k_means.m

L'algoritmo è stato testato con tutte le statistiche fin'ora calcolate e di seguito se ne mostrano i risultati.

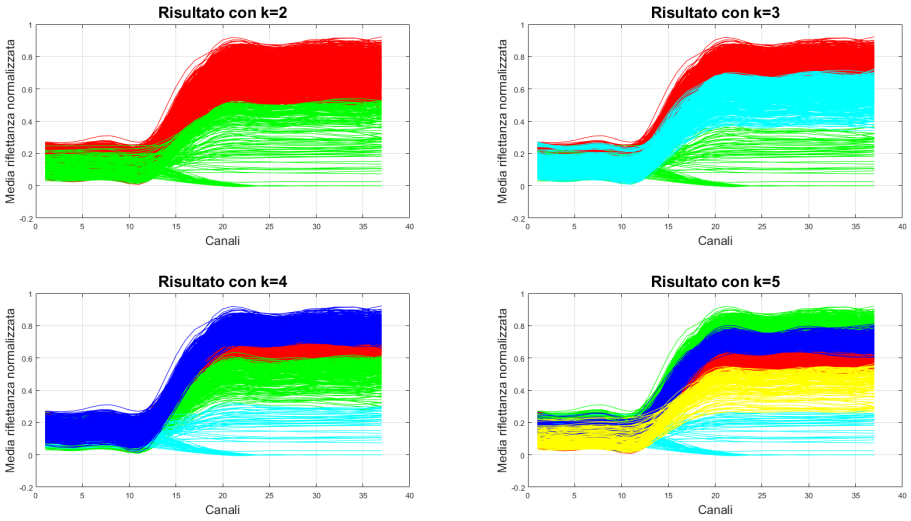


Figura 5.11 – Risultati del k-means ottenuti sfruttando la media dei valori della riflettanza

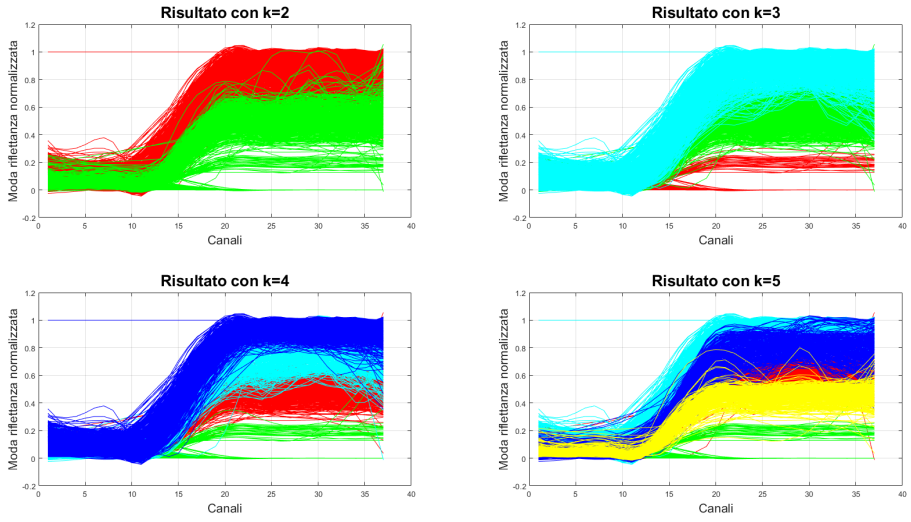


Figura 5.12 – Risultati del k-means ottenuti sfruttando la moda dei valori della riflettanza

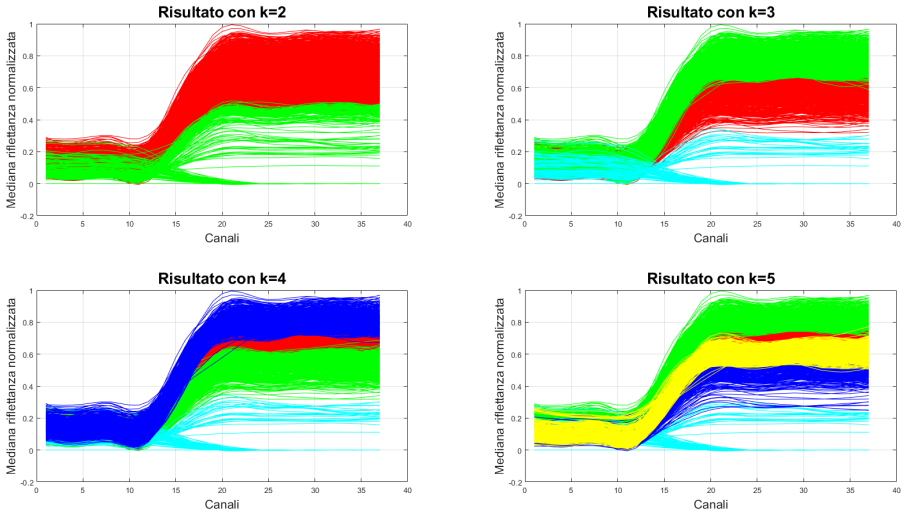


Figura 5.13 – Risultati del k-means ottenuti sfruttando la mediana dei valori della riflettanza

Analizzando quanto ottenuto è possibile fare alcune considerazioni:

- i colori utilizzati cambiano ogni volta che viene eseguito l'algoritmo. Questo aspetto è dovuto alla scelta casuale del centroide iniziale del *k-means++*.
- ovunque si riscontra la presenza di alcune *"curve piatte"* dovute al fallimento dell'algoritmo di pulizia delle bande più che ad una scarsa performance del *k-means*.
- i risultati ottenuti per **media** e **mediana** posso essere considerati **ottimi**, per la moda invece si nota del *"rumore"* sull'andamento di alcune curve, dovuto, come nel caso precedente, al fallimento del processamento in R.

L'aspetto più importante di cui tener conto è senz'altro il comportamento delle olive in immagini **VIS** rispetto ad immagini **NIR**: il grafico mostra un andamento *"piatto"* fino al canale 15, corrispondente di fatti all'ultimo canale VIS, per poi avere un incremento dei valori di riflettanza. Questo evidenzia come nelle immagini **VIS** sia quasi impossibile riuscire a distinguere i cluster ottenuti, mentre nel **NIR** i diversi gradi di maturazione sono ben percepibili.
Concludendo, il ***k-means*** si è dimostrato essere un algoritmo molto efficace considerato il suo funzionamento. Infatti, agendo sulla distanza tra il dato ed un punto definito come centroide del cluster, vista inoltre la struttura del dataset, riesce a distinguere con un certo grado di precisione i vari **layer**.

- ***dbscan:***
  Proprio come il ***k-means***, *MATLAB* dispone della funzione `dbscan` [45] che sfrutta di default la **distanza euclidea**. Fornendo i valori di ε, *minPts* e un dataset di dimensioni $M \times N$, l'output è costituito da una matrice $M \times 1$, dove l'unica colonna rappresenta a quale *cluster* è stato assegnata l'osservazione.

  Come detto in precedenza, l'efficienza del *dbscan* dipende soprattutto dalla scelta dei parametri ε e *minPts*.

  – per ε è stato considerato un numero molto piccolo poiché i valori di riflettanza sono normalizzai tra 0 e 1. Il punto di partenza è stato il grafico ***k-distance*** che ha evidenziato un *ginocchio* nel punto di coordinata $(x, y) = (1687, 0.1039)$.

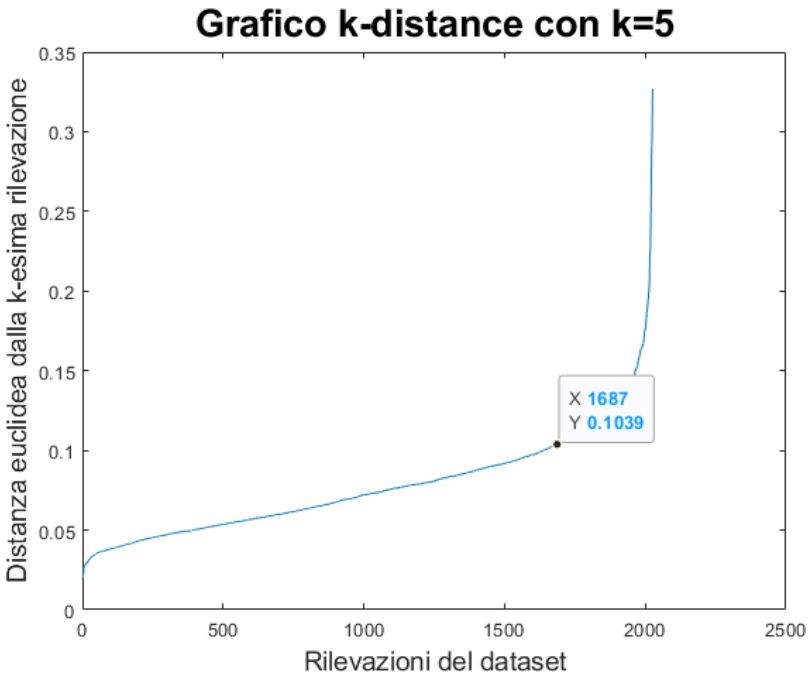


Figura 5.14 – Grafico k-distance

  Partendo quindi da $\varepsilon = 0.1$, sono state condotte varie prove, trovando come valore ottimo $\varepsilon = 0.05$

  – per la scelta di *minPts*, 4 è stato il valore che ha consentito di ottenere un buon compromesso tra qualità e numero di cluster.

Proprio riguardo quest'ultimo, invece che **5**, alla fine si è preferito scegliere i due parametri così da ottenere un numero di cluster maggiore, pari a **9** per l'esattezza, per i motivi affrontati tra poco.
Lanciando lo script *MATLAB*, contenuto nel file **dbscan.m**, il risultato è il seguente.

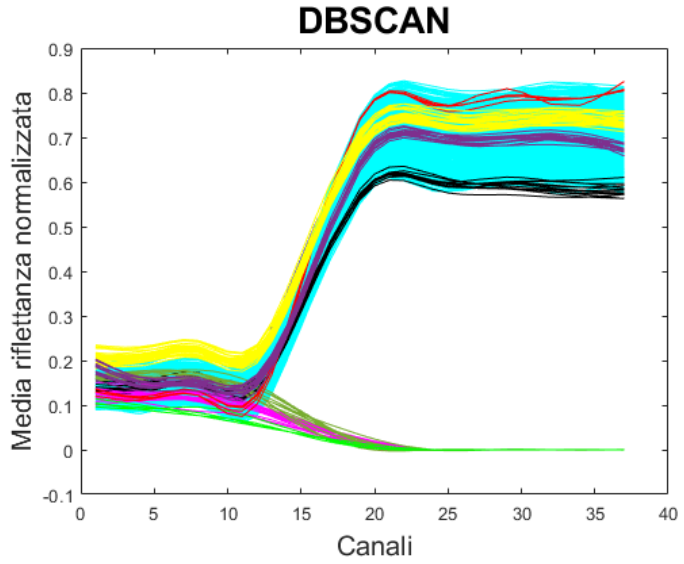


Figura 5.15 – Risultati del dbscan ottenuti sfruttando la media dei valori della riflettanza

L'algoritmo **dbscan**, definendo delle rilevazioni ***noise point***, esclude in questo caso un grossa fetta di valori non riuscendoli ad inserire in nessun cluster.

Dividendo i canali **VIS** da quelli **NIR** si ha un miglioramento nella *detection* dei cluster.

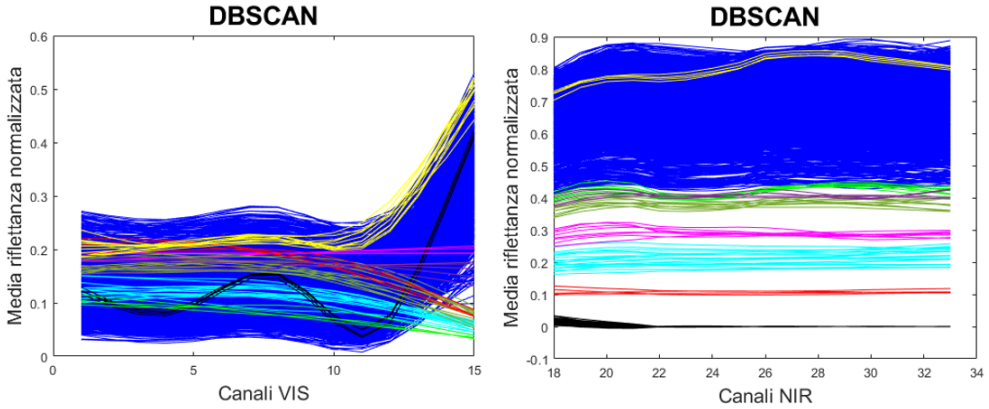


Figura 5.16 – Risultati del dbscan ottenuti sfruttando la media dei valori della riflettanza, distinguendo i canali VIS da quelli NIR

Ancora una volta, per le immagini **NIR** i risultati dimostrano come si riesca a distinguere meglio il comportamento delle olive rispetto alle **VIS**. Nonostante questo però, essendo il **dbscan** un algoritmo ***density-based***, la sua applicazione in questo contesto non risulta affidabile: in Figura 5.16, nel grafico di destra, il cluster di colore blu è quello predominante. Ciò accade proprio per la natura dell'algoritmo: essendo la maggior parte di questi dati molto vicini tra loro, una distinzione è possibile per i soli gruppi di rilevazioni situati ad una distanza sufficientemente alta da altri. I **9** raggruppamenti sono quindi serviti per dimostrare l'inefficienza di questo metodo nel presente caso di studio, portando il ***k-means*** ad essere l'unico metodo consistente in fase di **clustering**.

Capitolo 6

Conclusioni e sviluppi futuri

L'obiettivo che vede lo sviluppo di un sistema in grado di riconoscere differenze nel processo di maturazione di olive e che si basa su immagini multispettrali è quindi raggiunto. In base a quanto detto nei precedenti capitoli, in particolare si evidenziano i seguenti aspetti:

- La **coregistrazione** è stata un aspetto fondamentale per riuscire ad estrarre le bande per ogni singola oliva. Si tenga conto che potrebbe essere necessario modificare la funzione `registerImages` in base al dataset a disposizione, in particolare potrebbero essere testati altri metodi oltre a quelli proposti.
- Il **"filtraggio" delle bande rumorose** funziona bene per media e mediana, mentre per migliorare i risultati ottenuti con la moda potrebbero essere sfruttati altri metodi di interpolazione.
- I risultati ottenuti nella classificazione del grado di maturazione sfruttando un metodo **unsupervised** come il **clustering** dimostrano come l'algoritmo ***k-means*** sia in grado di separare i vari *stages*, mentre il ***dbscan*** fallisce nel compito.

Analizzando i risultati ottenuti, l'utilizzo di immagini multispettrali dimostra quindi maggior capacità di discriminazione rispetto ad immagini RGB classiche. Per valutare al meglio le prestazioni del sistema proposto, potrebbero essere in ogni caso necessari lavori futuri di etichettatura delle olive del dataset da parte di un team di esperti.

6.1 Sviluppi futuri

In conclusione, il presente elaborato pone le basi per la sperimentazione con reti neurali sfruttando modelli di **machine learning** per l'apprendimento supervisionato.
Sarebbe infatti interessante capire il comportamento di questa tipologia di approccio su immagini con una risoluzione così bassa.

Bibliografia

[1] Osservatorio AgriFood. L'agroalimentare è sempre più digitale: l'agricoltura 4.0 vale 450 mln di euro (+22%). April 23 2020.

[2] Internation Society of Precision Agricolture. Precision ag definition. https://www.ispag.org/about/definition.

[3] C. Starr, C.A. Evers, and L. Starr. *Biology: Concepts and Applications*. Brooks/Cole biology series. Thomson, Brooks/Cole, 2006.

[4] T.J. Bruno and P.D.N. Svoronos. *CRC Handbook of Fundamental Spectroscopic Correlation Charts*. CRC Press, 2005.

[5] IMEC. Hyperspectral sensors. 2018.

[6] Wikipedia Contributors. Digital camera. https://en.wikipedia.org/wiki/Digital_camera#Methods_of_image_capture.

[7] Wikipedia Contributors. Digital image. https://en.wikipedia.org/wiki/Digital_image#Raster_file_formats.

[8] Wikipedia Contributors. Rgb. https://it.wikipedia.org/wiki/RGB.

[9] Wikipedia Contributors. File:additivecolormixing.svg. https://it.wikipedia.org/wiki/File:AdditiveColorMixing.svg.

[10] Wikipedia Contributors. File:subtractivecolormixing.png. https://it.wikipedia.org/wiki/File:SubtractiveColorMixing.png.

[11] Comprendere ed utilizzare gli istogrammi. https://www.fotoritoccoprofessionale.it/articoli/comprendere-utilizzare-istogrammi-fotografia-digitale.htm. Fonte consultata in data 02/02/2020, non più disponibile al momento della pubblicazione.

[12] Lisa Gottesfeld Brown. A survey of image registration techniques(abstract). *ACM Computing Survey archive*, volume 24, December 1992.

[13] Parvez Rahi Sombir Singh Bisht, Bhumika Gupta. Image registration concept and techniques: A review. *International journal of engineering research and applications*, volume 4, April 2014.

[14] Math Works Dev Team. Techniques supported by registration estimator app. https://it.mathworks.com/help/images/techniques-supported-by-registration-estimator-app.html.

[15] David Arthur and Sergei Vassilvitskii. k-means++: The advantages of careful seeding. 2007.

[16] Nagesh Singh Chauhan. Dbscan clustering algorithm in machine learning. https://www.kdnuggets.com/2020/04/dbscan-clustering-algorithm-machine-learning.html.

[17] Jörg; Ester Martin; Kriegel Hans Peter; Xu Xiaowei Schubert, Erich; Sander. Dbscan revisited, revisited: Why and how you should (still) use dbscan. *ACM Trans. Database Syst.*, Volume 42, No.3, July 2017.

[18] MatLab Dev Team. Download r2020a. https://it.mathworks.com/downloads/web_downloads/download_release?release=R2020a, 2020.

[19] Wikipedia Contributors. File:matlab logo.png. https://it.wikipedia.org/wiki/File:Matlab_Logo.png.

[20] MatLab Dev Team. Image processing tololbox - matlab. https://it.mathworks.com/products/image.html.

[21] MatLab Dev Team. Statistics and machine learning toolbox - matlab. https://www.mathworks.com/products/statistics.html.

[22] Labelbox Dev Team. Labelbox site. https://labelbox.com/.

[23] Labelbox logo. https://www.kindpng.com/imgv/hRJJbbb_labelbox-logo-fabelio-hd-png-download/.

[24] Project Jupiter Dev Team. Project jupiter. https://jupyter.org/, 2018.

[25] Project Jupiter Dev Team. Overview. https://jupyterlab.readthedocs.io/en/stable/getting_started/overview.html, 2018.

[26] Wikipedia Contributors. File:jupyter logo.svg. https://commons.wikimedia.org/wiki/File:Jupyter_logo.svg.

[27] The R foundation. What is r? https://www.r-project.org/about.html.

[28] Rob Hyndman. forecast v8.13. https://www.rdocumentation.org/packages/forecast/versions/8.13, 2020.

[29] Simon Wood. mgcv v1.8-33. https://www.rdocumentation.org/packages/mgcv/versions/1.8-33, 2020.

[30] Wikipedia Contributors. Putty. https://en.wikipedia.org/wiki/PuTTY.

[31] Wikipedia Contributors. Github. https://en.wikipedia.org/wiki/GitHub.

[32] GitHub Dev Team. Github logo. https://github.com/logos, 2020.

[33] Wikipedia Contributors. Spectralon. https://en.wikipedia.org/wiki/Spectralon.

[34] NumPy Dev Team. Numpy. https://numpy.org/.

[35] MatLab Dev Team. imresize. https://it.mathworks.com/help/images/ref/imresize.html.

[36] MatLab Dev Team. adapthisteq. https://it.mathworks.com/help/images/ref/adapthisteq.html.

[37] Json vs csv. https://it.mathworks.com/help/images/ref/imresize.html.

[38] MatLab Dev Team. jsondecode. https://it.mathworks.com/help/matlab/ref/jsondecode.html.

[39] MatLab Dev Team. Register images using registration estimator app. https://it.mathworks.com/help/images/register-images-using-the-registration-estimator-app.html.

[40] MatLab Dev Team. Register images with projection distortion using control poin-

ts. https://it.mathworks.com/help/images/registering-an-aerial-photo-to-an-orthophoto.html.

[41] MatLab Dev Team. fitgeotrans. https://it.mathworks.com/help/images/registering-an-aerial-photo-to-an-orthophoto.html.

[42] MatLab Dev Team. rectangle. https://it.mathworks.com/help/matlab/ref/rectangle.html.

[43] MatLab Dev Team. kmeans. https://it.mathworks.com/help/stats/kmeans.html.

[44] Thaliath Amal Benson. Progettazione e sviluppo di un algoritmo basato su deep learning per la classificazione del grado di maturazione di olive mediante immagini. Master's thesis, Università Politecnica delle Marche, Anno accademico 2019-2020.

[45] MatLab Dev Team. dbscan. https://it.mathworks.com/help/stats/dbscan.html.

Elenco delle figure