UNIVERSITÀ POLITECNICA DELLE MARCHE

FACOLTÀ DI INGEGNERIA



Corso di Laurea Triennale in Ingegneria Informatica e dell'Automazione

### Progettazione e Sviluppo di un algoritmo basato su deep learning per la classificazione del grado di maturazione di olive mediante immagini

Design and Development of a deep learning algorithm to classify the ripening stage of olives from images

Relatore: Dott. Mancini Adriano Laureando: THALIATH AMAL BENSON

ANNO ACCADEMICO 2019-2020

## Indice

1	Intr	roduzione 4
	1.1	Studi e ricerche recenti
	1.2	Obiettivi
		1.2.1 Prima Fase 5
		1.2.2 Seconda Fase
		1.2.3 Terza fase
	1.3	Struttura della Tesi
<b>2</b>	Ma	teriali e metodi 7
	2.1	Dataset
	2.2	Python
		2.2.1 Pvcharm
		2.2.2 Google Colab
		2.2.3 Package principali
	2.3	MATLAB
	2.4	Repository del progetto
<b>0</b>		······································
3	Ola	Estrazione del grado di maturazione 11
	ა.1 იი	Dete set pro processing
	3.2	Data-set pre-processing   13     2.2.1   Drimi tontatini
	<u></u>	5.2.1       Primi tentativi       13         Algorithms di gileron ente in MATI AD       15
	3.3	Algoritmo di rilevamento in MALLAB
		3.3.1 Script MATLAB 16
	0.4	3.3.2 Interfacciamento con Python
	3.4	Mask R-ONN
		3.4.1 Implementazione $\dots$ 1/
	0.5	3.4.2 Risoluzione rilevamenti multipli
	3.5	$\mathbf{T}  (  \cdot  \cdot  \cdot  \cdot  \cdot  \cdot  \cdot  \cdot  \cdot$
	0.0	Trasformazioni utili
	3.6	Trasformazioni utili    19      Ciclo completo    21
	$3.6 \\ 3.7$	Trasformazioni utili    19      Ciclo completo    21      Creazione del training-set    22
	$3.6 \\ 3.7$	Trasformazioni utili    19      Ciclo completo    21      Creazione del training-set    22      3.7.1    Istogrammi    22
	3.6 3.7	Trasformazioni utili       19         Ciclo completo       21         Creazione del training-set       22         3.7.1       Istogrammi       22         3.7.2       Grado di maturazione       24
	3.6 3.7 3.8	Trasformazioni utili19Ciclo completo21Creazione del training-set223.7.1IstogrammiIstogrammi223.7.2Grado di maturazioneModelli di classificazione con Scikit-Learn25
	3.6 3.7 3.8	Trasformazioni utili19Ciclo completo21Creazione del training-set223.7.1Istogrammi223.7.2Grado di maturazione24Modelli di classificazione con Scikit-Learn253.8.1Implementazione26
	<ul> <li>3.6</li> <li>3.7</li> <li>3.8</li> <li>3.9</li> </ul>	Trasformazioni utili19Ciclo completo21Creazione del training-set223.7.1Istogrammi223.7.2Grado di maturazione24Modelli di classificazione con Scikit-Learn253.8.1Implementazione26Sperimentazione Transfer Learning con ResNet27

		3.9.2	Implementazione	27				
4	$\mathbf{Rist}$	ıltati		<b>31</b>				
	4.1	Hardw	are utilizzato	31				
	4.2	Rileva	mento maschere	31				
		4.2.1	Metodi di rilevamento	31				
		4.2.2	Analisi qualitativa degli insuccessi	32				
	4.3	Scikit-	Learn	33				
		4.3.1	Classificazione con 5 gradi	34				
		4.3.2	Classificazione con 3 gradi	35				
		4.3.3	Considerazioni	36				
	4.4	ResNe	t-18	37				
<b>5</b>	Con	clusio	ni e sviluppi futuri	39				
	5.1	Soluzio	oni ed applicazioni	39				
Bi	bliog	grafia		41				
El	Elenco delle figure							
Elenco delle tabelle								

# Capitolo 1 Introduzione

Con l'avvento dell'**Agricoltura 4.0**, anche detta *Smart Farming*, nuove tecnologie di telematica e gestione di grandi quantità di dati sono state combinate con il già conosciuto principio dell'*agricoltura di precisione*; questo ha permesso così, a chi lavora nel settore, di generare i dati necessari per le scelte strategiche di produzione direttamente nelle proprie fattorie attraverso sistemi di elaborazione dati e strumenti di cattura delle immagini, quali droni e UAV. Ma secondo molti studi entro il 2050 il mondo dovrà aumentare la produzione di cibo del 60%. A tale scopo fa appello la nuova **Agricoltura 5.0** che punta ad aumentare l'efficienza della produzione utilizzando *in primis* tecnologie che coinvolgono processi automatizzati e sistemi di decisione autonomi basati sull'intelligenza artificiale [1].

Per questa tesi di laurea verrà trattato in particolare la classificazione del grado di maturazione dei frutti tramite algoritmi di *machine learning*.

#### 1.1 Studi e ricerche recenti

Negli ultimi anni sono stati messi a punto solo un ristretto numero di algoritmi per rilevare i frutti maturi. Questi progetti non sono ancora pronti per il pieno utilizzo nel settore ma i risultati sono sempre molto promettenti.

I ricercatori dell'Università di Campinas e del Londrinas State University hanno provato a sviluppare un algoritmo di machine learning per stabilire la maturazione delle papaie in modo da aiutare produttori e consumatori in un prossimo futuro [2]. Non è stato utilizzato però un algoritmo di deep learning non avendo a disposizione una grande quantità di frutti, perciò il software ha una percentuale di accuratezza dell'84,4% per l'ultimo stadio di maturazione. Tuttavia questo risultato è anche dovuto al fatto che non sempre il colore della buccia rispecchia l'effettiva maturazione della polpa: miglioramenti futuri potrebbero portare l'algoritmo a portata di tutti nel proprio smartphone.

Un'altra ricerca ha avuto invece successo nel classificare la maturazione dei pomodori attraverso l'utilizzo di *Principal Component Analysis* (PCA), dei *Support Vector Machines* (SVM) e dell'*analisi discriminante lineare* (ADL) con una percentuale massima di accuratezza del 90,8% [3].

Infine un prototipo in grado classificare le arance è stato sviluppato dall'azienda

See Tree [4]. Un drone provvede a catturare le foto degli alberi, le arance vengono successivamente rilevate dall'algoritmo Faster R-CNN e classificate dal modello InceptionResnet V2. Secondo il team di sviluppo la performance dell'algoritmo è paragonabile a quella di una persona e può essere ri-allenata per classificare altri tipi di frutta.



Figura 1.1 – L'algoritmo di SeeTree in azione, la maturazione è valutata in percentuale. [4].

### 1.2 Obiettivi

Lo scopo del progetto è lo sviluppo di un algoritmo per la classificazione del grado di maturazione delle olive, sfruttando modelli di *machine learning* e la sperimentazione di un modello di rete convoluzionale per il *deep learning*. Il dataset utilizzato dal progetto è stato raccolto in collaborazione con il prof. Davide Neri del D3A - Università Politecnica delle Marche, fornendo immagini con le posizioni delle relative olive. Perciò per il seguente lavoro di tesi ci si concentrerà sul lato software, in particolare sulla parte di classificazione. Il progetto si suddivide in tre principali fasi di seguito riportate.

#### 1.2.1 Prima Fase

Isolamento delle aree di interesse (le olive) dalle immagini utilizzando i file di label contenenti le coordinate necessarie.



Figura 1.2 – Esempi di risultati previsti dall'estrazione delle olive dalle immagini del dataset

#### 1.2.2 Seconda Fase

La seconda fase prevede il miglioramento delle immagini rimuovendo tutto ciò che non fa parte dell'oliva, che si traduce nel dover generare una maschera di separazione tra sfondo e frutto. Questa è la fase più corposa del progetto.



Figura 1.3 – Esempi dei risultati da ottenere applicando le maschere di separazione.

#### 1.2.3 Terza fase

Nell'ultima fase si generano i training set e vengono implementati e collaudati i modelli di *machine learning* per stabilire le maturazioni. Per concludere si producono le metriche per l'analisi dei risultati.

#### 1.3 Struttura della Tesi

Per esporre al meglio tutti gli aspetti del progetto la tesi sarà strutturata nei seguenti capitoli:

- Capitolo 2: Elencherà le tecnologie principali utilizzate per lo sviluppo e fornirà la modalità per scaricare il materiale ed il codice associato.
- Capitolo 3: Illustrerà in dettaglio i singoli passi seguiti per sviluppare l'algoritmo di classificazione con estratti di codice e le immagini più significative.
- Capitolo 4: Capitolo riservato all'analisi dei risultati ottenuti durante i test dei modelli di *machine learning*.
- **Capitolo 5:** Conclusioni e sviluppi futuri per migliorare i risultati e le prestazioni dell'algoritmo.

# Capitolo 2 Materiali e metodi

Il progetto è stato sviluppato e scritto nel linguaggio *Python* con una piccola porzione di codice in *MATLAB*. Sono state utilizzate librerie per la manipolazione delle immagini digitali e modelli di intelligenza artificiale per l'apprendimento supervisionato. La maggior parte del software è eseguita in locale, la fase sperimentale con l'algoritmo di *deep learning* è invece eseguita tramite un servizio di *cloud computing*.

#### 2.1 Dataset

Il dataset utilizzato per questo progetto è composto da un set di immagini, ciascuna accompagnata da alcune varianti generate dal processo di *data augmentation*<sup>1</sup>, e da un corrispondente set di label. Queste *labelbox* sono state convertite in formato *YOLO* tramite il framework per *machine learning* **DarkNet** e sono file testuali contenenti dimensioni e coordinate del *bounding box*<sup>2</sup> [5]. Il dataset è disponibile nel repository indicato nella sezione 2.4.

### 2.2 Python

*Python* è un linguaggio interpretato e di conseguenza in media più lento nell'esecuzione di codice a differenza dei comuni linguaggi compilati, eppure negli ultimi anni è diventato uno delle scelte più popolari per la ricerca e lo sviluppo di algoritmi di intelligenza artificiale, questo grazie alla grandissima quantità di librerie di alta qualità che esso dispone e la lista è in costante aumento. Per questo progetto è stato utilizzato **Python 3.7**, tuttavia il codice può essere facilmente adattato a versioni successive.



 $<sup>^1\</sup>mathrm{Processo}$  per aumentare la quantità del dataset effettuando semplici trasformazioni come rotazione, inversione lungo gli assi e sfocatura

<sup>&</sup>lt;sup>2</sup>Rettangolo che racchiude l'oliva

#### 2.2.1Pycharm

Pycharm di Jetbrains [6] è l'IDE utilizzato per lo sviluppo locale di gran parte del progetto. I package utilizzati sono installati localmente nell'ambiente virtuale associato al progetto e il versioning del codice è gestito da Github. Per il repository vedere la sezione 2.4.



Colaboratory [7] è un servizio di cloud computing di Google per l'esecuzione di codice Python in formato Jupiter Notebook.<sup>3</sup> Il vantaggio è la disponibilità di una GPU dedicata per velocizzare ad esempio l'allenamento di reti di machine *learning*. Per questo progetto Google Colab è stato utilizzato per la parte finale di sperimentazione con il modello di deep learning ResNet-18.

#### 2.2.3Package principali

I package principali utilizzati sono i seguenti:

#### Numpy: [8]

Numpy è il package principale per creare e lavorare su array e matrici rendendolo un'alternativa valida a MATLAB. È spesso incluso o richiesto in altri package come nel caso di OpenCV che lo utilizza per memorizzare le immagini.

#### **OpenCV:** [9]

OpenCV è uno dei più popolari package per Python per la gestione e manipolazione di immagini e video, grazie ad una vasta selezione di funzioni. Ha anche alcuni moduli per la gestione di reti neurali profonde.

#### Scikit-image: [10]

Scikit-image è un'altra libreria di manipolazione immagini molto simile ad OpenCV. Ha però alcune funzioni in più per rilevare e disegnare figure geometriche nelle immagini.







 $<sup>{}^{3}</sup>$ È un ambiente Python interattivo che permette di eseguire sia codice script che comandi console utilizzando lo stesso stack di variabili.

#### Scikit-learn: [11]

Package composto da moltissimi strumenti per il *machine learning*, inclusi i modelli più popolari, funzioni per la normalizzazione del dataset e per il calcolo delle metriche.

#### **Pytorch:** [12]

Pytorch è un importante framework per lo sviluppo di algoritmi di *machine learning* che si focalizza soprattutto su modelli di *deep learning*. Inoltre il package permette di eseguire codice sfruttando l'accelerazione GPU. Nel progetto il package è utilizzato solo in *Google Colaboratory*.

#### Matplotlib: [13]

Matplotlib è una libreria per creare visualizzazioni statiche, animate o interattive, utilizzata sopratutto per tracciare grafici di vario genere. Per questo progetto è stato utilizzato per visualizzare gli istogrammi delle immagini.

Altri package come ad esempio *Scipy* [14] non vengono utilizzati direttamente ma sono inclusi e scaricati automaticamente durante l'installazione dei sopra elencati package.

### 2.3 MATLAB

Matlab è un potente linguaggio utilizzato per progettare algoritmi di calcolo matematico e scientifico ampiamente utilizzato da moltissime aziende ed organizzazioni. È distribuito attraverso la relativa applicazione desktop ricca di molti tool che vanno dal campo dell'automazione industriale all'intelligenza artificiale. In questa tesi è stata utilizzata la versione **2020a** [15] con licenza accademica fornita dall'*Università Politecnica delle Marche*. Il linguaggio non è stato utilizzato in modalità standalone ma bensì tramite il package *matlab engine* che funziona da interfaccia per poter eseguire i suoi script all'interno di *Python*. Per l'installazione vedere il README presente nel repository.



Il codice può essere scaricato dal repository *Github* https://github.com/ABTCoder/ Tirocinio-AnalisiMaturazione una volta richiesto il permesso d'accesso all'email amalbenson98@gmail.com.

MATLAB



# Ú



È possibile creare da zero l'ambiente virtuale del progetto in caso di migrazione ad un'altra versione di Python o in caso di problemi con i package. È importante inoltre seguire la procedura di installazione del *matlab engine* e della compilazione del file generateEllipseCandidates.cpp indicato nel README.

Il file *Jupiter Notebook* eseguito su *Google Colab* è disponibile al seguente link: https://colab.research.google.com/drive/1yjzZE31H4o0\_du4PwEvoPKgGRBZYDXox? usp=sharing.

Il dataset utilizzato per questa parte si trova all'interno delle cartelle images e single olives nel repository Github.

### Capitolo 3

## Classificazione del grado di maturazione

Questo capitolo spiegherà in dettaglio le tecniche e gli algoritmi utilizzati per stabilire il grado di maturazione delle olive partendo dalla manipolazione iniziale delle immagini del dataset al *feeding* delle reti di *machine learning*.

Per concludere verrà mostrata la sperimentazione di un algoritmo di *deep lear*ning e il principio del *transfer learning* con la rete *ResNet*.

Il work flow principale del progetto è indicato approssimativamente nel seguente schema:



Figura 3.1 – Il Workflow del progetto, punto fulcro è la generazione del dataset

#### 3.1 Estrazione delle olive

Le *labels* e i *bounding boxes* messi a disposizione insieme alle immagini del dataset sono in formato *DarkNet YOLO*. Per ogni oliva sono definiti 5 valori: il primo è la label che indica che è un'oliva<sup>1</sup>, gli altri quattro valori sono nell'ordine **xcenter**, **ycenter**, **width** e height che indicano posizione e dimensioni del *bounding box*, ovvero il rettangolo che racchiude l'oliva di interesse nell'immagine intera. Questi valori però sono normalizzati rispetto alle dimensioni dell'immagine in modo da poter essere utilizzate anche qualora si decidesse di ridimensionarla. Il relativo codice di estrazione è il seguente e si trova nel file **utils.py** 

```
def darknet_bbox(filename, height, width, padding=0):
1
2
       with open(filename, 'r') as f:
           vals = []
3
           for line in f:
4
\mathbf{5}
               nums = line.split()
               for num in nums:
6
                   vals.append(num)
7
8
           ndarray = np.array(vals, dtype=np.longfloat)
9
           ndarray = ndarray[ndarray > 0]
10
           k = int(ndarray.shape[0] / 4)
11
12
           ndarray = ndarray.reshape(k, 4)
           b_list = []
13
           for coord in ndarrav:
14
15
               x_center = int(round(coord[0] * width))
               y_center = int(round(coord[1] * height))
16
17
               half_width = int(round((coord[2] * width)/2))
               half_height = int(round((coord[3] * height)/2))
18
               x_start = x_center - half_width
19
               y_start = y_center - half_height
20
21
               x_stop = x_center + half_width
               y_stop = y_center + half_height
22
               if padding:
23
24
                   x_start = max(x_start-padding, 0)
                   y_start = max(y_start-padding, 0)
25
                   x_stop = min(x_stop + padding, width)
26
                   y_stop = min(y_stop + padding, height)
27
               b_list.append([x_start, x_stop, y_start, y_stop])
28
       return b_list
29
```

I valori sono tutti scritti su un unica riga, perciò nelle righe **2 - 12** si leggono tutti i numeri, li si aggiungono ad una lista temporanea, si rimuovono le label e si converte la lista in una matrice Numpy  $k \times 4$  dove k è il numero di olive presenti nell'immagine.

Di seguito si scorre con un ciclo tra le righe della matrice ottenuta e si convertono i parametri del bounding box nel formato x\_start, x\_stop, y\_start e y\_stop per compatibilità immediata con l'indexing di *Numpy*.

Si restituisce infine la nuova lista di bounding box.

Il parametro padding serve per ampliare le dimensioni del bounding box, l'utilizzo verrà motivato al punto 1 della sezione 3.5.

<sup>&</sup>lt;sup>1</sup>Sempre 0 nel dataset utilizzato essendo state rilevate solo le olive.

#### 3.2 Data-set pre-processing

Per risultati più accurati sia durante l'allenamento che durante il testing dei classificatori è necessario rimuovere dalle immagini tutto ciò che non fa parte del frutto, come sfondo, foglie e olive adiacenti. Per tale scopo sono stati testati funzioni di segmentazione delle immagini già disponibili in OpenCV e algoritmi di rilevamento delle ellissi più complessi. L'obiettivo principale è ottenere dunque una maschera binaria da passare alla funzione di calcolo dell'istogramma in modo che vengano ignorati tutti i pixel che non fanno parte dell'oliva.

#### 3.2.1 Primi tentativi

Sono riportate di seguito alcune funzioni di OpenCV e di SciKit Image, molto semplici da implementare ma dipendenti da vari parametri che potrebbero non essere sempre adatti ad ogni immagine.

#### • Thresholding

È il metodo più rapido e diretto per separare il primo piano dallo sfondo, utilizzato con la funzione cv.threshold() [16].

Dopo la conversione in scala di grigi si passa l'immagine alla funzione specificando il valore di soglia nell'intervallo [0, 255]. Ma se determinati valori risultano ottimali per una certa immagine potrebbero non esserlo per altre, ad esempio in caso di illuminazione differente o diverso grado di maturazione dell'oliva. Per questo si è optato per il *metodo Otsu* [17] che applica la soglia minimizzando la varianza dell'intensità intra-classe, ovvero nel caso di due picchi di intensità diversi nell'istogramma dell'immagine. È necessario inoltre rimuovere il rumore dall'immagine prima di applicare la soglia attraverso una sfocatura Gaussiana [18].

```
1 import cv2 as cv
2
3 gray = cv.cvtColor(image, cv.COLOR_BGR2GRAY)
4 gray = cv.GaussianBlur(gray, (5, 5), 0)
5 _, mask = cv.threshold(gray, 0, 255, cv.THRESH_BINARY + cv.THRESH_OTSU)
```



Figura 3.2 – Risultato abbastanza pulito e preciso della soglia con metodo Otsu.

Tuttavia anche questo metodo in alcuni casi può fallire producendo una maschera molto sporca o del tutto inutilizzabile.



Figura 3.3 – Un oliva matura potrebbe dare un risultato scarso, perché le ombre si confondono con i bordi.

```
• Canny edge detection + Hough Ellipse
```

Questo metodo sfrutta innanzitutto l'algoritmo multifase di rilevamento dei contorni *Canny* [19] utilizzando la funzione cv.canny() [20]. Di base questa funzione ha bisogno di due parametri: threshold1 e threshold2, rispettivamente la prima e la seconda soglia per il processo di isteresi interno all'algoritmo. Essi vanno manualmente regolati per ogni immagine, tuttavia è possibile impostarli automaticamente tramite un semplice calcolo della mediana dei valori dei pixel [21]:

```
import numpy as np
import cv2 as cv
gray = cv.cvtColor(image, cv.COLOR_BGR2GRAY)
sigma = 0.6
v = np.median(gray)
r lower = int(max(0, (1.0 - sigma) * v))
upper = int(min(255, (1.0 + sigma) * v))
gray = cv.Canny(gray, lower, upper)
```



Figura 3.4 – Alcuni risultati dell'algoritmo canny, anche in questo caso ci sono imperfezioni.

Si passa il risultato all'algoritmo di rilevamento di ellissi *Hough Ellipses* [22] del pacchetto *Scikit-Image* che produrrà una lista di possibili ellissi; va dunque estratta quella con il valore dell'*accumulatore* più alto. Infine si ricavano i parametri necessari per l'equazione dell'ellisse:

$$\frac{((x-h)\cos(A) + (y-k)\sin(A))^2}{(a^2)} + \frac{((x-h)\sin(A) - (y-k)\cos(A))^2}{(b^2)} = 1$$

Dove  $h \in k$  sono rispettivamente i centri xc e yc, ed A la rotazione rispetto all'asse X, salvata nella variabile rotation. Questi parametri sono

utilizzati nella funzione fill\_ellipse() del package skimage.draw [23] che provvede a disegnare l'area dell'ellisse nella maschera.

```
1 import numpy as np
2 import cv2 as cv
3 from skimage.transform import hough_ellipse
4 from skimage.draw import ellipse as fill_ellipse
5
6 height, width = gray.shape[0:2]
7 mask = np.zeros((height, width), np.uint8)
8 ellipses = hough_ellipse(gray, threshold=100, accuracy=100, min_size=
       round(max(height, width) / 2),
                            max_size=round(min(height, width)))
9
10 if ellipses.size > 0:
11
      ellipses.sort(order='accumulator')
      best = list(ellipses[-1])
12
13
      yc, xc, a, b = [int(round(x)) for x in best[1:5]]
       rotation = best[5]
14
       if a == 0 or b == 0:
15
          mask = None
16
       else:
17
          rr, cc = fill_ellipse(yc, xc, a, b, mask.shape, rotation)
18
          mask[rr, cc] = 255
19
```

Purtroppo come mostrato nella seguente figura il rilevamento non è sempre corretto.



Figura 3.5 – Risultato del rilevamento con Hough Ellipse riferito all'esempio 3.4(a).

Paradossalmente in alcuni casi anche se l'isolamento dei contorni con Can-ny è più accurato, come in figura 3.4(b), l'algoritmo potrebbe fallire completamente e produrre ellissi con uno dei due assi nullo.

Ma il problema più grande di questo metodo rimane il tempo di esecuzione che spesso è nell'ordine di diversi minuti.

#### 3.3 Algoritmo di rilevamento in MATLAB

Questo algoritmo, sviluppato in *MATLAB* da Changsheng Lu [24] è basato sul risultato dell'articolo Arc-Support Line Segments Revisited: An Efficient High-Quality Ellipse Detection [25].

È il migliore in termini di velocità di esecuzione<sup>2</sup> e precisione delle ellissi.

 $<sup>^2 \</sup>rm Principalmente per il fatto che una parte dell'algoritmo è compilato in C++, vedere il processo di installazione al capitolo 2.3$ 

#### 3.3.1 Script MATLAB

Questa è la parte MATLAB dell'algoritmo, scritta sottoforma di funzione con restituzione del risultato, ovvero la lista delle ellissi rilevate:

#### 3.3.2 Interfacciamento con Python

Dopo aver correttamente caricato il **MATLAB Engine** l'interfacciamento con *Python* viene eseguito nel seguente modo:

```
1 import matlab.engine
2 from skimage.draw import ellipse as fill_ellipse
3 import numpy as np
\mathbf{4}
5 def extract_matlab_ellipses(image, engine):
       image_mat = matlab.uint8(list(image.ravel(order='F')))
6
7
       ch = 3
       if len(image.shape) == 2:
8
          ch = 1
9
10
       image_mat.reshape((image.shape[0], image.shape[1], ch))
11
       ellipses = engine.get_ellipses(image_mat, nargout=1)
12
       mask = np.zeros((image.shape[0], image.shape[1]), np.uint8)
13
14
15
       for el in ellipses:
           rr, cc = fill_ellipse(el[1], el[0], el[3], el[2], mask.shape, -el[4])
16
           mask[rr, cc] = 255
17
^{18}
       if ellipses.size[0] == 0:
19
20
           mask = None
^{21}
       return mask
22
```

Nelle righe 8 - 12 si converte l'immagine OpenCV in un formato compatibile con MATLAB per poi passarla alla funzione tramite l'engine. Di seguito allo stesso modo del metodo 3.2.1 si estraggono i 5 parametri per ogni ellisse<sup>3</sup> e li si passano alla funzione fill\_ellipse().

 $<sup>^{3}</sup>$ Anche se in questo modo potrebbero essere disegnate più ellissi, nei rari casi in cui succede sono per lo più identiche e la loro combinazione va a migliorare la maschera



Figura 3.6 – Esempi di immagini, maschere e sovrapposizione per constatare la precisione dell'algoritmo in MATLAB.

L'algoritmo può fallire se ci sono molte ombre, se lo sfondo è poco distinguibile dall'oliva e se ci sono molte foglie in primo piano.

#### 3.4 Mask R-CNN

Un'altra soluzione per ottenere la maschera delle olive è l'algoritmo di machine learning Mask R-CNN [26], una rete neurale convoluzionale che oltre a identificare e indicare la posizione del soggetto nell'immagine è in grado di definire una maschera approssimativa dello stesso. È stato allenato sul dataset COCO [27], in particolare è stato utilizzato una revisione compatibile con cv.dnn(), il modulo per algoritmi deep neural network di OpenCV [28]. Questa soluzione ha un tasso di successo più alto ma produce maschere leggermente sporche.

#### 3.4.1 Implementazione

Per prima cosa vanno caricati nella rete due file presenti nel repository:

- frozen\_inference\_graphp.pb
- mask\_rcnn\_inception\_v2\_coco\_2018\_01\_28.pbtxt

rispettivamente i pesi della rete pre-allenata sul dataset *COCO* e le impostazioni di configurazione della rete.

```
1 import numpy as np
2 import cv2 as cv
3 import utils
4
5 net = cv.dnn.readNetFromTensorflow("mask-rcnn-coco/frozen_inference_graph.pb",
                                       "mask-rcnn-coco/
6
       mask_rcnn_inception_v2_coco_2018_01_28.pbtxt")
7
8 def extract_cnn_mask(image):
9
       (H, W) = image.shape[:2]
10
      newMask = np.zeros((H, W), np.uint8)
11
      blob = cv.dnn.blobFromImage(image, swapRB=True, crop=False)
12
      net.setInput(blob)
13
14
       (boxes, masks) = net.forward(["detection_out_final", "detection_masks"])
15
16
17
       for i in range(0, boxes.shape[2]):
```

```
classID = int(boxes[0, 0, i, 1])
18
19
           confidence = boxes[0, 0, i, 2]
20
           if confidence > 0:
21
                box = boxes[0, 0, i, 3:7] * np.array([W, H, W, H])
22
                (startX, startY, endX, endY) = box.astype("int")
23
                center_x = int(round((startX + endX) / 2))
24
                center_y = int(round((startY + endY) / 2))
25
26
27
                if utils.check_box_center(W, H, center_x, center_y):
                    boxW = endX - startX
boxH = endY - startY
28
29
                    mask = masks[i, classID]
30
                    mask = cv.resize(mask, (boxW, boxH),
31
32
                                       interpolation=cv.INTER_LANCZOS4)
                    mask = (mask > 0.3)
33
                    newMask[startY:endY, startX:endX][mask] = 255
34
35
```

36 **return** newMask

La funzione extract\_cnn\_mask(image), in main.py, si occupa di generare la maschera<sup>4</sup>. Alla rete viene passata non l'immagine ma un cosidetto *blob* [29], ovvero regioni dell'immagine che presentano caratteristiche e proprietà simili e quindi raggruppabili.

Nelle righe **13 - 15** si imposta dunque l'input e si esegue il *feed forward* della rete ottenendo i *bounding boxes* che delimitano le regioni di interesse e le maschere associate.

Normalmente si dovrebbe impostare una soglia minima per la confidenza della predizione ma essa è riferita alla classificazione e non alla maschera, di conseguenza si prendono tutti i boxes (righe **19 - 21**).

La maschera, una volta estratta, a riga **30** viene riportata alle dimensioni del suo bounding box, questo perché in una rete convoluzionale a livelli di convoluzione seguono spesso livelli di *pooling* [30] che riducono le dimensioni dell'input per diminuire la complessità. Infine la maschera a valori decimali viene utilizzata per creare la maschera in scala di grigi a riga **34**.



Figura 3.7 – Esempio delle immagini e diverse sovrapposizioni generate da Mask R-CNN

#### 3.4.2 Risoluzione rilevamenti multipli

In alcuni casi la rete potrebbe rilevare anche le olive adiacenti e di conseguenza combinarne le maschere. Questo è un risultato indesiderato, specialmente se sono di gradi di maturazione diversi.

 $<sup>^4 \</sup>rm Non$  sono riportati nel codice le funzioni opzionali utilizzati per la visualizzazione e l'assegnazione delle label, per il codice completo vedere il file.main.py



Figura 3.8 – In questa immagine viene rilevata anche l'oliva in alto a sinistra

Per risolverlo si controlla che il centro del bounding box si trovi all'interno di un area rettangolare centrata nell'immagine ma con dimensioni pari al 30%. Di seguito è riportata la relativa funzione , presente in **utils.py** ed utilizzata a riga **27** del codice a 3.4.1:

```
1 def check_box_center(w, h, x, y):
        allowed_h = int(round(0.30 * h))
2
        allowed_w = int(round(0.30 * w))
3
        sx = int(round((w - allowed_w) / 2))
sy = int(round((h - allowed_h) / 2))
4
\mathbf{5}
        ex = sx + allowed_w
6
        ey = sy + allowed_h
7
8
        if ex > x > sx and ey > y > sy:
9
             return True
10
        else:
11
             return False
12
```



Figura 3.9 – La funzione di controllo esclude correttamente l'oliva nell'angolo. Il bordo netto è dovuto ad altro.

Tuttavia è ancora presente dell'imprecisione ma può essere eventualmente risolto allenando la rete con un dataset specifico per le olive.

#### 3.5 Trasformazioni utili

I due algoritmi precedentemente elencati rilevano l'oliva nella gran parte dei casi ma per aumentare ulteriormente la probabilità di successo si possono eseguire alcune semplici trasformazioni all'input per migliorarne le caratteristiche e renderlo più facile da rilevare.

#### 1. Aumento delle dimensioni del bounding box:

I bounding box generati dalla rete DarkNet sono molto precisi nel racchiudere le olive, ma in alcuni casi possono essere troppo ristretti e tagliare parti dell'oliva. Ciò influisce negativamente sul rilevamento delle ellissi. Una semplice soluzione è quella di aumentare le dimensioni dei box di un determinato valore lungo ogni lato, senza ovviamente superare i bordi dell'immagine.

In base ai vari test il valore ottimale del padding è di 10 pixel. Per il codice fare riferimento alla sezione 3.1.

#### 2. Raddoppiamento delle dimensioni:

Soluzione semplice e molto rapida che migliora il rilevamento nella maggior parte dei casi.

L'interpolazione utilizzata è Lanczos4 che utilizza una funzione sinc finestrata come nucleo di convoluzione [31].

```
1 nw = int(image.shape[1] * 2)
2 nh = int(image.shape[0] * 2)
3 dim = (nw, nh)
4 image = cv.resize(image, dim, interpolation=cv.INTER_LANCZOS4)
```

Tuttavia in rarissimi casi il ridimensionamento può peggiorare o persino far fallire il rilevamento, per questo viene disattivato nell'ultimo ciclo di data pre-processing mostrato nella successiva sezione.

#### 3. Aumento nitidezza:

Per eliminare invece la sfocatura si è tentato prima con un resize 2x seguito da un resize 0.5x per riportarlo alle dimensioni originali ma con interpolazione bilineare, ma il numero di rilevamenti è rimasto pressoché invariato.

Più successo ha avuto invece un filtro di nitidezza tramite un nucleo di convoluzione apposito [32] applicato con la funzione cv.filter2D() [33].

```
1 kernel = np.array([[0, -1, 0], [-1, 5, -1], [0, -1, 0]])
2 image = cv.filter2D(image, -1, kernel)
```



Figura 3.10 – Esempio di filtro di nitidezza. La sfocatura è quasi del tutto rimossa.

#### 4. Equalizzazione dell'istogramma:

Immagini in cui i valori dei pixel (in scala di grigi) sono accumulati su un intervallo stretto nel corrispondente istogramma appaiono come immagini "piatte" a livello visivo. Per mettere in risalto la separazione del primo piano dallo sfondo si possono perciò distribuire i valori in un intervallo più ampio tramite la seguente funzione [34]:

```
1 gray = cv.cvtColor(image, cv.COLOR_BGR2GRAY)
```

```
2 eq = cv.equalizeHist(gray)
```



Figura 3.11 – Immagine e istogramma originali in scala di grigi



Figura3.12– Immagine e istogramma dopo l'equalizzazione. I valori sono ora distribuiti su tutto il range.

#### 3.6 Ciclo completo

A questo punto per il miglior tasso di successo possibile ogni oliva viene passata con dimensioni raddoppiate all'algoritmo MATLAB, e se non viene rilevata o se la maschera è troppo piccola per via di un falso rilevamento<sup>5</sup> si ritenta con l'equalizzazione di istogramma. Di nuovo si ricontrolla la validità della maschera e in caso di insuccesso si ripete il processo ma con Mask R-CNN. Questo è il primo ciclo, in caso di insuccesso parte il secondo ciclo, identico ma con filtro di nitidezza, ed infine il terzo ciclo disattiva il raddoppiamento delle dimensioni per quei rari casi citati nel punto 2 nella sezione 3.5.

Il seguente codice, in **main.py**, mostra in dettaglio il ciclo:

```
1 half_size = False
2 double_size = True
3 sharpening = False
4 for n in range(3):
5     processed = pre_processing(roi_bgr_extra, half_size, double_size, sharpening)
```

<sup>&</sup>lt;sup>5</sup>Tramite la funzione calc\_white\_percentage(mask) che calcola la percentuale di pixel bianchi nell'immagine, in generale la percentuale deve essere maggiore del 20%

```
rgb_processed = cv.cvtColor(processed, cv.COLOR_BGR2RGB)
6
       mask = extract_matlab_ellipses(rgb_processed) # METODO 4 MATLAB
7
8
       white_p = calc_white_percentage(mask)
9
       if mask is None or white_p < min_mask:</pre>
10
           eq = cv.cvtColor(processed, cv.COLOR_BGR2GRAY)
11
           eq = cv.equalizeHist(eq)
12
           mask = extract_matlab_ellipses(eq)
13
14
           white_p = calc_white_percentage(mask)
15
       if mask is None or white_p < min_mask:</pre>
16
17
           mask = extract_cnn_mask(processed)
18
           white_p = calc_white_percentage(mask)
19
20
       if mask is None or white_p < min_mask:</pre>
           eq = cv.cvtColor(processed, cv.COLOR_BGR2GRAY)
21
           eq = cv.equalizeHist(eq)
22
           eq = cv.cvtColor(eq, cv.COLOR_GRAY2BGR)
23
           mask = extract_cnn_mask(eq)
24
25
           white_p = calc_white_percentage(mask)
26
       if mask is not None and white_p > min_mask:
27
           break;
28
29
       sharpening = True
       if n == 1:
30
           double_size = False
31
```

Se dopo tutti i e tre i cicli l'oliva non è stata rilevata allora l'istogramma verrà calcolato senza maschera ma senza i 10 pixel di padding per il bounding box.

#### 3.7 Creazione del training-set

Il training-set è stato creato con i seguenti parametri:

- con e senza maschera;
- spazio di colori RGB e HSV;
- 8, 16, 32 bin per gli istogrammi;
- 5 e 3 classi per il grado di maturazione.

In questo modo sono disponibili 24 tipi di *training-set* per una ottenere una grande varietà di risultati durante l'allenamento delle reti. Questi parametri sono spiegati in dettaglio nelle sezioni successive.

#### 3.7.1 Istogrammi

Calcolare l'istogramma di un'immagine significa contare le frequenze dei pixel per ogni intervallo del canale di riferimento. Normalmente gli intervalli corrispondono ai 256 valori possibili per un canale, ma per una rete di *machine learning* risultano essere eccessivi come numero di input, ovvero di *feature*. Per questo gli intervalli vengono ampliati e quindi ridotti in numero, possibilmente in potenze di 2, e in particolare sono stati scelti 8, 16, e 32 bin (intervalli).

L'istogramma è stato calcolato con l'apposita funzione in OpenCV [35] che permette inoltre di passare direttamente la maschera:

hist = cv.calcHist([img],[0],mask,[256],[0,256])



Figura 3.13 – Istogrammi della stessa immagine con bin diversi.

Gli istogrammi non sono stati calcolati solo per i canali rosso, verde e blu del più comune spazio di colori **RGB** ma anche per lo spazio **HSV** [36], *hue-saturation-value*, che si avvicina al modo in cui le persone percepiscono i colori.

*Hue* corrisponde ai gradi di rotazione attorno alla ruota dei colori, *saturation* indica la quantità della tonalità rispetto al bianco mentre *value* rispetto al nero. In questo modo i valori per il canale *Hue* sono consistenti anche in caso di illuminazione diversa.



Figura 3.14 – Rappresentazione grafica tridimensionale dei due spazi di colore [36].

Per concludere sono stati generati anche gli istogrammi senza l'applicazione delle maschere. Questo per accertarsi che l'algoritmo di classificazione eventualmente abbia successo anche senza di esse, considerando che il rilevamento della maschera impiega il suo tempo e non sempre va a buon fine.

Il modo in cui i valori calcolati sono passati alle reti è accostando i risultati dei tre canali uno di fianco all'altro, di conseguenza il numero di *feature* è uguale a  $n^{o} bin \cdot 3$ .

```
È riportato di seguito un estratto dal training-set hsv_8bin_masked.txt:<sup>6</sup>
```

#### 3.7.2 Grado di maturazione

Il gradi di maturazione delle olive, ovvero le label del training-set, sono stati manualmente scelti in base alle seguenti gradazioni presenti in figura:



Figura 3.15 – I cinque gradi di maturazione di riferimento utilizzati nel progetto.

I valori da sinistra sono 1, 2, 3, 4 e 5. Tuttavia in molte immagini è difficile distinguere il grado 1 dal 2 o il 4 dal 5, per questo, anche per una questione di semplificazione dell'architettura delle reti, le gradazioni sono state anche convertite con il seguente formato: 1, 2 = 1; 3 = 2; 4, 5 = 3.

Allo stesso modo dei valori di input le label sono stati salvati in semplice formato testuale per essere facilmente caricati con Numpy.<sup>7</sup>

<sup>&</sup>lt;sup>6</sup>Ogni riga corrisponde ad una oliva

<sup>&</sup>lt;sup>7</sup>Una label per riga, inoltre la conversione in tre classi è eseguita durante il caricamento.

#### 3.8 Modelli di classificazione con Scikit-Learn

Questo importante package per *Python* mette a disposizione molteplici modelli di *machine learning* e tantissime funzioni utili per l'ambito del *training* e della valutazione dei risultati.

Per lo scopo di questa tesi sono stati utilizzati i seguenti modelli basati sull'allenamento supervisionato:

#### • K-Nearest Neighbors:

Anche conosciuto semplicemente come K-NN, è il più semplice algoritmo di riconoscimento di pattern per la classificazione di oggetti. Il funzionamento si basa sul parametro k; infatti per classificare un oggetto controlla quale classe sia la maggioranza fra i k oggetti più vicini ad esso [37].

#### • Linear SVM:

Il Support Vector Machine è un modello che mappa i dati di training come punti in uno spazio multidimensionale e cerca di stabilire, ad esempio nel caso più semplice di due classi di oggetti, l'iperpiano che separa al meglio i due gruppi di dati. La classificazione di nuovi elementi avviene verificando in quale parte dello spazio diviso dall'iperpiano, o dagli iperpiani, essi si trovano [38].

#### • Albero di decisione:

È un modello predittivo rappresentato da una struttura ad albero dove i nodi rappresentano diverse variabili, i percorsi ai nodi figli possibili valori per essi, mentre le foglie corrispondo alla classe predetta per l'oggetto. La struttura viene creata tramite il training set e la scelta manuale di alcuni parametri come la profondità (altezza) massima dell'albero [39].

#### • Foresta casuale:

Metodo di classificazione o regressione basato sulla media delle predizioni di molteplici alberi di decisione generati casualmente. Questo metodo ha meno possibilità di *overfitting*<sup>8</sup> rispetto all'algoritmo usato per il singolo albero di decisione [40].

#### • Multi Layer Perceptron:

L'MLP è uno dei più conosciuti modelli di machine learning. È formato da diversi livelli costituiti da un numero variabile di nodi (neuroni). Il primo livello di input ha un numero di nodi pari alle feauture del dataset, l'ultimo pari al numero di classi del dataset. I neuroni di un livello sono collegati a tutti quelli del livello successivo tramite connessioni pesate. L'apprendimento avviene tramite la discesa del gradiente per trovare i minimi locali [41].

#### • AdaBoost:

Abbreviamento di Adaptive Boost, è un algoritmo che si appoggia ai precedenti modelli di machine learning, migliorando il loro algoritmo di apprendimento focalizzandosi sugli errori e i casi più difficili di classificazione.

 $<sup>^{8}</sup>$ Quando un modello di machine learning è specializzato solo nel classificare i dati del training set e non riesce dunque a generalizzare e classificare nuovi elementi.

Per questa tesi il modello di base utilizzato da AdaBoost è l'albero di decisione [42].

#### 3.8.1 Implementazione

Verranno ora illustrati parti del codice che gestiscono il dataset e l'allenamento dei modelli.

Un metodo per ottenere un gruppo per l'allenamento e un gruppo per la valutazione dallo stesso dataset è la funzione *Kfold* [43], che suddivide il dataset in n parti di cui n - 1 suddivisioni vengono utilizzati come training set mentre il rimanente come test set. L'operazione è gestita con la seguente funzione in **utils.py**:

```
1 from sklearn.model_selection import KFold
2 import numpy as np
3
4 def split_data(x, y):
5      kf = KFold(n_splits=5, shuffle=True, random_state=3)
6      for train_index, test_index in kf.split(x):
7            x_train, x_test = x[train_index], x[test_index]
8            y_train, y_test = y[train_index], y[test_index]
9      return x_train, x_test, y_train, y_test
```

I parametri x e y sono rispettivamente l'input, ovvero gli istogrammi di tutte le olive, e l'output, cioè le label dei gradi di maturazione associati.

Successivamente nella funzione test\_classifiers() in main.py viene eseguito il *fitting* e la valutazione dei sei modelli scelti:

```
1 from sklearn import tree
2 from sklearn.ensemble import RandomForestClassifier, AdaBoostClassifier
3 from sklearn.svm import SVC
4 from sklearn.neural_network import MLPClassifier
5 from sklearn.neighbors import KNeighborsClassifier
6
7
8
      classifiers = [
              KNeighborsClassifier(3),
9
10
              SVC(kernel="linear", C=0.025),
              tree.DecisionTreeClassifier(),
11
              RandomForestClassifier(n_estimators=200),
12
              MLPClassifier(alpha=1, max_iter=2000),
13
              AdaBoostClassifier()]
14
          names = ["Nearest Neighbors", "Linear SVM", "Decision Tree", "Random
15
      Forest", "Neural Net", "AdaBoost"]
          for name, clf in zip(names, classifiers):
16
17
              clf = clf.fit(x_train, y_train)
              y_pred = clf.predict(x_test)
18
19
```

Una volta inizializzati ciascun classificatore viene allenato tramite la funzione clf.fit(x\_train, y\_train) e i risultati delle predizioni vengono ottenuti tramite clf.predict(x\_test).

I risultati verranno discussi nel prossimo capitolo.

#### 3.9 Sperimentazione Transfer Learning con ResNet

Il *transfer learning* è un approccio di machine learning molto popolare che utilizza modelli pre-allenati come punto di partenza per creare una nuova rete ottimizzata per la classificazione o la regressione di una diversa popolazione di elementi. In questo modo si evita di creare un nuovo modello *ad-hoc* da zero e si riducono di gran lunga i tempi di apprendimento [44].

#### 3.9.1 Il modello ResNet-18

ResNet è un *Residual Neural Network*, una rete neurale convoluzionale profonda 152 livelli.

A differenza di altri modelli ResNet non soffre della cosiddetta *scomparsa del gradiente*, problema molto comune per le reti neurali profonde dove il continuo moltiplicarsi del gradiente durante il processo di *backpropagation* lo potrebbe rendere infinitivamente piccolo e di conseguenza rallentare l'apprendimento. Ciò è possibile grazie alla tecnica dell'*identity shortcut connection* che permette di saltare uno o più livelli durante la retropropagazione [45].

Per il progetto è stato utilizzato ResNet-18, profondo 18 livelli ed allenato sul dataset *ImageNet*.

Layer Name	Output Size	ResNet-18
conv1	$112\times112\times64$	$7 \times 7$ , 64, stride 2
		$3 \times 3$ max pool, stride 2
conv2_x	$56 \times 56 \times 64$	$\left[\begin{array}{c} 3\times3,64\\ 3\times3,64\end{array}\right]\times2$
conv3_x	28  imes 28  imes 128	$\left[\begin{array}{c} 3 \times 3, 128\\ 3 \times 3, 128 \end{array}\right] \times 2$
conv4_x	$14\times14\times256$	$\left[\begin{array}{c} 3 \times 3,256\\ 3 \times 3,256 \end{array}\right] \times 2$
conv5_x	$7\times7\times512$	$\left[\begin{array}{c} 3 \times 3,512\\ 3 \times 3,512 \end{array}\right] \times 2$
average pool	$1\times1\times512$	$7 \times 7$ average pool
fully connected	1000	$512 \times 1000$ fully connections
softmax	1000	

Figura 3.16 – Schema dell'architettura di ResNet-18 [46].

#### 3.9.2 Implementazione

Per lo sviluppo è stata seguita la guida sul *Transfer Learning* disponibile al link https://pytorch.org/tutorials/beginner/transfer\\_learning\\_tutorial.html.

A differenza dei modelli utilizzati con Scikit-learn , *ResNet-18* essendo una rete convoluzionale non riceve come input i valori d'istogramma ma direttamente le immagini. Saranno i vari livelli di convoluzione e pooling a trasformare gradualmente l'immagine in una quantità di dati gestibile dagli ultimi livelli che sono

completamente connessi come le normali reti neurali. A tal proposito sono state salvate durante il processo di rilevamento della maschera le coppie di immagini con e senza maschera<sup>9</sup> di tutte le olive del dataset. Le immagini mascherate hanno inoltre lo sfondo nero.



Figura 3.17 – Esempio delle foto salvate con e senza maschera applicata

Allo stesso tempo sono state smistate in cartelle rappresentanti i loro gradi di maturazione e suddivise in due gruppi, uno per l'allenamento ed uno per la valutazione. Questa gerarchia permette l'immediato caricamento delle immagini con le label associate tramite le funzioni di caricamento dei dataset di *Pytorch*.

```
from torchvision import datasets, models, transforms
2
  import torch
3 import os
5 data_transforms = {
       ˈtrain': transforms.Compose([
6
7
           transforms.RandomResizedCrop(224),
           transforms.RandomHorizontalFlip(),
8
9
           transforms.ToTensor(),
           transforms.Normalize([0.485, 0.456, 0.406], [0.229, 0.224, 0.225])
10
11
      ]).
       'val': transforms.Compose([
12
           transforms.Resize(256)
13
           transforms.CenterCrop(224)
14
           transforms.ToTensor(),
15
           transforms.Normalize([0.485, 0.456, 0.406], [0.229, 0.224, 0.225])
16
17
      ]),
18 }
19
20 # Dataset location
21 data_dir = '/content/drive/My Drive/Colab Notebooks/Olives/mask5'
22 image_datasets = {x: datasets.ImageFolder(os.path.join(data_dir, x),
       data_transforms[x]) for x in ['train', 'val']}
23
24 dataloaders = {x: torch.utils.data.DataLoader(image_datasets[x], batch_size=4,
        shuffle=True, num_workers=4)
```

È importante tenere presente che nella riga 5 viene creato un dizionario contenente le trasformazioni da eseguire all'atto del caricamento delle immagini rispettivamente per il set di training e di valutazione. Per il primo viene effettuato un processo di *data augmentation* ridimensionando e ribaltando le immagini orizzontalmente in modo casuale. Mentre per entrambi i set le immagini sono ritagliate alle dimensioni  $224 \times 224$  richieste da ResNet, e i canali del colore normalizzati sulle medie e deviazioni standard di quest'ultimi.

 $<sup>^9\</sup>mathrm{Ovviamente}$ l'immagine con maschera applicata è salvata solo se il rilevamento va a buon fine



Figura 3.18 – Esempi di data augmentation applicato al training set

Infine, se si decide di utilizzare label numeriche, per evitare errori di esecuzione con l'accelerazione GPU è importante far partire le label da 0. Infatti, da come si può notare in figura 3.19, i gradi di maturazione sono stati scalati di un'unità.



Figura 3.19 – Gerarchia delle cartelle utilizzate dal dataset.

Senza soffermarsi troppo sulla funzione di allenamento in se, i passi importanti per caricare ed ottimizzare il modello *ResNet-18* sono i seguenti:

```
import torch.nn as nn
import torch.optim as optim
from torch.optim import lr_scheduler
from torchvision import models
6 model_ft = models.resnet18(pretrained=True)
7 num_ftrs = model_ft.fc.in_features
```

```
8 # Si imposta la dimensione d'output della rete in base al numero di classi
  model_ft.fc = nn.Linear(num_ftrs, len(class_names))
9
10
11 model_ft = model_ft.to(device)
12
13 criterion = nn.CrossEntropyLoss()
14
15 optimizer_ft = optim.SGD(model_ft.parameters(), lr=0.001, momentum=0.9)
16
17 # Riduce il tasso di apprendimento di un fattore di 0.1 ogni 7 epoche
18 exp_lr_scheduler = lr_scheduler.StepLR(optimizer_ft, step_size=7, gamma=0.1)
19
20 model_ft = train_model(model_ft, criterion, optimizer_ft, exp_lr_scheduler,
                          num_epochs=25)
21
```

Il modello viene caricato pre-allenato.

Viene lasciata invariata la dimensione d'input mentre quella d'output viene impostata al numero di classi del dataset utilizzato.

A riga **11** il modello viene regolato per la corretta esecuzione sul dispositivo scelto (GPU o CPU).

Successivamente vengono impostati i parametri principali per l'allenamento:

- la funzione di costo **criterion** che per questo progetto è una combinazione della trasformazione logaritmica della funzione di verosomiglianza [47] con la funzione di attivazione *LogSoftmax*.
- l'algoritmo di ottimizzazione optimizer\_ft che in questo caso è la discesa stocastica del gradiente [48].
- lo scheduler exp\_lr\_scheduler che riduce il tasso di apprendimento di un fattore 0.1 ogni 7 epoche.

Il modello viene allenato per 25 epoche.

Per questo progetto verrà saltata la fase di allenamento con blocco dei parametri dei livelli di *convoluzione* e *pooling*, dedicati all'estrazione delle *feature*, permettendo solo la l'alterazione dei pesi del livello finale.

Questa fase normalmente viene effettuata al termine del processo di *transfer learning*, ma in questo caso non garantirebbe miglioramenti rilevanti perché il dataset non è abbastanza grande e vario.

### Capitolo 4

## Risultati

In questo capitolo verranno mostrati ed analizzati i risultati dei rilevamenti del processo di dataset pre-processing e delle predizioni dei diversi modelli di classificazione adottati.

Per l'analisi si valuteranno tempo di esecuzione, tasso di successo e per le reti matrici di confusione e l'F1 Score.

#### 4.1 Hardware utilizzato

Il progetto Pycharm è stato eseguito su un computer Windows 7 64 bit con processore AMD FX-8350 8 GB di RAM con la versione 64 bit di Python.

Le informazioni sulla GPU utilizzata dal runtime del progetto *Google Colaboratory* non sono normalmente reperibili, è nota solo la quantità di ram che corrisponde a 12 GB.

#### 4.2 Rilevamento maschere

Durante il processo di generazione delle maschere tramite la funzione spiegata nel capitolo 3.6 è stato scritto un file di log per determinare quali olive sono state elaborate correttamente, quale metodo di rilevamento maschera è stato usato e il tempo di esecuzione totale dell'immagine intera.

Sono state rilevate **425** maschere su **471** olive ovvero il **90,23** %, mentre il tempo di esecuzione totale con l'hardware sopra specificato è pari a **9779** secondi con un tempo medio di **20,76** secondi ad oliva.

#### 4.2.1 Metodi di rilevamento

Andando a verificare quali metodi hanno avuto più successo si può subito notare che la maggioranza è stata rilevata da *Mask R-CNN*, il che è normale essendo un algoritmo di *machine learning*. Inoltre l'equalizzazione d'istogramma ha aiutato nel **20,94** % dei casi.

La seguente tabella mostra i risultati in dettaglio con percentuale calcolata sul numero totale di successi:

METODO	HISTEQUALIZE	OCCORRENZE	PERCENTUALE
MATLAB	NO	97	22.82~%
MATLAB	SI	34	8.00 %
Mask R-CNN	NO	239	56.24~%
Mask R-CNN	SI	55	12.94 %

Tabella 4.1 – Percentuale dei metodi di rilevamento delle maschere.

Per quanto riguarda le altre due trasformazioni utilizzate per migliorare il tasso di successo dei rilevamenti, ovvero il filtro di nitidezza e il raddoppiamento delle dimensioni, i risultati sono riportati nella tabella 4.2. Le righe rappresentano le combinazioni utilizzate rispettivamente dai tre cicli dell'algoritmo.

NITIDEZZA	DIM. 2X	OCCORRENZE	PERCENTUALE
NO	SI	380	89.41~%
SI	SI	26	6.12~%
SI	NO	19	4.47~%

Tabella 4.2 – Percentuale di applicazione del filtro di nitidezza e ridimensionamento 2X.

#### 4.2.2 Analisi qualitativa degli insuccessi

I casi di insuccesso sono dovuti il più delle volte al fatto che l'area di interesse contenente l'oliva è troppo piccola. In queste circostanze raddoppiare le dimensioni ed aumentare la nitidezza risulta essere insufficiente per mettere in risalto i contorni.

Sono riportati qui sotto alcuni esempi e le loro dimensioni originali.



Figura 4.1 – Immagini troppo piccole e perciò con contorni troppo vaghi per il rilevamento.

Fra gli altri motivi ci sono anche la presenza di troppe foglie o rami che coprono la sagoma dell'oliva, come ad esempio nella figura 4.2(a); oppure molte ombre che nascondono e rendono indistinguibili i bordi come nel caso 4.2(b).



Figura 4.2 – Esempi di olive coperte da foglie e ombre

Questi insuccessi sono in parte giustificati dal fatto che *Mask R-CNN* è allenato solo sul dataset *COCO*, che non possiede molti esempi di frutti celati dietro il fogliame.

L'algoritmo di MATLAB invece risulta più efficace su forme con contorni ben definiti e fa fatica a riconoscere sagome irregolari o interrotte troppo spesso.

#### 4.3 Scikit-Learn

Per stabilire l'accuratezza di un algoritmo di *machine learning* si utilizzano spesso due particolari metodi di valutazione:

#### 1. Matrice di confusione:

Anche detta tabella di errata classificazione, è una matrice quadrata in cui il valore di una cella ij rappresenta il numero di volte che un elemento della classe i è stato classificato come appartenente alla classe j. In termini più semplici le righe rappresentano le classi vere mentre le colonne le classi predette e la diagonale rappresenta le predizioni corrette. I valori sono inoltre spesso normalizzati in base al numero reale di elementi appartenenti alla classe, di conseguenza più si avvicinano ad 1 i valori lungo la diagonale più si può ritenere accurato il modello [49].

#### 2. F1 SCORE:

L'F-SCORE o F measure è una misura dell'accuratezza di un modello [50]. La formula generale è la seguente:

$$F = (1 + \beta^2) \cdot \frac{p \cdot r}{(\beta^2 \cdot p) + r}$$

Dove p è la precisione, ovvero il rapporto tra i veri positivi (predizioni corrette) e tutti i positivi predetti; r il recupero, definito come rapporto tra i veri positivi e tutti gli elementi che sarebbero dovuti risultare positivi:

$$p = \frac{VP}{VP + FP}; \quad r = \frac{VP}{VP + FN}$$

 $\beta$  è un parametro che enfatizza o attenua l'influenza dei falsi negativi. In particolare la misura bilanciata è data dall'**F1 score** dove  $\beta = 1$ :

$$F_1 = 2 \cdot \frac{p \cdot r}{p+r}$$

Per il calcolo e la visualizzazione di queste metriche sono state utilizzate funzioni del package *Scikit-learn*.

Dalla combinazione delle 24 varianti del dataset di istogrammi con i 6 modelli scelti sono stati ricavati 144 risultati differenti. Per un'analisi più chiara verranno suddivise nelle due categorie a 5 e 3 gradi di maturazione.

#### 4.3.1 Classificazione con 5 gradi

Sono riportati in ordine di punteggio, calcolato come media pesata dei punteggi delle 5 classi, le prime 10 combinazioni di modelli e dataset più accurate.

Classificatore	Bins	Maschera	Colori	F1 1	F1 2	F1 3	F1 4	F1 5	Media
Nearest Neighbors	32	SI	HSV	0.951	0.824	0.714	0.848	0.917	0.889
Neural Net	32	SI	HSV	0.950	0.800	0.706	0.774	0.818	0.857
Linear SVM	32	SI	HSV	0.935	0.783	0.667	0.667	0.818	0.823
Nearest Neighbors	16	NO	HSV	0.892	0.588	0.588	0.894	0.833	0.820
Nearest Neighbors	16	SI	HSV	0.907	0.571	0.714	0.800	0.846	0.820
Nearest Neighbors	32	NO	HSV	0.902	0.471	0.667	0.870	0.800	0.810
Nearest Neighbors	8	NO	HSV	0.860	0.167	0.706	0.958	0.960	0.808
Linear SVM	32	NO	HSV	0.895	0.545	0.636	0.810	0.846	0.802
Linear SVM	16	SI	HSV	0.916	0.625	0.778	0.667	0.783	0.801
Linear SVM	32	SI	RGB	0.935	0.667	0.875	0.684	0.556	0.799

Tabella 4.3 – I10 migliori risultati con 5 classi, in grassetto i punteggi migliori assoluti.

Dalla tabella si può constatare che i punteggi singoli per le classi 2 e 3 sono relativamente bassi, tuttavia essendo la quantità di olive con questi gradi di maturazione molto minore rispetto alle altre la media ponderata non è eccessivamente influenzata.

Si può notare inoltre come la maggior parte dei modelli più performanti utilizzino lo spazio di colore HSV, che rispetto al RGB varia di meno quando ci sono differenze di ombre e luci.

I modelli migliori per le singole classi sono invece i seguenti:

Classe	Classificatore	Bins	Maschera	Colori	F1 SCORE
1	Nearest Neighbors	32	SI	HSV	0.951
2	Nearest Neighbors	32	SI	HSV	0.824
3	Linear SVM	8	SI	RGB	0.933
4	Nearest Neighbors	8	NO	HSV	0.958
5	Nearest Neighbors	8	NO	HSV	0.960

Tabella 4.4 – I migliori risultati per ciascuna delle 5 classi.

Ad eccezione della classe 3 che ha il minor numero di immagini, i punteggi più alti per la prima e la seconda classe sono detenuti dal modello e dal dataset migliori, e la quarta e la quinta dal modello in settima posizione.

Per una rappresentazione grafica più intuitiva dell'accuratezza sono mostrate alcune matrici di confusione, in particolare delle voci 1, 24, 48 e 72 in modo da avere un ampia visione su tutti i punteggi.





(a) 32 bins, con maschera, HSV, F1 0.889

(b) 8 bins, con maschera, HSV, F1 0.754



(c) 16 bins, senza maschera, HSV, F1 0.657

(d) 8 bins, con maschera, RGB, F1 0.388

Figura 4.3 – Alcune matrici di confusione dei modelli a 5 classi.

#### 4.3.2 Classificazione con 3 gradi

Nel caso di soli 3 gradi di maturazione i risultati sono naturalmente più accurati.

Classificatore	Bins	Maschera	Colori	F1 1	F1 2	F1 3	Media
Neural Net	8	SI	HSV	0.990	0.875	0.945	0.964
Linear SVM	16	SI	HSV	0.990	0.824	0.926	0.953
Linear SVM	8	SI	RGB	0.961	0.857	0.963	0.952
Nearest Neighbors	32	SI	HSV	0.990	0.714	0.947	0.950
Linear SVM	32	NO	RGB	0.958	0.762	0.986	0.948
Neural Net	32	NO	HSV	0.959	0.800	0.971	0.947
Nearest Neighbors	8	NO	HSV	0.969	0.706	0.973	0.943
Linear SVM	32	SI	RGB	0.959	0.824	0.945	0.942
Random Forest	8	SI	HSV	0.979	0.800	0.915	0.941
Linear SVM	16	SI	RGB	0.950	0.800	0.963	0.940

Tabella 4.5 – I 10 migliori risultati con 3 classi, in grassetto i punteggi migliori assoluti.

Nell'immediato è chiaro che in confronto ai test con 5 classi le medie F1 differiscono dall'un l'altro solo di pochi decimi. Stesso vale per i punteggi singoli il cui valore minimo fra queste prime 10 voci è 0.706 a differenza del 0.167 del caso con più stadi di maturazione. Inoltre i dataset a 8 bin, quindi con 24 input, e spazio di colore RGB sono più frequenti fra i punteggi alti.

Come in precedenza sono riportati i migliori risultati per classe ed alcune matrici di confusione.

Classe	Classificatore	Bins	Maschera	Colori	F1 SCORE
1	Neural Net	8	NO	HSV	1.000
2	Neural Net	8	SI	HSV	0.875
3	Linear SVM	32	NO	RGB	0.986

Tabella 4.6 – I migliori risultati per ciascuna delle 3 classi.





(a) 8 bins, con maschera, HSV, F1 0.964

(b) 16 bins, senza maschera, HSV, F1 0.920



(c) 8 bins, senza maschera, RGB, F1 0.874 (d) 32 bins, con maschera, RGB, F1 0.781

Figura 4.4 – Alcune matrici di confusione dei modelli a 3 classi.

#### 4.3.3 Considerazioni

Dai precedenti risultati è facile appurare che il dataset a 3 gradi di maturazione è più semplice da apprendere e predire, con un punteggio F1 massimo di 0.964 e un minimo di 0.781. Al contrario con 5 classi si ha una maggiore distinzione fra i gradi di maturazione ma l'accuratezza è molto minore e i punteggi si estendono in un range più ampio con un massimo e un minimo di 0.889 e 0.388. Tuttavia i risultati possono migliorare ampliando il dataset, specialmente per il secondo e terzo grado.

Un aspetto molto interessante è il fatto che gli istogrammi calcolati senza l'ausilio della maschera hanno dimostrato di essere perfettamente validi per buone classificazioni, e solo leggermente inferiori alle varianti con maschera.

#### 4.4 ResNet-18

Anche per la valutazione di ResNet sono state utilizzate le matrici di confusione e il punteggio F1 ma è interessante osservare prima alcuni esempi di predizione.



Figura 4.5 – Predizioni con 5 classi, senza e con maschera.

Da come si può vedere *ResNet-18* è in grado di classificare correttamente anche le olive coperte da foglie o ombre prominenti e la sfocatura non influisce in nessun modo sulle predizioni.

Di seguito alcuni esempi con sole 3 classi.



Figura 4.6 – Predizioni con 3 classi, senza e con maschera.

DATASET	F1 1	F1 2	F1 3	F1 4	F1 5	MEDI
5 classi	0.966	0.824	0.818	0.88	0.947	0.917
5 classi - maschera	0.953	0.8	0.823	0.852	0.882	0.893
3 classi	0.980	0.8	0.951			0.950
3 classi - maschera	0.978	0.727	0.952			0.941

I punteggi F1 sono i seguenti:

Tabella 4.7 – F1 score di ResNet-18 sui vari dataset.

I punteggi per i dataset a 5 gradi di maturazione sono più alti rispetto ai comuni modelli non *deep learning*. Questo aumento si deve sopratutto ad un aumento nell'accuratezza per i gradi intermedi da come si può verificare dai punteggi singoli e dalle matrici di confusione nella figura 4.7.

Mentre con 3 gradi gli F1 score sono per lo più paragonabili ai punteggi migliori dei modelli non convoluzionali.

In generale vi è sicuramente margine di miglioramento, considerando che il dataset utilizzato è abbastanza piccolo e non permette dunque un esecuzione ottimale della fase finale di allenamento spiegata nel capitolo 3.9.2.



Figura 4.7 – Le matrici di confusione ottenute dai risultati di ResNet-18.

## Capitolo 5

## Conclusioni e sviluppi futuri

L'obiettivo della tesi è quindi raggiunto; i modelli di *machine learning* standard e il modello convoluzionale di *deep learning* sono in grado di classificare con una buona accuratezza le maturazioni delle olive.

Le reti con le migliori prestazioni sono state il *Multi-layer perceptron*, *Linear* SVM e ResNet-18, sebbene abbiano ancora margine di miglioramento.

La fase intermedia di rimozione del superfluo dalle immagini delle olive si è rivelata la parte più complessa del progetto. Questa può essere ottimizzata ulteriormente permettendo quindi la creazione di un dataset più grande con maschere più pulite da utilizzare per l'allenamento dei modelli. Nonostante i problemi della suddetta fase, i risultati hanno mostrato come non sia strettamente necessario separare il frutto dallo sfondo per ottenere buone predizioni, ma questo potrà essere confermato in revisioni future con il perfezionamento delle tecnologie o l'utilizzo di nuove.

#### 5.1 Soluzioni ed applicazioni

Fra le soluzioni possibili che si possono adottare per migliorare la resa dell'algoritmo le più importanti sono:

- 1. Ampliare il dataset considerevolmente con immagini di buone dimensioni e medesimo numero di olive per grado di maturazione. Pertanto i modelli possono essere allenati più a lungo senza rischio di *overfitting* o *bias* verso un grado specifico.
- 2. Conversione dell'algoritmo di rilevamento ellissi da MATLAB a Python. Ciò permette di fare a meno del matlab-engine che oltre ad impiegare molto tempo al caricamento del codice impedisce di condividere facilmente il codice o di eseguirlo ad esempio in servizi online come Jupiter Notebook. Questo perché è un package legato alla propria licenza di MATLAB e che non può essere ottenuto in versione standalone.
- 3. Allenare Mask R-CNN su esempi specifici di olive. Così facendo aumenterà il tasso di rilevamenti, specialmente per le olive sfocate o parzialmente coperte, e si potrà utilizzare la rete per ottenere direttamente sia i bounding box che le maschere, e quest'ultime tra l'altro saranno più pulite e

consistenti. In seguito si potranno escludere dal ciclo di rilevamento della maschera tutti gli altri algoritmi riducendo nettamente i tempi di esecuzione e rendendo più compatto il codice. Tuttavia la creazione di questo dataset di allenamento è un lavoro abbastanza laborioso.

4. Utilizzare modelli per la regressione. A differenza della classificazione che consiste nell'etichettare gli input con un numero discreto di classi, la regressione invece punta ad associare valori reali continui approssimando la funzione che lega la caratteristica studiata alla popolazione [51]. In questo modo si potrebbe stabilire il grado di maturazione con valori percentuali da 0 a 100, fornendo una descrizione più precisa dell'oliva.

Come nel caso del progetto dell'azienda *SeeTree* [4] esposto nell'introduzione, una volta effettuate tutte le ottimizzazioni necessarie il software potrà essere installato ad esempio in un sistema centralizzato con a disposizione droni per la cattura delle immagini, e provvedere al calcolo della media delle maturazione di tutti gli alberi del campo, aiutando così a stabilire il periodo ottimale per la raccolta.



Figura 5.1 – Drone utilizzato per stabilire l'impatto della potatura sulla salute degli ulivi [52].

## Bibliografia

- [1] Verónica Saiz-Rubio and Francisco Rovira-Más. From smart farming towards agriculture 5.0: A review on crop data management. Agronomy, 10(2), 2020.
- Jeremy Hsu. Ai detects papaya ripeness. https://spectrum.ieee.org/ tech-talk/robotics/artificial-intelligence/ai-detects-papaya-ripeness, March 18, 2018.
- [3] Nashwa El-Bendary, Esraa El-hariri, Aboul Ella Hassanien, and Amr Badr. Using machine learning techniques for evaluating tomato ripeness. *Expert Systems with Applications*, 42, 10 2014.
- [4] CIKLUM. See tree | ai prototype for ripe fruit detection. https://www.ciklum. com/case-studies/seetree/.
- [5] Joseph Redmon. Darknet convolutional neural networks. https://github.com/ pjreddie/darknet.
- [6] Jetbrains. Pycharm ide. https://www.jetbrains.com/pycharm/.
- [7] Google. Colaboratory. https://colab.research.google.com/.
- [8] S. van der Walt, S. C. Colbert, and G. Varoquaux. The numpy array: A structure for efficient numerical computation. *Computing in Science Engineering*, 13(2):22– 30, 2011.
- [9] G. Bradski. The OpenCV Library. Dr. Dobb's Journal of Software Tools, 2000.
- [10] Stéfan van der Walt, Johannes L. Schönberger, Juan Nunez-Iglesias, François Boulogne, Joshua D. Warner, Neil Yager, Emmanuelle Gouillart, Tony Yu, and the scikit-image contributors. scikit-image: image processing in Python. *PeerJ*, 2:e453, 6 2014.
- [11] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.
- [12] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, Advances in Neural Information Processing Systems 32, pages 8024–8035. Curran Associates, Inc., 2019.
- [13] J. D. Hunter. Matplotlib: A 2d graphics environment. Computing in Science & Engineering, 9(3):90–95, 2007.

- [14] Pauli Virtanen, Ralf Gommers, Travis E. Oliphant, Matt Haberland, Tyler Reddy, David Cournapeau, Evgeni Burovski, Pearu Peterson, Warren Weckesser, Jonathan Bright, Stéfan J. van der Walt, Matthew Brett, Joshua Wilson, K. Jarrod Millman, Nikolay Mayorov, Andrew R. J. Nelson, Eric Jones, Robert Kern, Eric Larson, C J Carey, İlhan Polat, Yu Feng, Eric W. Moore, Jake VanderPlas, Denis Laxalde, Josef Perktold, Robert Cimrman, Ian Henriksen, E. A. Quintero, Charles R. Harris, Anne M. Archibald, Antônio H. Ribeiro, Fabian Pedregosa, Paul van Mulbregt, and SciPy 1.0 Contributors. SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python. Nature Methods, 17:261–272, 2020.
- [15] MATLAB. 9.8.0.1396136 (R2020a) Update 3. The MathWorks Inc., Natick, Massachusetts, 2020.
- [16] Alexander Mordvintsev and Abid K. Image thresholding. https: //opencv-python-tutroals.readthedocs.io/en/latest/py\_tutorials/py\_ imgproc/py\_thresholding/py\_thresholding.html.
- [17] Wikipedia Contributors. Otsu's method. https://en.wikipedia.org/wiki/Otsu% 27s\_method.
- [18] Wikipedia Contributors. Gaussian blur. https://en.wikipedia.org/wiki/ Gaussian\_blur.
- [19] Wikipedia Contributors. Canny edge detector. https://en.wikipedia.org/wiki/ Canny\_edge\_detector.
- [20] OpenCV Dev Team. Feautre detection. https://docs.opencv.org/2.4/modules/ imgproc/doc/feature\_detection.html?highlight=canny.
- [21] Adrian Rosebrock. Zero-parameter, automatic canny edge detection with python and opency. https://www.pyimagesearch.com/2015/04/06/ zero-parameter-automatic-canny-edge-detection-with-python-and-opency/, April 6, 2015.
- [22] Scikit image Dev Team. Circular and elliptical hough transforms. https://scikit-image.org/docs/stable/auto\_examples/edges/plot\_ circular\_elliptical\_hough\_transform.html.
- [23] Scikit image Dev Team. Module:draw. https://scikit-image.org/docs/stable/ api/skimage.draw.html#ellipse.
- [24] Changsheng Lu. High-quality ellipse detection. https://github.com/AlanLuSun/ High-quality-ellipse-detection.
- [25] Changsheng Lu, Siyu Xia, Ming Shao, and Yun Fu. Arc-support line segments revisited: An efficient high-quality ellipse detection. *IEEE Transactions on Image Processing*, 29:768–781, 2019.
- [26] Waleed Abdulla. Mask r-cnn for object detection and instance segmentation on keras and tensorflow. https://github.com/matterport/Mask\_RCNN, 2017.
- [27] Tsung-Yi Lin, Michael Maire, Serge Belongie, Lubomir Bourdev, Ross Girshick, James Hays, Pietro Perona, Deva Ramanan, C. Lawrence Zitnick, and Piotr Dollár. Microsoft COCO: Common Objects in Context. arXiv e-prints, page arXiv:1405.0312, May 2014.
- [28] Adrian Rosebrock. Mask r-cnn with opencv. https://www.pyimagesearch.com/ 2018/11/19/mask-r-cnn-with-opencv/, November 2019.
- [29] Wikipedia Contributors. Blob detection. https://en.wikipedia.org/wiki/Blob\_ detection.

- [30] Wikipedia Contributors. Convolutional neural network pooling. https://en. wikipedia.org/wiki/Convolutional\_neural\_network#Pooling.
- [31] Wikipedia Contributors. Lanczos resampling. https://en.wikipedia.org/wiki/ Lanczos\_resampling.
- [32] Wikipedia Contributors. Matrice di convoluzione. https://it.wikipedia.org/ wiki/Matrice\_di\_convoluzione.
- [33] OpenCV Dev Team. Image filtering filter2d. https://docs.opencv.org/2.4/ modules/imgproc/doc/filtering.html#filter2d.
- [34] OpenCV Dev Team. Histogram equalization. https://docs.opencv.org/2.4/ modules/imgproc/doc/filtering.html#filter2d.
- [35] Alexander Mordvintsev and Abid K. Histograms 1:find, plot, analyze!!! https://opencv-python-tutroals.readthedocs.io/en/latest/py\_tutorials/ py\_imgproc/py\_histograms/py\_histogram\_begins/py\_histogram\_begins.html# histogram-calculation-in-opencv.
- [36] Wikipedia Contributors. Hsl and hsv. https://en.wikipedia.org/wiki/HSL\_and\_ HSV.
- [37] Wikipedia Contributors. K-nearest neighbors. https://it.wikipedia.org/wiki/ K-nearest\_neighbors.
- [38] Lorenzo Govoni. Algoritmo support vecotr machine. https://lorenzogovoni. com/support-vector-machine/.
- [39] Wikipedia Contributors. Albero di decisione. https://it.wikipedia.org/wiki/ Albero\_di\_decisione.
- [40] Wikipedia Contributors. Foresta casuale. https://it.wikipedia.org/wiki/ Foresta\_casuale.
- [41] Wikipedia Contributors. Multilayer perceptron. https://en.wikipedia.org/ wiki/Multilayer\_perceptron.
- [42] Wikipedia Contributors. Adaboost. https://en.wikipedia.org/wiki/AdaBoost.
- [43] Scikit learn Dev Team. Kfold. https://scikit-learn.org/stable/modules/ generated/sklearn.model\_selection.KFold.html.
- [44] Jason Brownlee. A gentle introduction to transfer learning. Machine Learning Mastery, December 2017.
- [45] Jason Brownlee. An overview of resnet and its variants. Towards Data Science, December 2017.
- [46] Josh Varty. Visualizing resnet18 activation. https://forums.fast.ai/t/ visualizing-resnet18-activations/43916.
- [47] Wikipedia Contributors. Likelihood function. https://en.wikipedia.org/wiki/ Likelihood\_function#Log-likelihood.
- [48] Wikipedia Contributors. Discesa stocastica del gradiente. https://it.wikipedia. org/wiki/Discesa\_stocastica\_del\_gradiente.
- [49] Wikipedia Contributors. Matrice di confusione. https://it.wikipedia.org/ wiki/Matrice\_di\_confusione.
- [50] Wikipedia Contributors. F1 score. https://it.wikipedia.org/wiki/F1\_score.

- [51] Ml | classification vs regression. https://www.geeksforgeeks.org/ ml-classification-vs-regression/, December 2 2019.
- [52] Jiménez-Brenes, F.M., López-Granados, F.de Castro, and A.I. et al. Quantifying pruning impacts on olive tree architecture and annual canopy growth by using uav-based 3d modelling. *Plant Methods* 13, 55, July 06 2017.

## Elenco delle figure

1.1	L'algoritmo di SeeTree in azione, la maturazione è valutata in percen- tuale. [4].	5
1.2	Esempi di risultati previsti dall'estrazione delle olive dalle immagini del	-
1.3	dataset	$5 \\ 6$
3.1	Il Workflow del progetto, punto fulcro è la generazione del dataset	11
$3.2 \\ 3.3$	Risultato abbastanza pulito e preciso della soglia con metodo Otsu Un oliva matura potrebbe dare un risultato scarso, perché le ombre si	13
0.4	confondono con i bordi.	14
3.4	Alcuni risultati dell'algoritmo canny, anche in questo caso ci sono im-	14
35	Bisultate del rilevamente con Hough Ellipse riferite all'esempie 3 4(a)	14
3.6	Esempi di immagini maschere e sovrapposizione per constatare la pre-	10
0.0	cisione dell'algoritmo in MATLAB.	17
3.7	Esempio delle immagini e diverse sovrapposizioni generate da Mask R-CNN	18
3.8	In questa immagine viene rilevata anche l'oliva in alto a sinistra	19
3.9	La funzione di controllo esclude correttamente l'oliva nell'angolo. Il bor-	
	do netto è dovuto ad altro.	19
3.10	Esempio di filtro di nitidezza. La sfocatura è quasi del tutto rimossa	20
3.11	Immagine e istogramma originali in scala di grigi	21
3.12	Immagine e istogramma dopo l'equalizzazione. I valori sono ora distri-	~ 1
0 10	butti su tutto il range.	21
3.13	Istogrammi della stessa immagine con bin diversi	23
3.14	Rappresentazione grafica tridimensionale dei due spazi di colore [30]	23
3.10 2.16	I cinque gradi di maturazione di riterimento utilizzati nel progetto	24
3.10 3.17	Esempio della fata salvata con a sonza maschara applicata	21
3.17	Esemplo delle loto salvate coll è senza maschera applicata	$\frac{20}{20}$
3.10	Gerarchia delle cartelle utilizzate dal dataset	$\frac{29}{29}$
0.15		20
4.1	Immagini troppo piccole e perciò con contorni troppo vaghi per il rile-	
	vamento	32
4.2	Esempi di olive coperte da foglie e ombre	33
4.3	Alcune matrici di confusione dei modelli a 5 classi	35
4.4	Alcune matrici di confusione dei modelli a 3 classi.	36
4.5	Predizioni con 5 classi, senza e con maschera.	37
4.6	Predizioni con 3 classi, senza e con maschera.	37
4.7	Le matrici di confusione ottenute dai risultati di ResNet-18	38

5.1	Drone utilizzato per s	tabilire l'impatto	della potatura sulla	salute degli
	ulivi [52]			40

## Elenco delle tabelle

4.1	Percentuale dei metodi di rilevamento delle maschere	32
4.2	Percentuale di applicazione del filtro di nitidezza e ridimensionamento 2X.	32
4.3	I 10 migliori risultati con 5 classi, in grassetto i punteggi migliori assoluti.	34
4.4	I migliori risultati per ciascuna delle 5 classi.	34
4.5	I 10 migliori risultati con 3 classi, in grassetto i punteggi migliori assoluti.	35
4.6	I migliori risultati per ciascuna delle 3 classi.	36
4.7	F1 score di ResNet-18 sui vari dataset	37