

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 il riconoscimento dello stato di maturazione
di olive mediante immagini multi-spettrali***

*Design and development of a Deep Learning algorithm to evaluate the ripening stage of
olives from multispectral images*

Relatore:
DOTT. MANCINI ADRIANO

Laureanda:
PROIETTI MELISSA

ANNO ACCADEMICO 2020-2021

Indice

1	Introduzione	1
1.1	Agricoltura 4.0	2
1.2	Il progetto	3
1.3	Struttura della tesi	3
2	Strumenti e metodi utilizzati	5
2.1	Deep Learning e Reti Neurali	5
2.2	Reti Neurali artificiali	6
2.2.1	Principio di funzionamento	7
2.2.2	Error Back-Propagation	7
2.3	YOLOv3	8
2.3.1	Funzionamento di YOLO	9
2.3.2	Parametri di YOLO	10
2.3.3	Iperparametri di YOLO	14
2.4	Labelbox	15
2.5	Python	15
2.6	Google Colab	15
2.7	Kaggle	16
2.8	GitHub	16
3	Labeling	18
3.1	Labeling	18
3.2	Labelbox	18
3.3	Esportazione in formato JSON	18
3.4	Parsing	19
3.5	Preparazione del dataset	21
3.6	Setup e primo train di YOLOv3	21
3.7	Risultati	22
4	Data Augmentation	24
4.1	Rotazione	24
4.2	Blurring	26
4.3	Flipping Orizzontale	27
4.4	Flipping Verticale	28

5	Train con il nuovo dataset	31
5.1	Il nuovo dataset	31
5.2	Risultati	32
	Conclusioni e sviluppi futuri	37
	Bibliografia	38

Capitolo 1

Introduzione

La seguente tesi sperimentale nel campo dell'intelligenza artificiale consiste nell'implementazione di un software basato su una rete neurale convoluzionale per il riconoscimento di olive mediante immagini multi-spettrali **VIS-NIR**.

Lo scopo di questo elaborato è di fornire una descrizione del procedimento attraverso il quale si è giunti alla realizzazione di tale applicazione e che potrà essere utilizzato, per scopi futuri, come punto di partenza per realizzare un software capace di stimare il grado di maturazione delle olive.

Il *dataset* usato per il *training* della rete neurale è stato realizzato dall'*Università Politecnica delle Marche* ed è costituito da un insieme di immagini scattate mediante innovativi dispositivi che prendono il nome di **camere multi-spettrali**.

La loro particolarità risiede nel fatto che sono camere in grado di rilevare diverse bande spettrali anche in range che vanno oltre le possibilità dell'occhio umano.

In particolare, le camere usate sono state due: una in grado di catturare la radiazione con la **lunghezza d'onda** compresa tra **470nm - 630nm** (campo del visibile o **VIS**) mentre l'altra in grado di catturare la radiazione con la **lunghezza d'onda** compresa tra **650nm - 950nm** (campo dell'infrarosso o **NIR**).

Una volta acquisite le immagini, una prima manipolazione è stata effettuata dal collega di lavoro *Vittorio Andreotti* il quale ha sviluppato algoritmi per l'elaborazione di immagini multi-spettrali e non.

Ha dapprima creato un *dataset* processando il materiale che aveva a disposizione, ruotando, ridimensionando e equalizzando le immagini; successivamente ne ha effettuato la **coregistrazione**: nel momento della rilevazione le immagini **VIS** e **NIR** vengono acquisite da due camere differenti. Ciò provoca un disallineamento tra ogni coppia dovuto alle diverse angolazioni dei sensori multi-spettrali. Questo fa sì che le operazioni successive richiedano un riallineamento in modo tale che ogni oliva di ogni canale abbia le stesse coordinate in pixel.

Queste operazioni hanno permesso di ottenere un *dataset* appropriato per l'addestramento della rete neurale.

Successivamente il *dataset* è stato arricchito attraverso manipolazioni delle stesse

immagini in modo da aumentare la generalizzazione della performance della rete. Le *label* delle immagini sono state create tramite la piattaforma **Labelbox** mentre il *training* e *testing* della rete neurale sono stati eseguiti in linguaggio **Python** su **Google Colab** e **Kaggle** per riuscire ad ottenere un risultato più efficiente sfruttando schede grafiche maggiormente performanti fornite in remoto da Google.

1.1 Agricoltura 4.0

Con **Agricoltura 4.0** si intende quell'insieme di tecniche innovative che vogliono rendere più sostenibile l'agricoltura in vista dei cambiamenti globali che si stanno verificando attualmente sul nostro pianeta.

Infatti tramite l'utilizzo delle nuove tecnologie, quali *Intelligenza Artificiale* (IA) e *Internet delle Cose* (IoT), siamo capaci di monitorare in tempo reale vari parametri come ad esempio lo stato di salute del raccolto, il livello di nutrienti nel terreno, l'umidità nell'aria ed eventuali malattie che potrebbero colpire la coltivazione.

Nello specifico, grazie a queste tecnologie, è possibile massimizzare la produzione della coltura sfruttando meno risorse: una resa maggiore a fronte di un minor utilizzo di pesticidi, fertilizzanti ed acqua.

Inoltre anche da un punto di vista delle emissioni di CO_2 è possibile notare un decremento sostanziale. [1]



Figura 1.1: Schema generale dell'Agricoltura 4.0

Fonte: <https://agricommerciogardencenter.edagricole.it/news/agricoltura-4-0-sicurezza-sostenibilita-e-tracciabilita/>

1.2 Il progetto

Tenuto conto di quanto appena detto, lo scopo del progetto è quello di implementare un software che sia in grado di riconoscere le olive sulla base di immagini VIS-NIR.

Ciò è stato possibile grazie all'utilizzo di una tecnica nota come **Data Augmentation** attraverso la quale è stato aumentato il *dataset* delle immagini ottenendo copie "alterate" (immagini-copia con caratteristiche differenti da quelle originali) delle immagini di partenza.

Per ogni immagine del *dataset* originario ne è stata creata manualmente una versione **blurred**, due versioni **flipped** (una orizzontale e una verticale) e una versione **ruotata**.

1.3 Struttura della tesi

Per rendere chiaro il più possibile il lavoro svolto, di seguito viene esposta la struttura del presente elaborato:

- **Capitolo 1:** visione generale del problema e dei passi necessari per la sua risoluzione.
- **Capitolo 2:** elenco dei principali strumenti utilizzati per l'implementazione del software ponendo particolare attenzione sulla rete neurale utilizzata.
- **Capitolo 3:** manipolazione del file JSON, ottenuto attraverso la piattaforma Labelbox, per la preparazione del (*dataset*) con successivo *setup* e primo *train* della rete neurale.
- **Capitolo 4:** spiegazione delle tecniche utilizzate per ottenere copie alterate del *dataset* di partenza.
- **Capitolo 5:** riaddestramento della rete neurale con il nuovo *dataset* e discussione dei risultati ottenuti.

Capitolo 2

Strumenti e metodi utilizzati

In questo capitolo viene fatta un'introduzione sul **machine learning** e sulle **reti neurali** per poi spostare l'attenzione sugli strumenti utilizzati durante lo sviluppo del progetto.

2.1 Deep Learning e Reti Neurali

Il **Deep Learning**, la cui traduzione strettamente letterale è **Apprendimento Profondo**, è una sottobranca del **Machine Learning** il cui scopo è quello di creare modelli di apprendimento su più livelli.

Il punto cardine di questa disciplina risiede nel fatto che l'apprendimento assume la forma di una piramide: **i concetti più alti sono appresi a partire dai livelli più bassi.**[2]

Tutti questi livelli danno luce ad una gerarchia di concetti da apprendere: ciascun livello successivo utilizza l'uscita del livello precedente come input in modo tale che l'informazione venga elaborata in maniera sempre più completa.

Oggi i sistemi di apprendimento profondo permettono di:

- identificare gli oggetti nelle immagini e nei video;
- trascrivere il parlato in testo;
- individuare e interpretare gli interessi degli utenti online mostrando i risultati più pertinenti per la loro ricerca.[3]

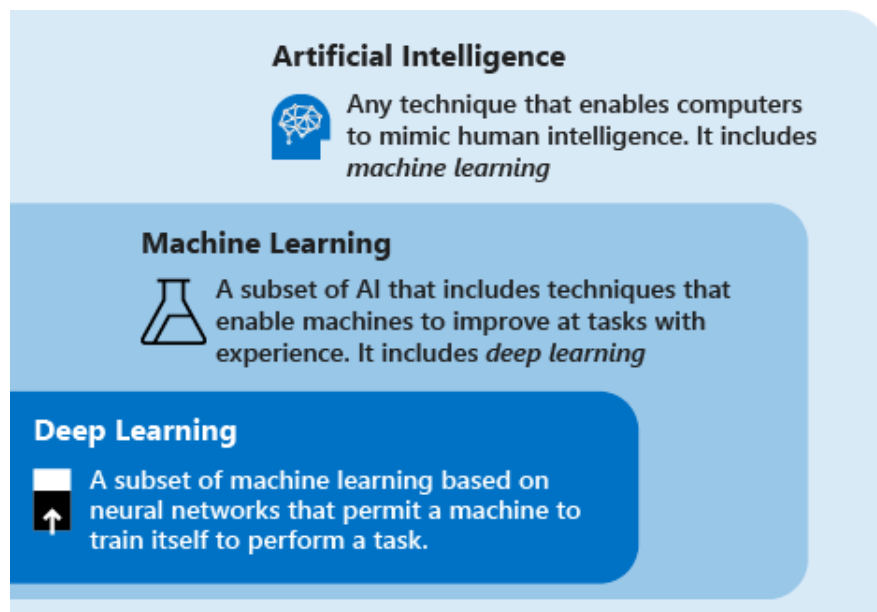


Figura 2.1: Struttura dell'Intelligenza Artificiale

Fonte: <https://docs.microsoft.com/it-it/azure/machine-learning/concept-deep-learning-vs-machine-learning>

2.2 Reti Neurali artificiali

Alla base del *Deep Learning* vi sono le **reti neurali artificiali** o **ANNs**.

Le ANN sono capaci di apprendere utilizzando dei meccanismi simili, almeno in parte, a quelli dell'intelligenza umana.[4]

La tecnologia delle ANNs si ispira al funzionamento delle **reti neurali biologiche**. Le reti neurali del cervello umano sono costituite da un'insieme di cellule nervose fittamente interconnesse tra loro, all'interno delle quali troviamo particolari componenti quali **neurotrasmettitori**, **assoni**, **dendriti** e **sinapsi**.

È proprio sul funzionamento delle sinapsi che le reti neurali artificiali trovano le fondamenta:

- un solo neurone può ricevere contemporaneamente segnali da più sinapsi. La sua capacità principale è quella di poter misurare il potenziale elettrico di questi segnali in modo tale da stabilire se è stata raggiunta la soglia di attivazione per generare un impulso nervoso.
- il numero di sinapsi può aumentare o diminuire a seconda dei vari stimoli che riceve la rete. Più sono numerosi e maggiori sono le connessioni sinaptiche che vengono create e viceversa.

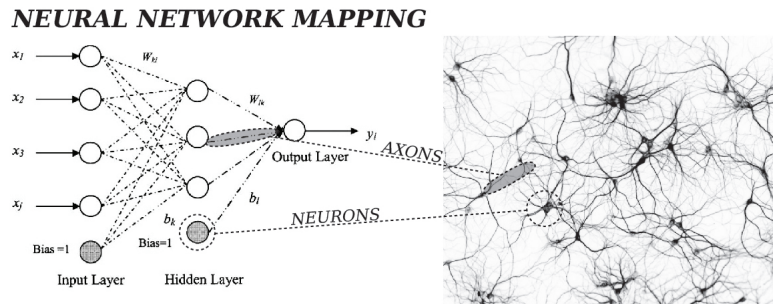


Figura 2.2: Confronto tra Rete Neurale Biologica e Artificiale.

Fonte:

<https://analisiidiborsa.altervista.org/reti-neurali-artificiali>

2.2.1 Principio di funzionamento

L'architettura di una rete *ANNs* è costituita da uno strato di **input**, il quale riceve dati dall'esterno allo scopo di riconoscere i vari *pattern*, uno strato di **output**, che fornisce la soluzione al problema, ed infine uno strato **nascosto intermedio** che separa il primo dal secondo strato e nel quale avviene la computazione.

La chiave del funzionamento di queste reti neurali risiede nel *pattern* di **input-elaborazione-output**, i cui input sono costituiti da dati sempre differenti e grazie ai quali i singoli nodi *apprendono* e forniscono output corretti.[5]

Gli **algoritmi di apprendimento** usati per addestrare le reti neurali sono divisi in 3 categorie:

- **apprendimento supervisionato:** viene fornito alla rete un insieme di input ai quali corrispondono output noti (**training set**). Analizzandoli la rete apprende il nesso che li unisce.
- **apprendimento non supervisionato:** viene fornito alla rete solo un insieme di variabili di input. Analizzandole, la rete le organizzerà secondo delle somiglianze comuni in modo tale da effettuare previsioni sugli input futuri.
- **apprendimento di rinforzo:** la rete neurale impara esclusivamente dall'interazione con l'ambiente. Viene considerato rinforzo l'azione che avvicina al risultato ed errore ciò la allontana.

La scelta su quale algoritmo sia più consono utilizzare dipende dal campo di applicazione della rete neurale e dalla sua tipologia (**feedback** o **feedforward**).

In questo lavoro di tesi è stato utilizzato l'apprendimento supervisionato.

2.2.2 Error Back-Propagation

Ogni volta che la macchina produce un output si provvede a correggerlo per migliorare la sua risposta. Questa operazione consiste nel variare i **pesi sinaptici**

(indicano la forza di una connessione tra due nodi).

Lo scopo di questa procedura è quello di aumentare i pesi che determinano gli output corretti e diminuire quelli che generano valori ritenuti non validi; tutto questo perché una corretta regolazione dei pesi garantisce tassi di errore inferiore rendendo il modello affidabile e aumentando la sua generalizzazione.

Tuttavia variare manualmente i pesi risulta un'operazione piuttosto complicata a livello operativo e per questo motivo viene utilizzato un particolare algoritmo, che prende il nome di **Error Back-Propagation**, per migliorare la precisione dell'output.

Questo algoritmo consente di mettere a punto i pesi della rete neurale in base al *tasso di errore* ottenuto nell'iterazione (epoch) precedente e consiste nel variare i pesi associati ad ogni output a ritroso, cioè partire dall'ultimo strato per arrivare al primo in modo da ridurre sempre di più l'errore totale.[6]

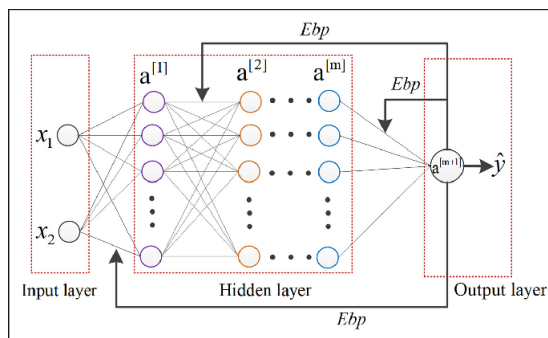


Figura 2.3: Error Back-Propagation

Fonte: <https://www.researchgate.net/figure/The-multilayer-feedforward-neural-network>

Un altro parametro molto importante da tenere in considerazione è la dimensione del training set, questo perché si vuole evitare il fenomeno di **overfitting**. Nello specifico, in informatica, si parla di **overfitting** quando un modello statistico molto complesso, nel nostro caso la rete neurale, si adatta in maniera forte ai dati osservati perché ha un numero eccessivo di parametri rispetto al numero di osservazioni. Per questo motivo è necessario avere il giusto rapporto fra le dimensioni del training set, le dimensioni della rete e l'abilità a generalizzare ciò che desidera ottenere.[7]

2.3 YOLOv3

YOLO, acronimo di **You Only Look Once**, è un algoritmo che utilizza una **rete neurale convoluzionale** per rilevare oggetti in tempo reale ¹. Come suggerisce il nome, l'algoritmo richiede una sola propagazione in avanti attraverso la rete neurale per rilevare gli oggetti.[8]

¹You Only Look Once: Unified, Real-Time Object Detection - Redmon et al, 2016

La versione utilizzata in questo progetto è **YOLOv3**, la più efficiente e generalizzata delle altre, ottenuta dal **repository GitHub** di **ultralytics**.

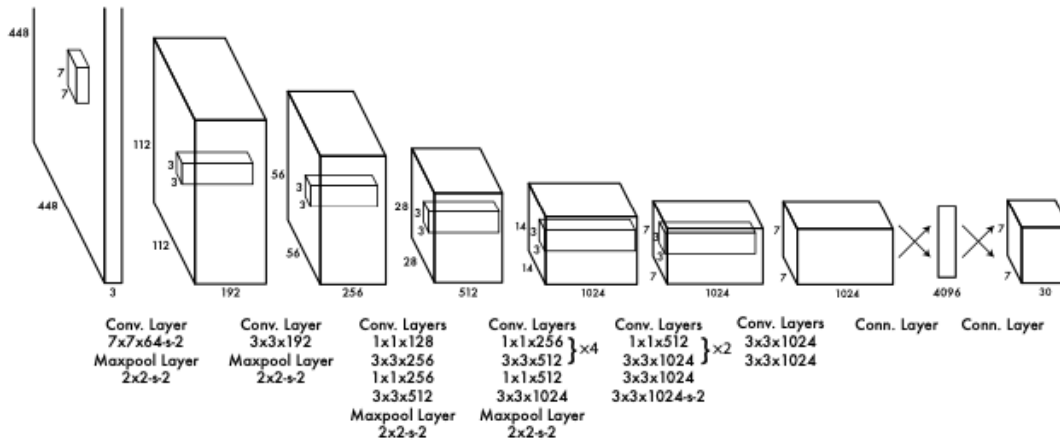


Figura 2.4: Architettura di YOLO

Fonte: <https://ichi.pro/it/>

rilevamento-di-oggetti-yolov3-in-tensorflow-2-x-151844547217064

2.3.1 Funzionamento di YOLO

Il primo passo per capire il funzionamento di YOLO è capire come codifica i suoi input. L'immagine in input viene divisa in una griglia di dimensione $S \times S$ di celle. Per ogni oggetto presente sull'immagine, ogni cella della griglia è responsabile della sua previsione (con centro dell'oggetto appartenente alla stessa cella) mediante un numero finito di **bounding box**.

Ogni **bounding box** possiede 5 componenti: 4 di esse sono relative alle coordinate del bbox mentre la quinta rappresenta l'**indice di confidenza**:

- **(top, left)**: coordinate che rappresentano il centro del bbox, relativo alla posizione della cella sulla griglia.
- **(height, width)**: coordinate che rappresentano le dimensioni del bbox.
- **indice di confidenza**: parametro che indica il valore della probabilità che ci sia un oggetto in quella bbox.

Ogni bbox, inoltre, ha un certo numero di **probabilità condizionali di classe** (una per classe), valori che indicano quanto è probabile che l'oggetto in questione appartenga a quella specifica classe.[9]

Una volta determinate tutte le bbox YOLO scarta quelle con indice di confidenza più basso (minore di 0.25) per poi selezionare il bbox migliore.

Questo bbox migliore sarà usato, poi, per calcolare un parametro fondamentale per l'efficienza della rete: la *funzione di loss* o **loss function**.

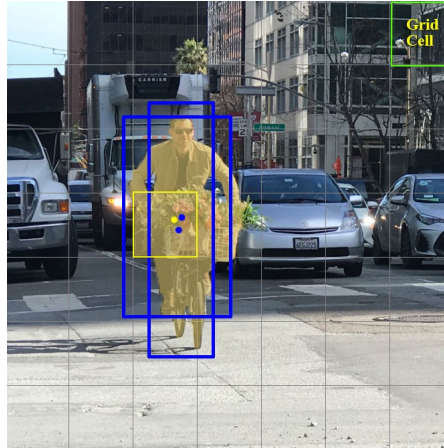


Figura 2.5: Griglia SxS di YOLO

Fonte:

<https://ichi.pro/it/rilevamento-di-oggetti-in-tempo-reale-con-yolo>

2.3.2 Parametri di YOLO

YOLO dispone di una vasta gamma di parametri per valutare la precisione e l'efficienza delle proprie predizioni.

I principali sono: **Loss Function**, **Intersect Over Union**, **Precision**, **Recall**, **F1**, **AP** e **MAP**.

La **Loss Function** è una funzione di costo che rappresenta la somma degli errori commessi nella predizione dell'oggetto. Viene calcolata tramite l'errore quadratico medio sul bbox *migliore* ed è a sua volta composta da 3 parametri, ognuno dei quali definisce un particolare tipo di errore nella predizione: **classification loss**, **localization loss** e **confidence loss**. [10]

$$\begin{aligned}
 & \lambda_{\text{coord}} \sum_{i=0}^{S^2} \sum_{j=0}^B \mathbb{1}_{ij}^{\text{obj}} \left[(x_i - \hat{x}_i)^2 + (y_i - \hat{y}_i)^2 \right] \\
 & + \lambda_{\text{coord}} \sum_{i=0}^{S^2} \sum_{j=0}^B \mathbb{1}_{ij}^{\text{obj}} \left[(\sqrt{w_i} - \sqrt{\hat{w}_i})^2 + (\sqrt{h_i} - \sqrt{\hat{h}_i})^2 \right] \\
 & + \sum_{i=0}^{S^2} \sum_{j=0}^B \mathbb{1}_{ij}^{\text{obj}} (C_i - \hat{C}_i)^2 \\
 & + \lambda_{\text{noobj}} \sum_{i=0}^{S^2} \sum_{j=0}^B \mathbb{1}_{ij}^{\text{noobj}} (C_i - \hat{C}_i)^2 \\
 & + \sum_{i=0}^{S^2} \mathbb{1}_i^{\text{obj}} \sum_{c \in \text{classes}} (p_i(c) - \hat{p}_i(c))^2
 \end{aligned}$$

Figura 2.6: Formula per il calcolo della Loss Function.

Di seguito vengono illustrati i parametri che compongono la Loss Function:

- **Classification Loss:** errore quadratico medio della probabilità condizionale di classe, calcolata per ogni classe.

$$\sum_{i=0}^{S^2} \mathbb{1}_i^{\text{obj}} \sum_{c \in \text{classes}} (p_i(c) - \hat{p}_i(c))^2$$

where

$\mathbb{1}_i^{\text{obj}} = 1$ if an object appears in cell i , otherwise 0.

$\hat{p}_i(c)$ denotes the conditional class probability for class c in cell i .

Figura 2.7: Formula per il calcolo della Classification Loss.

- **Localization Loss:** misura l'errore sulla posizione e sulla grandezza del bbox.

$$\begin{aligned} & \lambda_{\text{coord}} \sum_{i=0}^{S^2} \sum_{j=0}^B \mathbb{1}_{ij}^{\text{obj}} \left[(x_i - \hat{x}_i)^2 + (y_i - \hat{y}_i)^2 \right] \\ & + \lambda_{\text{coord}} \sum_{i=0}^{S^2} \sum_{j=0}^B \mathbb{1}_{ij}^{\text{obj}} \left[\left(\sqrt{w_i} - \sqrt{\hat{w}_i} \right)^2 + \left(\sqrt{h_i} - \sqrt{\hat{h}_i} \right)^2 \right] \end{aligned}$$

where

$\mathbb{1}_{ij}^{\text{obj}} = 1$ if the j th boundary box in cell i is responsible for detecting the object, otherwise 0.

λ_{coord} increase the weight for the loss in the boundary box coordinates.

Figura 2.8: Formula per il calcolo della Localization Loss.

- **Confidence Loss:** errore relativo alla probabilità che un bbox contenga un oggetto.

$$\sum_{i=0}^{S^2} \sum_{j=0}^B \mathbb{1}_{ij}^{obj} (C_i - \hat{C}_i)^2$$

where

\hat{C}_i is the box confidence score of the box j in cell i .

$\mathbb{1}_{ij}^{obj} = 1$ if the j th boundary box in cell i is responsible for detecting the object, otherwise 0.

Figura 2.9: Formula per il calcolo della Confidence Loss.

Torniamo alla spiegazione dei principali parametri di valutazione di YOLO.

- **Intersect Over Union (IoU):** è un numero compreso tra 0 e 1 che valuta il grado di sovrapposizione di due bbox (quella originale e quella determinata dalla rete neurale). L'interpretazione del valore ottenuto è piuttosto intuitiva: un valore vicino o uguale ad 1 sta a significare che il bbox originale e quella determinata dalla rete si sovrappongono mentre un valore corrispondente a 0 indica che i due bbox non si sovrappongono.[11]

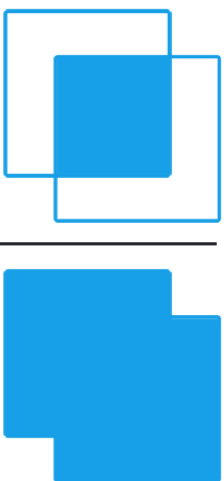
$$\text{IoU} = \frac{\text{Area of Overlap}}{\text{Area of Union}}$$


Figura 2.10: Formula per il calcolo dell'IoU.

Fonte: [https://towardsdatascience.com/](https://towardsdatascience.com/iou-a-better-detection-evaluation-metric-45a511185be1)

[iou-a-better-detection-evaluation-metric-45a511185be1](https://towardsdatascience.com/iou-a-better-detection-evaluation-metric-45a511185be1)

- **Precision, Recall e F1:** sono 3 metriche che determinano l'efficienza della predizione.

La **precision** corrisponde all'accuratezza con cui il sistema di machine learning prevede le classi positive ed è definito come il rapporto tra i *true positive*

e la somma dei *true positive* e *false positive*.

La **recall** indica il rapporto di istanze positive correttamente individuate dal sistema.

F1 è una metrica composta dalla fusione delle due precedenti. Corrisponde ad una **media armonica**, ovvero il reciproco della media aritmetica dei reciproci. Rispetto ad una media convenzionale, questo tipo di media attribuisce un peso maggiore ai valori piccoli. In questo modo un classificatore ottiene un alto punteggio di F1 solo quando precision e recall sono entrambi alti.[12]

$$\text{Precision} = \frac{TP}{TP + FP}$$

TP = True positive

$$\text{Recall} = \frac{TP}{TP + FN}$$

TN = True negative

FP = False positive

FN = False negative

$$F1 = 2 \cdot \frac{\text{precision} \cdot \text{recall}}{\text{precision} + \text{recall}}$$

Figura 2.11: Formule per il calcolo di Precision, Recall e F1.

Fonte: <https://jonathan-hui.medium.com/>

map-mean-average-precision-for-object-detection

- **AP (Average Precision)**: corrisponde all'area sottesa alla curva **precision-recall**.

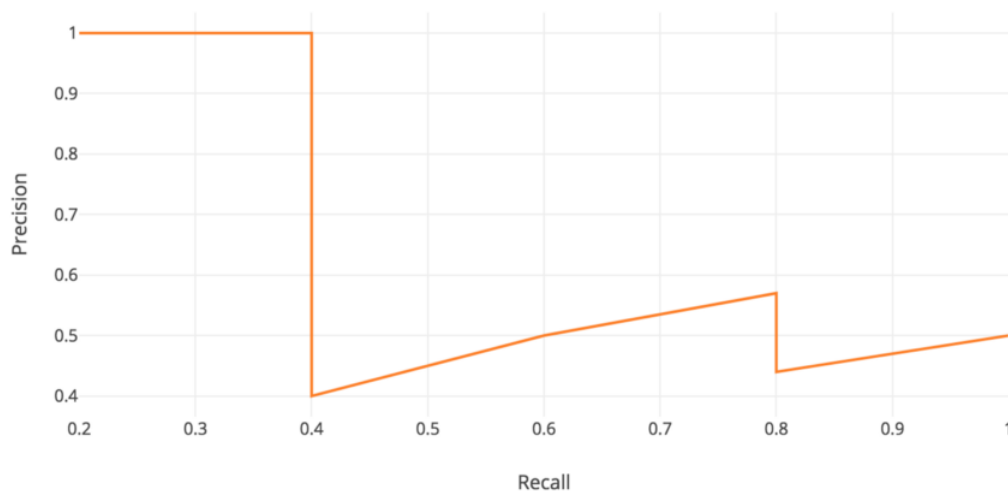


Figura 2.12: Esempio di curva Precision-Recall.

Fonte: <https://jonathan-hui.medium.com/>

map-mean-average-precision-for-object-detection-45c121a31173

Viene calcolato come segue:

$$\text{AP} = \int_0^1 p(r) dr$$

Figura 2.13: Calcolo dell'AP.

Fonte: <https://jonathan-hui.medium.com/map-mean-average-precision-for-object-detection-45c121a31173>

- **MAP (Mean Average Precision)**: corrisponde all'AP ma calcolato per tutte le classi, cioè la media dell'AP per ogni classe.

2.3.3 Iperparametri di YOLO

Sono dei parametri di sistema che gestiscono l'algoritmo di apprendimento e determinano l'efficienza del training della rete neurale. Si dividono principalmente in due categorie: quelli relativi alla *loss function* e quelli relativi al *data augmentation*.

Gli iperparametri relativi al data augmentation sono stati impostati tutti a zero in quanto le immagini che compongono il dataset sono state modificate attraverso degli script realizzati ad hoc.

Per gli iperparametri relativi alla loss function è stato usato il comando **evolve.py** fornito da YOLO. Consiste in un algoritmo evolutivo che parte dall'insieme iniziale di iperparametri per poi modificarlo fino a raggiungere dei valori ottimali.

```
# Hyperparameters
hyp = {'giou': 3.54, # giou loss gain
      'cls': 37.4, # cls loss gain
      'cls_pw': 1.0, # cls BCELoss positive_weight
      'obj': 64.3, # obj loss gain (*=img_size/320 if img_size != 320)
      'obj_pw': 1.0, # obj BCELoss positive_weight
      'iou_t': 0.20, # iou training threshold
      'lr0': 0.01, # initial learning rate (SGD=5E-3, Adam=5E-4)
      'lrf': 0.0005, # final learning rate (with cos scheduler)
      'momentum': 0.937, # SGD momentum
      'weight_decay': 0.0005, # optimizer weight decay
      'fl_gamma': 0.0, # focal loss gamma (efficientDet default is gamma=1.5)
      'hsv_h': 0.0138, # image HSV-Hue augmentation (fraction)
      'hsv_s': 0.678, # image HSV-Saturation augmentation (fraction)
      'hsv_v': 0.36, # image HSV-Value augmentation (fraction)
      'degrees': 1.98 * 0, # image rotation (+/- deg)
      'translate': 0.05 * 0, # image translation (+/- fraction)
      'scale': 0.05 * 0, # image scale (+/- gain)
      'shear': 0.641 * 0} # image shear (+/- deg)
```

Figura 2.14: Iperparametri di YOLO.

2.4 Labelbox

Labelbox è un tool di **data labeling** usato per creare le varie bbox sulle immagini del dataset. Il principio di funzionamento di questo tool è il seguente: i due valori (x e y) che vengono assegnati ad ogni label seguono un sistema di assi cartesiani che parte da in alto a sinistra.



Figura 2.15: Logo di Labelbox.

2.5 Python

È un linguaggio di programmazione di *alto livello*, orientato ad oggetti. È il linguaggio utilizzato per lo sviluppo del seguente lavoro di tesi.



Figura 2.16: Logo di Python.

2.6 Google Colab

È una piattaforma che consente di eseguire codice sul cloud. La particolarità di questa piattaforma è quella di fornire da remoto GPU ai suoi utenti permettendo di svolgere calcoli anche molto complessi.



Figura 2.17: Logo di Google Colab.

2.7 Kaggle

È una consociata di Google LLC con le stesse funzionalità di Google Colab.



Figura 2.18: Logo di Kaggle.

2.8 GitHub

È un servizio di hosting per progetti software. Viene principalmente utilizzato da sviluppatori che caricano il codice sorgente dei loro programmi e lo rendono scaricabile dagli utenti. Questi ultimi possono interagire con lo sviluppatore al fine di migliorare il codice e risolvere eventuali bug oppure utilizzarlo per i loro scopi.[13]

In questo progetto di tesi è stata scaricata la versione **YOLOv3** dalla repository github <https://github.com/ultralytics/yolov3>.



Figura 2.19: Logo di GitHub.

Capitolo 3

Labeling

3.1 Labeling

Per la creazione del primo dataset ci si è serviti della piattaforma **Labelbox**. Sono state selezionate circa **56 immagini NIR** prelevate da una collezione di foto resa disponibile dall'*Università Politecnica delle Marche* e successivamente caricate su tale piattaforma.

Grazie a *Labelbox* è stato possibile applicare le **label** manualmente ed esportare localmente i risultati ottenuti in formato **json**.

In seguito è stato creato un **algoritmo di parsing** la cui funzione è quella di estrarre le coordinate in formato **Darknet**¹ (un file di testo con estensione **.txt**) e quella di **normalizzare** queste coordinate ottenute, per via del fatto che il riferimento cartesiano utilizzato da *Labelbox* è diverso da quello usato da *YOLOv3*.

3.2 Labelbox

Una volta iscritti alla piattaforma *Labelbox* è stato generato un nuovo progetto dalla sezione **New Project** e specificata la classe **Olive** nel tool di labeling. Effettuate queste operazioni è stato possibile procedere alla fase la *labellizzazione*.

Labelbox offre la possibilità di creare le *bbox* in varie forme. Per questo progetto la scelta è ricaduta sulla forma rettangolare.

Al termine di questa fase sono state totalizzate, tra le 56 immagini, ben *960 bounding box*, con l'accortezza di aver preso in considerazione le olive migliori tra tutte quelle presenti.

3.3 Esportazione in formato JSON

Attraverso il comando **export** di *Labelbox* è stato effettuato il download in locale del file contenente le *label* in formato *JSON*, le quali presentano la seguente

¹Darknet: Open Source Neural Networks in C - Redmon, 2016

struttura:

```

7   "Created By": "proietti.melissa@gmail.com",
8   "DataRow ID": "ckrv0gxhb1p0x0yt7fvod9u7y",
9   "Dataset Name": "Olives",
10  "External ID": "08_NIR_1603201836671.npy.png",
11  "Has Open Issues": 0,
12  "ID": "ckrv0q63203p20yb67gex6vbd",
13  "Label": {
14    "objects": [
15      {
16        "featureId": "ckrv0qh8s00013b697rf1vbx",
17        "schemaId": "ckrv0kf2o015s0ya199vt737r",
18        "color": "#1CE6FF",
19        "title": "Oliva",
20        "value": "oliva",
21        "bbox": {
22          "top": 132,
23          "left": 326,
24          "height": 28,
25          "width": 20
26        },
27        "instanceURI": "https://api.labelbox.com/masks/feature/ckr
28      },

```

Figura 3.1: Schema generale di una singola Label.

Come si nota dall'immagine sovrastante, il file JSON è strutturato come un array di oggetti dove ogni campo rappresenta un'immagine etichettata.

Il campo *Label* contiene un array di oggetti chiamato **objects**.

Ogni oggetto contenuto in *objects* rappresenta un bounding box, quindi un'oliva. Tra i vari attributi di questi oggetti troviamo il titolo dell'etichetta, il colore usato per il bounding box, le coordinate e le dimensioni del bounding box:

- l'origine degli assi è posto in alto a sinistra (*top - left*)
- *height - width* costituiscono le dimensioni del bbox.

3.4 Parsing

Visto che il sistema di coordinate non è compatibile con il formato **Darknet** di YOLOv3 è stato necessario effettuare una conversione in cui per ogni oggetto otteniamo una stringa così strutturata:

$\langle object - class \rangle \langle x - center \rangle \langle y - center \rangle \langle width \rangle \langle height \rangle$

dove:

- $\langle object - class \rangle$ indica il numero relativo alla classe. In questo caso è presente una sola classe, ovvero 0, quella delle olive.

- $\langle x - center \rangle$ indica l'ascissa del centro del bounding box.
- $\langle y - center \rangle$ indica l'ordinata del centro del bounding box.
- $\langle width \rangle$ indica la lunghezza del bounding box.
- $\langle height \rangle$ indica l'altezza del bounding box.

Tale conversione è stata effettuata attraverso il seguente algoritmo:

- importazione del file JSON e creazione di apposite cartelle che conterranno le immagini e le relative label.
- per ogni oggetto del file JSON viene effettuato il download dell'immagine e le viene assegnato un nome pari al valore del contatore incrementale.
- calcolo dell'altezza e del centro della label.
- trasformazione di tali valori nel formato **Darknet**.
- scrittura di una riga nel file di testo relativo ad ogni immagine contenente le misure delle label sopra calcolate.

```
import json
import os
import requests
from PIL import Image

os.mkdir('Olive')
os.mkdir('Olive/Immagini')
os.mkdir('Olive/Labels')

json_file = open(r'C://Users/mely7/OneDrive/Desktop/export-2021-08-22T17_06_07.224Z.json')
JSON = json.load(json_file)

for imagecounter, row in enumerate(JSON):
    if row['Label'] != 'Skip':
        url = row['Labeled Data']
        filename = f'Olive/Immagini/{imagecounter}.jpg'
        r = requests.get(url, allow_redirects=True)
        open(filename, 'wb').write(r.content)

for counter, row in enumerate(JSON):
    if row['Label'] != 'Skip':
        picture = Image.open(f'Olive/Immagini/{counter}.jpg')
        picture_width, picture_height = picture.size
        with open(f'Olive/Labels/{counter}.txt', 'w+') as output:
            for element in row['Label']['objects']:
                x_center_normalized = (element["bbox"]["left"] + element["bbox"]["width"] / 2) / picture_width
                y_center_normalized = (element["bbox"]["top"] + element["bbox"]["height"] / 2) / picture_height
                width_normalized = element["bbox"]["width"] / picture_width
                height_normalized = element["bbox"]["height"] / picture_height
                output.write(f'0 {x_center_normalized} {y_center_normalized} {width_normalized} {height_normalized}\n')
```

Figura 3.2: Codice del Parsing.

Il risultato ottenuto è il seguente:

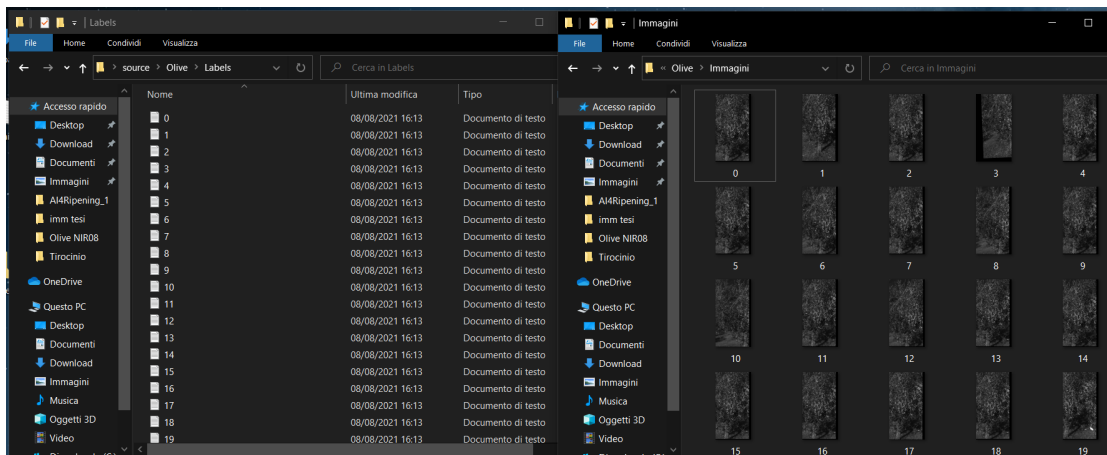


Figura 3.3: Overview del dataset.

3.5 Preparazione del dataset

Il primo *train* della rete è stato effettuato con il *dataset* appena presentato dividendolo in un **rapporto 80:20** tra **training set**, l'insieme delle immagini usato per addestrare la rete, e **validation set**, un piccolo set di immagini che è periodicamente oggetto di un test da parte della rete neurale.

Questa suddivisione è necessaria per evitare il problema dell'**overfitting**.

Analizzando, inoltre, i parametri generati dai test sul validation set e attraverso il confronto con quelli generati sul training set si riesce a capire quanto la rete neurale sia generalizzata.

3.6 Setup e primo train di YOLOv3

Per preparare la rete alla fase di training è necessario creare 3 file di configurazione e modificare un file di tipo *cfg*.

I tre file di configurazione vanno inseriti nella cartella *data*, all'interno della directory di YOLO:

- il primo file da creare è un file che prende il nome di **olives.names**, che andrà a contenere la lista di tutte le classi del dataset. In questo caso la sola classe **olives**.
- il secondo file da creare è un file che prende il nome di **olives.txt**, contenente una lista comprendente, per ogni immagine del dataset, il relativo percorso.

- il terzo file da creare è un file **olives.data** il quale contiene il percorso del file **olives.names**, il percorso del file **olives.txt** usato per l'addestramento, il percorso del file **olives.txt** usato per la validazione (in questo progetto viene usato lo stesso file per entrambe le fasi) e il numero delle classi.

All'interno della cartella *cfg*, contenuta nella directory di YOLO, si trova il file **yolov3.cfg** di cui è necessario modificare alcuni parametri di configurazione:

- il numero di filtri viene settato a **18** per ogni strato della rete neurale ($filters = [5 + n] * 3$, dove $n = class\ count$).
- il numero di classi è stato settato a **1**.
- il **batch-size** è stato settato a 8
- il **subdivision** è stato settato a 16.

3.7 Risultati

Di seguito sono mostrati i risultati del primo train:

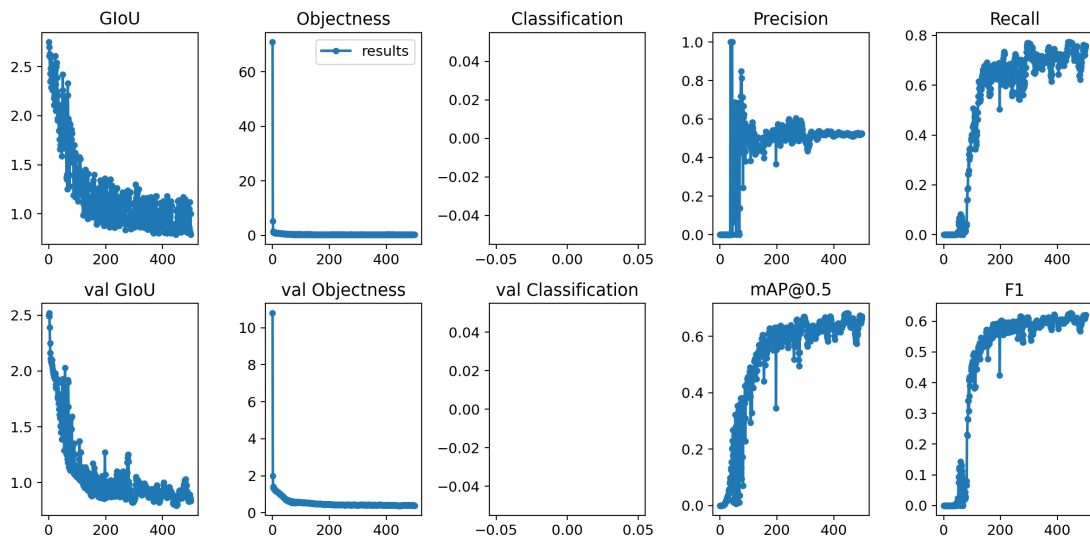


Figura 3.4: Risultati del primo train.

Tutte le funzioni di costo presentano delle oscillazioni causate dal ridotto numero di immagini che costituiscono il dataset anche se nel complesso i risultati sono buoni.

Non è presente l'*overfitting* in quanto il valore delle funzioni di costo *validation GloU* e *validation Objectness* si riduce nel tempo senza mai incrementare nuovamente.

La precisione risulta essere *bassa*; tale valore può essere migliorato attraverso la *data augmentation*.

Capitolo 4

Data Augmentation

Data Augmentation, letteralmente *aumento dei dati*, è un insieme di tecniche che amplia il *dataset* a disposizione applicando ai dati già esistenti dei cambiamenti. Questa tecnica viene effettuata in modo tale da addestrare la rete a riconoscere un maggior numero di oggetti.[14]

In questo caso, sono state effettuate quattro tipi di trasformazioni differenti: **rotazione**, **blurring**, **flipping orizzontale** e **flipping verticale**.

4.1 Rotazione

Per ogni immagine è stata scelta una rotazione in senso antiorario di un angolo di 30° .

Per realizzare questa operazione è necessario prima creare un'immagine-copia ruotata e successivamente cambiare le coordinate del *bounding box*. Inoltre, è necessario creare anche un nuovo file che contiene le *label* con le coordinate aggiornate dopo la rotazione.

In genere, la rotazione è una trasformazione, dipendente da un angolo θ , che trasforma un vettore (x, y) in:

- $x' = x\cos(\theta) - y\sin(\theta)$
- $y' = x\sin(\theta) + y\cos(\theta)$

Tuttavia il centro di YOLO è posizionato in alto a sinistra, con gli assi x e y rivolti rispettivamente verso destra e verso il basso, motivo per cui è stato invertito il segno della y :

- $x' = x\cos(\theta) + y\sin(\theta)$
- $y' = -x\sin(\theta) - y\cos(\theta)$

Queste equazioni però sono valide per un sistema di riferimento che ha il centro posizionato in $(0, 0)$, ovvero il centro dell'immagine.

Nel calcolo delle nuove coordinate perciò, prima viene traslato ogni singolo punto e poi vengono calcolate le coordinate del punto soggetto a rotazione. Infine vengono traslate nel sistema di riferimento usato da YOLO.

Riassumendo:

1. Normalizzazione delle coordinate:

- $x = x - (\text{imagewidth})/2$
- $y = y - (\text{imageheight})/2$

2. Rotazione delle coordinate:

- $x' = x\cos(\theta) + y\sin(\theta)$
- $y' = -x\sin(\theta) - y\cos(\theta)$

3. Cambio di coordinate:

- $x' = x' - (\text{imagewidth})/2$
- $y' = y' - (\text{imageheight})/2$

L'algoritmo appena presentato è stato implementato tramite il seguente script utilizzando la libreria *OpenCV*:

```
import cv2
import os
from PIL import Image
from scipy import ndimage
import math
import json

json_file = open(r'C://Users/mely7/OneDrive/Desktop/export-2021-08-22T17_06_07.224Z.json')
JSON = json.load(json_file)

for counter, row in enumerate(JSON):
    txt_file = open(r'C://Users/mely7/OneDrive/Desktop/Olive/Labels/' + str(counter) + '_r.txt', 'w+')
    img = cv2.imread(r'C://Users/mely7/OneDrive/Desktop/Olive/Immagini/' + str(counter) + '.jpg')
    rows, cols, ht = img.shape
    matrix = cv2.getRotationMatrix2D((cols/2, rows/2), 30, 1)
    new_img = cv2.warpAffine(img, matrix, (cols, rows))
    cv2.imwrite(r'C://Users/mely7/OneDrive/Desktop/Olive/Immagini/' + str(counter) + '_r.jpg', new_img)
```

Figura 4.1: Generazione dell'immagine-copia ruotata.

```

for counter, row in enumerate(JSON):
    if row['Label'] != 'Skip':
        picture = Image.open(f'C:/Users/mely7/OneDrive/Desktop/Olive/Immagini/{counter}_r.jpg')
        rows, cols, ht = img.shape
        with open(f'C:/Users/mely7/OneDrive/Desktop/Olive4/Labels/{counter}_r.txt', 'w+') as output:
            for element in row['Label']['objects']:
                x = (element["bbox"]["left"] + element["bbox"]["width"]/2)
                y = (element["bbox"]["top"] + element["bbox"]["height"]/2)
                center_x = (element["bbox"]["left"] + element["bbox"]["width"]/2) - cols/2
                center_y = (element["bbox"]["top"] + element["bbox"]["height"]/2) - rows/2
                appoggio_x = (element["bbox"]["left"] + element["bbox"]["width"]/2) - cols/2
                appoggio_y = (element["bbox"]["top"] + element["bbox"]["height"]/2) - rows/2
                new_center_x = appoggio_x * math.cos(math.pi/6) + appoggio_y * math.sin(math.pi/6)
                new_center_y = appoggio_x * math.sin(math.pi/6) - appoggio_y * math.cos(math.pi/6)
                new_center_x = new_center_x + cols/2
                new_center_y = new_center_y + rows/2
                relatives_center_x = new_center_x / cols
                relatives_center_y = new_center_y / rows
                width_normalized = element["bbox"]["width"] / cols
                height_normalized = element["bbox"]["height"] / rows
                #controllo che il centro della label sia ancora contenuta nell'immagine, se non lo è la label non è valida e non la scrivo nel txt
                if (((relatives_center_x > 0) and (relatives_center_y > 0)) and (relatives_center_x < 1) and (relatives_center_y < 1)):
                    output.write(f'0 {relatives_center_x} {relatives_center_y} {width_normalized} {height_normalized}\n')

```

Figura 4.2: Calcolo nuove coordinate.

Il risultato ottenuto è il seguente:

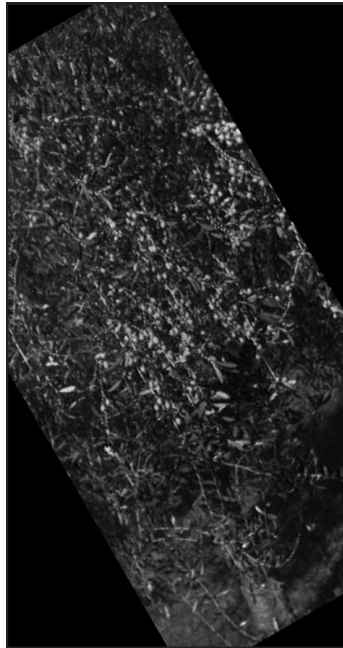


Figura 4.3: Immagine ruotata.

4.2 Blurring

Processo di sfocatura dell'immagine originale realizzato tenendo conto del parametro **Blur Radius**.

Questa operazione non modifica la posizione delle label nelle immagini, motivo per cui vengono soltanto generate delle copie dei file *.txt*.

```
import os
from PIL import Image
import cv2
from scipy import ndimage
import math
import json

json_file = open(r'C://Users/mely7/OneDrive/Desktop/export-2021-08-22T17_06_07.224Z.json')
JSON = json.load(json_file)

for counter, row in enumerate(JSON):
    file3 = open(r'C://Users/mely7/OneDrive/Desktop/Olive/Labels/' + str(counter) + '_b.txt', 'w+')
    img = cv2.imread(r'C://Users/mely7/OneDrive/Desktop/Olive/Immagini/' + str(counter) + '.jpg')
    blur = cv2.blur(img, (5, 5))

    cv2.imwrite((r'C://Users/mely7/OneDrive/Desktop/Olive/Immagini/' + str(counter) + '_b.jpg'), blur)

for counter, row in enumerate(JSON):
    if row['Label'] != 'Skip':
        picture = Image.open(f'C://Users/mely7/OneDrive/Desktop/Olive/Immagini/{counter}_b.jpg')
        picture_width, picture_height = picture.size
        with open(f'C://Users/mely7/OneDrive/Desktop/Olive4/Labels/{counter}_b.txt', 'w+') as output:
            for element in row['Label']['objects']:
                x_center_normalized = (element["bbox"]["left"] + element["bbox"]["width"] / 2) / picture_width
                y_center_normalized = (element["bbox"]["top"] + element["bbox"]["height"] / 2) / picture_height
                width_normalized = element["bbox"]["width"] / picture_width
                height_normalized = element["bbox"]["height"] / picture_height
                output.write(f'0 {x_center_normalized} {y_center_normalized} {width_normalized} {height_normalized}\n')
```

Figura 4.4: Script per il blurring.

Il risultato ottenuto è il seguente:

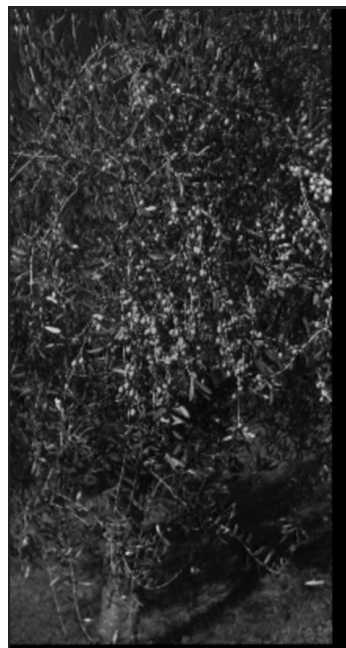


Figura 4.5: Immagine con effetto blur.

4.3 Flipping Orizzontale

Il processo con cui si invertono le righe e le colonne di un'immagine in pixel, in orizzontale, prende il nome di **Horizontal Flip**.

Dopo aver creato l'immagine-copia bisogna creare i nuovi file *.txt* contenenti le nuove coordinate delle label: in questo caso la coordinata Y resta invariata, mentre la coordinata X risulta simmetrica rispetto all'asse passante per il centro.

```
import os
from PIL import Image
import cv2
from scipy import ndimage
import math
import json

json_file = open(r'C://Users/mely7/OneDrive/Desktop/export-2021-08-22T17_06_07.224Z.json')
JSON = json.load(json_file)

for counter, row in enumerate(JSON):
    file2 = open(r'C://Users/mely7/OneDrive/Desktop/Olive/Labels/' + str(counter) + '_fh.txt', 'w+')
    img = cv2.imread(r'C://Users/mely7/OneDrive/Desktop/Olive/Immagini/' + str(counter) + '.jpg')
    flipHorizontal = cv2.flip(img, 1)

    cv2.imwrite((r'C://Users/mely7/OneDrive/Desktop/Olive/Immagini/' + str(counter) + '_fh.jpg'), flipHorizontal)

for counter, row in enumerate(JSON):
    if row["Label"] != "Skip":
        picture = Image.open(f'C://Users/mely7/OneDrive/Desktop/Olive/Immagini/{counter}_fh.jpg')
        picture_width, picture_height = picture.size
        with open(f'C://Users/mely7/OneDrive/Desktop/Olive/Labels/{counter}_fh.txt', 'w+') as output:
            for element in row["Label"]["objects"]:
                x = (element["bbox"]["left"] + element["bbox"]["width"])/2
                new_x = picture_width - x
                x_center_normalized = new_x / picture_width
                y_center_normalized = (element["bbox"]["top"] + element["bbox"]["height"] / 2) / picture_height
                width_normalized = element["bbox"]["width"] / picture_width
                height_normalized = element["bbox"]["height"] / picture_height
                output.write(f'\0 {x_center_normalized} {y_center_normalized} {width_normalized} {height_normalized}\n')
```

Figura 4.6: Script per l'Horizontal Flip.

Il risultato ottenuto è il seguente:



Figura 4.7: Immagine flippata orizzontalmente.

4.4 Flipping Verticale

Il processo con cui si invertono le righe e le colonne di un'immagine in pixel, in verticale, prende il nome di **Vertical Flip**.

Dopo aver creato l'immagine-copia bisogna creare i nuovi file *.txt* contenenti le nuove coordinate delle label: in questo caso la coordinata X resta invariata, mentre la coordinata Y risulta simmetrica rispetto all'asse passante per il centro.

```
import cv2
from PIL import Image
import os
from scipy import ndimage
import math
import json

json_file = open(r'C://Users/mely7/OneDrive/Desktop/export-2021-08-22T17_06_07_224Z.json')
JSON = json.load(json_file)

for counter, row in enumerate(JSON):
    file = open(r'C://Users/mely7/OneDrive/Desktop/Olive/Labels/' + str(counter) + '_fv.txt', 'w+')
    img = cv2.imread(r'C://Users/mely7/OneDrive/Desktop/Olive/Immagini/' + str(counter) + '.jpg')
    flipVertical = cv2.flip(img, 0)

    cv2.imwrite((r'C://Users/mely7/OneDrive/Desktop/Olive/Immagini/' + str(counter) + '_fv.jpg'), flipVertical)

for counter, row in enumerate(JSON):
    if row['Label'] != 'Skip':
        picture = Image.open(f'C://Users/mely7/OneDrive/Desktop/Olive/Immagini/{counter}_fv.jpg')
        picture_width, picture_height = picture.size
        with open(f'C://Users/mely7/OneDrive/Desktop/Olive/Labels/{counter}_fv.txt', 'w+') as output:
            for element in row['Label']['objects']:
                y = (element["bbox"]["top"] + element["bbox"]["height"])/2
                new_y = picture_height - y
                x_center_normalized = (element["bbox"]["left"] + element["bbox"]["width"] / 2) / picture_width
                y_center_normalized = new_y / picture_height
                width_normalized = element["bbox"]["width"] / picture_width
                height_normalized = element["bbox"]["height"] / picture_height
                output.write(f'0 {x_center_normalized} {y_center_normalized} {width_normalized} {height_normalized}\n')
```

Figura 4.8: Script per il Vertical Flip.

Il risultato ottenuto è il seguente:



Figura 4.9: Immagine flippata verticalmente.

Capitolo 5

Train con il nuovo dataset

Una volta eseguite le operazioni di *Data Augmentation* sul *dataset* originario si può procedere al nuovo train della rete neurale.

Tuttavia, il fatto di aver aumentato la grandezza del *dataset* ha creato problematiche relative all'utilizzo di **Google Colab**.

Più precisamente, la durata dell'esecuzione dell'addestramento della rete neurale eccedeva il tempo messi a disposizione per sfruttare la *macchina virtuale* offerta da *Google Colab*: la durata massima a cui può arrivare la macchina virtuale è di 12 ore ma questa andrà incontro a disconnessioni in caso di inattività prolungata. Inoltre, la durata massima e la gestione dei tempi può variare a seconda dell'utilizzo; questo affinché *Colab* possa offrire risorse computazionali gratuitamente agli utenti che intendono avvalersi del servizio di *Google*.^[15]

Utilizzando **Kaggle** è stato possibile aggirare i problemi appena descritti.^[16]

5.1 Il nuovo dataset

Il dataset risulta adesso costituito da una collezione ampliata di immagini e file `.txt`.

Attraverso il data augmentation è stato ottenuto un nuovo dataset contenente 560 elementi così divisi:

- **280 immagini**: immagini originali, blurred, ruotate e flipate
- **280 file .txt**: file contenenti le label in formato Darknet.

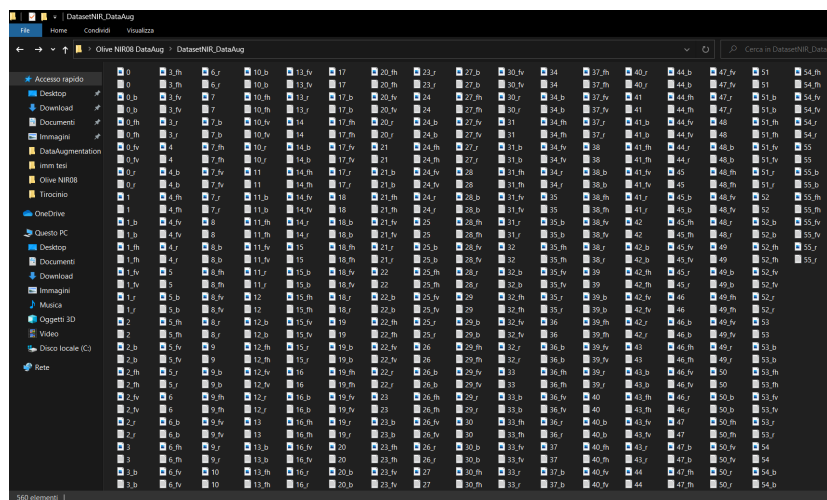


Figura 5.1: Il nuovo dataset.

5.2 Risultati

I risultati ottenuti dal nuovo train della rete neurale sono i seguenti:

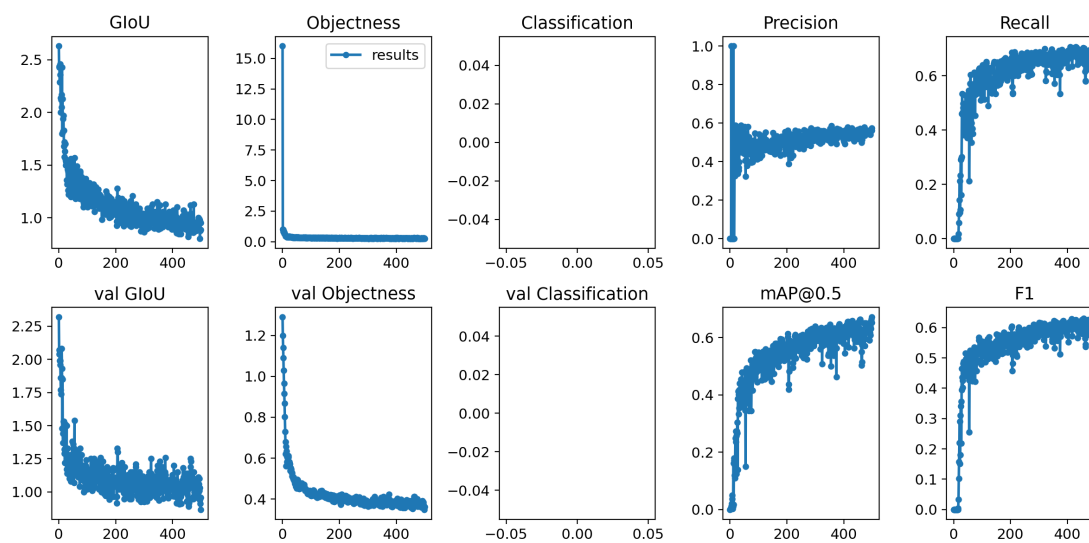


Figura 5.2: Risultati del nuovo train.

Alcune considerazioni:

1. Le funzioni di costo non presentano grandi oscillazioni grazie all'aumento del dataset.
2. Non c'è presenza di **overfitting**: il valore delle funzioni di costo **Validation GloU** e **Validation Objectness** si riduce nel tempo.

3. La funzione *Precision* è leggermente migliorata rispetto al train precedente, ma risulta essere ancora bassa.

Ulteriori test, al fine di migliorare alcuni parametri, sono stati effettuati dapprima con immagini in falso colore e in seguito sperimentando la composizione di due differenti canali NIR.

La scelta dei canali su cui effettuare il test è stata fatta tenendo in considerazione la visibilità delle olive, cercando di selezionare canali in cui le olive si distinguessero maggiormente dallo sfondo circostante.

I risultati ottenuti con il *train* della rete neurale effettuato con immagini su falso colore sono i seguenti:

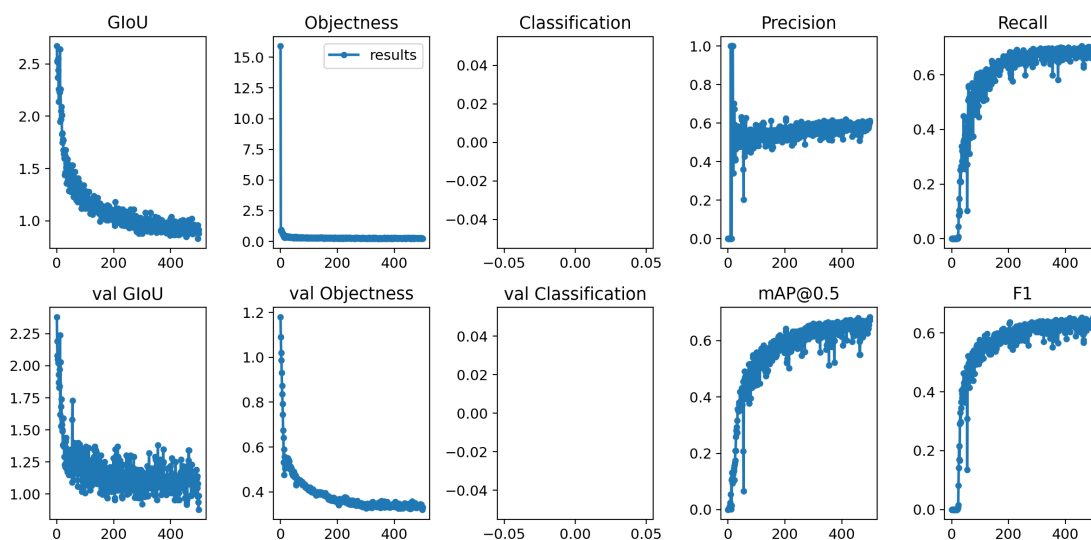


Figura 5.3: Risultati del train con finte immagini RGB.

Come si può vedere dall'immagine sovrastante, anche qui, le funzioni di costo non presentano elevate oscillazioni:

- non c'è presenza di **overfitting**: infatti le funzioni di costo **Validation Objectness** e **Validation GloU** si riducono nel tempo.
- la funzione *Precision* sembra essere ancora bassa, seppur leggermente migliorata rispetto al caso precedente.
- la funzione *Recall* è migliorata rispetto al train precedente.

Per quanto riguarda, invece, i risultati ottenuti attraverso il *train* della rete neurale, effettuato sulla composizione di due canali NIR, presentano il seguente comportamento:

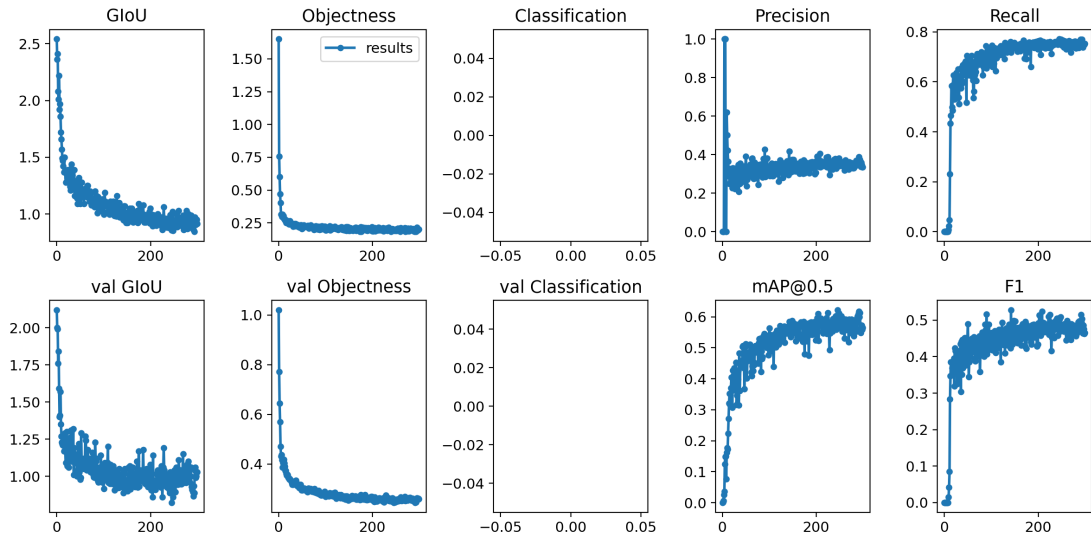


Figura 5.4: Risultati del train con composizione di due canali.

In questo caso la situazione sembra migliorare leggermente nel complesso:

- tutte le funzioni di costo non presentano oscillazioni importanti
- non c'è presenza di **overfitting**
- la funzione *Precision* si è abbassata notevolmente
- la funzione *Recall* è migliorata in maniera importante.

Conclusioni e sviluppi futuri

La rete neurale addestrata fino ad adesso ha ottenuto risultati abbastanza soddisfacenti, questo perché le immagini utilizzate presentano più disturbi e inquadrature da maggiori distanze, sebbene abbiano ancora margine di miglioramento.

Negli ultimi anni sono stati realizzati altri algoritmi mirati alla stima del grado di maturazione dei frutti con risultati molto promettenti. L'*Università di Campinas*, in concerto con la *Londrina State University*, ha messo a punto un algoritmo per la stima del grado di maturazione dei frutti della pianta della papaya. I risultati ottenuti sono estremamente promettenti: si è raggiunta una precisione del 94,7% che però si riduce fino all' 84,4% per campioni all'ultimo stato di maturazione. Questo perché il colore della buccia non corrisponde perfettamente all'effettiva maturazione della polpa; motivo per cui lo studio verrà nuovamente ripreso in futuro andando ad aumentare il numero di campioni da analizzare ¹.

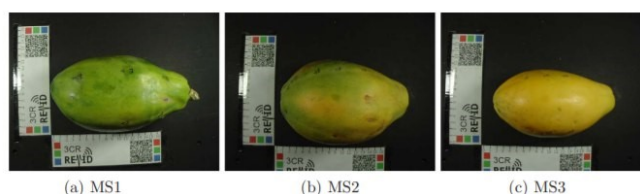


Fig. 3. Sample images from each maturity stage of the papaya fruit. The three maturity stages MS1, MS2, and MS3 are derived from pulp firmness measurements.

Figura 5.5: Classi di maturazione del frutto della papaya.

Fonte: L.F-Santos-Pereira-et-al.

¹Predicting the ripening of papaya fruit with digital imaging and random forests - L.F Santos Pereira et al, 2018

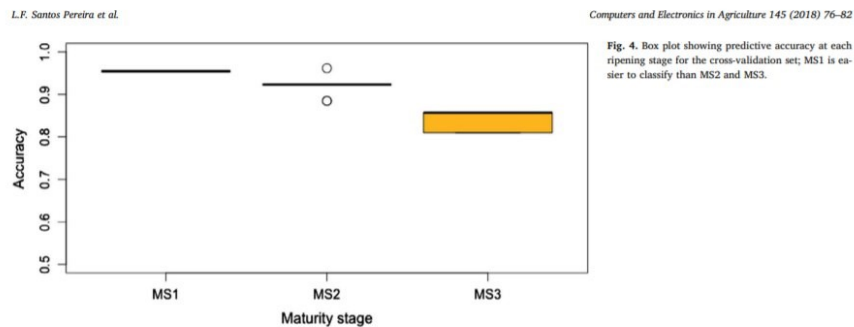


Figura 5.6: Accuratezza predetta ad ogni stato di maturazione.

Fonte: L.F-Santos-Pereira-et-al.

Questa tecnica è stata sfruttata anche per valutare il grado di maturazione delle fragole²: in questo caso si è arrivati ad ottenere una precisione del 98,6% utilizzando una rete neurale *pretrained AlexNet CNN*.

In questo caso, gli autori di questo studio, hanno analizzato immagini iper-spetttrali (con caratteristiche simili a quelle multi-spetttrali) catturate in condizioni di ottima visibilità ottenendo risultati promettenti che tuttavia non consentono un utilizzo a portata di tutti nel proprio smartphone. Questo perché le immagini analizzate sono state scattate con la pretesa di voler essere quanto più vicino possibile ad un *dataset* ideale (foto scattate in condizioni di ottima visibilità: ad esempio un frutto nascosto o poco visibile o lontano dall'obiettivo non viene incluso nel *dataset*).

L'obiettivo della tesi è quindi raggiunto: il modello convoluzionale di *deep learning* è in grado di riconoscere con una accuratezza soddisfacente le olive utilizzando come *dataset* immagini multi-spetttrali catturate direttamente sul campo. Una volta generalizzata la rete al riconoscimento delle olive sarà finalmente possibile addestrare una nuova rete al fine di trovare la classe di maturazione. Ciò richiede un insieme più grandi di immagini al fine di migliorare le metriche rendendo il sistema più vicino al mondo dell'utilizzatore finale.

²Real-time hyperspectral imaging for the in-field estimation of strawberry ripeness with deep learning - Z. Gao et al, 2020

Bibliografia

- [1] *What is Agriculture 4.0*. URL: <https://www.eib.org/en/stories/what-is-agriculture-4-0/>.
- [2] *Deep Learning (appendimento approfondito)*. URL: <https://www.intelligenzaartificiale.it/deep-learning/>.
- [3] *Appendimento Approfondito*. URL: https://it.wikipedia.org/wiki/Appendimento_profondo/.
- [4] *Reti neurali, cosa sono*. URL: <https://www.intelligenzaartificiale.it/reti-neurali/>.
- [5] *Artificial Neural Network*. URL: <https://www.sciencedirect.com/topics/engineering/artificial-neural-network/>.
- [6] Anas Al-Masri. *How does Back-Propagation in Artificial Neural Network Work*. URL: <https://towardsdatascience.com/how-does-back-propagation-in-artificial-neural-networks-work-c7cad873ea7>.
- [7] *Overfitting*. URL: <https://it.wikipedia.org/wiki/Overfitting%22>.
- [8] *Introduction to YOLO Algorithm for Object Detection*. URL: <https://www.section.io/engineering-education/introduction-to-yolo-algorithm-for-object-detection/%22>.
- [9] Jonathan Hui. *Real-time object Detection with YOLO, YOLOv2 and now YOLOv3*. URL: <https://jonathan-hui.medium.com/real-time-object-detection-with-yolo-yolov2-28b1b93e2088>.
- [10] Hacker Noon. *Understanding YOLO*. URL: <https://hackernoon.com/understanding-yolo-f5a74bbc7967>.
- [11] Eric Hofesmann. *IoU a better detection evaluation metric*. URL: <https://towardsdatascience.com/iou-a-better-detection-evaluation-metric-45a511185be1>.
- [12] *Precision and Recall con F1 Score*. URL: <https://andreaprovino.it/precision-and-recall-precisione-e-recupero/>.
- [13] *GitHub*. URL: <https://it.wikipedia.org/wiki/GitHub>.
- [14] *Augmented data: cosa sono e a che serve la data augmentation*. URL: <https://www.bigdata4innovation.it/big-data/augmented-data/>.

-
- [15] *Google Colab: il tool gratuito di Google a servizio dei data scientist*. URL: <https://www.miriade.it/google-colab-il-tool-gratuito-di-google-a-servizio-dei-data-scientist/>.
- [16] *Kaggle*. URL: <https://en.wikipedia.org/wiki/Kaggle>.

Elenco delle figure

1.1	Schema generale dell'Agricoltura 4.0	2
2.1	Struttura dell'Intelligenza Artificiale	6
2.2	Confronto tra Rete Neurale Biologica e Artificiale.	7
2.3	Error Back-Propagation	8
2.4	Architettura di YOLO	9
2.5	Griglia SxS di YOLO	10
2.6	Formula per il calcolo della Loss Function.	10
2.7	Formula per il calcolo della Classification Loss.	11
2.8	Formula per il calcolo della Localization Loss.	11
2.9	Formula per il calcolo della Confidence Loss.	12
2.10	Formula per il calcolo dell'IoU.	12
2.11	Formule per il calcolo di Precision, Recall e F1.	13
2.12	Esempio di curva Precision-Recall.	13
2.13	Calcolo dell'AP.	14
2.14	Iperparametri di YOLO.	14
2.15	Logo di Labelbox.	15
2.16	Logo di Python.	15
2.17	Logo di Google Colab.	15
2.18	Logo di Kaggle.	16
2.19	Logo di GitHub.	16
3.1	Schema generale di una singola Label.	19
3.2	Codice del Parsing.	20
3.3	Overview del dataset.	21
3.4	Risultati del primo train.	22
4.1	Generazione dell'immagine-copia ruotata.	25
4.2	Calcolo nuove coordinate.	26
4.3	Immagine ruotata.	26
4.4	Script per il blurring.	27
4.5	Immagine con effetto blur.	27
4.6	Script per l'Horizontal Flip.	28
4.7	Immagine flippata orizzontalmente.	28
4.8	Script per il Vertical Flip.	29

4.9	Immagine flippata verticalmente.	29
5.1	Il nuovo dataset.	32
5.2	Risultati del nuovo train.	32
5.3	Risultati del train con finte immagini RGB.	33
5.4	Risultati del train con composizione di due canali.	34
5.5	Classi di maturazione del frutto della papaya.	36
5.6	Accuratezza predetta ad ogni stato di maturazione.	37