Politecnico di Torino

Politecnico di Torino

Computer Networks and Cloud Computing

A.a 2023/2024

Sessione di Laurea Luglio 2024

DIVINE

DIagnosi delle malattie della VIte per immagini tramite le reti NEurali e il deep learning

Relatore
Prof. BOTTINO Andrea

Candidato
SANINO Fabrizio

Sommario

I cambiamenti climatici stanno mettendo a dura prova le coltivazioni: si stima che il 40% delle colture a livello mondiale muoia prima del raccolto a causa delle malattie. A livello nazionale, una tra le piantagioni più importanti è la vite. L'irregolarità delle precipitazioni e l'aumento delle temperature facilitano la proliferazione di malattie funginee su questo tipo di piante. Per proteggere i vigneti, si è dunque costretti ad incrementare le quantità di agrofarmaci utilizzati. Inoltre, nella maggior parte dei casi, il trattamento viene eseguito senza discriminazione su tutta la coltivazione, poichè non si dispone di strumenti diagnostici con un livello di dettaglio sufficiente per consentire un intervento mirato. L'uso massiccio di prodotti chimici, derivato dalle cause appena descritte, comporta non solo una crescita dei costi di produzione, ma soprattutto effetti negativi sulla salute umana e sull'ambiente.

Partendo da queste problematiche, gli obiettivi principali di questo studio sono tre: (1) analizzare lo stato dell'arte dei metodi per la diagnosi automatica delle malattie nelle vigne, esplorando varie tecniche e modelli derivanti dal campo dell'Intelligenza Artificiale (IA), in particolar modo del Deep Learning; (2) definire ed realizzare una procedura di collezione dei dati che sfrutti strumenti e metodi di ultima generazione, come l'annotazione assistita dall'IA, al fine di creare una dataset specifico per diverse malattie delle vigne, in grado di fornire un supporto utile all'addestramento delle reti neurali; (3) implementare ed addestrare una rete neurale in grado di facilitare la diagnosi automatica delle malattie nelle vigne, facilmente addestrabile per nuove patologie. Per raggiungere tali obiettivi, la Computer Vision offre diverse soluzioni, tra cui tecniche di Classificazione e di Object Detection. Tali sistemi vengono dunque impiegati per analizzare fotografie di piante e pareti fogliari con lo scopo di individuare la posizione delle foglie all'interno dell'immagine ed indicare per ogni foglia se questa è sana o malata, specificandone l'eventuale malattia. Queste tecnologie quindi rendono possibile l'intervento con prodotti chimici esclusivamente nelle aree di interesse evitando inutili sprechi e supportando l'ambiente.

Dalla revisione della letteratura è emerso che non esiste un dataset ampio e

pubblico adatto ad addestrare modelli di Object Detection e Classificazione per le malattie delle vigne. Tuttavia sono stati individuati 4 dataset, che dopo essere stati trattati, annotati e migliorati, sono stati unificati nel Vine Leaf Vision Dataset. Questo nuovo dataset contiene più di 10.000 immagini di foglie e pareti fogliari di viti, con oltre 30.000 annotazioni (bounding box). Le annotazioni, raccolte da operatori non esperti, utilizzano etichette basate su pattern visivi piuttosto che su diagnosi specifiche, limitando così le ambiguità derivanti dall'inesperienza degli annotatori. Per valutare l'efficacia del dataset sono stati condotti diversi esperimenti, concentrandosi sul fine-tuning cioè la capacità dei modelli di adattarsi nell'identificazione di una specifica patologia. Le analisi effettuate su alcuni modelli di object detection hanno dimostrato come un addestramento sui dati del Vine Leaf Vision Dataset consente ai modelli di migliorare significativamente le loro performance quando successivamente subiscono un'operazione di fine-tuning su nuove malattie non presenti nel dataset.

"da la meta mai
Non torcer gli occhi"
Alessandro Manzoni

Indice

Elenco delle tabelle VIII

Elenco delle figure IX

Acronimi XII

1 Introduzione 1

2 Panoramica della letteratura 4
2.1 Classificazione 4
2.2 Object detection 8
2.3 Costruzione del dataset 10

3 Metodologie 13
3.1 Collezione dei dati 13
3.1.1 ESCA-dataset 14
3.1.2 Grapevine Disease Images 18
3.1.3 Vitis Vinifera 20
3.1.4 Plants Diseases Detection and Classification Image Dataset . 25
3.2 Strumento per l'annotazione 26
3.2.1 Machine Learning backend 27
3.3 Unione dei dataset 29
3.4 Data augmentation 30
3.5 Metriche per il confronto dei risultati 31
3.5.1 Metriche per i modelli di classificazione 31
3.5.2 Metriche per i modelli di object detection 32
3.6 Operazione di Classificazione 34
3.7 Operazione di Object Detection 39

4 Architettura dei modelli utilizzati 43
4.1 Classificazione 43

4.1.1 ResNet - Residual Neural Network . . . 43
4.1.2 ViT - Vision Transformer . . . 46
4.1.3 MobileNet . . . 49
4.2 Object Detection . . . 51
4.2.1 YOLO - You Only Look Once . . . 51
4.2.2 DETR - DEtection TRansformer . . . 55

5 Risultati 58
5.1 Classificazione . . . 58
5.2 Object Detection . . . 60

6 Conclusioni 64
6.1 Classificazione . . . 64
6.2 Object Detection . . . 64
6.3 Sviluppi futuri . . . 65

A Operazioni tra tensori 68
A.1 Convoluzione [36] . . . 68
A.2 ReLU [38] . . . 69
A.3 Softmax [39] . . . 69

Bibliografia 71

Elenco delle tabelle

3.1 Distribuzione delle classi all'interno di ESCA-dataset 15
3.2 Distribuzione delle classi all'interno di Grapevine Disease Images . . 19
3.3 Contenuto di Vitis Vinifera dopo l'operazione di annotazione 24
3.4 Contenuto di Plants Diseases Detection and Classification Image Dataset 26
3.5 Numero di immagini per ciascuna partizione derivanti dalla suddivisione di Vine Leaf Vision Dataset 40

5.1 Caratteristiche dei modelli di classificazione comparati 59
5.2 Risultati dell'operazione di classificazione raccolti utilizzando la partizione di test di ESCA-dataset 59
5.3 Specifiche dei modelli di object detection comparati 60
5.4 Risultati per gli esperimenti del gruppo A, raccolti utilizzando la partizione di test del dataset di fine-tuning 60
5.5 Comparazione tra il miglior esperimento del gruppo A rispetto all'architettura DETR utilizzando la partizione di test del dataset di fine-tuning 61
5.6 Risultati dell'operazione di object detection raccolti utilizzando la partizione di test del dataset di training 62

Elenco delle figure

2.1 Esempio del processo di classificazione delle immagini 5
2.2 La prima colonna mostra le immagini di input, la seconda le stesse dopo la conversione nel CIELAB color model, infine, la terza presenta i risultati dell'operazione di clustering eseguita dall'algoritmo k-means. Le aree con lo stesso colore indicano i componenti del medesimo cluster. Fonte: [2] 6
2.3 Risultato ottenuto dopo l'elaborazione di un'immagine tramite un modello di Object Detection. Fonte: [10] 8

3.1 Fotografia etichettata come esca 15
3.2 Fotografia etichettata come healthy 15
3.3 Pattern visivo di una foglia classificata come esca 17
3.4 Pattern visivo di una foglia classificata come healthy 17
3.5 Pattern visivo di una foglia classificata come white speckle dapple . 17
3.6 Pattern visivo di una foglia classificata come golden fleck healthy . . 17
3.7 Pattern visivo di una foglia classificata come golden fleck esca . . . 17
3.8 Pattern visivo di una foglia classificata come partially visible 17
3.9 Numero di foglie per ogni etichetta in ESCA-dataset 18
3.10 Esempio di una foglia classificata come healthy 20
3.11 Esempio di una foglia classificata come black rot 20
3.12 Esempio di una foglia classificata come esca 20
3.13 Esempio di una foglia classificata come leaf blight 20
3.14 Esempio immagine vitis vinifera verde 21
3.15 Esempio immagine vitis vinifera a chiazze rosse 21
3.16 Foglia estratta dall'immagine intera utilizzando le coordinate della bounding box 24
3.17 Maschera per ottenere la sola foglia all'interno della bounding box, ottenuta invertendo la maschera dello sfondo. La maschera è rappresentata dal colore giallo 24
3.18 Maschera per ottenere l'area verde della foglia. Siccome la foglia è verde, la quantità di colore giallo è predominante 24

3.19 Grafico che mostra, per ogni foglia classificata come *healthy*, l'area della parte rossa. I punti di colore rosso indicano gli outliers cioè le foglie la cui area del colore rosso si discosta dalla media 25
3.20 Fotografia che mostra le bounding box a cui è associata l'etichetta grape leaf 26
3.21 Fotografia che mostra le bounding box a cui è associata l'etichetta grape leaf black rot 26
3.22 Esempio di un'immagine dopo l'operazione di annotazione 28
3.23 Numero di foglie per ogni etichetta in Vitis Leaf Vision. Tra le parentesi tonde è indicato il valore numerico della classe 30
3.24 Matrice di confusione. Fonte: [16] 31
3.25 IoU 33
3.26 Passaggi per il calcolo dell'average precision su di una singola classe. Fonte: [18] 34
3.27 Dettaglio dell'architettura sviluppata dall'articolo. Fonte: [4] 36

4.1 Architettura ResNet32. I collegamenti tra i vari layer indicano le skip-connections, quelli tratteggiati quando c'è anche un cambiamento della dimensione di input. Fonte: [21] 44
4.2 Residual block. Fonte: [19] 45
4.3 Transformer Encoder. Fonte: [23] 47
4.4 Mappa di self-attention. Fonte: [25] 47
4.5 Architettura ViT. Fonte: [23] 48
4.6 Architettura MobileNet. In blu è rappresentato il dettaglio della depthwise separable convolution. Fonte: [28] 50
4.7 Architettura YOLO. Fonte: [30] 52
4.8 Funzionamento YOLO. Fonte: [31] 53
4.9 Architettura DETR. Fonte: [32] 56

5.1 Andamento del training loss per l'esperimento A4 62
5.2 Andamento del validation loss per l'esperimento A4 63

A.1 Operazione di convoluzione. Si parte con il filtro in alto a sinistra e si finisce con il pixel in basso a destra. Fonte: [37] 68

Acronimi

DIVINE
Diagnosi delle malattie della vite per immagini tramite la reti neurali

IA
Intelligenza artificiale

ML
Machine Learning

DL
Deep Learning

CNN
Convolutional Neural Network

AP
Average Precision

SVM
Support Vector Machine

Capitolo 1

Introduzione

L'Italia è uno tra i principali produttori di vino al mondo. I cambiamenti climatici però stanno mettendo a dura prova le coltivazioni viticole. Le recenti fluttuazioni, caratterizzate da eventi di precipitazioni estreme, seguite da periodi prolungati di siccità ed un significativo aumento delle temperature durante l'autunno hanno portato e porteranno problematiche nella gestione del suolo e delle risorse idriche per la viticoltura. Questo comporterà, oltre ad una riduzione delle aree adatte alla coltivazione, ad un aumento dell'incidenza e della gravità delle malattie delle piante, rendendo necessaria una ristrutturazione delle pratiche fitosanitarie[1]. Le principali malattie che colpiscono la vite sono di natura fungina. Queste possono causare una riduzione della resa e della qualità del raccolto, portando a consistenti perdite economiche. Di conseguenza, la maggior parte dei trattamenti sulla vite sono mirati alla difesa contro queste malattie, con una media di 6-12 trattamenti a stagione, a seconda delle condizioni climatiche. Questi interventi comportano costi significativi sia dal punto di vista economico che soprattutto dal punto di vista ambientale.

Generalmente, la diagnosi della malattia di una pianta viene effettuata da un esperto che, dopo aver attentamente esaminato un campione, definisce qual è la malattia che l'ha colpito e quali sono i trattamenti che dovranno essere effettuati [1]. Questo metodo però è molto costoso, richiede molto tempo e può essere impreciso. Grazie alla tecnologia e sopratutto all'**intelligenza artificiale (IA)** e il **deep learning (DL)** è possibile effettuare la diagnosi delle malattie in modo automatico, preciso ed efficiente.

Le prime ricerche nell'ambito dell'IA si sono focalizzate sull'utilizzo di algoritmi di **machine learning (ML)** come **k-means clustering** e quindi basate

[1]Insieme di pratiche volte al controllo di organismi nocivi per le coltivazioni viticole.

sull'elaborazione deterministica delle immagini [2]. Il problema di questi metodi sono il complesso e costoso processo di trattamento iniziale delle immagini e l'estrazione successiva delle **features**[2]. Inoltre, il risultato è insoddisfacente per quanto riguarda l'accuratezza e la precisione nell'identificazione della malattia.

Successivamente, si è passati all'utilizzo di algoritmi più avanzati e veloci di DL come le **convolutional neural network (CNN)**. Queste reti estraggono le features direttamente da esempi di immagini evitando le manipolazioni complesse degli algoritmi di ML e quindi riducendo anche le capacità computazionali richieste per l'esecuzione di questi algoritmi [3] [4].

Un ulteriore passo in avanti è stato compiuto con l'introduzione dell'**object detection** cioè la possibilità di individuare all'interno dell'immagine degli oggetti. Queste tecniche permettono di fornire una granularità più fine e quindi scendere più nel dettaglio della classificazione perché invece di restituire come output una sola classe per immagine, è possibile indicare la posizione delle singole foglie ed assegnarne la classe di appartenenza. Queste tecnologie risultano però meno efficienti e richiedono quindi una disponibilità computazionale più elevata [5] [6] [7].

Infine, è cruciale sottolineare l'importanza della creazione di un **dataset** ben strutturato. La qualità di quest'ultimo rappresenta la base fondamentale per l'addestramento accurato di modelli di DL. Un insieme diversificato e ben etichettato di immagini consente ai modelli di apprendere in modo efficace e di generalizzare meglio su nuovi dati, migliorando così le prestazioni complessive del sistema di visione artificiale [4] [5] [8] [7] [9].

Gli obiettivi principali di questo studio sono quindi tre: (1) analizzare lo stato dell'arte dei metodi per la diagnosi automatica delle malattie nelle vigne, esplorando varie tecniche e modelli derivanti dal campo dell'Intelligenza Artificiale (IA), in particolar modo del Deep Learning. Esaminare i dataset disponibili e le metodologie di raccolta e annotazione di tali dati; (2) definire ed implementare una procedura di collezione dei dati che sfrutti strumenti e metodi di ultima generazione, come l'annotazione assistita dall'IA, al fine di creare un dataset specifico per diverse malattie delle vigne, in grado di fornire un supporto utile all'addestramento delle reti neurali; (3) implementare ed addestrare una rete neurale in grado di facilitare la diagnosi automatica delle malattie nelle vigne, facilmente addestrabile per nuove patologie.

[2]Estrazione di informazioni rilevanti che sono contenute all'interno di un'immagine.

La tesi è così organizzata:

- **Capitolo 2 Panoramica della letteratura**: questo capitolo contiene una panoramica della ricerca negli ambiti trattati dalla tesi, quali sono le problematiche riscontrate da questi studi e quali sono i margini di miglioramento.
- **Capitolo 3 Metodologie**: questo capitolo contiene delucidazioni su quali sono stati i passaggi compiuti per creare il set di dati e le metriche con cui verranno creati e confrontati gli esperimenti.
- **Capitolo 4 Modelli**: questo capitolo tratta nel dettaglio le architetture dei modelli utilizzati, come questi funzionano e quali sono i vantaggi e svantaggi di ciascuna.
- **Capitolo 5 Risultati**: questo capitolo presenta ed analizza i risultati ottenuti utilizzando le metriche ed applicando le metodologie ed i modelli descritti nei capitoli precedenti.
- **Capitolo 6 Conclusioni**: questo capitolo illustra i risultati in relazione agli obiettivi stabiliti e suggerisce potenziali sviluppi futuri.

I risultati e gli algoritmi utilizzati nel progetto sono fruibili attraverso GitHub al seguente link: `https://github.com/fabriziosanino/grape_leaves_detection`.

Capitolo 2

Panoramica della letteratura

2.1 Classificazione

La **classificazione** è una tecnica di DL di tipo **supervisionato** il cui obiettivo è la creazione di un modello capace di categorizzare l'input ricevuto in una **classe** o **etichetta** scelta da un insieme predefinito. Nel caso specifico della seguente tesi, l'input che verrà fornito al modello sarà un immagine che contiene foglie di viti, sia singole che intere pareti.

Esistono vari tipi di classificazione tra cui la classificazione **binaria** e **multi-classe**. La prima si occupa di assegnare un'etichetta al dato di input scegliendola tra un gruppo di due, le cui opzioni sono mutuamente esclusive (es. spam o non spam). La seconda, invece, assegna un'etichetta scegliendola però da un gruppo più ampio di possibili categorie.

I modelli, per poter funzionare, devono essere allenati su di un insieme di dati già etichettati, quindi, per ogni immagine dovrà essere associata l'indicazione della classe di appartenenza ed è questo il motivo per cui vengono chiamati supervisionati. In questo modo, durante la fase di addestramento, il modello imparerà quali sono i pattern o gli attributi comuni delle diverse classi e sarà in grado di predire, durante la fase di inferenza, la classe di appartenenza delle nuove immagini che non sono mai state analizzate o elaborate dal modello (Figura 2.1).

La classificazione delle malattie sulle foglie effettuata dagli algoritmi permette di velocizzare l'ottenimento del risultato. Come descritto in [1], l'operazione di diagnosi di una malattia su di una foglia o pianta fatta a mano da una persona richiede molto tempo e sopratutto sono necessari professionisti del settore. La categorizzazione delle malattie deve essere compiuta attraversando diverse fasi: la prima cosa che deve essere fatta, un volta che il campione da esaminare è stato raccolto, è l'identificazione della specie. Questo perché, in base a quest'ultima, l'esperto dovrà concentrarsi su di una specifica lista di malattie ed inoltre data la

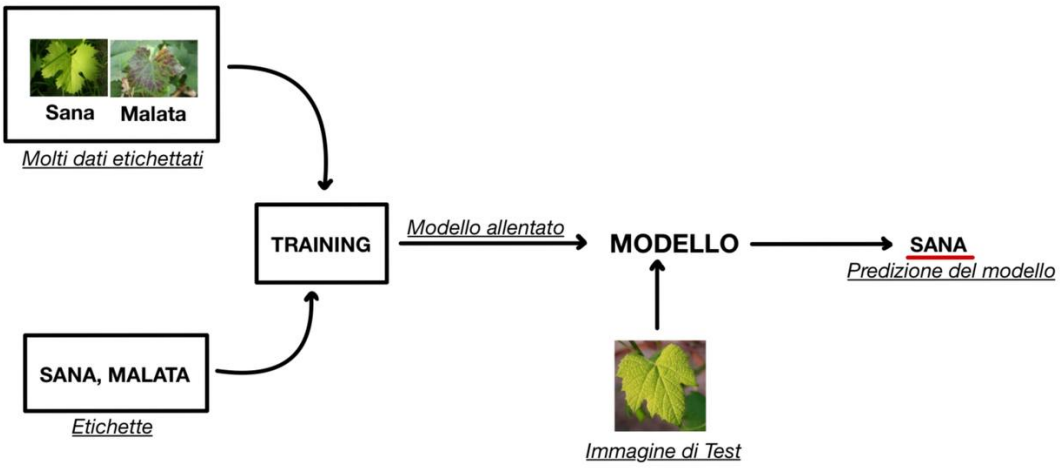

Figura 2.1: Esempio del processo di classificazione delle immagini

specie è possibile conoscere qual è il pattern visivo che un campione sano dovrebbe avere. Completata questa fase si può passare ad esaminare se il campione presenta i sintomi caratteristici di una determinata malattia e la quantità di foglia coperta da essa. Grazie a questo si può determinare lo stadio di avanzamento della malattia e quindi delineare il percorso da seguire per riportare in salute la pianta. Ovviamente, se ciò non bastasse, si può proseguire con un'analisi chimica in laboratorio per ottenere più dettagli. La procedura appena descritta permette di comprendere quanti e quali siano i costi e gli sforzi da compiere da parte di un esperto per poter determinare la quantità di prodotto fitosanitario da utilizzare.

Al fine di ridurre il tempo impiegato nella diagnosi, la ricerca è dunque passata all'utilizzo di algoritmi automatici. Le prime soluzioni che sono state proposte utilizzano tecniche di **clustering** come il **k-means**. Queste tecnologie sono di tipo **non supervisionato** cioè apprendono da dati non etichettati permettendo di identificare strutture e/o gruppi nascosti all'interno dei dati stessi. In particolare, il clustering è una tecnica che raggruppa i dati in input fornendo insiemi di elementi tra loro coerenti.

In [2] è stato utilizzato l'algoritmo k-means per classificare quattro tipi di malattie presenti in fotografie di foglie acquisite in laboratorio. Nello specifico, i pixel delle immagini vengono raggruppati in base a somiglianze tra i loro valori cromatici. Prima di procedere con il clustering sono state eseguite alcune operazioni di elaborazione; come prima cosa sono state utilizzate alcune tecniche per rimuovere il rumore e aumentare il contrasto all'interno dell'immagine. Successivamente, la stessa è stata convertita utilizzando il **CIELAB color model** per poter essere

processata in modo corretto dall'algoritmo di clustering. L'output ottenuto è stato utilizzato per etichettare ciascun pixel all'interno dell'immagine in base ai gruppi e similitudini scovate (Figura 2.2). Infine, tramite il classificatore **support vector machine (SVM)**, le features estratte dal clustering sono state utilizzate come risorsa di allenamento su cinque diverse malattie. Il SVM è un algoritmo di apprendimento supervisionato il cui obbiettivo è trovare l'iperpiano migliore in uno spazio multidimensionale che separi in modo ottimale gli elementi dei diversi gruppi. L'iperpiano in questione viene trovato massimizzando la distanza tra i punti più vicini delle diverse classi. Nel caso della ricerca citata l'obiettivo è di raggruppare i pixel che presentano la stessa malattia.

L'accuratezza, indipendentemente dal tipo di malattia, è risultata superiore al 95%. Un ostacolo comune in tecniche di questo tipo è il livello di complessità nell'elaborazione iniziale delle immagini che si traduce in tempi più lunghi prima di poter utilizzare i risultati ottenuti.

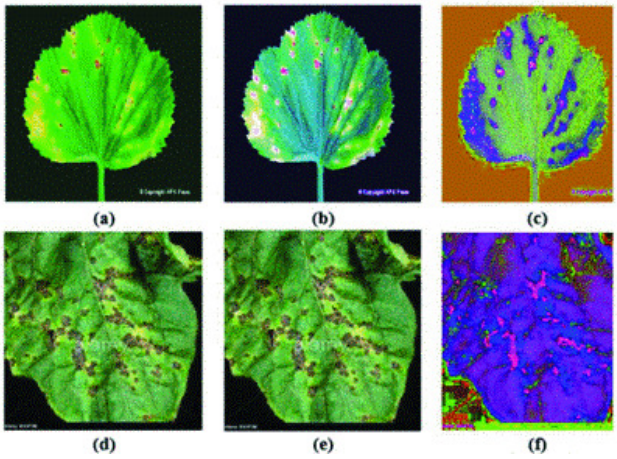

Figura 2.2: La prima colonna mostra le immagini di input, la seconda le stesse dopo la conversione nel CIELAB color model, infine, la terza presenta i risultati dell'operazione di clustering eseguita dall'algoritmo k-means. Le aree con lo stesso colore indicano i componenti del medesimo cluster. Fonte: [2]

Soluzioni più avanzate tecnologicamente sono quelle che si basano sul DL utilizzando le **CNN**. Le reti in questione hanno un potenziale più elevato in termini di accuratezza rispetto ad altre architetture di reti neurali, specialmente nel riconoscimento di immagini e dell'elaborazione visiva.

La ricerca [3] ha comparato diversi studi che utilizzano modelli di DL per il riconoscimento di malattie sulle foglie generiche di piante. Sono stati trattati modelli più vecchi come AlexNet (2012) per arrivare fino a DeepLab (2019). E'

stato evidenziato come soprattutto i modelli di nuova generazione (per l'epoca della ricerca) riescano a raggiungere un livello di accuratezza molto alto. Nonostante ciò, sono state portate alla luce alcune criticità e lacune in molti degli studi analizzati:

- nella maggior parte delle ricerche ed articoli esaminati il dataset utilizzato per valutare l'accuratezza ed allenare le reti neurali è **PlantVillage**. Questo insieme comprende una vasta collezione di immagini (circa 54.000) di foglie, sia sane che malate, e piante appartenenti a specie diverse. Il problema è che le fotografie sono scattate in laboratorio quindi lo sfondo dell'immagine è artificiale. La ricerca suggerisce come, in uno scenario più pratico come quello della tesi dove lo strumento da realizzare verrà applicato in campo, dovranno essere utilizzati dataset più affini alla realtà cioè che contengano immagini di pareti fogliari e non solo fotografie scattate in laboratorio.

- la **severità di una malattia varia nel tempo e nelle forme** pertanto i modelli creati dovranno essere continuamente aggiornati ed allenati per riconoscere questi cambiamenti. Ciò implica la necessità di creare un dataset ampio e comprensivo capace di coprire tutte le possibili varianti e scenari della malattia. Inoltre, è essenziale includere dati provenienti da diverse fonti e contesti per garantire che il modello mantenga un'elevata accuratezza e robustezza nel riconoscimento e nella classificazione dei vari stadi e manifestazioni della malattia.

- il modello creato deve essere efficiente indipendentemente dalle **condizioni di illuminazione** delle immagini. Di conseguenza, il dataset non deve includere solo immagini che hanno un'esposizione perfetta ma anche immagini con un'illuminazione ridotta, dove le foglie risultano meno visibili. È fondamentale che il set di dati presenti una varietà di condizioni di illuminazione, comprese ombre, riflessi e luce soffusa, per garantire che il modello possa generalizzare in modo corretto e mantenere alte le prestazioni anche in situazioni di illuminazione non ottimale.

Sono inoltre presenti alcune ricerche che si concentrano solo sulla diagnosi di malattie delle foglie di vite. L'articolo [4] del 2021 si è focalizzato sulla creazione di una rete CNN ad hoc specifica per il riconoscimento della malattia *mal dell'esca della vite*. Questo perché la malattia in questione può arrecare gravi danni alle coltivazioni arrivando fino a causare la morte delle piante affette. Come prima cosa, lo studio si è preoccupato di creare il dataset. Quest'ultimo contiene 1770 immagini raccolte in campo, quindi pareti fogliari con sfondi reali e sono presenti sia fotografie in cui sono visibili foglie sane sia malate. Siccome il numero di immagini non è risultato sufficiente per ottenere buoni risultati in termini di classificazione, tramite procedure di **data augmentation** è stato aumentato a 24780 immagini.

La rete neurale utilizzata è stata progettata e creata utilizzando le operazioni di **convoluzione** dato che l'obbiettivo dello studio era la creazione di una CNN. Il risultato ottenuto è soddisfacente in termini di prestazioni infatti raggiunge addirittura il 99.16% di accuratezza su immagini in input di dimensione 1280 x 720 (pixel). Un altro significativo contributo dell'articolo è la messa a disposizione del dataset creato per studi e sviluppi futuri.

2.2 Object detection

Le tecniche di classificazione permettono di ottenere buoni risultati nella diagnosi delle malattie delle viti ma sono limitate. Infatti, in una sola immagine possono esserci più foglie e quindi anche diversi tipi di malattie. La classificazione non è in grado di distinguerle perché per ogni immagine in input viene restituita una sola classe. Un miglioramento può essere portato implementando l'**object detection**. Questa tecnica è progettata per individuare e classificare gli oggetti che sono presenti all'interno di un'immagine fornendo non solo l'identificazione ma anche la posizione precisa. L'informazione aggiuntiva della posizione fornita da questo tipo di reti neutrali è estremamente utile per applicare i fitofarmaci in modo mirato ed efficace, riducendo lo spreco di sostanze chimiche e minimizzando l'impatto ambientale. Inoltre, l'uso dell'object detection può migliorare significativamente la gestione delle risorse agricole, premettendo un monitoraggio più accurato della salute delle piante ed una risposta tempestiva ad eventuali problemi di salute.

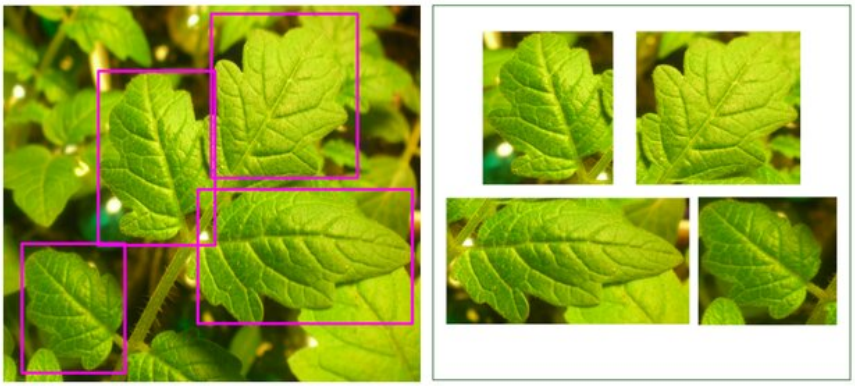

Figura 2.3: Risultato ottenuto dopo l'elaborazione di un'immagine tramite un modello di Object Detection. Fonte: [10]

Ci sono vari articoli pubblicati che trattano il tema dell'object detection applicato alla diagnosi delle malattie delle piante. Nello specifico, [6] tratta le malattie che

affliggono le foglie di pomodoro (*oidio*, *peronospora*, *muffa* e *virus del mosaico del pomodoro*). L'obbiettivo della ricerca è stato quello di migliorare il modello **Faster RCNN** già esistente. Le operazioni di modifica che sono state compiute per migliorarne le prestazioni sono le seguenti:

- è stato sostituito il *feature extractor* di base ovvero VGG16 con una *depth residual network* ovvero **ResNet**. Questo perché ResNet adotta una struttura più profonda rispetto a VGG16 e quindi ciò migliora le capacità di estrazione delle features. Inoltre, l'architettura di ResNet risolve il problema della dispersione del gradiente all'aumentare del numero di layers (Sezione 4.1.1).

- è stato introdotto l'algoritmo di clustering k-means in modo tale da aggregare le bounding box generate al fine di ottimizzare l'ancoraggio predefinito impiegato dalla **region proposal network (RPN)** presente nell'architettura di Faser RCNN. Questo consente di migliorare l'attendibilità e la precisione delle regioni di interesse proposte dalla rete.

Il dataset utilizzato per la verifica della rete creata è composto da 4178 immagini. Le ottimizzazioni e innovazioni apportate a Faster RCNN hanno portato il valore di mAP al 98.54% pari ad un incremento di 1.5 punti percentuali delle prestazioni rispetto alla rete non modificata. Questo evidenzia come i modelli, opportunamente adattati al contesto di utilizzo, possano raggiungere una granularità estremamente alta. Tuttavia, l'unico inconveniente di questo studio è l'utilizzato di un dataset composto esclusivamente da immagini acquisite in laboratorio. Di conseguenza, i livelli di accuratezza riportati potrebbero risultare superiori a quelli ottenibili nelle reali condizione operative.

Esistono anche studi più specifici che si concentrano solo sulle malattie delle viti. L'articolo [5] del 2020 si è focalizzato sullo sviluppo di una rete CNN personalizzata per diagnosticare e localizzare le quattro malattie più comuni delle viti: *mal dell'esca*, *marciume nero*, *peronospora* e *acaro del grappolo*. Lo scopo principale di questa ricerca è colmare le lacune dei modelli di object detection standard i quali non sono in grado di diagnosticare queste malattie in tempo reale a causa delle diverse forme e della possibilità che compaiano sulla stessa foglia.

Dopo aver creato un ampio dataset di immagini di foglie, sia con sfondo artificiale che naturale, è stato eseguito un processo di data augmentation per aumentarne il numero e la differenza tra le stesse. In totale, il dataset risultante comprende 62286 immagini. Successivamente, da parte degli autori dell'articolo è stata eseguita l'operazione di annotazione. Procedura che ha richiesto molto tempo perché necessita l'inserimento di una *bounding box* per ogni area malata della foglia.

Il modello proposto, **Faster DR-IACNN**, è riuscito ad ottenere ottimi risultati. Infatti, la media dell'accuratezza registrata per ogni classe (mAP) è 81.1% con una velocità di predizione di 15.01 FPS (frame per secondo). Da questo articolo emerge

un dato significativo: la rete sviluppata ha raggiunto i livelli di accuratezza riportati utilizzando un numero molto elevato di iterazioni nella fase di addestramento, pari a 280.000. Questo valore risulta notevolmente superiore rispetto ad altri studi e ciò suggerisce quindi che potrebbe essere notevolmente ridotto.

Similmente, l'articolo [7] ha trattato la diagnosi delle malattie della vite concentrandosi in particolare sulla *peronospora*. Lo studio ha analizzato e migliorato le prestazioni del noto algoritmo di object detection **YOLOv5** creandone una versione avanzata chiamata YOLOv5-CA. Come in molte altre ricerche, anche in questa il primo passo è stato la raccolta e la creazione del dataset. Il set di dati dello studio citato contiene 820 immagini raccolte sotto diverse condizioni di luminosità, con foglie sovrapposte e malattie che si presentano a vari livelli di intensità.

La particolarità di questa ricerca è che all'architettura standard di YOLO è stato aggiunto un meccanismo di **coordinate attention** cioè una tecnica che serve per catturare le relazioni spaziali tra le caratteristiche in modo più efficace e quindi essere in grado di distinguere parti diverse dell'immagine. Questo ha permesso di aumentare l'abilità del modello nell'estrazione delle features e quindi migliorare la ricerca della malattia sulle foglie. Il risultato ottenuto ha incrementato il valore di mAP ottenibile con YOLO raggiungendo l'89.55%, ovvero circa 2 punti percentuali in più rispetto a quello standard. Un altro importante risultato è la capacità di elaborazione del modello, infatti è stata raggiunta la velocità di 58.82 FPS ovvero circa 30 FPS in più rispetto alla versione standard di YOLO.

2.3 Costruzione del dataset

La qualità dei dati è il pilastro su cui si erge il successo dei modelli di intelligenza artificiale. Da modelli di ML a sistemi di DL, la costruzione di dataset accurati e ben strutturati è essenziale per ottenere risultati affidabili e scalabili. Sono state quindi esplorate le metodologie e le pratiche consigliate da varie ricerche e articoli per la creazione di dataset robusti che alimentano la crescita e il progresso nell'ambito dell'IA. Il dataset è fondamentale perché è quello che verrà utilizzato per addestrare la rete neurale che si ha intenzione di utilizzare.

Spesso la quantità dei dati che si hanno a disposizione non è sufficiente per il compito che è necessario svolgere. L'articolo [4], già citato in precedenza, mostra quanto sia fondamentale utilizzare tecniche di **data augmentation** quando gli input a disposizione non sono abbastanza. Infatti, gli autori avendo a disposizione solo 882 immagini di pareti fogliari di viti hanno utilizzato la tecnica sopra citata per aumentarne il numero a disposizione. Questa procedura, come riportato, è molto utile perché permette di incrementare la diversità dei dati presenti all'interno del dataset andando ad applicare varie trasformazioni su ciascuna immagine. Le alterazioni che sono state applicate sono le seguenti: ribaltamento orizzontale e

verticale, rotazione dell'immagine con un angolo di 40°, traslazione della lunghezza e della larghezza dell'immagine, zoom, sfocatura, aumento della luminosità, modifica del contrasto e della tonalità. Queste trasformazioni sono estremamente utili poiché, quando le fotografie verranno scattate in condizioni reali, come nel caso della tesi a progetto operativo, è molto probabile che non siano perfette come quella realizzate da utenti esperti durante la fase di analisi. Tuttavia, è importante prestare attenzione alle alterazioni delle tonalità perché queste potrebbero modificare e compromettere la visibilità delle malattie.

In modo analogo anche l'articolo [5] ha sottolineato quanto sia importante il processo di data augmentation nonostante si avessero a disposizione già 4449 immagini. Questo perché, come indicato dagli autori, più immagini si hanno e più queste sono diverse tra di loro, meno probabilità si ha di avere problemi di **overfitting** ed inoltre una quantità maggiore di immagini permette al modello di imparare più pattern durante il processo di addestramento. L'overfitting si manifesta quando un algoritmo di IA si adatta in modo troppo fedele o addirittura esattamente, ai dati che sono stati utilizzati nella fase di addestramento. Il risultato di questo problema è che il modello non sarà in grado di effettuare previsioni o conclusioni accurate su dati che sono diversi da quelli già osservati.

Le operazioni di trasformazione compiute da questo studio sul dataset di partenza sono molto simili a quelle descritte in precedenza. Queste ultime hanno portato, per ogni immagine, alla creazione di 14 diverse. Per avvalorare quanto riportato nell'articolo è stata evidenziata la differenza nella metrica mAP ottenuta senza l'uso della data augmentation. Evitando questa tecnica la mAP raggiunge solo il 74.3%, mentre con la data augmentation sale all'81.1% cioè un incremento di circa 7 punti percentuali. Questo confronto dimostra l'importanza significativa dell'uso della data augmentation nel migliorare le prestazioni del modello.

Ci sono poi altri due studi [9] e [7] che ulteriormente sottolineano quanto siano importanti i procedimenti di manipolazione sulle immagini presenti nel dataset raccolto.

Il primo articolo, così come [5], ha evidenziato il problema dell'overfitting e di come la data augmentation possa risolverlo. Viene infatti riportato che questo problema si presenta quando il modello si concentra sulle interferenze invece che sulle relazioni più significative che le features hanno tra di loro. Quando il set di dati è molto vasto il modello è in grado di identificare un numero maggiore di pattern irrilevanti in modo da ridurre il rischio di overfitting. Sono stati presentati vari modi con cui è possibile incrementare le dimensioni del dataset. Ad esempio, si può utilizzare la sfocatura Gaussiana per simulare delle condizioni meteo nuvolose oppure il valore di luminosità di ciascun pixel può essere modificato in modo casuale introducendo una non linearità nei dati. Queste sono solo alcune delle tecniche che sono state adottate ma che fanno capire quanto siano varie le trasformazioni applicabili. Il risultato più importante da cogliere da questo studio è che queste

operazioni vanno necessariamente effettuate sul set di dati e che esse cambiano in base all'obiettivo che si intende raggiungere.

Il secondo articolo ha inoltre investigato se la dimensione delle immagini di input influisse nell'accuratezza. Per fare questo sono state confrontate 5 diverse dimensioni: 112x112, 224x224, 320x320, 416x416 e 512x512. Ciò che è emerso è che maggiore è la dimensione delle immagini, maggiore è l'accuratezza perché banalmente si hanno maggiori dettagli all'interno dell'immagine. Infatti, si passa da 76.71% di mAP@0.5 con immagini 112x112 per arrivare a 87.89% con immagini 512x512. Nonostante ciò, c'è un inversione di tendenza per quanto riguarda la velocità di manipolazione: più la dimensione è grande, più il modello sarà lento. Basti pensare che si passa da 102.04 FPS per immagini 112x112 e si raggiungono solo 45.45 FPS con immagini di 512x512. Ciò dimostra quanto sia importate comprendere il dominio di interesse a cui la rete dovrà essere applicata ed in base a quello determinare quale sia la metrica più importante da preservare.

Infine, una conclusione estremamente significativa è stata raggiunta dalla ricerca [8]. Gli autori si sono concentrati, tra le varie cose, a rispondere alla seguente domanda: "Esiste un set di dati pubblico adeguato per il ML e il DL nel riconoscimento e l'identificazione della malattie sulle piante?". Per rispondere alla domanda è stata riportata una vasta lista di dataset che potrebbero essere utilizzati, tra cui collezioni multi specie e dataset specifici per singole specie, come ad esempio PlantVillage o RiceDataset. Tuttavia, è emerso che mancano dei dataset in cui sono presenti immagini di piante raccolte direttamente in campo anziché in laboratorio. Le fotografie acquisite in laboratorio non sono adatte per applicazioni che richiedono un funzionamento in tempo reale. Inoltre, è stato notato un ulteriore problema legato alla mancanza di un dataset unificato per ogni varietà di pianta, il che rende difficile confrontare i risultati ottenuti da diverse ricerche in modo accurato: ogni studio utilizza il proprio dataset rendendo i risultati non sempre comparabili in modo ottimale.

Capitolo 3

Metodologie

3.1 Collezione dei dati

Nello sviluppo delle reti neurali, la creazione del set di dati per addestrare ed ottenere le prestazioni di una rete detiene un ruolo fondamentale. La sfida che si è presentata durante le fasi di costruzione del dataset è stata la disponibilità limitata di immagini che rappresentino in modo esaustivo le condizioni reali che possono essere incontrate nei vigneti, mentre esistono molti dataset che contengono immagini di foglie prelevate in laboratorio. Inoltre, la quasi totalità delle collezioni di dati disponibili non presenta alcuna annotazione se non a livello di singola immagine. Questo rende impossibile l'utilizzo dei dati al di fuori della sola operazione di classificazione. Per affrontare questa lacuna si è deciso di combinare e migliorare i dati provenienti dalle varie ricerche o articoli pubblicati. L'approccio adottato mira a garantire una maggiore rappresentatività del set di dati includendo una vasta gamma di condizioni e variabili che sono presenti nelle diverse fonti. In questo modo, il dataset risultante sarà più completo e robusto, permettendo di ottenere risultati affidabili, comparabili ed utilizzabili in applicazioni pratiche. Inoltre, l'archivio di dati che verrà creato sarà reso disponibile alla comunità dei ricercatori in modo che, come citato in molte ricerche, le future analisi comparative possano essere effettuate sulla stessa solida base di dati.

Il dataset sviluppato è stato nominato **Vine Leaf Vision Dataset** ed è l'unione di quattro raccolte di immagini pubblicate online. Siccome tre di queste presentano una sola etichetta per immagine, cioè l'indicazione della malattia o salubrità delle foglie contenute all'interno, le immagini possono essere utilizzate solo per la classificazione. Per poter sfruttare appieno questi dati anche nel compito dell'object detection, le immagini sono state arricchite con le **bounding box** tramite uno

specifico strumento di annotazione[1].

Per ciascun insieme di immagini utilizzato verrà dettagliato il modo con cui l'operazione di annotazione e miglioramento è stata compiuta. I dataset collezionati sono i seguenti:

- **ESCA-dataset** pubblicato nella ricerca [4].
- **Grapevine Disease Images** in [11].
- **Vitis Vinifera** pubblicato dall'articolo [12].
- **Plants Diseases Detection and Classification Image Dataset** pubblicato in [13].

3.1.1 ESCA-dataset

Questo insieme di dati è costituito da immagini di pareti fogliari raccolte direttamente in campo risultando quindi molto simili a quelle che la rete dovrà elaborare una volta selezionata ed allenata. Nello specifico sono presenti 2 classi:

1. **esca**: pareti fogliari in cui è visibile la malattia *mal dell'esca*. Questa malattia si manifesta con macchie necrotiche marroni e/o rosse sulla superficie di foglie giallastre o rosse.
2. **healthy**: pareti fogliari che contengono solo foglie in stato di salute ottimo cioè completamente verdi. Le tonalità di verde variano in base allo stato di maturazione della foglia.

Il numero di immagini per classe è mostrato nella tabella 3.1 mentre gli esempi di fotografie contenute in esso sono visibili nelle figure 3.1 e 3.2. Come si può notare, le immagini presenti sono ideali per l'addestramento della reti neurali poiché rappresentano fedelmente le immagini che verranno catturate ed utilizzate per individuare la posizione della malattia una volta che il modello sarà operativo.

L'attività di perfezionamento applicata al dataset in questione consiste nell'annotazione tramite bounding box per ciascuna immagine. Queste ultime, aggiunte manualmente, sono state tracciate su ogni foglia visibile e ciascuna è stata etichettata in base al pattern visivo che questa presenta permettendo una classificazione accurata delle caratteristiche rilevanti. Il riquadro di delimitazione è stato apposto con la massima precisione possibile in modo tale da delimitare accuratamente il perimetro della foglia, riducendo al minimo il disturbo che potrebbe essere provocato dall'inclusione di parti di sfondo. In genere, l'operazione di annotazione richiede di

[1]Per i dettagli sullo strumento di annotazione si faccia riferimento alla sezione 3.2.

Classe	Numero di immagini
healthy	888
esca	882
Totale	1770

Tabella 3.1: Distribuzione delle classi all'interno di ESCA-dataset

Figura 3.1: Fotografia etichettata come esca

Figura 3.2: Fotografia etichettata come healthy

coinvolgere esperti nel settore a cui la si vuole applicare, quindi degli agronomi. Tuttavia, non essendo a disposizione, invece che etichettare la foglia in base alla malattia che essa presenta, sono state etichettate le foglie in base a delle caratteristiche (**pattern**) della stessa che sono facilmente riconoscibili. Naturalmente, è probabile che si siano verificati errori durante la fase di annotazione. Nonostante ciò, questi non dovrebbero influenzare significativamente sui risultati poiché l'elevato numero di annotazioni contribuisce a mitigare l'impatto dell'interferenza portata da eventuali etichettature errate.

Il processo di annotazione appena descritto ha identificato i seguenti pattern visivi:

- **healthy**: la foglia presenta una colore verde acceso ed uniforme, senza ingiallimenti, macchie o decolorazioni. La superficie è liscia e non presenta lacerazioni o fori ed i margini sono regolari. La foglia è visibile per almeno il 90% del suo perimetro. Esempio in figura 3.4.

- **esca**: la foglia presenta macchie necrotiche o striature di colore rosso o marrone spesso circondate da un alone giallastro. I bordi della foglia possono apparire secchi e non uniformi ed inoltre la superficie è irregolare. La foglia è visibile per almeno il 90% del suo perimetro. Esempio in figura 3.3.

- **white speckle dapple**: la foglia presenta piccoli puntini bianchi distribuiti in modo non uniforme senza quindi un pattern ben preciso. Questi punti possono avere dimensioni diverse tra di loro ma sono ben visibili e sono presenti su foglie che sono completamente verdi. La foglia è visibile per almeno il 90% del suo perimetro. Esempio in figura 3.5.

- **golden fleck healthy**: la foglia mostra una transizione graduale di colore, dal verde chiaro al giallo con variazioni di intensità lungo il perimetro della stessa. Le varie tonalità di giallo possono includere sfumature più chiare o più scure e non ci sono segni visibili di danni. La foglia è visibile per almeno il 90% del suo perimetro. Esempio in figura 3.6.

- **golden fleck esca**: la foglia mostra una transizione graduale di colore, dal verde chiaro al giallo con variazioni di intensità lungo il perimetro della stessa. Sulla sua superficie sono presenti macchie marroni o aree marroni che contrastano rispetto al colore predominante. Le macchie marroni possono variare in forma e dimensione e sono distribuite in modo irregolare sulla superficie della stessa. La foglia è visibile per almeno il 90% del suo perimetro. Esempio in figura 3.7.

- **partially visible**: La foglia non visibile per almeno il 90% del suo perimetro. Questa etichetta è valida per ciascuna classe definita in precedenza. Esempio in figura 3.8. Questo gruppo di etichette è stata introdotta in modo da poter aumentare il numero di esperimenti che possono essere eseguiti in futuro. Ciò non toglie il fatto che, in caso di risultati non soddisfacenti nel riconoscimento di questa classe, le etichette possano essere collassate automaticamente nella classe reale a cui appartengono. Ad esempio le foglie etichettate con *partially visible healthy* possono essere trasformate in *healthy* con un semplice algoritmo. L'operazione inversa non potrebbe essere effettuata perché richiederebbe di eseguire una scansione manuale completa del dataset per poter applicare le etichette.

Nel grafico 3.9 sono riportate, per ogni etichetta, il numero di foglie che sono state individuate durante l'operazione manuale di annotazione. Come si può notare c'è una disparità tra le diverse classi. In particolare, il numero di bounding box di tipo *pvl healthy* è molto alto rispetto a tutte le altre. Questa differenza così importante potrebbe comportare diversi problemi ai modelli di IA durante la fase di addestramento:

- il modello è probabile che abbia uno sbilanciamento nei confronti della classe più presente, andando ad ignorare o sottovalutare le altre classi durante la fase di allenamento. Questo quindi potrebbe ridurre la capacità di generalizzazione del modello.

- le classi che sono meno rappresentate è probabile che vengano predette con una bassa precisione perché il modello non ha abbastanza esempi per imparare quali sono le features caratterizzanti di quelle determinate classi.

Ci sono varie strategie che possono essere utilizzate per evitare questi problemi come ad esempio la **data augmentation** o il **resampling** (Vedi sezione 3.4). Queste operazioni vengono impiegate per aumentare la quantità di dati presenti in un dataset. In questo caso, dovrebbero essere utilizzate per bilanciare il numero di elementi di ciascuna classe. Pertanto, si potrebbe considerare l'idea di applicare alcune trasformazioni alle classi meno rappresentate, al fine di avvicinare il loro numero a quello della classe più numerosa.

Figura 3.3: Pattern visivo di una foglia classificata come esca

Figura 3.4: Pattern visivo di una foglia classificata come healthy

Figura 3.5: Pattern visivo di una foglia classificata come white speckle dapple

Figura 3.6: Pattern visivo di una foglia classificata come golden fleck healthy

Figura 3.7: Pattern visivo di una foglia classificata come golden fleck esca

Figura 3.8: Pattern visivo di una foglia classificata come partially visible

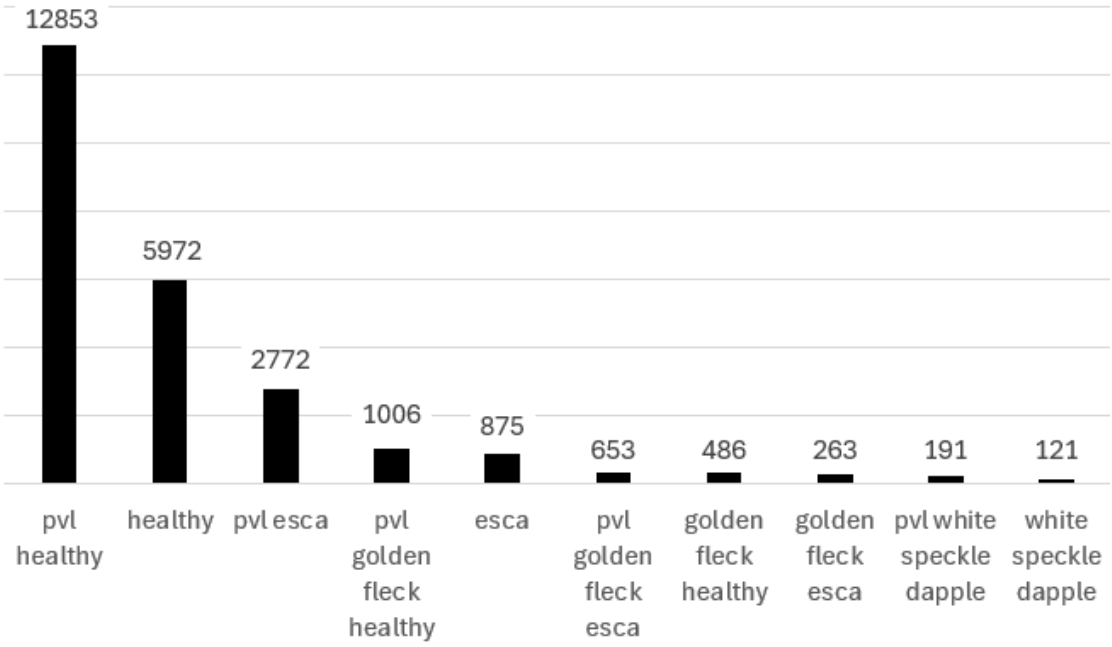

Figura 3.9: Numero di foglie per ogni etichetta in ESCA-dataset

3.1.2 Grapevine Disease Images

Questa collezione di dati contiene fotografie di foglie di vite raccolte in laboratorio cioè che mostrano sfondi artificiali. Più precisamente, il dataset è un sottoinsieme delle sole foglie di vite presenti nella più conosciuta raccolta di immagini chiamata **PlantVillage**. Questa collezione di dati ha etichette a livello di fotografia, ovvero una singola classe per l'intera immagine siccome ognuna di queste contiene una sola foglia. Le classi presenti sono quattro:

- **healthy**: foglia, di colore verde, in stato di salute ottimo che non presenta nessun tipo striature, macchie o decolorazioni.
- **black rot**: foglia affetta dalla malattia *marciume nero*. Sulle foglie colpite da questa malattia compaiono delle aree necrotiche di forma circolare sulle quali si sviluppano minuscoli corpi neri (picnidi), i quali contengono conidi che sono i responsabili della diffusione della malattia.
- **esca**: foglia affetta dalla malattia *mal dell'esca*. Essa presenta macchie necrotiche o striature di colore rosso o marrone spesso circondate da un alone

giallastro. I bordi della foglia possono apparire secchi e non uniformi ed inoltre la superficie è irregolare.

- **leaf blight**: foglia affetta dalla malattia *peronospora*. La malattia si manifesta attraverso macchie clorotiche (di colore giallo pallido o verde chiaro) sulle foglie che possono presentare muffa bianca o grigia nella parte inferiore.

Il numero di immagini per classe congiuntamente ad alcuni esempi di fotografie è mostrato nella tabella 3.2.

Classe	**Numero di immagini**	**Esempio**
healthy	420	Figura 3.10
black rot	1180	Figura 3.11
esca	1376	Figura 3.12
leaf blight	1084	Figura 3.13
Totale	**4062**	-

Tabella 3.2: Distribuzione delle classi all'interno di Grapevine Disease Images

Anche per questo dataset è stata effettuata l'operazione di annotazione tramite l'utilizzo delle bounding box sulle singole foglie in modo da poter sfruttare la collezione di fotografie anche per l'object detection. Siccome in ogni immagine è presente solo una foglia, non si è proceduto in modo manuale, utilizzando uno strumento di annotazione. La strategia adottata è stata quella di procedere in modo deterministico: applicare un modello di object detection pre-allenato pubblicato in rete [14] per posizionare le bounding box in modo automatico. Questo modello, basato su YOLOv8, riesce ad individuare e classificare foglie di diverse piante, comprese quelle di vite, con un valore di mAP@0.5 di 94.6%. L'elevato valore di accuratezza permette di applicare le bounding box con molta precisione. Il modello appena citato, quindi, è stato utilizzato per tracciare il riquadro delimitante attorno alla foglia, mentre la classe assegnata alla stessa è stata scartata per essere sostituita con quella fornita dal dataset. Questo procedimento ha permesso di ridurre drasticamente il tempo richiesto per completare la procedura di annotazione perché non ha richiesto, durante tale operazione, la presenza di un operatore.

La distribuzione delle etichette dopo questa operazione non viene specificata poiché è identica alla distribuzione delle classi all'interno del dataset, dato che ogni fotografia contiene una sola foglia.

Figura 3.10: Esempio di una foglia classificata come healthy

Figura 3.11: Esempio di una foglia classificata come black rot

Figura 3.12: Esempio di una foglia classificata come esca

Figura 3.13: Esempio di una foglia classificata come leaf blight

3.1.3 Vitis Vinifera

Questo dataset contiene fotografie di foglie appartenenti a più di 1200 diverse varietà di viti anche chiamate *Vitis vinifera*. Ogni fotografia presenta una o più foglie (fino ad un massimo di quattro) ed è scattata utilizzando un vetro non riflettente, con una fonte luminosa posizionata in modo opposto alla fotocamera per rendere il perimetro della foglia chiaramente visibile. Quindi anche in questo dataset le fotografie sono state raccolte in laboratorio con sfondo artificiale.

Le immagini presenti al suo interno sono 4303 e sono state classificate in base alla varietà che è un codice alfanumerico a cui appartengono. Per ogni tipo sono presenti 5/6 immagini raggiungendo un totale di circa 10 foglie. Dato che ci sono troppe poche immagini per ciascuna classe non è stato possibile utilizzare il dataset

nella suo forma originale anche per la sola classificazione: i modelli non sarebbero stati in grado di estrarre gli elementi caratterizzanti di ciascuna specie con così pochi esempi. Questa conclusione è stata raggiunta grazie ad un'attenta analisi dei dati disponibili: dopo aver estratto e processato le componenti di ciascun immagine, in relazione alla varietà a cui sono state associate, tramite **PCA** non è stato possibile raggrupparle in insiemi distinti a causa della distribuzione lineare dei dati. Di conseguenza, non si è potuto combinare varietà diverse nonostante le caratteristiche simili. Quindi utilizzare il tipo di varietà come classe non è possibile a causa del numero limitato di immagini.

Sono state quindi effettuate una serie di operazioni per rendere fruibile il dataset in questione all'interno del progetto. Durante l'analisi preliminare, utilizzando la PCA, è emerso che le due caratteristiche principali che accomunano le immagini non sono le sagome delle foglie (determinate dalla varietà) ma piuttosto i loro colori e le componenti visive. Come si può vedere da alcuni campioni presi dal set di dati, le differenze cromatiche sono molto significative. La figura 3.14 mostra due foglie completamente verdi mentre la figura 3.15 due foglie con delle macchie o parti completamente rosse. Non è possibile affermare con certezza che le macchie presenti nella seconda immagine siano dovute ad una malattia, in quanto non esperti del settore fitopatologico ma la differenza tra i due tipi di foglie è notevole. Nonostante ciò, è stata adottata la stessa strategia già utilizzata per ESCA-dataset: etichettare le immagini in base a pattern visivi chiaramente riconoscibili in modo da ridurre il più possibile l'errore durante la fase di annotazione.

Figura 3.14: Esempio immagine vitis vinifera verde

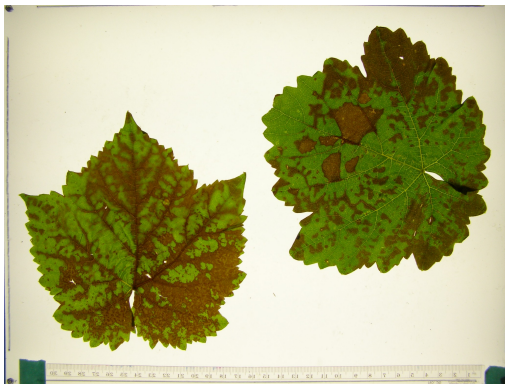

Figura 3.15: Esempio immagine vitis vinifera a chiazze rosse

Sulla base di quanto osservato, le etichette che sono state scelte e successivamente applicate sono le seguenti:

- **healthy**: la foglia è completamente verde, con una colorazione uniforme e

vivace. Possono essere presenti piccolissime macchie rosse sparse in modo irregolare sulla superficie della foglia. Le macchie presenti non influenzano però sull'aspetto generale.

- **red spot**: la foglia in questione è contraddistinta da macchie rosse di grandi dimensioni che possono arrivare a coprire l'intera superficie della foglia. Queste ultime possono essere distribuite uniformemente oppure possono essere concentrate in aree specifiche creando un forte contrasto con il verde della foglia, rendendole particolarmente evidenti.

Come per tutti i dataset trattati in precedenza è stata eseguita l'annotazione per singola foglia. La prima operazione è stata l'applicazione delle bounding box per delimitare il perimetro di ciascuna, così come in Grapevine Disease Images dataset l'annotazione è stata compiuta in modo automatico tramite l'utilizzo di una rete neurale. Ciò è stato possibile poiché le foglie, anche se presenti in più di una in ciascuna immagine, sono facilmente distinguibili dallo sfondo, rendendo il compito relativamente semplice per il modello utilizzato. La rete neurale è stata sfruttata esclusivamente per applicare le bounding box; per l'assegnazione delle etichette è stata invece adottata la libreria **OpenCV-Python** e le sue funzionalità, poiché il dataset non fornisce una classe utilizzabile, come osservato in precedenza. In particolare le operazioni compiute per la determinazione della classe di ciascuna bounding box sono state, nell'ordine, le seguenti:

1. ciascuna figura è stata convertita nello spazio **HUE saturation**. Questo è un formato per le immagini che permette di separare le informazioni del colore da quelle della luminosità e quindi di ottenere risultati migliori nell'operazione di identificazione di specifiche tonalità.

2. utilizzando la bounding box applicata in modo automatico, da ogni immagine è stata estratta la singola foglia con una procedura di *crop* (Figura 3.16). E' stato necessario suddividere le immagini in singole foglie poiché in ciascuna fotografia è possibile che siano presenti foglie che appartengono a classi diverse. Di conseguenza, non è possibile classificare l'intera fotografia con una singola etichetta ma è necessario classificare ogni foglia individualmente.

3. per ottenere una rappresentazione precisa delle foglie, sono state create delle **maschere**. A differenza delle bounding box, che sono quadrate e non riescono a seguire esattamente i contorni delle foglie, le maschere sono molto più precise in quanto etichettano ciascun pixel all'interno dell'immagine. Questo permette di catturare in modo molto più accurato la sagoma della foglia evitando di considerare lo sfondo, come accadrebbe utilizzando le bounding box. Per evitare di creare le maschere manualmente, è stata adottata una strategia basata sul rilevamento del colore dello sfondo: tutte le variazioni cromatiche

che appartengono allo sfondo non fanno parte delle tonalità della foglia. Quindi per estrarre i pixel della sola foglia è necessario prima isolare lo sfondo. Questo operazione è stata realizzata campionando le sfumature di colore dello sfondo, espresse nel formato RGB, e definendo un intervallo minimo e massimo di colori possibili che corrispondono a quest'ultimo. Per creare la maschera è stata utilizzata la funzione
`maschera_sfondo = cv2.inRange(foglia, lower-bound, upper-bound)`.
Questa prende in input la foglia ed il range di tonalità e restituisce la maschera binaria dove per ogni pixel è indicato se esso appartiene al range selezionato o no. Eseguendo questa operazione con le sfumature dello sfondo si ottiene una maschera che identifica solo lo sfondo. Siccome la foglia è tutto ciò che non fa parte delle sfondo bisogna invertire il risultato appena ottenuto per estrarla. Questo è stato compiuto tramite la funzione `cv2.bitwise_not(maschera_sfondo)`. Grazie a quest'ultima è stato possibile ottenere la maschera che identifica solo la foglia all'interno della bounding box ritagliata (Figura 3.17).

4. una volta che si ha a disposizione la sola area della foglia questa deve essere suddivisa nella parte verde e nella parte rossa. Come per lo sfondo, anche in questo caso sono state identificate, tramite campionamenti, le tonalità di verde e di rosso e successivamente sono state ottenute ed applicate le maschere (Figura 3.18). Date le tre aree, è stato possibile calcolare la percentuale di foglia verde e quella rossa rispetto all'area totale della foglia. Grazie a questi valori è stata assegnata la classe a ciascuna bounding box. In particolare, la foglia viene classificata come *healthy* se la percentuale di area rossa è inferiore al 5% e quella verde è superiore al 90%, mentre tutte le altre foglie vengono classificate come *red spot*. I valori appena citati sono stati determinati attraverso uno studio sperimentale, nel quale sono stati eseguiti diversi esperimenti al fine di individuare i limiti ed i range ottimali per la migliore classificazione delle foglie.

Siccome la procedura descritta in precedenza si basa su campionamenti di colori è possibile, nonostante i numerosi esperimenti compiuti, che delle foglie abbiano ricevuto una classe sbagliata. Per ridurre al minimo questa possibilità è stato eseguito uno studio aggiuntivo sul risultato della classificazione. Nel dettaglio, sono state raccolte, in modo separato per ogni classe e per ogni foglia, le aree delle tonalità dei colori presenti in esse. Su questi valori è stato calcolato lo **z-score** cioè una misura statistica che quantifica la distanza che si manifesta tra ciascun punto e la media di tutti i punti. Prendendo come esempio la classe *healthy*, lo z-score indica, per ogni foglia, quanto i suoi colori siano differenti rispetto alla media dei colori di tutte le immagini che appartengono alla stessa classe. Questa operazione è servita per l'identificazione degli **outliers** cioè le foglie che sono state classificate

Figura 3.16: Foglia estratta dall'immagine intera utilizzando le coordinate della bounding box

Figura 3.17: Maschera per ottenere la sola foglia all'interno della bounding box, ottenuta invertendo la maschera dello sfondo. La maschera è rappresentata dal colore giallo

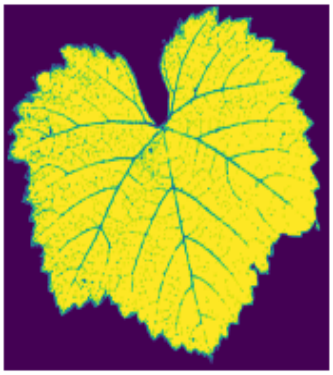

Figura 3.18: Maschera per ottenere l'area verde della foglia. Siccome la foglia è verde, la quantità di colore giallo è predominante

in modo sbagliato. La figura 3.19 mostra gli outliers identificati per la classe citata in precedenza. Un volta determinati, si è proceduti attraverso un'annotazione manuale in modo tale da risolvere gli errori presenti.

Il risultato di questa serie di operazioni ha portato come risultato l'identificazione di un numero elevato di foglie, per ogni classe, che possono essere osservati nella tabella 3.3.

Classe	**Numero di foglie**	**Esempio**
healthy	2388	Figura 3.14
red spot	6206	Figura 3.15
Totale	**8594**	-

Tabella 3.3: Contenuto di Vitis Vinifera dopo l'operazione di annotazione

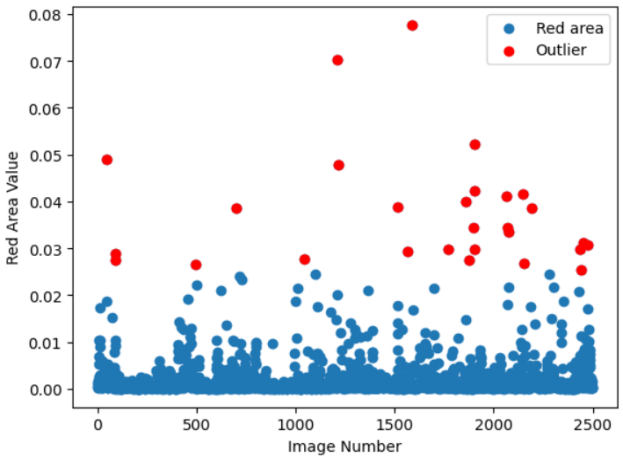

Figura 3.19: Grafico che mostra, per ogni foglia classificata come *healthy*, l'area della parte rossa. I punti di colore rosso indicano gli outliers cioè le foglie la cui area del colore rosso si discosta dalla media

3.1.4 Plants Diseases Detection and Classification Image Dataset

Questo dataset è composto da immagini di foglie appartenenti a 48 diverse classi tra cui meli, pomodori, patate e viti. Le fotografie scattate sia in laboratorio che in campo ammontano in totale a 2516. Una caratteristica distintiva di questo dataset rispetto a tutti gli altri pubblicati è che ciascun'immagine è già annotata tramite l'utilizzo delle bounding box.

Dall'insieme di fotografie sono state selezionate solo le immagini di interesse ovvero quelle contenenti foglie o pareti fogliari di vite, che in totale sono 130. In particolare, queste immagini sono etichettate utilizzando due classi:

- **grape leaf**: foglia di vite sana il cui colore predominante è il verde acceso. Il totale di fotografie presenti per questa classe è 64. Un esempio è visibile in figura 3.20.
- **grape leaf black rot**: foglia di vite affetta dalla malattia *black rot*. Il totale di fotografie presenti per questa classe è 66. Un esempio è visibile in figura 3.21.

Nonostante il numero di immagini estratte sia relativamente ridotto, è importante integrarle nel dataset complessivo perché presentano caratteristiche e features uniche non riscontrabili negli altri dataset già trattati. Questo permette di aumentare la variabilità dei dati ed evitare che i modelli possano soffrire di *overfitting*.

Figura 3.20: Fotografia che mostra le bounding box a cui è associata l'etichetta grape leaf

Figura 3.21: Fotografia che mostra le bounding box a cui è associata l'etichetta grape leaf black rot

Per il set di dati in questione non è stato necessario procedere con alcuna operazione di miglioramento poiché i dati sono già stati forniti con le bounding box applicate. Nella tabella 3.4 è possibile osservare il numero di foglie che sono presenti per ogni classe.

Classe	**Numero di foglie**
grape leaf	231
grape leaf black rot	124
Totale	**355**

Tabella 3.4: Contenuto di Plants Diseases Detection and Classification Image Dataset

3.2 Strumento per l'annotazione

Gli strumenti per l'annotazione sono software che permettono agli annotatori, cioè persone incaricate ad eseguire il compito di etichettatura, di creare le bounding box attorno agli oggetti presenti nelle immagini in base a determinate caratteristiche definite a priori. Questi software consentono la marcatura delle immagini in modo intuitivo perché, grazie ad elementi grafici come forme geometriche, il compito può essere eseguito in modo più semplice.

In commercio sono presenti vari strumenti di annotazione, quello che però è stato scelto è **Label Studio** perché permette di poter essere potenziato in termini di funzionalità. LabelStudio è un software open source che concede di annotare dataset non solo di immagini ma anche video e audio. L'efficacia dello strumento è data dal fatto che presenta un'interfaccia grafica molto semplice e sbrigativa, essenziale in compiti come il seguente che richiedono molto tempo. Le annotazioni, una volta introdotte nell'immagine, possono essere esportate in vari formati tra cui JSON o **YOLO**. Per poter utilizzare questo software si è resa necessaria la creazione di un container locale tramite **Docker** seguendo le istruzioni presenti sulla pagina web ufficiale di Label Studio. Il container viene utilizzato come *wrapper* per lo strumento di etichettatura. Una volta creato, il sistema si utilizza tramite browser raggiungendo l'indirizzo ip associato al contenitore appena creato.

Dopo aver allestito correttamente l'ambiente di esecuzione, la prima cosa da fare è caricare all'interno della piattaforma di annotazione il dataset. Successivamente, è necessario definire le etichette che potranno essere utilizzate. Una volta fatto questo è possibile iniziare con l'annotazione vera è propria, quindi, per ogni foglia identificata si deve capire a quale pattern visivo appartiene, selezionare l'etichetta corretta ed infine tracciare una forma rettangolare in modo da racchiuderla al suo interno. Questa operazione deve essere ripetuta per ogni foglia all'interno dell'immagine. È possibile trovare un esempio di immagine annotata nella figura 3.22.

Il procedimento di annotazione descritto in precedenza è stato utilizzato per ESCA-dataset. Ogni etichetta richiede, in genere, 10/15 secondi per essere posizionata. In ciascuna immagine ci sono 25/30 etichette con picchi anche di 50. Quindi, sono necessari circa 4 minuti per portare a compimento la procedura di marcatura su ciascuna immagine. Siccome il dataset contiene più di 1700 immagini che devono essere elaborate, utilizzando questa serie di passi devono essere impiegate circa 115 ore di lavoro cioè quasi 3 settimane lavorative. La quantità di ore che dovrebbero essere utilizzate è troppo elevata, quindi sono state cercate alternative per risolvere tale limitazione.

Per ridurre il tempo impiegato durante l'operazione di annotazione è stata introdotta ed utilizzata la funzionalità aggiuntiva di LabelStudio chiamata **Lablel Studio ML backend**.

3.2.1 Machine Learning backend

Si tratta di un **SDK (software developement kit)** progettato per integrare un modello di ML all'interno di Label Studio. Essenzialmente, l'SDK consente di creare un **backend** in cui inserire il codice di un modello desiderato. Successivamente, questo viene posizionato all'interno di un container trasformandolo in un web server per poter abilitare la comunicazione con Label Studio. Il modello incluso

Figura 3.22: Esempio di un'immagine dopo l'operazione di annotazione

all'interno del backend permette di predire la posizione e l'etichetta della bounding box, facilitando così l'operazione di annotazione. La rete che è stata utilizzata è **Plant leaf Detection and Classification** [14]. Quest'ultima è basata sul framework di **YOLO** ed è stata creata in modo tale da localizzare la posizione di diversi tipi di foglie, tra cui quelle delle viti. Dato che questo modello non è specificamente addestrato sulle foglie di vite, la precisione nel riconoscere la posizione delle stesse non è ottima. Tuttavia, anche se queste non sono identificate in modo perfetto, il backend accelera il processo di annotazione poiché molte bounding box sono già posizionate, anche se talvolta necessitano di aggiustamenti, modifica della classe o l'inserimento di alcune nuove che è comunque molto meno oneroso rispetto alla creazione manuale completa. Per mitigare il problema della precisione è stato deciso di ri-addestrare il modello ogni circa 200 nuove immagini annotate. Questo approccio consente alla rete di acquisire gradualmente una maggiore familiarità con le caratteristiche specifiche delle foglie di vite e delle malattie in esame, contribuendo a migliorare la precisione nel tempo e riducendo la durata complessiva dell'annotazione.

Come per Label Studio, anche il backend è eseguito all'interno di un container. Una volta scaricato ed installato[2] è necessario aggiungere al suo interno la logica del modello. Per fare questo bisogna modificare la funzione `predict(self, tasks, context, **kwargs)` nel file `model.py`. Questa riceve

[2]Per i dettagli seguire le indicazioni presente al seguente link: `https://labelstud.io/guide`.

come parametro una lista di *tasks* che sono le immagini che necessitano di essere annotate e deve restituire una lista di *predictions* (una per ogni task) cioè le bounding box predette dal modello con l'etichetta associata.

Una volta completata l'integrazione del modello, l'ultimo passo è l'abilitazione della comunicazione tra Label Studio ed il backend. E' un operazione molto semplice perché nel menù **Machine Learning** di Label Studio basta inserire l'indirizzo ip e la porta del container in cui è eseguito il backend. Quando la comunicazione è stabilita, ogni volta che si procede con l'annotazione di una nuova immagine, automaticamente verranno mostrate su di essa alcune bounding box che sono il risultato della predizione effettuata dal modello eseguito nel backend.

Questa operazione ha permesso di ridurre il tempo di annotazione del 40% circa. Infatti, si è passati da impiegare 5 minuti per immagine a soli 3 minuti.

3.3 Unione dei dataset

Dopo aver raccolto ed ottimizzato ciascun dataset considerato, è stato eseguito il processo di fusione in uno unico che è stato chiamato **Vine Leaf Vision Dataset**. L'operazione di unificazione è particolarmente delicata poiché richiede di utilizzare un formato comune sia per le etichette che per le bounding box.

Come prima cosa, è stato deciso di unificare le etichette. Questo implica trovare un numero adeguato di classi e assegnare a ciascuna un valore numerico a partire da 0. Tale approccio consente di identificare ogni classe presente in ciascun dataset considerato, evitando duplicazioni. Ad esempio, siccome nella maggior parte delle collezioni di dati è presente la classe *healthy*, nel associazione finale questa dovrà essere presente una sola volta. Il risultato ottenuto da questa fase ha portato alle associazioni presenti nel grafico 3.23 dove per ogni classe, all'interno delle parentesi tonde, è mostrato il valore numerico associato alla stessa. Il totale delle annotazioni ammonta a **38.201**.

Successivamente si è passati alla fase di normalizzazione delle bounding box utilizzando il formato YOLO, poiché è l'unico accettato dal modello che si intenderà utilizzare. Per ogni bounding box, la configurazione scelta prevede di avere 5 valori:

1. indicazione della classe associata alla bounding box. È un numero che parte da 0 per arrivare fino 12. Questo valore è frutto della associazione eseguita al passo precedente.
2. coordinata x del centro della bounding box
3. coordinata y del centro della bounding box
4. altezza h della bounding box
5. larghezza w della bounding box

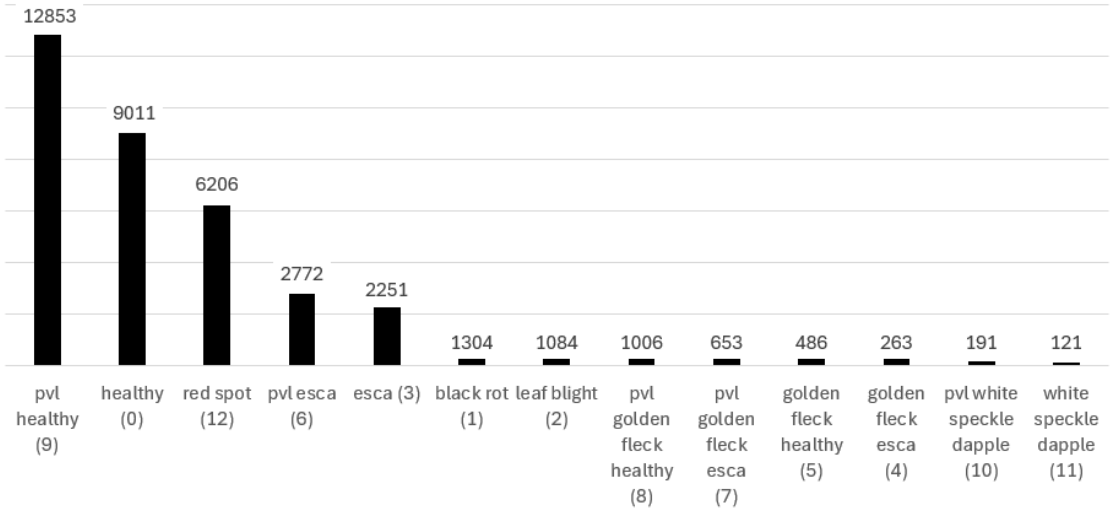

Figura 3.23: Numero di foglie per ogni etichetta in Vitis Leaf Vision. Tra le parentesi tonde è indicato il valore numerico della classe

Gli ultimi 4 valori sono standardizzati cioè $0 \leq x, y, h, w \leq 1$. In sostanza sono in percentuale rispetto alle dimensioni dell'intera immagine. Per rendere utilizzabile il dataset anche con altre architetture, è stato messo a disposizione un algoritmo che consente la conversione delle annotazioni in qualsiasi altro formato.

Nonostante l'obiettivo sia la classificazione di foglie in pareti fogliari, è molto importante includere nel dataset finale molte immagini di foglie singole raccolte in laboratorio. Questo perché, siccome queste ultime vengono raccolte in condizioni controllate e con sfondi artificiali, permettono di ottenere informazioni più chiare e prive di interferenze facilitando l'addestramento iniziale del modello. Inoltre, le fotografie raccolte in laboratorio aiutano il modello ad imparare e riconoscere le caratteristiche distintive delle differenti malattie senza la possibile distrazione dovuta allo sfondo dell'immagine.

3.4 Data augmentation

L'operazione di data augmentation è estremamente importante, come evidenziato in numerosi articoli citati in precedenza. Questa permetterebbe di aumentare la variabilità dei dati ma soprattutto garantirebbe di eguagliare il numero di elementi per ogni classe. Tuttavia, verrà eseguita in futuro poiché la priorità attuale è verificare il funzionamento del dataset creato e dei modelli selezionati. Se questi

dimostreranno di funzionare correttamente, l'applicazione della data augmentation sul dataset non potrà che migliorare ulteriormente i risultati.

3.5 Metriche per il confronto dei risultati

Le **metriche di valutazione**, nell'ambito del ML sono cruciali perché permettono di determinare l'efficacia di un modello. Infatti, la corretta interpretazione delle performance è fondamentale per migliorare ed ottimizzare gli algoritmi.

3.5.1 Metriche per i modelli di classificazione

Una volta che è stato individuato il modello per il compito della classificazione è necessario valutarne le prestazioni. Per effettuare questa operazione, in questo ambito, possono essere utilizzate diverse **metriche** [15]:

- **accuracy**: è il rapporto tra il numero di istanze correttamente classificate per il numero totale di istanze. Più il valore è alto, più il modello è abile nella classificazione. E' una metrica che però soffre quando il dataset è sbilanciato cioè quando c'è disparità tra il numero di elementi appartenenti ad ogni classe.
- **confusion matrix**: è una matrice che riporta, per ogni classe, il numero di:[3]
  - **true positive (TP)**: la classe prevista è SI ed è uguale alla classe effettiva.
  - **true negative (TN)**: la classe prevista è SI ma è diversa dalla classe effettiva (che è NO).
  - **false positive (FP)**: la classe prevista è NO ed è uguale alla classe effettiva.
  - **false negative (FN)**: la classe prevista è NO ma è diversa dalla classe effettiva (che è SI).

		Actual Class	
		Positive (**P**)	Negative (**N**)
Predicted Class	Positive (**P**)	True Positive (**TP**)	False Positive (**FP**)
	Negative (**N**)	False Negative (**FN**)	True Negative (**TN**)

Figura 3.24: Matrice di confusione. Fonte: [16]

[3]supponendo di utilizzare un classificatore binario dove le classi possibili sono **SI** e **NO**

più i valori sulla diagonale sono alti e gli altri bassi, più il modello sta classificando in modo corretto. La matrice viene utilizzata per capire dove sono i margini di miglioramento e quali sono le classi che il modello identifica con una minore accuratezza.

- **precision**: è il numero delle previsioni corrette di una classe sul totale delle volte che il modello prevede quella stessa classe.

$$precision = \frac{TP}{TP + FP} \tag{3.1}$$

- **recall**: è il rapporto tra le previsioni corrette per un classe sul totale dei casi in cui la classe è quella effettiva.

$$recall = \frac{TP}{TP + FN} \tag{3.2}$$

- **f-score**: è la media armonica tra precision e recall. Viene utilizzata quando i valori di precision e recall sono molto diversi tra un gruppo di modelli che si intendono comparare. Utilizzando l'f-score si garantisce l'uniformità perché le due metriche vengono considerate congiuntamente.

$$f - score = \frac{2 * recall * precision}{recall + precision} \tag{3.3}$$

3.5.2 Metriche per i modelli di object detection

Così come per la classificazione, anche per l'object detection sono presenti delle metriche di valutazione che permettono di misurare quali sono le prestazioni del modello analizzato [17]:

- **intersection over union(IoU)**: misura la quantità di sovrapposizione tra la bounding box predetta e quella **ground trouth**[4]. E' calcolata come il rapporto tra l'area dell'intersezione per l'unione delle due aree. Varia tra 0 e 1, dove 0 indica senza sovrapposizione (predizione completamente sbagliata), 1 indica sovrapposizione perfetta.

[4]Bounding box associata nella fase di annotazione in modo manuale o semi-manuale. E' la bounding box da considerarsi nella posizione corretta. L'obiettivo del modello è avvicinarsi il più possibile a questa posizione.

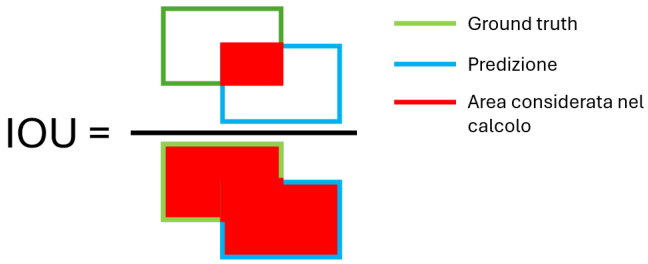

Figura 3.25: IoU

- **confusion matrix**: è una matrice, molto simile a quella per la classificazione ma con alcune differenze:
  - **true positive (TP)**: il modello ha predetto la bounding box in una determinata posizione (positive) ed è corretta (true).
  - **true negative (TN)**: il modello non ha predetto la bounding box in una determinata posizione (negative) ed è corretto (true). Questo corrisponde allo sfondo dell'immagine, cioè quella parte che non presenta le bounding box e che non viene utilizzata per il calcolo delle metriche.
  - **false positive (FP)**: il modello ha predetto la bounding box in una determinata posizione (positive) ma è sbagliata (false).
  - **false negative (FN)**: il modello non ha predetto la bounding box in una determinata posizione (negative) ed è sbagliato (false). Ad esempio quando la bounding box ground trouth esisteva in quella posizione.
- **average precision(AP)**: è una metrica il cui calcolo si basa su IoU, confusion matrix, precision e recall (Vedi sezione 3.5.1). Indica le performance di un algoritmo di object detection nel localizzare le istanze di una determinata classe. Il calcolo di questa metrica segue i passaggi mostrati nella figura 3.26. La **Precision-Recall Curve** è un grafico che mostra il variare della precision e del recall per diversi valori della soglia di **threshold**. Questa soglia è un valore definito dall'utente che indica il valore minimo di confidenza.

  Il punteggio di AP è l'area al di sotto di questa curva ed è un valore scalare. La misura di AP è alta quando sia la precision che il recall sono alti, bassa altrimenti.

$$AP = \int_{r=0}^{1} p(r)\, dr \tag{3.4}$$

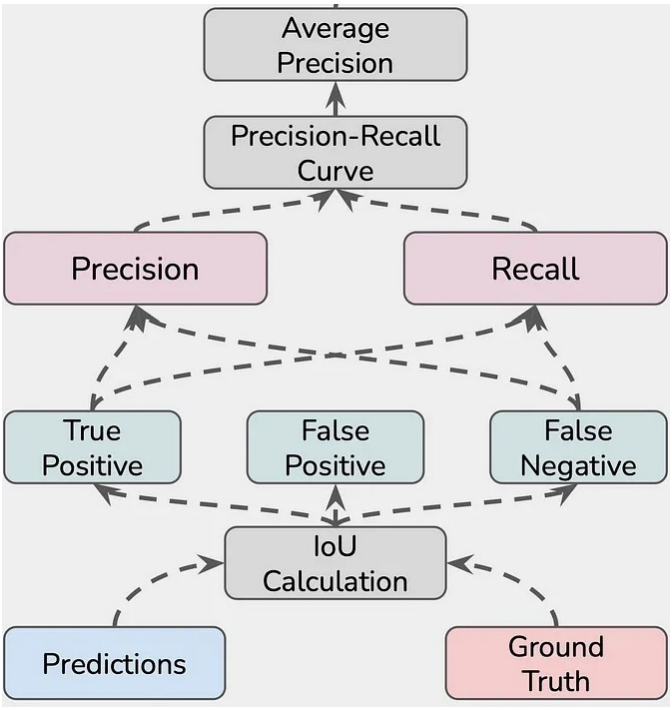

Figura 3.26: Passaggi per il calcolo dell'average precision su di una singola classe. Fonte: [18]

- **mean average precision (mAP)**: è la media dell'AP calcolata per ogni classe.

$$mAP = \frac{1}{k} \sum_{i}^{k} AP_i \tag{3.5}$$

  Si divide in **mAP@50** e **mAP@50-95**. La prima viene calcolata per valori di IoU minori di 0.5 e misura l'accuratezza del modello considerando solo i rilevamenti più semplici. La seconda, invece, considera valori di IoU tra 0.5 e 0.95, offrendo una visione più completa sulle prestazioni del modello attraverso vari livelli di difficoltà.

3.6 Operazione di Classificazione

Nella prima fase di questa tesi si è cercato di capire quali tipi di reti neurali si adattassero maggiormente al compito di classificazione delle malattie sulle pareti fogliari delle viti. Le reti neurali prese in considerazione sono:

- **ResNet**: questa architettura di tipo CNN era lo stato dell'arte prima dell'introduzione dei transformer. È stato deciso di utilizzare la seguente rete neurale in quanto è presente in diverse varianti (cioè con un numero diverso di livelli convoluzionali), è basata sulla tecnologia delle convoluzioni e garantisce un giusto bilanciamento tra l'accuratezza ed il suo peso. Per maggiori dettagli riguardo all'architettura si faccia riferimento alla sezione 4.1.1.

- **ViT**: questa architettura è in contrapposizione con le CNN perché non utilizza le convoluzioni ma utilizza i transformers. È stato scelto questo tipo di rete neurale perché raggiunge prestazioni migliori in termini di accuratezza e perché utilizza una tecnologia diversa dalle ResNet. Il punto debole dei modelli, una volta allenati, è che sono molto pesanti e quindi potrebbero avere problemi nell'essere eseguite in sistemi embedded. L'architettura di queste è mostrata nella sezione 4.1.2.

- **MobileNet**: architetture di tipo CNN molto leggere e versatili. Il motivo per cui è stato scelto di utilizzarle è perché possono essere eseguite in sistemi con capacità computazionali molto ridotte. Siccome l'obbiettivo finale del progetto DIVINE, in cui si inserisce questa tesi, ha come scopo l'utilizzo di questi sistemi direttamente all'interno dei vigneti in tempo reale, questi modelli potrebbero essere la soluzione adatta. È possibile consultare nella sezione 4.1.3 i dettagli dell'architettura.

Sono state comparate alcune varianti specifiche delle architetture sopra citate rispetto ai risultati ottenuti dall'articolo scientifico [4]. Come già menzionato in precedenza, in questa pubblicazione oltre ad essere stato raccolto un dataset di circa 1800 immagini di pareti fogliari di vite, è stata anche proposta una semplice pipeline di DL per la classificazione di queste immagini. L'architettura che è stata sviluppata nello studio è una CNN ad hoc con soli cinque livelli convoluzionali. La realizzazione completa è mostrata nella figura 3.27.

Per garantire una comparazione accurata e corretta dei risultati tra i diversi modelli e lo studio in questione, sono state adottate due principali strategie. In primo luogo, durante la fase di addestramento, è stato utilizzato lo stesso dataset e le stesse tecniche di data augmentation descritte nello studio di riferimento. In secondo luogo, poiché sempre lo studio fornisce l'algoritmo per la suddivisione del dataset in set di addestramento, validazione e test, sono state applicate le medesime operazioni in modo da avere gli stessi dati e le stesse distribuzioni delle classi in ciascun insieme. Questo ha portato ad una suddivisione del dataset in 60% training, 15% validation e 25% test.

Dopo aver allineato la parte di dataset con lo studio comparato, si è passati

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 1280, 720, 32)	896
activation (Activation)	(None, 1280, 720, 32)	0
max_pooling2d (MaxPooling2D)	(None, 640, 360, 32)	0
conv2d_1 (Conv2D)	(None, 640, 360, 32)	9248
activation_1 (Activation)	(None, 640, 360, 32)	0
max_pooling2d_1 (MaxPooling2	(None, 320, 180, 32)	0
conv2d_2 (Conv2D)	(None, 320, 180, 64)	18496
activation_2 (Activation)	(None, 320, 180, 64)	0
max_pooling2d_2 (MaxPooling2	(None, 160, 90, 64)	0
conv2d_3 (Conv2D)	(None, 160, 90, 64)	36928
activation_3 (Activation)	(None, 160, 90, 64)	0
max_pooling2d_3 (MaxPooling2	(None, 80, 45, 64)	0
conv2d_4 (Conv2D)	(None, 80, 45, 32)	18464
activation_4 (Activation)	(None, 80, 45, 32)	0
max_pooling2d_4 (MaxPooling2	(None, 40, 22, 32)	0
flatten (Flatten)	(None, 28160)	0
dense (Dense)	(None, 64)	1802304
activation_5 (Activation)	(None, 64)	0
dropout (Dropout)	(None, 64)	0
dense_1 (Dense)	(None, 2)	130
activation_6 (Activation)	(None, 2)	0

Figura 3.27: Dettaglio dell'architettura sviluppata dall'articolo. Fonte: [4]

alla fase di **fine-tuning**[5]. Nello specifico, le varianti delle architetture che sono state analizzate sono **ResNet18** cioè con 18 livelli convoluzionali, **MobileNet V3** l'ultima versione disponibile ed infine **ViT b 16** cioè con sotto-sezioni dell'immagine di dimensione 16x16 pixel. Tutti i modelli presi in considerazione sono pre-addestrati sul dataset ImagenetV2. Questo consente di avviare gli esperimenti da una base già capace di estrarre features dalle immagini; successivamente, applicando le tecniche

[5]Operazione di ottimizzazione delle prestazioni di un modello già allenato nei confronti di un dominio più specifico.

di fine-tuning, i modelli vengono ri-addestrati sul dataset di interesse per eseguire compiti più specifici. Il fine-tuning può essere effettuato in diversi modi, inclusa la possibilità di fissare alcuni pesi del modello, tutti, o una combinazione di essi. Quando un peso viene bloccato, l'algoritmo di **backpropagation**[6] non aggiorna il suo valore durante la fase di allenamento, mantenendolo così costante. Queste scelte progettuali non solo influenzano le performance del modello ma anche la sua capacità di generalizzazione e di adattamento a nuovi domini (**domain shifting**) e, inoltre, sono determinate dalle risorse computazionali disponibili durante la fase di addestramento. In particolare, nei modelli che sono stati sperimentati l'intera rete viene mantenuta fissa eccetto l'ultimo strato, che deve essere inizializzato zero in quando responsabile della definizione delle etichette finali.

Gli iperparametri selezionati per portare a compimento la fase di addestramento sono stati scelti al fine di ottimizzare le prestazioni dei modelli. Per una comparazione migliore sarebbe ideale utilizzare gli stessi parametri menzionati nell'articolo di riferimento ma poiché questi non sono stati forniti, si è proceduto a selezionare i valori più appropriati in base ad alcune sperimentazioni preliminari:

- numero di **epoche**: 50. Questo è il numero di volte che nella fase di addestramento vengono processate le immagini presenti nel dataset. Il valore di questo parametro è stato scelto sulle base delle indicazioni presenti nello studio di riferimento.

- **learning rate (lr)**: 0.01 per le reti convoluzionali, 0.001 per le reti basate sui transformers. È un parametro che determina l'entità della modifica dei pesi della rete rispetto all'errore calcolato durante la fase di addestramento. È stato scelto un valore di lr più alto per le reti CNN per consentire una convergenza più rapida delle stesse avendo a disposizione solo 50 epoche, mentre un valore più basso per l'altro tipo di reti in modo da garantire una convergenza più stabile e prevenire possibili instabilità.

- **batch size**: 16 per la fase di addestramento perché è il valore standard, mentre 1 per la fase di validazione in quando si deve simulare l'applicazione reale della rete. Questo parametro indica il numero di immagini che vengono processate in contemporanea.

- **workers**: 8 poiché è un valore che viene comunemente utilizzato. Indica il numero di processi che verranno utilizzati nella fase di caricamento in memoria del dataset.

[6]Algoritmo utilizzato per andare ad aggiornare i pesi dei collegamenti nella rete neurale, minimizzando l'errore tra l'output previsto e quello effettivo.

- funzione di **ottimizzazione**: *stochastic gradient descending (sgd).* È la funzione che viene utilizzata per ottimizzare ed addestrare i modelli. In particolare, questa è una variante della più comune *gradient descent (gd)* che consente di avere aggiornamenti più veloci ed evitare di rimanere bloccati in minimi locali.

- funzione per il calcolo della **loss**: *cross entropy loss.* Questa permette di misurare quanto le predizioni del modello deviano dal risultato corretto e viene calcolata nel seguente modo:

$$H(p,q) = -\sum_{x \in classi} p(x) log q(x) \tag{3.6}$$

  dove q è la distribuzione della probabilità delle predizioni mentre p è la distribuzione delle probabilità corretta.

Per semplificare e velocizzare le operazioni di addestramento è stata creata una libreria ad hoc che è composta da quattro classi più il file *main.py* che le combina:

- **Trainer**: è il cuore della libreria perché è la classe che effettivamente esegue la fase di addestramento e la fase di validazione in base agli iperparametri specificati.

- **CustomModel**: classe che permette di avviare, tra i modelli disponibili, quello che si intende addestrare.

- **CustomDataset**: classe che permette di reperire le immagini con cui si intende allenare il modello.

- **SplitDataset**: classe che divide il dataset in tre partizioni: training, validation e test in base alle percentuali specificate.

Modificando le variabili presenti nel *main.py* è possibile allenare un modello. In particolare, è necessario scegliere i valori degli iperparametri definiti in precedenza, specificare il nome del modello da addestrare tra quelli disponibili e infine la cartella in cui è presente il dataset da utilizzare. Una volta eseguito lo script, la libreria gestirà automaticamente l'addestramento e la validazione del modello e, in tempo reale, invierà su **Comet** le metriche e i grafici per monitorarne l'andamento. Per maggiori informazioni e dettagli è possibile consultare il seguente link Github in cui la libreria è stata pubblicata:
`https://github.com/fabriziosanino/code_base_ai`.

3.7 Operazione di Object Detection

Durante la fase principale della tesi l'attenzione è stata dedicata all'analisi di **Vine Leaf Vision Dataset**, con l'obiettivo di ottimizzare le prestazioni dei modelli di object detection nel compito dell'individuazione e classificazione di nuove patologie non presenti all'interno del dataset. Le architetture prese in considerazione sono state:

- **YOLO**: è un modello di DL il cui obiettivo è la rilevazione in tempo reale degli oggetti in un'immagine, che suddivide la stessa in una griglia e prevede simultaneamente la bounding box e le probabilità di classe per ogni regione. Grazie alla sua architettura unificata, YOLO è noto per la velocità e precisione, rendendolo ideale per applicazioni che richiedono rilevamenti in tempi rapidi.

- **DETR**: è un modello di DL per la rilevazione di oggetti in un'immagine che combina reti CNN con la tecnologia dei transformer. Questo approccio unifica il processo di rilevazione e ottiene risultati di alta precisione senza la necessità di componenti post-elaborazione complessi.

L'obiettivo è fornire un modello in grado di apprendere nuove patologie della vite attraverso il fine-tuning con un dataset limitato di immagini. A tal proposito, dal Vine Leaf Vision Dataset sono state scorporate tutte le immagini facenti parte della classe *black rot* che rappresenta la malattia da apprendere, oggetto degli esperimenti. Questo approccio simula un contesto realistico in cui il modello pre-addestrato su foglie di vite può essere adattato, tramite l'operazione di fine-tuning, per riconoscere una nuova malattia utilizzando un dataset ristretto di immagini. Inoltre, lo stesso set di dati destinato alla fase di fine-tuning può essere rapidamente annotato in modo semi-automatico, con l'ausilio dell'IA, impiegando la strategia presentata nella sezione 3.2.

Per garantire quindi l'integrità degli esperimenti, è stato necessario rimuovere dal dataset originale tutte le immagini contenenti la malattia di interesse. Le fotografie rimosse sono state introdotte nel dataset di fine-tuning. Insieme alle immagini delle foglie malate, all'interno dello stesso set di dati sono state aggiunte quelle appartenenti alla classe *healthy*, estratte in modo proporzionale rispetto a quelle malate. Questo perché l'obiettivo del modello è distinguere le foglie malate da quelle infette. Il risultato di questa prima fase sono quindi due dataset: uno per la fase di pre-training ed uno per la fase di fine-tuning.

Successivamente, ciascun dataset è stato suddiviso in tre partizioni utili per l'addestramento: training (70%), validation (10%) ed infine test (20%). Per garantire l'uniformità delle classi tra le varie partizioni è stata utilizzata la suddivisione di tipo **stratified**. Questo metodo mantiene le proporzioni delle classi del dataset non partizionato all'interno delle partizioni, evitando sbilanciamenti durante la fase

di addestramento. Ad esempio, se nessuna immagine di una determinata malattia fosse nella partizione di training, il modello non sarebbe in grado di riconoscerla, rendendo quindi le metriche inaffidabili. Il risultato di questa operazione ha portato alle suddivisioni e proporzioni osservabili nella tabella 3.5.

Dataset	**#Immagini training**	**#Immagini validation**	**#Immagini test**
training dataset	5386	769	1538
fine-tuning dataset	1476	211	422

Tabella 3.5: Numero di immagini per ciascuna partizione derivanti dalla suddivisione di Vine Leaf Vision Dataset

Per valutare l'efficacia del dataset Vine Leaf Vision sono stati eseguiti quattro esperimenti:

- **A1** la versione yolov8n senza addestramento iniziale, addestrata sul fine-tuning dataset per riconoscere la differenza tra foglie affette da *black rot* e *healthy*. I pesi iniziali della rete sono stati inizializzati in modo casuale anziché tutti a zero poiché altrimenti ogni neurone riceverebbe lo stesso valore di gradiente durante la fase di backpropagation, portando tutti quelli dello stesso livello ad apprendere le stesse caratteristiche e limitando la velocità di convergenza.

  Le metriche ottenute da questo esperimento determineranno se il modello può apprendere la specifica malattia della vite con un numero limitato di esempi.

- **A2** la versione con pre-training di yolov8 fornita da [13], sottoposta ad un'operazione di fine-tuning per riconoscere la differenza tra foglie affette da *black rot* e *healthy*, adattando così la conoscenza delle piante generiche ad una singola malattia della vite. Il modello di base utilizzato possiede un valore di mAP molto alto ma non è specificamente addestrato sulle foglie delle viti.

  I risultati permetteranno di comprendere se la rete, con la conoscenza pregressa di altre specie di foglie, può specializzarsi su una nuova malattia di una singola pianta con un numero limitato di immagini.

- **A3** la versione yolov8n senza addestramento iniziale, addestrata prima utilizzando la porzione di pre-training e poi allenata sul dataset di fine-tuning per riconoscere la differenza tra foglie affette da *black rot* e *healthy*.

  Le metriche ottenute da questo esperimento permetteranno di capire se il dataset creato migliora l'apprendimento della rete neurale (rispetto all'esperimento A1) nel riconoscere la nuova malattia.

- **A4** la versione con pre-training di yolov8 fornita da [13], sottoposta ad un'ulteriore fase di addestramento con la porzione di pre-training ed infine un'operazione di fine-tuning per riconoscere la differenza tra foglie affette da *black rot* e *healthy*.

  I risultati che saranno ottenuti serviranno a determinare se l'ulteriore addestramento di una rete già allenata, permette di riconoscere in modo ancora più preciso (rispetto all'esperimento A2) la malattia presente sulle foglie delle viti.

Infine, come ultimo test, si è deciso di comparare l'accuratezza dell'architettura di YOLO con quella di DETR, dato che utilizzano due tecnologie diverse. Per effettuare questo confronto è stato condotto un solo esperimento:

- **B** DETR (con backbone ResNet50) senza pre-training, addestrata prima sulla porzione di pre-training e poi sul dataset di fine-tuning per riconoscere la differenza tra foglie affette da *black rot* e *healthy*. Questo esperimento si focalizza non tanto sull'efficacia del dataset ma piuttosto su quale architettura ottenga le migliori metriche, eseguendo le stesse operazioni dell'esperimento A3.

  Le metriche migliori indicheranno quale architettura si adatta maglio ai dati che sono a disposizione.

Per una comparazione ancora più accurata dei risultati si dovrebbero utilizzare gli stessi iperparametri implementati dallo studio [13] che viene utilizzato come fondamento per due esperimenti (A2 e A4). Dato che questi valori non sono stati rilasciati è stato deciso di utilizzare quelli standard definiti dalla libreria **ultralytics**, utilizzata per l'addestramento. In particolare gli iperparametri impostati sono i seguenti (il significato di ciascuno è presente nella sezione 3.6):

- numero di **epoche**: 500.
- **learning rate**: 0.01.
- **batch size**: 16.
- **workers**: 8.
- funzione di **ottimizzazione**: viene scelta in modo automatico dalla libreria in base alle configurazioni del modello. Nella maggior parte dei casi viene utilizzata la **SGD** oppure la **Adam**. La seconda è una versione che combina i vantaggi di due algoritmi: *AdaGrad* e *RMSProp* e tiene quindi traccia sia del valore medio dei gradienti sia del valore medio delle loro variazioni.

- **momentum factor**: 0.937. Questo valore accelera la convergenza verso il minimo globale riducendo le oscillazioni.

Come precedentemente introdotto, per poter eseguire l'addestramento è stata utilizzata la libreria ultralyitics. Questa fornisce la classe $YOLO$ che accetta come unico parametro il file contenente i pesi del modello desiderato. L'inizializzazione della classe restituisce un oggetto su cui è possibile invocare la funzione *train* nella quale si possono specificare gli iperparametri da utilizzare. Inoltre, è necessario definire il dataset da sfruttare durante la fase di addestramento. A tal fine, si deve creare un file $conf.yaml$ in cui vengono specificati il percorso per raggiungere il dataset e le etichette in esso contenute. La classe crea automaticamente una cartella in cui salva i risultati dell'addestramento e le metriche calcolate. Inoltre, è possibile anche configurare la connessione con Comet per avere una visualizzazione più chiara dei risultati.

È possibile consultare il dataset e gli esperimenti al seguente link Github: `https://github.com/fabriziosanino/vine_leaf_vision_dataset`.

Capitolo 4

Architettura dei modelli utilizzati

4.1 Classificazione

L'obiettivo di questo gruppo di reti neurali è l'assegnazione di un'etichetta ad una fotografia in base al suo contenuto. Esistono una grande varietà di modelli che permettono di eseguire questa operazione. In questa tesi l'attenzione è stata posta sulle seguenti reti:

- **ResNet (Residual Neural Network)**: reti di tipo CNN la cui architettura innovativa non solo consente un'efficace estrazione delle caratteristiche dalle immagini ma garantisce un'alta efficienza durante la fase di addestramento.

- **ViT (Vision Transformer)**: applicano il concetto visionario dei **Transformer** al riconoscimento di immagini. Potenzialmente, queste reti offrono prestazioni più competitive rispetto alle reti convoluzionali.

- **MobileNet**: famiglia di reti neurali di tipo CNN efficienti e leggere, progettate per essere eseguite su dispositivi mobili e apparati con risorse computazionali limitate come sistemi embedded.

4.1.1 ResNet - Residual Neural Network

Le ResNet sono un gruppo di reti neurali CNN, nello specifico Deep CNN, quindi basate su un numero molto elevato di layer **convoluzionali** (Vedi appendice A.1). Sono state introdotte nel 2015 all'interno dell'articolo "Deep Residual Learning for Image Recognition" [19].

Le reti standard Deep CNN presentano alcune problematiche note che sono state affrontate con l'introduzione di questa nuova architettura (ResNet). In particolare, i problemi di cui soffrivano erano due [20]:

- **vanishing/exploding gradient**: quando il numero di livelli dell'architettura aumenta, durante la fase di back-propagation il valore del gradiente calcolato per la funzione di loss diventa estremamente piccolo rispetto ai pesi della rete. Questo causa, nei livelli dell'architettura più vicini all'output, un problema nell'apprendimento. In altre parole, per questi livelli è più difficile apprendere features ed aggiornare i pesi in modo da avere un impatto nell'estrazione di queste ultime.

- **problema di degradation**: questo fenomeno si osserva quando l'aumentare della profondità della rete, non corrisponde un aumento delle performance ma bensì una degradazione di queste. Ciò significa che gli strati aggiuntivi inseriti non giustificano l'aumento della complessità dell'architettura. Questo perché le performance della rete dovrebbero andare di pari passo con l'aumento dei livelli della stessa.

Elementi chiave dell'architettura

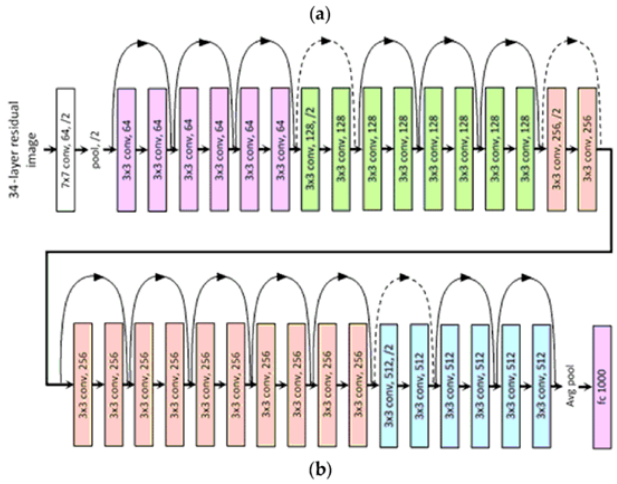

Figura 4.1: Architettura ResNet32. I collegamenti tra i vari layer indicano le skip-connections, quelli tratteggiati quando c'è anche un cambiamento della dimensione di input. Fonte: [21]

- **conv1**: questa è la prima operazione convoluzionale che viene applicata. Il suo obiettivo è quello di preparare l'input, quindi l'immagine, per poter essere processata dall'intera rete. Questa azione viene seguita da un'operazione di **Max Pooling**[1].

- **residual block e skip-connections**: tali attività sono state introdotte nelle ResNet per ovviare ai problemi di *exploding gradient* e *degradation*. In particolare, nelle reti che adottano questo blocco l'output $H(x)$ può essere rappresentato nel seguente modo:

$$H(x) = F(x) + x \tag{4.1}$$

  quindi l'input del blocco, x, viene sommato all'output del blocco $F(x)$ creando una residual connection. La presenza di x permette al gradiente di passare in modo più semplice da un livello ad un altro. Come si può osservare dalla figura 4.2, la freccia più esterna che collega l'input all'output è la **skip-connection** che permette all'input di bypassare lo strato convoluzionale. Questa soluzione consente di risolvere i problemi descritti in precedenza perché se un livello della rete danneggia le performance questo verrà evitato grazie alle skip-connection. Nella figura 4.1 si può osservare come il Residual Block è un'operazione che

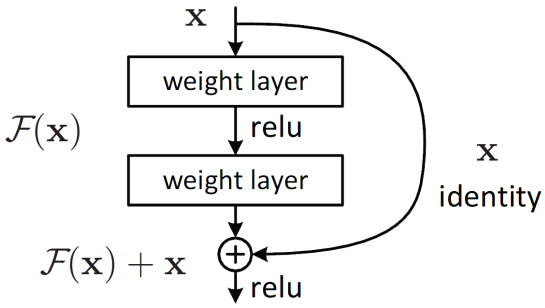

Figura 4.2: Residual block. Fonte: [19]

  viene effettuata per ogni livello della rete. Più nello specifico, ogni blocco è formato da due operazioni di convoluzione seguite dalla funzione di attivazione **ReLU** (Vedi appendice A.2).

- **fully connected layer**: questo è l'ultimo livello della rete il cui compito è prendere le features estratte dall'immagine e trasformale in una singola classe

[1]Operazione che calcola il valore massimo per ogni *patches* e usa i valori per creare un *downsample*.

che corrisponde all'output della rete. Il numero di neuroni che sono presenti nell'ultimo strato dipende da quante classi in output si vogliono ottenere. Ad esempio, per creare un classificatore multi-classe, basta inserire in questo livello un numero di neuroni maggiore di uno.

Per riassumere, il funzionamento della rete è questo: grazie alle convoluzioni vengono estratte delle informazioni riguardo all'immagine in input. I livelli convoluzionali più vicini all'input imparano a riconoscere forme più semplici, mentre quelli più lontani imparano forme geometriche più complesse. Una volta che le features sono state estratte vengono classificate e quindi viene prodotto l'output.

Ci sono varie versioni di ResNet che differiscono per il numero di livelli convoluzionali che presentano[2]. Più il numero di livelli aumenta, più lo farà anche il numero di parametri e quindi la rete sarà più dispendiosa in termini di calcolo e tempo richiesto. Inoltre, idealmente, più il numero di livelli è alto più la rete riuscirà a trovare delle features complesse. Ciò ovviamente dipende molto dalla complessità dell'input che viene fornito alla rete.

4.1.2 ViT - Vision Transformer

I ViT sono un gruppo di reti neurali di nuova generazione che sono state presentate come le alternative alle reti CNN perché si basano su un concetto totalmente diverso. Al giorno d'oggi, questo tipo di reti sono diventate lo stato dell'arte per i compiti di computer vision poiché riescono a raggiungere livelli di accuratezza mai visti prima. Questa architettura è stata introdotta nel 2020 all'interno dell'articolo "An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale" [22].

La peculiarità di queste reti è che non utilizzano operazioni convoluzionali ma bensì i **transformer encoder** (Figura 4.3). Il transformer è un concetto che proviene dai modelli di elaborazione del linguaggio naturale (**NLP**). Questo è stato poi successivamente trasferito ed adattato ai compiti di classificazione delle immagini.

Meccanismo di self-attention [24]

Il **self-attention** è lo strumento chiave delle architetture che si basano sui transformers. Ciò consente di catturare le dipendenze, anche molto lontane, che potrebbero essere presenti all'interno di un immagine. I modelli basati su questa tecnologia possono quindi elaborare aree dell'immagine in modo differenziato, tenendo conto

[2]Il numero di livelli è indicato nel nome della rete. Ad esempio, ResNet32 ha 32 livelli convoluzionali.

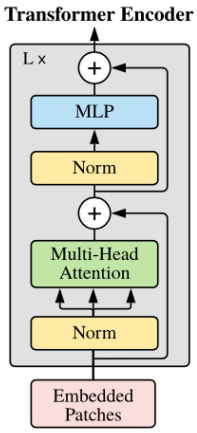

Figura 4.3: Transformer Encoder. Fonte: [23]

del tipo e della rilevanza della regione. Per funzionare, il meccanismo calcola una somma pesata dei dati in input, dove i pesi derivano dalle somiglianze che le features hanno tra di loro. Questa somma viene utilizzata successivamente per calcolare le **mappe di self-attention** (Figura 4.4). Queste ultime mostrano l'importanza delle diverse parti all'interno di un'immagine e possono essere visualizzate come **heatmap** dove quindi più il colore è acceso, più l'elemento è importante. Questa operazione è fondamentale perché permette ai modelli ViT di focalizzarsi solo sugli oggetti di interesse che sono presenti all'interno delle immagini e quindi, ad esempio, scartare lo sfondo che non contiene informazioni rilevanti.

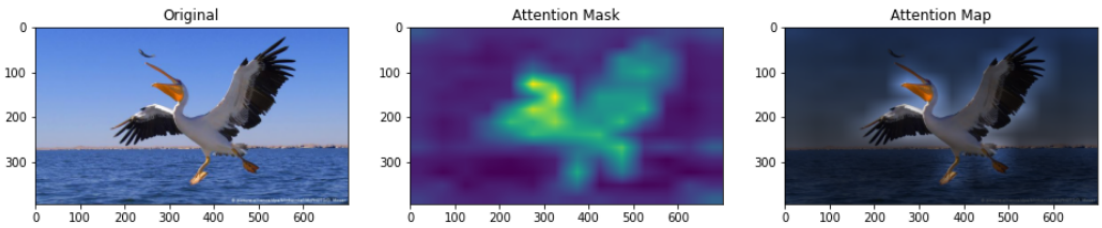

Figura 4.4: Mappa di self-attention. Fonte: [25]

I transformer encoder, implementano il meccanismo sopra citato ed includono i seguenti elementi:

- **Multi-Head Self Attention Layer (MSP)**: questo livello applica il meccanismo di self-attention per più volte ed in parallelo alle differenti sottosezioni dell'immagine. Ciò da la possibilità al modello di apprendere i pattern

dell'input in parallelo.

- **Multi-Layer Perceptrons (MLP)**: riceve l'output dell'operazione di self-attention ed applica una procedura di trasformazione non lineare che è chiamata **Gaussian Error Linear Unit (GELU)**. Questa fase fornisce al modello la possibilità di avere più flessibilità ed espressività.
- **Layer Norm**: tecnica che viene utilizzata per normalizzare le funzioni di attivazione provenienti dai diversi layer della rete. Permette di stabilizzare il processo di addestramento e le sue prestazioni.

Operazioni chiave dell'architettura

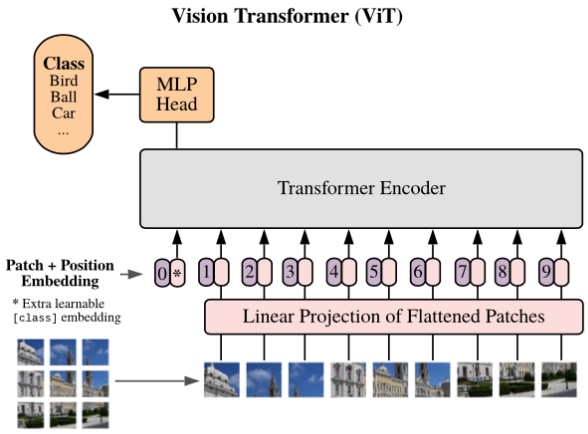

Figura 4.5: Architettura ViT. Fonte: [23]

Diversi sono i modelli che seguono ed adottano, anche in modo differente, le nuove tecnologie introdotte da ViT. Nonostante ciò, le operazioni che vengono in generale eseguite possono essere riassunte in questo modo:

- l'immagine di input viene suddivisa in **patches** (sotto-sezioni) di dimensione fissa. La misura delle sotto-sezioni dipende dal tipo di modello. Come per ResNet, anche in ViT il nome della rete porta con se alcune informazioni. In questo caso, nel nome viene indicato il numero di pathes che verranno generate. Ad esempio ViT_b_16 utilizza sotto-sezioni di dimensione 16x16 pixel.
- ciascuna sotto-sezione viene ridotta, nel numero di dimensioni, in modo da poter essere processata.

- al risultato dell'operazione precedente viene applicata una **lower-dimensional linear embeddings**.
- siccome le sotto-sezioni sono disconnesse tra di loro, ad ognuna deve essere associato un valore che ne indichi la posizione all'interno dell'immagine di partenza. In questa fase viene fatta questa operazione di associazione tra patch e posizione.
- la sequenza di sotto-sezioni viene mandata in input al transformer encoder mostrato in precedenza. Per ogni rete possono essere presenti più livelli di questo tipo.

4.1.3 MobileNet

Si tratta di una famiglia di architetture di reti CNN progettate per essere leggere ed efficienti, ideali per dispositivi con risorse limitate come smartphone e altri apparecchi con capacità di elaborazione simili. Queste reti sono state introdotte nel 2017 nell'articolo "MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications" [26].

Le prime versioni di MobileNet si basano sull'idea della **depthwise separable convolution**, operazione che riduce significativamente il numero di parametri e la complessità computazionale richiesta rispetto ad una CNN standard, senza però sacrificare troppo la precisione della rete. Questo è possibile grazie ad una struttura efficiente per la realizzazione delle convoluzioni che prevede di scomporne l'operazione in due parti [27]:

1. **depth-wise convolution**: applica un singolo filtro per ogni canale di input. Questi canali generalmente sono 3 cioè rosso (R), verde (G) e blu (B). Questa operazione è diversa da ciò che viene effettuato nella convoluzione tradizionale perché in quella vengono applicati più filtri per ogni canale di input.

   Il costo della convoluzione standard può essere calcolato in questo modo:

   $$Costo_{conv-standard} = D_k \cdot D_k \cdot M \cdot N \cdot D_f \cdot D_f \tag{4.2}$$

   dove D_f è la dimensione di un lato della feature di input[3], D_k è la dimensione di un lato del kernel, M è il numero di canali in input infine N è il numero di canali in output.

   Invece, il costo della depth-wise convolution è questo:

   $$Costo_{depth-wise-conv} = D_k \cdot D_k \cdot M \cdot D_f \cdot D_f \tag{4.3}$$

[3]Si presuppone di avere in input feature e kernel di dimensione quadrata.

dove M è il numero di kernel che verranno applicati, di dimensione fissa ad 1. Come si può osservare, in questo tipo di operazione non c'è una dipendenza del costo rispetto ai canali di output. Questa variante della convoluzione può però essere utilizzata solo per filtrare il canale di input e non può essere utilizzata per altri scopi. Per questo motivo è stata introdotta la **point-wise convolution**.

2. **point-wise convolution**: combina gli input per produrre delle nuove features. Calcola la combinazione lineare dell'output che si ottiene dalla convoluzione depth-wise utilizzandone una di dimensione 1x1.

   Il costo di questa operazione è:

   $$Costo_{point-wise} = M \cdot N \cdot D_f \cdot D_f \tag{4.4}$$

   e quindi il costo totale per effettuare una depthwise separable convolution è:

   $$Costo_{depth-wise} = Costo_{depth-wise-conv} + Costo_{point-wise} \tag{4.5}$$

   Comparando il costo della convoluzione adottata dalle reti MobileNet rispetto alla convoluzione classica si ottiene:

   $$\frac{Costo_{depth-wise}}{Costo_{conv-standard}} = \frac{1}{N} + \frac{1}{D_k^2} \tag{4.6}$$

   Questo risultato mostra che, la convoluzione tradizionale ha circa 9 volte più operazioni di moltiplicazione rispetto alla versione ottimizzata.

Operazioni chiave dell'architettura

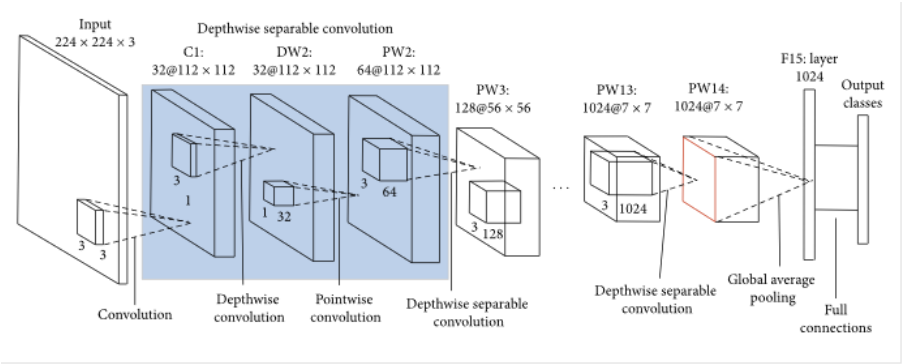

Figura 4.6: Architettura MobileNet. In blu è rappresentato il dettaglio della depthwise separable convolution. Fonte: [28]

In MobileNet ogni livello convoluzionale è seguito da un'operazione di normalizzazione e successivamente da un'operazione di attivazione di tipo ReLU. Come in tutte le altre reti neurali analizzate è inoltre presente un ultimo livello completamente connesso che permette di ridurre la dimensione del risultato e quindi restituire la classe di output.

4.2 Object Detection

I modelli di object detection sono stati utilizzati nella seconda parte della tesi. Questo perché permettono di raggiungere un livello di dettaglio più elevato poiché individuano gli oggetti di interesse nell'immagine e li classificano. Nel dettaglio, in questo progetto, le architetture che sono state utilizzate sono le seguenti:

- **YOLO (You Only Look Once)**: è un modello di object-detection principalmente utilizzato per compiti in real-time. Queste architetture sono in grado di predire la bounding box e la probabilità che questa appartenga ad una determinata classe in simultanea (da qui il nome). Inoltre, sono basate su operazioni convoluzionali.
- **DETR (DEtection TRansformer)**: sono delle reti di nuova generazione che utilizzano il concetto di transformers eliminando la necessità di dover utilizzare componenti standard come le convoluzioni nelle operazioni più importanti.

4.2.1 YOLO - You Only Look Once

Gli YOLO sono un gruppo di reti neurali basate su diversi livelli convoluzionali. Sono state introdotte nel 2015 all'interno dell'articolo "You Only Look Once: Unified, Real-Time Object Detection"[29].

La caratteristica principale di queste architetture è che sono **single-shot** cioè sull'immagine di input contemporaneamente identificano l'oggetto e ne assegnano la classe di appartenenza. Questo permette di ottenere delle performance molte elevate ed è questo il motivo per cui spesso vengo utilizzate per le identificazioni in tempo reale. La velocità, però, è a discapito della precisione infatti i modelli **two-shot**, cioè che prima estraggono gli oggetti tramite degli algoritmi di *region proposal* e successivamente associano una classe ad essi, sono più precisi perché le due fasi sono separate e quindi la rete può concentrarsi, in modo più approfondito su ciascuna.

Funzionamento dell'architettura (YOLOv1)

Come indicato in [31], il funzionamento dell'architettura può essere riassunto in questo modo: prima di tutto è presente una convoluzione il cui obiettivo è

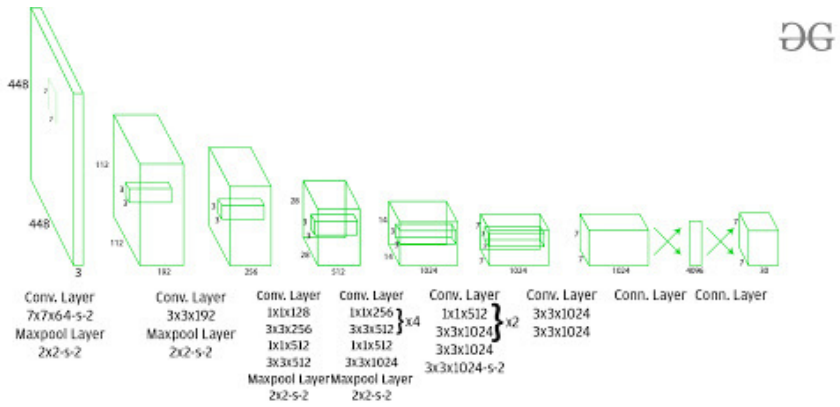

Figura 4.7: Architettura YOLO. Fonte: [30]

tracciare una griglia di dimensione SxS al di sopra dell'immagine. Ogni cella identificata produce due output: il primo è una serie di bounding box di una qualsiasi dimensione, dove però il centro di queste deve cadere all'interno della cella. Assieme alla bounding box è associato un livello di confidenza che indica quanto è probabile che ci sia un oggetto al suo interno. Il secondo output è una **mappa di probabilità** cioè per ogni cella è indicata la classe dell'oggetto (o sfondo) che è presente all'interno della stessa (Figura 4.8). Questi due output vengono combinati per ottenere solo le bounding box che effettivamente hanno una rilevanza nella predizione.

Più nel dettaglio, per ogni cella vengono prodotti due vettori: il primo contiene la probabilità di ogni classe (4.7) mentre il secondo una serie di bounding box che sono il risultato dell'operazione di predizione (4.8).

$$[p(c_1), p(c_2), ..., p(c_n)] \tag{4.7}$$

$$\begin{aligned} [x_1, y_1, \sqrt{w_1}, \sqrt{h_1}, C_1], \\ [x_2, y_2, \sqrt{w_2}, \sqrt{h_2}, C_2], \\ \dots, \\ [x_B, y_B, \sqrt{w_B}, \sqrt{h_B}, C_B] \end{aligned} \tag{4.8}$$

dove $p(c)$ è la probabilità della classe considerata, x e y sono le coordinate del centro della bounding box, w e h larghezza ed altezza rispettivamente, calcolata in base alla cella, mentre C è il livello di confidenza di ciascuna. YOLO restituisce le coordinate delle bounding box in forma normalizzata cioè $0 \leq x, y, w, h \leq 1$.

Questi vettori vengono calcolati, come mostrato dall'architettura visibile in figura 4.7, da una serie di operazioni convoluzionali. Nell'immagine 4.8 è possibile osservare il funzionamento di YOLO su di un esempio.

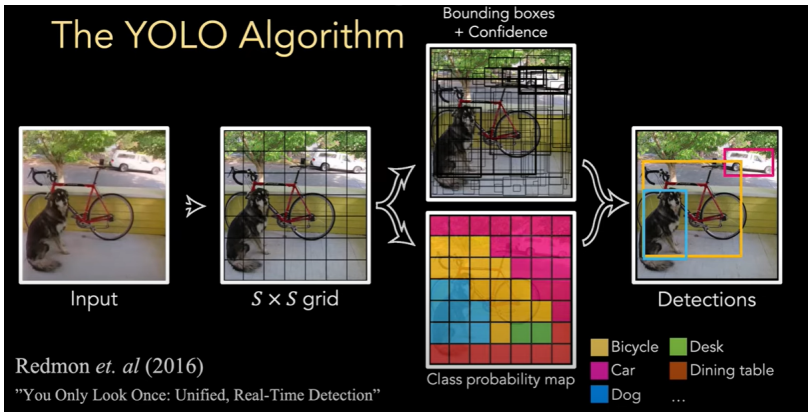

Figura 4.8: Funzionamento YOLO. Fonte: [31]

Funzionamento del training (YOLOv1)

Come per tutte le reti, anche questa necessita della fase di addestramento prima di poter essere utilizzata. YOLO richiede in input un gruppo di immagini e i riquadri di delimitazione per ciascun oggetto di interesse. Per ognuno deve essere anche presente la classe dell'oggetto contenuto. Le bounding box fornite in input, nella fase di addestramento, sono chiamate **ground truth** perché sono quelle utilizzate come fondamento per poter apprendere. Ognuna ha un centro, determinato dalle coordinate x ed y, che identifica la cella che sarà la responsabile nel predirla.

Il valore di confidenza viene calcolato come indice di IoU (3.25) tra la predizione ed il valore corretto ma solo per celle in cui si trova il centro di una bounding box di tipo ground truth. Per tutte le altre, il valore di confidenza viene impostato automaticamente a zero. Questo implica che verranno considerate solo le predizioni provenienti dalla stessa cella in cui è presente il centro di una ground trouth. Siccome ogni cella può generare più bounding box, si sceglie, per il calcolo della loss, di mantenerne solo una che corrisponde a quella con il valore più alto di confidenza. La formula che viene applicata è la seguente:

$$C_{finale} = argmax(C_1, C_2, ..., C_n) \tag{4.9}$$

Questa operazione prende il nome di **non max suppression** ed in sostanza permette di ridurre il numero di bounding box considerate e quindi diminuire quelle che sono sovrapposte.

Funzione di loss

Il calcolo della **loss** viene effettuato sommando due funzioni:

$$L = L_{cls} + L_{loc} \tag{4.10}$$

Il primo valore di loss L_{cls} opera sulle probabilità delle classi mentre il secondo L_{loc} sui parametri delle bounding box.

La *class loss* (L_{cls}) è determinata con la seguente procedura:

$$L_{cls} = \sum_{i=0}^{S^2} \mathbb{I}_i^{obj} \sum_{c \in classes} (p_i(c) - \hat{p}_i(c))^2 \tag{4.11}$$

dove $\mathbb{I}_i^{obj}$ vale uno se c'è un oggetto in quella cella i mentre altrimenti vale zero. Essenzialmente, la class loss non tiene conto del sfondo dell'immagine ma solo quei casi in cui un oggetto è stato scambiato per un altro oggetto.

Questa loss quindi calcola la somma al quadrato dell'errore tra le classi predette rispetto a quelle ground truth. La somma viene eseguita per tutte le classi e tutte le celle. La differenza sostanziale è che questa architettura non utilizza funzioni di loss standard come ad esempio la **cross entropy loss** che invece viene usata nella maggior parte delle altre reti.

La *localization loss* (L_{loc}) viene a sua volta calcolata come somma di altre due funzioni di loss:

$$L_{loc} = L_{coord} + L_{conf} \tag{4.12}$$

dove la *coordinate loss* (L_{coord}) è calcolata in questo modo:

$$L_{coord} = \lambda_{coord} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{I}_{ij}^{obj} l \tag{4.13}$$

$$l = (\sqrt{w_i} - \sqrt{\hat{w}_i})^2 + (\sqrt{h_i} - \sqrt{\hat{h}_i})^2 + (x_i - \hat{x}_i)^2 + (y_i - \hat{y}_i)^2 \tag{4.14}$$

cioè come la sommatoria tra tutte le celle e tutte le bounding box. Come in precedenza questa sommatoria vale solo per le celle i che sono responsabili nel predire la bounding box j.

La *confidence loss* (L_{conf}) è invece valutata nella seguente modalità:

$$L_{conf} = \sum_{i=0}^{S^2} \sum_{j=0}^{B} [\mathbb{I}_{ij}^{obj} (C_i - \hat{C}_i)^2] + \lambda_{no-obj} \sum_{i=0}^{S^2} \sum_{j=0}^{B} [\mathbb{I}_{ij}^{no-obj} (C_i - \hat{C}_i)^2] \tag{4.15}$$

il calcolo di questa è molto simile alla L_{coord}. Ciò che cambia è che le celle che non contengono un oggetto sono scalate rispetto al parametro λ_{no-obj} mentre quelle che contengono un oggetto non sono scalate.

Come si può vedere dalle formule utilizzate, l'object detection può essere visto come una sorta di problema di **regressione** e quindi questo rende YOLO in un certo senso semplice.

Non esiste una sola versione di YOLO ma ne esistono parecchie, partendo dalla meno recente v1 per arrivare sino a quella più recente cioè la v9. Così come per ResNet, anche per YOLO esistono varianti della stessa versione che hanno più o meno livelli.

4.2.2 DETR - DEtection TRansformer

DETR è un'architettura di nuova generazione utilizzata per l'object detection. Questa utilizza il meccanismo dei transformers per andare a sostituire le convoluzioni. È stata introdotta nel 2020 nell'articolo "End-to-End Object Detection with Transformers" [32].

Le reti in questione trattano il compito di object detection in modo completamente differente rispetto ad altre architetture come Faster R-CNN. Gli approcci principali utilizzati sono [33]:

- **predizione diretta**: invece che utilizzare il processo convenzionale che richiede prima una fase di **region proposal network (RPNs)** e successivamente la classificazione degli oggetti individuati, DETR effettua ciò in una singola operazione. In altre parole, considera tutti gli oggetti che sono presenti nell'immagine e predice le classi di ciascuna bounding box nello stesso momento.
- **transformer self-attention**: questo meccanismo permette al modello di imparare le relazioni o dipendenze complesse che sono presenti tra i vari oggetti nell'immagine.
- **predizione parallela**: utilizzando le informazioni ottenute dal meccanismo di self-attention, DETR predice in modo simultaneo la classe e la posizione di ogni oggetto. Generalmente, nelle altre architetture, questa operazione è eseguita in modo sequenziale.

Elementi chiave dell'architettura [34]

Sono tre i più importanti elementi che compongono l'architettura di DETR:

- **CNN backbone**: questa viene utilizzata per estrarre una versione compatta ma rappresentativa delle features presenti nell'immagine di input in modo che le operazioni successive possano compiere la localizzazione degli oggetti

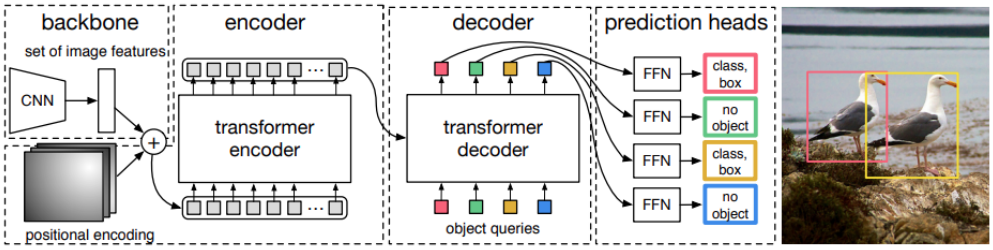

Figura 4.9: Architettura DETR. Fonte: [32]

in modo corretto. Generalmente viene utilizzata ResNet50 ma può essere modificata e sostituita con qualsiasi altro tipo di rete.

- **transformer encoder-decoder**: le operazioni eseguite dal transformer sono divise in due parti:

  1. **encoder**: prima di tutto è presente una convoluzione 1x1 che serve per ridurre la dimensione del canale di input. Successivamente, per ogni livello di encoder si applica il meccanismo di *multi-head self-attention*. Così come per ViT, anche qui il risultato viene aumentato con l'indicazione posizionale della patch.
  2. **decoder**: trasforma le patch o **embeddings** ricevute in input utilizzando l'**encoder-decoder attention**. La differenza più importante rispetto al decoder tradizionale è che in questo caso le patch vengono processate in parallelo per ogni livello di decoder. Grazie agli embeddings il modello fa delle previsioni di dove gli oggetti sono posizionati all'interno del contesto complessivo dell'immagine.

- **feed forward network (FFN)**: è un **preceptron**[4] con 3 livelli, avente come funzione di attivazione la ReLU. Questi layers predicono le coordinate del centro, altezza e larghezza della bounding box mentre il layer lineare predice l'etichetta della stessa utilizzando la funzione Softmax. Siccome viene predetto un numero N di bounding box, dove N è molto più grande del numero di oggetti che possiamo avere all'interno dell'immagine di input, viene aggiunta una classe che rappresenta "nessun oggetto".

La funzione di loss si ottiene in due fasi: nella prima si calcolano le migliori previsioni rispetto alle bounding box ground truth utilizzando un grafo con una

[4]Algoritmo di apprendimento supervisionato per la classificazione binaria sviluppato negli anni '50. È il più semplice modello di rete neurale.

funzione di costo. Nella seconda fase, la loss penalizza le classi e le posizioni delle bounding box che sono state predette erroneamente.

Il vantaggio di DETR, a differenza degli altri modelli di object detection, è che è completamente **end-to-end**. Questo significa che non richiede fasi multiple di training né aggiustamenti manuali dei parametri, poiché riesce automaticamente ad allenare simultaneamente diversi elementi dell'architettura.

Capitolo 5

Risultati

In questo capitolo saranno presentate le metriche ottenute dell'esecuzione degli esperimenti descritti nei capitoli precedenti. È importante sottolineare che la relazione tra le caratteristiche del modello e le metriche raggiunte è cruciale per gli obiettivi di questa tesi. I modelli selezionati per l'identificazione delle malattie dovranno essere implementati su sistemi embedded che dispongono di risorse hardware limitate. Questo perché l'obiettivo finale del progetto **DIVINE**, a cui questa tesi è correlata, è l'identificazione in tempo reale delle patologie direttamente nei vigneti. Pertanto, è essenziale trovare un equilibrio tra prestazioni, capacità di generalizzazione del modello (entrambe tendono ad aumentare con l'incremento dei parametri della rete, il che comporta un aumento del peso del modello), efficienza in termini di memoria occupata ed infine velocità di elaborazione durante la fase di inferenza, in modo tale da rispettare i limiti temporali previsti delle applicazioni utilizzate in tempo reale.

5.1 Classificazione

I risultati ottenuti dalle reti neurali dopo l'esecuzione degli esperimenti sono mostrati nella tabella 5.2. I dati fanno riferimento alla partizione di test di 6195 immagini corrispondenti al 25% di ESCA-dataset. Inoltre, nella tabella 5.1, sono presenti le principali caratteristiche delle architetture esaminate, con particolare attenzione al peso del modello e al numero di parametri. Il tempo di inferenza mostrato nella stessa è calcolato come media dei tempi di elaborazione di una singola immagine, ripetuto su 100 immagini differenti, selezionate da ESCA-dataset in modo casuale. L'hardware utilizzato dispone di una scheda grafica NVIDIA GTX con 4GB di GPU ed un sistema con 16GB di RAM.

È notevole come la tecnologia dei transformer adottata da ViT, pur avendo più parametri ed occupando più memoria, risulti più veloce di MobileNet. Questa

Modello	**#Parametri (Milioni)**	**Peso (Mb)**	**Tempo inferenza (sec)**
CNN [4]	1.9	6	-
ResNet18	11	44.7	**0.0064**
MobileNet v3	**2**	**6**	0.0148
ViT b 16	86	327	0.0165

Tabella 5.1: Caratteristiche dei modelli di classificazione comparati

velocità deriva dall'architettura della rete che consente un notevole parallelismo. È quindi necessario sottolineare che tali tempi sono possibili grazie all'utilizzo di hardware che supporta il calcolo in parallelo (GPU), risorse che non sono comunemente disponibili su sistemi embedded.

I risultati degli esperimenti (tabella 5.2) mostrano come ResNet18, nonostante abbia più parametri ed una architettura più complessa rispetto allo studio comparato, raggiunga prestazioni inferiori in termini di accuratezza. Questo perché l'architettura CNN creata ad hoc è stata progettata specificamente per il compito al contrario di ResNet che è generale. Nonostante ciò, tutte le altre architetture esaminate hanno ottenuto ottime performance nelle varie metriche analizzate. Come già anticipato, la scelta del modello ottimale varia in base alle risorse disponibili: ViT, con la sua elevata precisione e capacità di apprendimento è preferibile su hardware di ultima generazione, mentre MobileNet è la soluzione ideale per sistemi con risorse limitate, mantenendo comunque un'accuratezza competitiva e garantendo un significativo risparmio di risorse. Infatti, con solo $\frac{1}{5}$ dei parametri di ResNet ed $\frac{1}{43}$ di quelli di ViT ottiene un incremento di circa 1 punto percentuale rispetto a ResNet mentre solo un decremento di 0.5 punti percentuali nei confronti di ViT.

Modello	**Accuracy %**	**Precision % (esca - healthy)**	**Recall % (esca - healthy)**
CNN [4]	99.16	-	-
ResNet18	98.66	98.44 - 97.48	97.05 - 98.43
MobileNet v3	99.55	99.62 - 96.95	97.02 - 99.61
ViT b 16	**1.0**	**100 - 98.29**	**98.31 - 100**

Tabella 5.2: Risultati dell'operazione di classificazione raccolti utilizzando la partizione di test di ESCA-dataset

5.2 Object Detection

I risultati ottenuti dalle reti neurali dopo l'esecuzione degli esperimenti sul dataset di fine-tuning sono osservabili nella tabella 5.4. In aggiunta, è possibile consultare in 5.3 le caratteristiche più importanti delle due architetture esaminate.

L'hardware utilizzato per l'addestramento e la valutazione dei modelli dispone di una scheda grafica NVIDA GeForce RTX 4070 con 12GB di GPU ed un sistema con 64GB di RAM.

Modello	**#Parametri (Milioni)**	**Peso (Mb)**	**Tempo inferenza (msec)**
YOLOv8n	**3**	**6.3**	**1.9**
DETR	41	160	4

Tabella 5.3: Specifiche dei modelli di object detection comparati

Test	**A1**	**A2**	**A3**	**A4**
Accuracy (%)	97.1	96.5	**98.1**	97.5
Precision (%)	97.9	97.8	98	**98.2**
Recall (%)	96.4	96.3	**97.9**	97
mAP50 (%)	98.4	98.1	**99.1**	99
mAP50-95 (%)	93.8	94.5	**94.9**	94.8

Tabella 5.4: Risultati per gli esperimenti del gruppo A, raccolti utilizzando la partizione di test del dataset di fine-tuning

I risultati della tabella 5.4 mostrano che l'operazione di pre-training eseguita utilizzando Vine Leaf Vision Dataset negli esperimenti A3 e A4, ha portato ad un miglioramento generale delle prestazioni rispetto agli esperimenti A1 e A2 che non hanno beneficiato del pre-training. In particolare, utilizzando la metrica mAP@50-95, che integra informazioni sia sulla classificazione che sull'object detection, si osserva un incremento di 1.1 punti percentuali da A1 a A3 e di 0.3 punti percentuali da A2 a A4. Questo miglioramento è dovuto al fatto che il modello ha acquisito, tramite il dataset, una maggiore conoscenza delle foglie di vite durante il pre-training, permettendogli di specializzarsi in modo più approfondito durante la fase di fine-tuning. Incrementi di questo tipo possono sembrare poco rilevanti ma

superando la soglia del 90% significa che l'accuratezza del modello è già molto alta e quindi il margine di miglioramento è ridotto.

L'esperimento che ha ottenuto le prestazioni migliori (A3) ha evidenziato come il punto di forza sia la rilevazione della posizione delle foglie (object detection), infatti le metriche di mAP sono molto alte in quell'ambito. Tuttavia, è presente un margine di miglioramento nella fase di classificazione della foglia stessa (accuracy, precision e recall). Questa lacuna potrebbe essere colmata in futuro, aumentando la varietà del dataset tramite la data augmentation in modo da dare maggiore rappresentatività alle classi meno presenti all'interno del dataset di pre-training. In aggiunta a questo, il numero di *false positive* nella confusion matrix può essere ridotto andando ad introdurre nel dataset almeno il 10% di immagini **background** (valore indicato dagli sviluppatori della libreria ultralytics) cioè immagini che non contengono alcun oggetto di interesse al loro interno. Il valore di questi è cruciale nel calcolo delle metriche di classificazione poiché, riducendolo, si potrà ottenere un miglioramento generale delle metriche.

Test	**A3**	**B**
Accuracy (%)	98.1	**98.5**
Precision (%)	**98**	97.6
Recall (%)	97.9	**98.5**
mAP50 (%)	99.1	**99.4**
mAP50-95 (%)	94.9	**95.7**

Tabella 5.5: Comparazione tra il miglior esperimento del gruppo A rispetto all'architettura DETR utilizzando la partizione di test del dataset di fine-tuning

Nonostante questi ottimi risultati forniti dall'architettura YOLO, quella che ottiene le metriche migliori a parità di condizioni rispetto all'esperimento A3, è DETR cioè l'architettura che si basa sulla tecnologia innovativa dei transformer (tabella 5.5). Quest'ultima supera YOLO di 0.8 punti percentuali nella metrica mAP@50-95. Tuttavia, il problema di questa è il numero di parametri ed il tempo di inferenza calcolati. Infatti, DETR ha oltre 10 volte la mole di parametri rispetto a YOLO ed impiega più del doppio del tempo per riconoscere e classificare le foglie in un'immagine. Le prestazioni, in termini di accuratezza, non vanno quindi a giustificare l'aumento consistente delle capacità computazionali richieste per la sua esecuzione.

Alcune osservazioni molto importanti possono essere effettuate consultando i risultati dopo la fase di pre-training. Nella tabella 5.6 sono riportate le metriche ottenute. Come era prevedibile, dato il numero elevato di malattie da riconoscere

ed una non omogeneità del numero di immagini tra le varie classi, i valori per le metriche di classificazione e quelli dell'object detection risultano relativamente bassi, se comparati a quelli dopo l'operazione di fine-tuning. In realtà non è così: come indicato da uno sviluppatore della libreria ultralytics in [35], il valore di mAP@50 quando è sopra al 50% è considerato molto positivo nella maggior parte delle applicazioni, mentre mAP@50-95 siccome è più stringente, valori sopra al 30% sono soddisfacenti sopratutto in dataset complessi come questo. Inoltre, l'obiettivo della fase di pre-training non è riconoscere in modo dettagliato varie malattie, bensì fornire un base solida al modello così che nella fase di fine-tuning si possa trasferire la conoscenza generale su di una specifica malattia.

Test	**A3**	**A4**	**B**
Accuracy (%)	69.9	69.7	**71.9**
Precision (%)	**70**	52.7	65.3
Recall (%)	54.4	**57.3**	56
mAP50 (%)	56.7	55.5	**58.1**
mAP50-95 (%)	46	45.2	**47.9**

Tabella 5.6: Risultati dell'operazione di object detection raccolti utilizzando la partizione di test del dataset di training

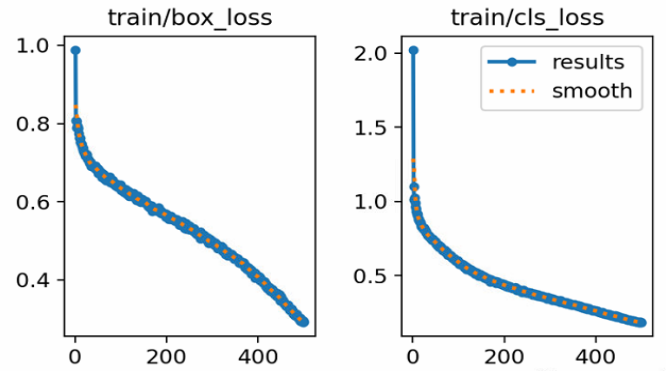

Figura 5.1: Andamento del training loss per l'esperimento A4

Infine, nel grafico presente in figura 5.1 è rappresentato l'andamento della *training loss* durante la fase di pre-training di 500 epoche sul Vine Leaf Vision Dataset per l'esperimento A4. Come si può osservare, sia la **box_loss** (loss relativa alle

bounding box) che la **cls_loss** (loss relativa alla classificazione delle bounding box) mostrano un calo progressivo man mano che l'addestramento procede, indicando che il modello sta apprendendo nuove caratteristiche del dataset e riducendo l'errore calcolato. Tuttavia, i valori corrispondenti per la *validation loss*, visibili in figura 5.2, iniziano ad aumentare dopo circa 170 epoche, segnalando un possibile problema di **overfitting**. Questo fenomeno indica che il modello inizia a sovra-adattarsi ai dati di addestramento, perdendo così la sua capacità di generalizzazione. Per questo motivo, nei successivi esperimenti è stato utilizzato il modello con le metriche migliori prima dell'inizio del possibile overfitting.

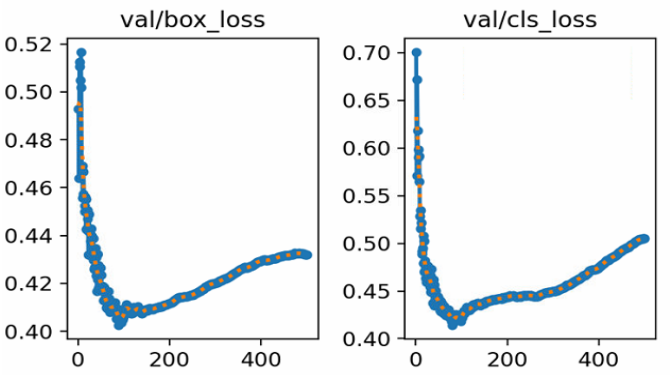

Figura 5.2: Andamento del validation loss per l'esperimento A4

Capitolo 6

Conclusioni

Le attività mostrate e presentate in questa tesi hanno permesso di raggiungere gli obiettivi prefissati, portando a significativi pregressi nell'ambito della diagnosi automatica delle malattie delle viti.

6.1 Classificazione

I risultati hanno dimostrato che l'utilizzo di architetture create su misura non raggiungono livelli di accuratezza adeguati per diagnosticare con precisione le malattie sulle foglie delle viti. La soluzione migliore consiste nello sfruttare le reti già progettate e consolidate come ViT o MobileNet. In particolare, **MobileNet** ha ottenuto i risultati migliori per l'ambito in cui dovrà essere applicata: sebbene sia leggermente meno accurata di ViT, è decisamente più adatta per l'esecuzione su sistemi embedded, poiché contiene meno parametri e quindi è più leggera.

6.2 Object Detection

L'analisi dello stato dell'arte, condotta per raggiungere l'obiettivo (1), ha fornito una comprensione approfondita del settore, rilevando lacune e potenziali contributi. Questa ha portato alla scoperta di quattro dataset pubblici contenti fotografie di singole foglie e pareti fogliari di vite.

Lo studio ha condotto alla creazione di Vine Leaf Vision Dataset, obiettivo (2), attraverso l'integrazione, annotazione e il miglioramento dei dataset pubblicati. Per eseguire l'operazione di annotazione e ridurre i tempi impiegati, sono state utilizzate e messe a disposizione tecniche assistite dall'AI. Il Vine Leaf Vision Dataset è composto da quasi **10.000** immagini di foglie e pareti fogliari, con oltre **38.000** annotazioni (bounding box). Queste sono basate su di un'attenta selezione dei pattern visivi più comunemente osservabili sulle foglie analizzate.

La creazione del dataset è un risultato importante poiché come emerso degli studi iniziali, la disponibilità di collezioni di dati ampie ed annotate con bounding box per ciascuna foglie è limitata o inesistente. Inoltre, lo stesso, può rappresentare una risorsa significativa per il pregresso della ricerca.

Infine, per garantire il corretto funzionamento del dataset è stato raggiunto l'obiettivo (3). Eseguendo le operazioni di pre-training sui modelli YOLO e DETR, si è evidenziato un incremento dell'accuratezza nel riconoscere le foglie affette da una nuova malattia rispetto agli stessi modelli a cui il pre-training non è stato applicato. La soluzione migliore è risultata essere **YOLO** poiché garantisce elevati livelli di accuratezza sia nella classificazione che nell'object detection, mantenendo un numero di parametri ed un peso accettabile per i sistemi embedded. DETR, pur ottenendo le metriche migliori, non è sostenibile a causa dell'enorme carico di lavoro richiesto dall'esecuzione di una rete con così tanti parametri.

Il dataset raccolto in questo studio, i modelli allenati per verificare il funzionamento del dataset, il codice per l'addestramento ed il fine-tuning sono stati resi pubblicamente disponibili. Questi rilasci costituiscono un ottimo punto di partenza per future ricerche nel campo, permettendo ad altri ricercatori di estendere e migliorare l'attuale dataset, nonché di effettuare benchmarking di altri modelli e tecniche grazie alla baseline prodotte da questo studio.

6.3 Sviluppi futuri

Utilizzando Vine Leaf Vision Dataset fornito da questo studio è possibile proseguire con ulteriori esperimenti e miglioramenti.

Per prima cosa, siccome il dataset presenta una disparità molto evidente tra le diverse classi, portando i modelli allenati ad essere più efficienti nel riconoscimenti delle malattie più rappresentate. Lo sviluppo che dovrà sicuramente essere apportato a questo set di dati è l'applicazione di alcune operazioni di data augmentation in modo tale da bilanciare la rappresentatività degli elementi di ciascuna classe. È stato dimostrato da diversi studi citati che questo tipo di operazioni permette di raggiungere notevoli miglioramenti in termini di accuratezza.

Successivamente, siccome nel dataset creato sono presenti sia immagini di singole foglie, che immagini di intere pareti fogliari, un'ulteriore esperimento dovrebbe essere quello di suddividere la fase di pre-training in due parti. Nella prima, si utilizzano solo foglie singole in modo che i modelli imparino in maniera precisa ed accurata le caratteristiche specifiche degli oggetti che dovranno essere identificati, cioè le foglie. Nella seconda fase si dovrà eseguire un ri-addestramento su immagini di intere pareti fogliari. Questo dovrebbe portare quindi ad un miglioramento dell'accuratezza

della predizione dei modelli in quanto hanno già sufficiente conoscenza da poter applicare per imparare pattern all'interno di fotografie più complesse.

Infine, dato che durante la prima parte della ricerca ci si è focalizzati sull'analisi di architetture per la sola classificazione, si potrebbero condurre esperimenti per valutare se, collegando attraverso una **pipeline** i modelli di object detection specifici per l'identificazione delle foglie a modelli di classificazione delle malattie, si possano ottenere metriche migliori. Questo perché ciascuna parte della pipeline sarebbe focalizzata su un compito ben specifico: la prima si occuperebbe di trovare le foglie nell'immagine mentre la seconda classificherebbe la malattia presente su di ciascuna di esse. Probabilmente la creazione della pipeline porterà ad una diminuzione delle prestazioni di inferenza poiché saranno richieste un maggior numero di operazioni, che però potrebbero essere giustificate da un aumento considerevole dell'accuratezza.

Appendice A

Operazioni tra tensori

A.1 Convoluzione [36]

La **Convoluzione** è l'operazione matematica fondamento delle CNN. Questa applica un **filtro** (più comunemente chiamato **kernel**) all'immagine di input e restituisce una versione raffinata dell'immagine originale. In particolare, il valore del kernel viene moltiplicato per ogni pixel presente nell'immagine di input. Successivamente, i risultati della moltiplicazione di ciascun elemento del filtro per il valore del pixel vengono sommati ed andranno a formare il nuovo valore. Infine, il filtro viene spostato sul pixel successivo finché non è stata coperta l'intera immagine.

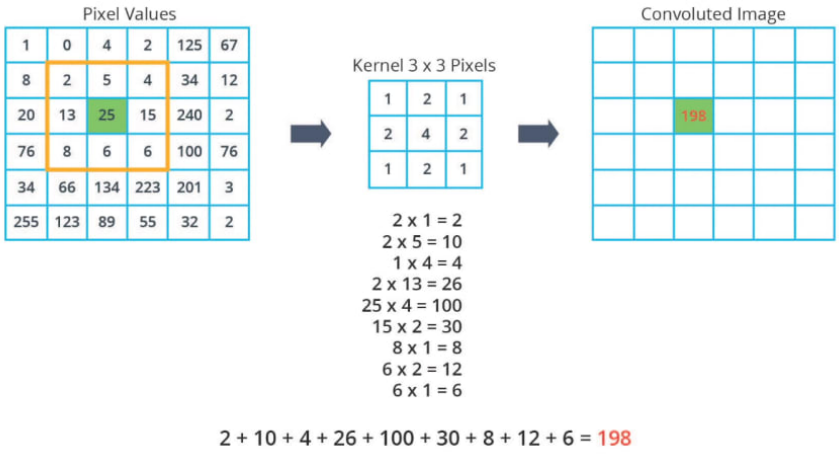

Figura A.1: Operazione di convoluzione. Si parte con il filtro in alto a sinistra e si finisce con il pixel in basso a destra. Fonte: [37]

Nel caso il cui kernel non avesse una dimensione multipla dell'immagine, è necessario aggiungere un bordo per tutto il suo perimetro evitando che vi siano moltiplicazioni nulle.

La convoluzione è un'operazione fondamentale quando si elaborano le immagini poiché permette di estrarre le caratteristiche più importanti in modo efficiente. Un esempio di informazioni che possono essere estratte dalle immagini sono le linee, gli angoli, le forme, ecc. Una peculiarità fondamentale di questa operazione è la proprietà **shift-invariant**. L'output dell'operazione di convoluzione è lo stesso indipendentemente dalla posizione in cui il filtro è applicato sull'immagine. Questo significa che è possibile effettuare l'operazione in parallelo sull'immagine, andando a risparmiare tempo nell'esecuzione.

A.2 ReLU [38]

La **Rectified Linear Unit** è la funzione di attivazione maggiormente utilizzata per introdurre la non linearità nei modelli di DL. Questa funzione ritorna 0 se l'input ricevuto è negativo, mentre per ogni valore positivo x, ritorna in output x. Il funzionamento può essere sintetizzato con questa formula:

$$ReLU(x) = max(0, x) \tag{A.1}$$

La funzione di attivazione ha principalmente due scopi:

1. aiutare il modello, a cui viene applicata, nel risolvere l'inconveniente dell'**interaction effects**. Questa criticità si manifesta quando il valore di una variabile, A, ha effetti differenti sulla predizione che dipendono dal valore di un'altra variabile B.

2. permette al modello di non avere un comportamento **lineare**. Questo evita che per ogni input il modello abbia sempre la stessa predizione. La versione semplificata della ReLU mostrata in A.1 permette la non sistematicità ma è troppo semplice. Infatti la maggior parte dei modelli include nella formula un **bias** per ogni nodo cioè un valore costante che viene calcolato durante il training della rete per introdurre un maggiore grado di non linearità.

A.3 Softmax [39]

La **softmax** è una funzione che trasforma l'output ottenuto da una rete neurale in un vettore di probabilità. In altre parole, fornisce la distribuzione della probabilità tra le possibili classi di output. Supponendo che la rete dia in output N classi, la softmax restituisce un vettore lungo N dove per ogni indice i è presente la probabilità che l'immagine di input appartenga alla classe i considerata.

Il calcolo della softmax è effettuato in questo modo:

$$softmax(x)_i = \frac{e_i^Z}{\sum_{j=0}^{N} e_j^Z} \tag{A.2}$$

dove z è il vettore di outputs che provengono dalla rete, e è la costante che vale circa 2.718 mentre i è la posizione all'interno del vettore di probabilità finale.

Molto importante è notare che tutti gli output della funzione softmax sono compresi tra 0 e 1 e la somma di tutti questi è uguale ad 1.

Bibliografia

[1] Melissa B Riley, Margaret R Williamson e Otis Maloy. «Plant disease diagnosis». In: *The plant health instructor* 10 (2016) (cit. alle pp. 1, 4).

[2] F.A. Princi Rani, S.N Kumar, A Lenin Fred, Charles Dyson, V. Suresh e P.S Jeba. «K-means Clustering and SVM for Plant Leaf Disease Detection and Classification». In: *2019 International Conference on Recent Advances in Energy-efficient Computing and Communication (ICRAECC)*. 2019, pp. 1–4. DOI: 10.1109/ICRAECC43874.2019.8995157 (cit. alle pp. 2, 5, 6).

[3] Muhammad Hammad Saleem, Johan Potgieter e Khalid Mahmood Arif. «Plant Disease Detection and Classification by Deep Learning». In: *Plants* 8.11 (2019). ISSN: 2223-7747. DOI: 10.3390/plants8110468. URL: https://www.mdpi.com/2223-7747/8/11/468 (cit. alle pp. 2, 6).

[4] M. Alessandrini, R. Calero Fuentes Rivera, L. Falaschetti, D. Pau, V. Tomaselli e C. Turchetti. «A grapevine leaves dataset for early detection and classification of esca disease in vineyards through machine learning». In: *Data in Brief* 35 (2021), p. 106809. ISSN: 2352-3409. DOI: https://doi.org/10.1016/j.dib.2021.106809. URL: https://www.sciencedirect.com/science/article/pii/S2352340921000937 (cit. alle pp. 2, 7, 10, 14, 35, 36, 59).

[5] Xiaoyue Xie, Yuan Ma, Bin Liu, Jinrong He, Shuqin Li e Hongyan Wang. «A Deep-Learning-Based Real-Time Detector for Grape Leaf Diseases Using Improved Convolutional Neural Networks». In: *Frontiers in Plant Science* 11 (2020). ISSN: 1664-462X. DOI: 10.3389/fpls.2020.00751. URL: https://www.frontiersin.org/journals/plant-science/articles/10.3389/fpls.2020.00751 (cit. alle pp. 2, 9, 11).

[6] Yang Zhang, Chenglong Song e Dongwen Zhang. «Deep Learning-Based Object Detection Improvement for Tomato Disease». In: *IEEE Access* 8 (2020), pp. 56607–56614. DOI: 10.1109/ACCESS.2020.2982456 (cit. alle pp. 2, 8).

[7] Zhao Zhang, Yongliang Qiao, Yangyang Guo e Dongjian He. «Deep Learning Based Automatic Grape Downy Mildew Detection». In: *Frontiers in Plant Science* 13 (2022). ISSN: 1664-462X. DOI: 10.3389/fpls.2022.872107. URL: https://www.frontiersin.org/journals/plant-science/articles/10.3389/fpls.2022.872107 (cit. alle pp. 2, 10, 11).

[8] Wasswa Shafik, Ali Tufail, Abdallah Namoun, Liyanage Chandratilak De Silva e Rosyzie Anna Awg Haji Mohd Apong. «A Systematic Literature Review on Plant Disease Detection: Motivations, Classification Techniques, Datasets, Challenges, and Future Trends». In: *IEEE Access* 11 (2023), pp. 59174–59203. DOI: 10.1109/ACCESS.2023.3284760 (cit. alle pp. 2, 12).

[9] Bin Liu, Zefeng Ding, Liangliang Tian, Dongjian He, Shuqin Li e Hongyan Wang. «Grape Leaf Disease Identification Using Improved Deep Convolutional Neural Networks». In: *Frontiers in Plant Science* 11 (2020). ISSN: 1664-462X. DOI: 10.3389/fpls.2020.01082. URL: https://www.frontiersin.org/journals/plant-science/articles/10.3389/fpls.2020.01082 (cit. alle pp. 2, 11).

[10] Davinder Singh, Naman Jain, Pranjali Jain, Pratik Kayal, Sudhakar Kumawat e Nipun Batra. «PlantDoc: A Dataset for Visual Plant Disease Detection». In: gen. 2020, pp. 249–253. DOI: 10.1145/3371158.3371196 (cit. a p. 8).

[11] PIYUSH MISHRA. In: *Kaggle* (2019). URL: https://www.kaggle.com/datasets/piyushmishra1999/plantvillage-grape (cit. a p. 14).

[12] Daniel Chitwood. «Data from: Vitis vinifera leaf images». In: (giu. 2021). DOI: 10.5061/dryad.g79cnp5mn. URL: https://doi.org/10.5061/dryad.g79cnp5mn (cit. a p. 14).

[13] Graduation Project 2023. *Plants Diseases Detection and Classification Dataset*. https://universe.roboflow.com/graduation-project-2023/plants-diseases-detection-and-classification. Open Source Dataset. visited on 2024-06-03. Ago. 2023. URL: https://universe.roboflow.com/graduation-project-2023/plants-diseases-detection-and-classification (cit. alle pp. 14, 40, 41).

[14] Nehul Agrawal e Pranjal singh Thakur. «YOLOv8s Leaf Detection and Classification». In: (2023). URL: https://huggingface.co/foduucom/plant-leaf-detection-and-classification (cit. alle pp. 19, 28).

[15] Andrea Minini. «Matrice di confusione». In: (2021). URL: https://www.andreaminini.com/ai/machine-learning/matrice-di-confusione (cit. a p. 31).

[16] ml-science. «Confusion Matrix». In: (2023). URL: https://www.ml-science.com/confusion-matrix (cit. a p. 31).

[17] Towards Data Science. «What is Average Precision in Object Detection and Localization Algorithms and how to calculate it?» In: (2022). URL: https://towardsdatascience.com/what-is-average-precision-in-object-detection-localization-algorithms-and-how-to-calculate-it-3f330efe697b (cit. a p. 32).

[18] Aqeel Anwar. «What is Average Precision in Object Detection and Localization Algorithms and how to calculate it?» In: (2022). URL: https://towardsdatascience.com/what-is-average-precision-in-object-detection-localization-algorithms-and-how-to-calculate-it-3f330efe697b (cit. a p. 34).

[19] Kaiming He, Xiangyu Zhang, Shaoqing Ren e Jian Sun. *Deep Residual Learning for Image Recognition*. 2015. arXiv: 1512.03385 [cs.CV] (cit. alle pp. 43, 45).

[20] Azeem - I. «Understanding ResNet Architecture: A Deep Dive into Residual Neural Network». In: (2023). URL: https://medium.com/@ibtedaazeem (cit. a p. 44).

[21] Siddhesh Bangar. «Resnet Architecture Explained». In: (2022). URL: https://medium.com/@siddheshb008/resnet-architecture-explained-47309ea9283d (cit. a p. 44).

[22] Alexey Dosovitskiy et al. *An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale*. 2021. arXiv: 2010.11929 [cs.CV] (cit. a p. 46).

[23] Google Research. «vision transformer». In: (2020). URL: https://github.com/google-research/vision_transformer (cit. alle pp. 47, 48).

[24] viso.ai. «Vision Transformers (ViT) in Image Recognition Read». In: (2024). URL: https://viso.ai/deep-learning/vision-transformer-vit/ (cit. a p. 46).

[25] jo1jun. «Vision-Transformer». In: (2021). URL: https://github.com/jo1jun/Vision_Transformer (cit. a p. 47).

[26] Andrew G. Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto e Hartwig Adam. *MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications*. 2017. arXiv: 1704.04861 [cs.CV] (cit. a p. 49).

[27] Srudeep PA. «An Overview on MobileNet: An Efficient Mobile Vision CNN». In: (2020). URL: https://medium.com/@godeep48 (cit. a p. 49).

[28] wikidocs. «K-04 Understanding of MobileNet - EN». In: (2021). URL: https://wikidocs.net/165429 (cit. a p. 50).

[29] Joseph Redmon, Santosh Divvala, Ross Girshick e Ali Farhadi. *You Only Look Once: Unified, Real-Time Object Detection*. 2016. arXiv: 1506.02640 [cs.CV] (cit. a p. 51).

[30] geeksforgeeks. «YOLO : You Only Look Once – Real Time Object Detection». In: (2022). URL: https://www.geeksforgeeks.org/yolo-you-only-look-once-real-time-object-detection/ (cit. a p. 52).

[31] DeepBean. «How YOLO Object Detection Works». In: (2023). URL: https://www.youtube.com/watch?v=svn9-xV7wjk&t=169s&ab_channel=DeepBean (cit. alle pp. 51, 53).

[32] Nicolas Carion, Francisco Massa, Gabriel Synnaeve, Nicolas Usunier, Alexander Kirillov e Sergey Zagoruyko. *End-to-End Object Detection with Transformers*. 2020. arXiv: 2005.12872 [cs.CV] (cit. alle pp. 55, 56).

[33] Petru Potrimba. «What is DETR». In: (2023). URL: https://blog.roboflow.com/what-is-detr/ (cit. a p. 55).

[34] Deval Shah. «Revolutionary Object Detection Algorithm from Facebook AI». In: (2020). URL: https://medium.com/visionwizard/detr-b677c7016a47 (cit. a p. 55).

[35] glenn-jocher. «What is Box(P R mAP50 mAP50-95)?» In: (2024). URL: https://github.com/ultralytics/ultralytics/issues/9446 (cit. a p. 62).

[36] Gabriel Rennó. «What is convolution?» In: (2022). URL: https://medium.com/latinxinai/what-is-convolution-ceb5a3bab020 (cit. a p. 68).

[37] Sandeep Balachandran. «Machine Learning - Convolution with color images». In: (2020). URL: https://dev.to/sandeepbalachandran/machine-learning-convolution-with-color-images-2p41 (cit. a p. 68).

[38] DANB. «Rectified Linear Units (ReLU) in Deep Learning». In: (2024). URL: https://www.kaggle.com/code/dansbecker/rectified-linear-units-relu-in-deep-learning (cit. a p. 69).

[39] Bala Priya C. «Softmax Activation Function: Everything You Need to Know». In: (2023). URL: https://www.pinecone.io/learn/softmax-activation/ (cit. a p. 69).

Ringraziamenti