Università degli Studi di Salerno

Dipartimento di Informatica

Corso di Laurea Triennale in Informatica

Agricoltura di Precisione ed Intelligenza Artificiale per la prevenzione delle Malattie della Vite

Relatori
Prof. Andrea Francesco Abate
Dott.ssa Chiara Pero

Candidato
Daniele Gregori
Matricola: 0512106722

Anno Accademico 2021/2022

Abstract

Gli organismi vegetali (come le foglie della vite) possono essere colpiti da fitopatie, suddivise in: fisiopatie (malattie non infettive o non parassitarie), malattie infettive e malattie parassitarie. In questo studio ci occuperemo delle malattie infettive della vite, con un particolare focus sulle malattie fungine. Mantenere in salute la pianta della vite è di vitale importanza per garantire la qualità finale del raccolto; purtroppo, i metodi tradizionali per il rilevamento delle malattie sono dispendiosi, dato il personale specializzato richiesto ed il continuo monitoraggio, laborioso ed imperfetto considerando l'errore umano. L'obiettivo di questo lavoro è sviluppare un modello previsionale basato sul Deep Learning per prevedere l'insorgenza di fitopatologie nei vigneti, in modo tale da fornire ai viticoltori uno strumento efficace per agire in tempo e contrastare il danneggiamento della pianta. A tal proposito, è stato possibile sviluppare un nuovo metodo di apprendimento completamente automatico, utilizzando le reti neurali convoluzionali mediante una raccolta di set di dati di addestramento. In primo luogo, il lavoro si è incentrato sulla raccolta di immagini rappresentando cinque malattie della vite più comuni: Downy Mildew (Peronospora), Powdery Mildew (Oidio), Black Rot (Marciume nero), Black Measles (Mal dell'Esca) e Leaf Blight (Ruggine della foglia). Dato l'esiguo numero di fotogrammi disponibili, sono state applicate alcune delle più note tecniche di aumento dei dati (data augmentation), utilizzando altresì algoritmi di segmentazione. Sempre a causa del limitato numero di dati a disposizione, si è optato per l'utilizzo di modelli pre-allenati noti in letteratura, in particolare: MobileNetV2, VGG19, Resnet50 ed InceptionV3. I risultati della sperimentazione sono incoraggianti: MobileNetV2 ed InceptionV3 hanno rispettivamente ottenuto un'accuratezza pari al 91.76% e al 93.26%. Le prestazioni raggiunte indicano che l'apprendimento profondo, combinato con tecniche di visione artificiale, possono essere vantaggiosamente utilizzati in ambito vitivinicolo per la rilevazione automatica delle malattie delle foglie della vite.

Indice

Elenco delle figure

Elenco delle tabelle

Capitolo 1

Introduzione

L'intelligenza artificiale (Artificial Intelligence, AI) è una delle tecnologie più innovative della nostra epoca. Questa ha il potenziale di rivoluzionare il nostro modo di vivere e lavorare, e già è possibile notare l'impatto significativo che sta avendo su un ampio spettro di settori industriali. Dal campo sanitario a quello finanziario, dal settore delle auto autonome a quello agricolo, l'intelligenza artificiale sta aiutando le imprese a migliorare la loro efficienza, a prendere decisioni strategiche e sta creando nuove opportunità innovative.

Uno dei vantaggi più rilevanti dell'intelligenza artificiale è la sua abilità di analizzare grandi quantità di dati. Con l'ascesa del fenomeno "Big Data", le imprese collezionano più informazioni che mai. Nonostante ciò, processare ed interpretare questi dati può rivelarsi un'operazione ardua per noi esseri umani. Gli algoritmi di intelligenza artificiale possono velocemente vagliare grandi quantitativi di dati ed individuare pattern ed informazioni che per noi umani sarebbero difficili da scorgere. Questo permette alle aziende di prendere decisioni sempre più consapevoli.

Con la popolazione globale proiettata al raggiungimento di 9.1 miliardi entro il 2050 [1], il bisogno di pratiche agricole efficienti e sostenibili continuerà incessantemente a crescere, in concomitanza con il bisogno di andare ad abbattere l'impatto ambientale di queste ultime. Proprio grazie a questa sfida, in questo elaborato il mio obiettivo è quello di discutere della crescente importanza dell'intelligenza artificiale nella società odierna, ed il suo potenziale per plasmare il futuro dell'agricoltura.

1.1 Intelligenza Artificiale e Viticoltura: domini applicativi

Precision Agriculture, anche conosciuta come Agricoltura di Precisione, è un approccio che fa uso dell'intelligenza artificiale per ottimizzare i raccolti, ridurre lo spreco e di conseguenza aumentare i profitti dell'agricoltore. Questa metodologia implica l'utilizzo di un'ampia varietà di sensori e tecniche di analisi dei dati, per il monitoraggio della crescita delle colture e delle condizioni del suolo; ciò permette, al bisogno, di effettuare modifiche alla piantatura, alla

fertilizzazione e all'irrigazione. Il Deep Learning (Apprendimento Profondo) e la Computer Vision (Visione Artificiale) possono ricoprire un ruolo cruciale nel processo [2], consentendo ai coltivatori, grazie alla tecnologizzazione delle attività rurali, di analizzare grosse quantità di dati visuali in tempo reale sui loro dispositivi smart connessi ad internet e di ragionare su strategie vincenti.

Come detto precedentemente, una delle applicazioni chiave del Deep Learning e della Computer Vision nell'agricoltura smart è il monitoraggio delle colture. I metodi di supervisione tradizionali prevedono un'ispezione manuale di ogni campo, il che si rivela un'operazione dispendiosa e soggetta all'errore umano [3]. Con la Computer Vision, è possibile usare sensori, droni o altri veicoli aerei equipaggiati con fotocamere, per acquisire immagini in campi molto vasti, ed utilizzare algoritmi di Deep Learning per l'analisi dei dati raccolti. Una delle tecnologie fondamentali utilizzate nell'agricoltura di precisione è il Global Positioning System (GPS); questo strumento permette agli agricoltori di mappare il proprio campo agricolo e raccogliere dati riguardanti fattori come l'umidità del suolo e i livello dei nutrienti contenuti al suo interno.

Con l'avvento dell'uso dei sensori remoti, i coltivatori riescono a collezionare moltissimi dati concernenti aree geografiche molto vaste, ma come afferma Tantalaki [4]: "questi dati non sono omogenei e provengono da varie fonti con differenti modalità spaziali, temporali e spettrali. Le tecniche di Machine Learning sono comunemente riportate in letteratura come flessibili, capaci di processare un gran numero di input, e di gestire problemi non lineari. I dati disponibili giocano un ruolo fondamentale nello sviluppo di questi modelli. Più complicato è il problema da risolvere, più dati saranno richiesti."

L'agricoltura di precisione ha anche il potenziale di ridurre gli sprechi, infatti andando a concentrarsi su specifiche aree del campo da trattare, i contadini possono ridurre l'ammontare di fertilizzanti e pesticidi usati. Questo non solo permette di risparmiare denaro che potrà essere investito in altri ambiti, ma aiuta anche nella protezione dell'ambiente, diminuendo solo allo stretto necessario il numero di prodotti chimici utilizzati per i trattamenti.

Gli algoritmi utilizzati possono rilevare dei pattern all'interno delle immagini raccolte, che indicano lo stato di salute e lo stato di crescita della coltura analizzata, fornendo agli agricoltori preziose informazioni riguardo le aree di coltivazione che potrebbero necessitare di maggiori cure e attenzioni.

Il Deep Learning e la Computer Vision sono anche largamente utilizzati per efficientare le attività agricole. Un esempio è l'uso di robot che si avvalgono della Computer Vision e di veicoli autonomi per la piantatura e la raccolta delle colture. Questi sistemi sono in grado di navigare attraverso i campi agricoli e svolgere compiti con una precisione elevata, il che può ridurre considerevolmente l'ammontare di lavoro richiesto ed incrementare l'efficienza. Con la continua evoluzione di queste tecnologie, il loro ruolo si rivelerà fondamentale per assicurare una fornitura efficiente e sostenibile per la popolazione globale, la cui crescente richiesta di

beni di consumo diventa sempre più difficile da soddisfare.

Di seguito, andremo ad illustrare alcuni degli svariati domini applicativi del Deep Learning nel settore dell'agricoltura, e spiegheremo come queste metodologie supportano l'agricoltura di precisione.

1.1.1 Qualità dei semi

La preoccupazione principale dei produttori è focalizzata sul garantire ai coltivatori e giardinieri semi di alta qualità, attraverso lo studio di vari aspetti come: colore, forma e struttura. Questa accurata ricerca porta benefici al rendimento delle coltivazioni, ove usare semi di qualità elevata risulta vitale.

Come fa notare Gulzar [5] nella sua ricerca: "tradizionalmente questa categorizzazione è svolta da specialisti che ispezionano visivamente ogni campione di seme, il che si rivela essere un'operazione alla lunga tediosa e che richiede tempo." Con l'aiuto delle tecniche di visione artificiale è possibile automatizzare questo processo, permettendo uno smistamento più semplice e veloce, oltre che ad una classificazione accurata che sia conforme agli standard internazionali di qualità [6]. Durante lo studio si prendono in considerazione svariati fattori, come: test di germinazione, test di purezza, integrità genetica e anche delle ispezioni fisiche riguardanti grandezza, forma e colore, per assicurarsi che questi fattori siano conformi agli standard industriali.

Ad oggi le tecniche di Deep Learning si rivelano fondamentali per lo studio dei semi; in particolare, le reti neurali convoluzionali si sono dimostrate efficienti per l'identificazione e la classificazione delle varietà di questi ultimi, con un'accuratezza addirittura del 99.9% [6].

1.1.2 Stato salutare della pianta

Come primo dominio applicativo andremo a commentare l'argomento principe dell'elaborato: l'analisi dello stato salutare della pianta. L'analisi dello stato salutare della pianta è un aspetto cruciale dell'agricoltura e dell'orticultura moderna; questo processo implica lo studio e la valutazione di varie caratteristiche della pianta, come il colore delle foglie, gli schemi di crescita e sintomi di eventuali malattie (Figura 1.1) o parassiti, per determinare la sua salute complessiva ed il suo benessere.

Esistono diverse tecniche che possono essere applicate per questa tipologia di analisi. La più comune è l'ispezione visiva, che prevede la disamina fisica della pianta e la ricerca di eventuali segni di sofferenza o malattia; questo può includere fattori come foglie appassite, decolorazione o schemi di crescita anormali.

Un'altra tecnica spesso utilizzata è il test diagnostico, dove vengono raccolti dei campioni di tessuto vegetale o di terreno, che vengono successivamente analizzati in laboratorio per l'identificazione di eventuali agenti patogeni o altre problematiche che potrebbero impattare

negativamente sul benessere della pianta.

Uno dei benefici chiave di questo approfondimento, è che consente agli agricoltori e agli orticoltori di rilevare precocemente l'insorgenza di eventuali malattie, consentendo un trattamento repentino dell'infezione prima che questa si aggravi, rendendo la pianta, o tutto il campo attorno ad essa, irrecuperabili. Questa modalità di agire può ridurre il costo complessivo della produzione agricola ed incrementare il rendimento del raccolto finale.

Oltre ad individuare ed affrontare vari problemi, l'analisi dello stato salutare della pianta può aiutare sotto il punto di vista della riduzione degli sprechi e con la diminuzione dell'impatto ambientale creato dagli agenti chimici utilizzati dal coltivatore. Per esempio, analizzando il livello di nutrienti nel terreno e gli schemi di crescita delle colture, gli agricoltori possono determinare la quantità e la tipologia di fertilizzanti o pesticidi più efficaci da utilizzare per aiutare il loro raccolto a crescere prospero.

Come asserisce Dhanya [3]: "l'analisi delle immagini può individuare efficacemente cambiamenti nella struttura della foglia. Gli approcci del Deep Learning attraverso la visione artificiale sono soluzioni valide per far fronte all'individuazione tempestiva di malattie ed evitare il consulto di esperti."

1.1.3 Analisi del suolo

L'analisi del suolo è il processo utile alla determinazione delle proprietà chimiche e fisiche del suolo, tra cui lo studio della temperatura, umidità e componenti del suolo. Attraverso l'ottenimento di queste informazioni, il coltivatore può capire l'andamento della produzione e grazie alla stime accurate sviluppare strategie per l'uso adeguato del campo utili alla sua gestione ottimale [7].

Queste pratiche di gestione del campo agricolo aiutano ad identificare carenze di nutrienti, che possono essere corrette con la fertilizzazione mirata; inoltre, attraverso la comprensione della struttura del suolo, gli agricoltori possono scegliere le tipologie di colture che si rivelano ideali per il tipo di suolo con cui si sta lavorando. Importante è anche la prevenzione da frane ed inondazioni per ridurre l'impatto sull'ambiente, grazie allo studio del ciclo idrologico mediante l'utilizzo di satelliti ed aerei appositi [8].

Con queste prassi vengono realizzate dettagliate mappe idrogeologiche per studi più approfonditi, fruttuosi per contrastare il cambiamento climatico. Le tecniche di Machine Learning forniscono dei metodi affidabili e low-cost per lo studio e la conseguente tenuta delle condizioni del suolo, sospendendo l'uso delle metodologie convenzionali rivelatesi impegnative.

1.1.4 Pianificazione e gestione sostenibile dell'irrigazione

La gestione dell'irrigazione è la pratica di controllare la quantità, la frequenza e le tempistiche di acqua richiesta dalle colture per crescere sane. Questa procedura si rivela essenziale in luoghi che soffrono di scarse risorse d'acqua e/o di fenomeni meteorologici imprevedibili a causa dei cambiamenti climatici che stiamo riscontrando al giorno d'oggi; inoltre, fornire solo l'acqua necessaria alle coltivazioni, non solo previene lo spreco di acqua preziosa, ma riduce anche i danni subiti da situazioni di irrigazione eccessiva.

In aggiunta, il rischio di malattie e parassiti diminuisce drasticamente grazie alla creazione di un ambiente avverso alla diffusione di agenti patogeni, ciò si ottiene limitando il numero di pesticidi usati, dimostratisi deleteri per la salute umana.

Nel suo lavoro di ricerca, Alibabaei [9] afferma che "i modelli per la valutazione delle colture aiutano a ridurre il consumo d'energia, incrementano la produttività e aiutano a stimare i requisiti per la raccolta e lo stoccaggio". Dai risultati del suo studio si evince che l'architettura di Deep Learning che ha ottenuto i risultati migliori sono le reti neurali ricorrenti (RNN), le quali, afferma: "offrono numerosi vantaggi rispetto ad altri modelli di Deep Learning e agli approcci di Machine Learning tradizionali".

1.1.5 Cambiamenti climatici

Gli agenti atmosferici ed il cambiamento climatico hanno un impatto notevole sull'agricoltura, che interessa la resa delle colture, la disponibilità di acqua e la diffusione di parassiti e malattie. Gli effetti di questi continui cambiamenti sono soliti variare in base alla regione e alla tipologia di coltivazione, però non bisogna sottovalutarli, poiché hanno il potenziale di danneggiare la produzione di cibo e attentare alla sua sicurezza.

I cambiamenti di temperatura, frequenza delle piogge ed eventi meteorologici estremi, possono minacciare lo sviluppo delle colture, portando nei casi peggiori alla carestia. Ad esempio, la siccità causata dall'innalzamento delle temperature, porta ad una riduzione dell'umidità del terreno, rendendo ardua la crescita delle piantagioni; allo stesso modo, le inondazioni tendono a danneggiare il seminato e conseguentemente l'accesso ai campi diventa difficoltoso per i coltivatori.

L'aumento delle temperature inoltre non permette a svariati tipi di piante di crescere in alcune regioni, e per di più agenti patogeni e parassiti si abituano a queste nuove condizioni di calore, riuscendo così a sopravvivere e a proliferare sempre di più. Anche le stagioni ormai non sono più puntuali come dovrebbero, con inverni caldi, ed estati torride, la cui conseguenza è la necessità di piantare in periodi diversi dal solito, cosicché si riesca ad ottimizzare la percentuale di crescita delle colture.

In questo ambito, le tecniche di Machine Learning possono aiutare a contrastare i cambiamenti climatici [10], automatizzando diversi aspetti con l'aiuto di sensori che catturano informazioni; ad esempio, posizionando questi ultimi su fiumi o edifici, si riesce a tenere sotto controllo ogni aspetto climatico o variazioni improvvise che potrebbero causare danni al raccolto.

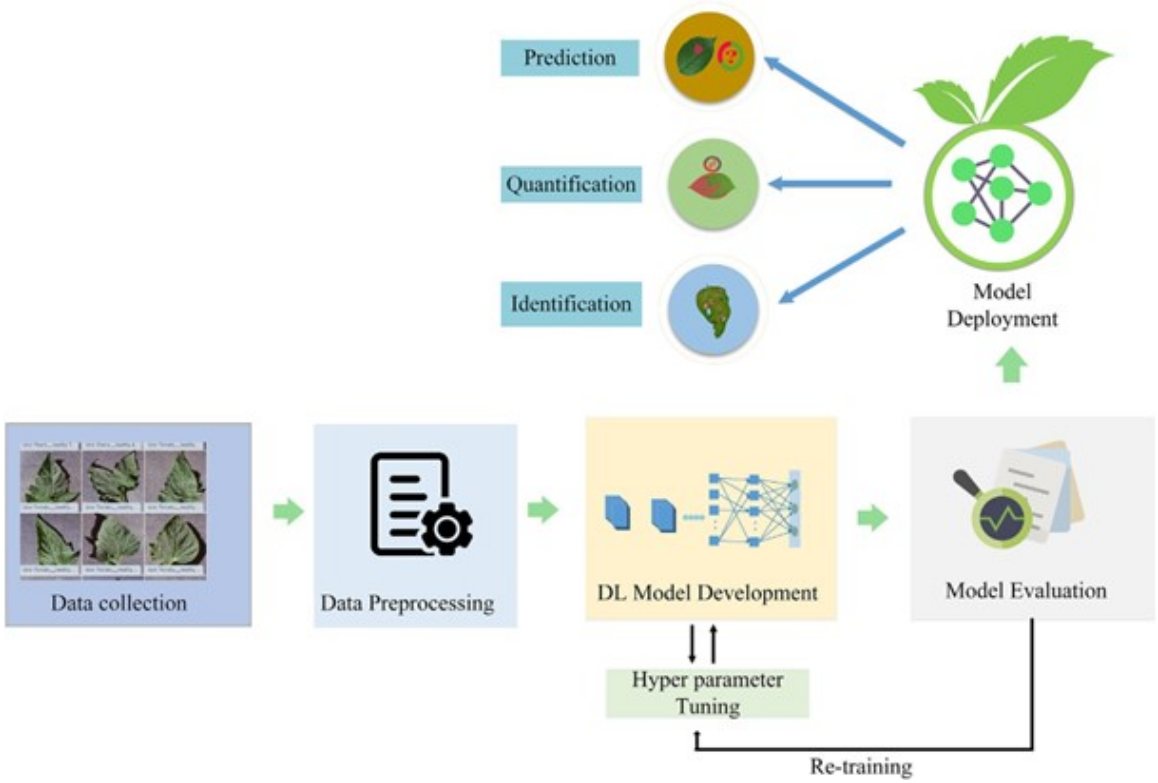

Figura 1.1: Tecniche di visione artificiale basate sull'apprendimento profondo per l'analisi dello stato salutare della pianta [3].

1.2 Apprendimento Profondo e Visione Artificiale

Il Deep Learning è una branca del Machine Learning che prevede l'allenamento di reti neurali artificiali per svolgere compiti come: il riconoscimento d'immagini, la traduzione linguistica e svariati processi decisionali. Queste reti neurali sono modellate per riprodurre il comportamento e la struttura del cervello umano, e sono capaci di imparare e prendere decisioni basandosi su una grande mole di dati [11].

Una delle caratteristiche fondamentali del Deep Learning è l'uso di molteplici layer di neuroni artificiali, o anche chiamati nodi, nella rete. Questi permettono alla rete di imparare rappresentazioni dei dati in input volta per volta sempre più complesse, cosicché si possano ottenere risultati più accurati ed affidabili. Nel Machine Learning tradizionale, il numero di layer è tipicamente limitato ad uno o due, mentre le reti di Deep Learning possono essere

costituite da dozzine o addirittura centinaia di layer, da qui il nome "Deep" Learning.

Il vantaggio principale del Deep Learning è che riesce ad imparare le caratteristiche dei dati in input automaticamente, senza che queste vengano definite manualmente dall'esperto, come si può vedere dalla Figura 1.2. Tuttavia, i modelli di Deep Learning richiedono una grande quantità di dati e un grande potere computazionale per riuscire a svolgere un allenamento adeguato. Questi particolari requisiti fanno insorgere diverse problematiche, riguardanti la ricerca di un numero sufficiente d'immagini in letteratura o l'acquisizione d'immagini ex novo che siano confacenti al lavoro che si sta svolgendo che siano state scattate con un'attrezzatura all'altezza; oltre alla spinosa questione di possedere un hardware specializzato, come delle GPU, che sia consono al lavoro che si sta conducendo, per non incappare in interminabili sessioni di allenamento della rete neurale.

Allenare una rete di Deep Learning è gravoso sia dal punto di vista dello spazio che da quello computazionale, a causa dei milioni di parametri che necessitano di essere iterativamente affinati. L'impatto di queste problematiche è commisurato al numero di parametri di cui la rete da noi creata, o preesistente, è composta, dalla sua profondità e dalla tipologia di dati che le diamo in ingresso. Anche il processo d'inferenza è computazionalmente oneroso, dipendentemente dall'alta dimensionalità dei dati in input, ad esempio immagini ad alta risoluzione.

Per soddisfare i requisiti computazionali richiesti dal Deep Learning, un approccio ricorrente è di sfruttare il cloud computing [12]. Per usare le risorse nel cloud, i dati devono essere mossi dalla posizione in cui si trovano i dati, come smartphone o sensori dell'Internet-of-Things, ad una locazione centralizzata nel cloud. Questa possibile soluzione di spostare i dati dalla fonte al cloud fa sorgere diverse sfide da affrontare:

- **Latenza**
  La latenza consiste nel tempo che ci mette una richiesta per essere processata e per una risposta ad essere ricevuta. Nel Deep Learning, la latenza che si manifesta con l'utilizzo del cloud computing può essere un problema, a causa della grande quantità di dati che necessitano di essere processati e della complessità dei modelli utilizzati. Questa problematica può portare a dei ritardi di propagazione durante l'allenamento della rete e durante il processo d'inferenza, rendendo complicata l'elaborazione di tanti dati in modo veloce ed efficiente e non soddisfacendo i rigorosi requisiti di bassa latenza richiesti per le applicazioni interattive. Per minimizzare la latenza è fondamentale usare dei framework ottimizzati per il Deep Learning, come TensorFlow e PyTorch, e predisporre abbastanza larghezza di banda e connessioni a bassa latenza tra l'origine dei dati e le risorse cloud.

- **Scalabilità**
  La scalabilità è l'abilità di un sistema di gestire un crescente carico di lavoro o numero di utenti. Gli algoritmi di Deep Learning possono richiedere un alto numero di risorse e le aziende potrebbero aver bisogno di ampliare le loro risorse di cloud computing proporzionalmente alla crescita dei loro progetti; inoltre, col crescere del numero di

dispositivi che si connettono e la quantità di memoria e stoccaggio richiesti per gestire il carico di lavoro, l'infrastruttura cloud può trasformarsi in un collo di bottiglia andando ad introdurre problemi di scalabilità. Anche caricare tutti i dati sul cloud si rivela una pratica inefficiente dal punto di vista dell'utilizzo delle risorse, in particolare se non tutti i dati sul cloud sono poi effettivamente utilizzati. Per affrontare questo fenomeno, l'utilizzo di tecniche di Deep Learning distribuite, come il parallelismo dei dati e dei modelli, può aiutare a rendere scalabile il processo di allenamento e d'inferenza per gestire al meglio i vari carichi di lavoro.

- **Privacy**
  Gli algoritmi di Deep Learning spesso richiedono dati sensibili, come informazioni personali o dati finanziari, da immagazzinare e processare nel cloud. Le aziende devono assicurare che questi dati siano adeguatamente protetti, per tutelarne i proprietari e le persone di cui si fa riferimento in questi, facendo si che solo coloro che sono autorizzati possano accedervi. Tutto ciò può essere difficile da conseguire, specialmente quando si ha a che fare con una vasta quantità di dati e molteplici fornitori di servizi cloud. Per attenersi alle regole per la privacy è importante attenersi alle regolazioni per la protezione dei dati, come il GDPR. Inoltre, l'utilizzo della crittografia, di protocolli sicuri per la comunicazione e strategie per il controllo degli accessi, possono aiutare a proteggere i dati e i modelli da accessi illeciti.

- **Costi**
  I costi possono essere una preoccupazione significativa quando si fa uso del cloud computing per il Deep Learning, dal momento che le risorse computazionali e la memorizzazione possono essere costose. I costi possono velocemente accumularsi, specialmente quando si allenano e implementano grandi modelli, o quando si eseguono molteplici esperimenti o iterazioni. Oltre a ciò, vista la grande quantità di dati che richiede un modello di Deep Learning, questo può risultare in costi elevati per il trasferimento di dati e per la memorizzazione di questi. Anche il costo del noleggio di GPU da remoto, come attraverso l'uso di Google Colab, può gravare sulle tasche delle aziende nel caso non fossero provviste di hardware adatto. Per far fronte al problema, si potrebbero usare modelli pre-allenati, ridurre il numero di parametri o usare tecniche di Deep Learning distribuite per ridurre le risorse computazionali richieste, portando ad un abbassamento dei costi. Un altro aspetto da considerare è l'uso di servizi cloud che permettono di pagare solo per le risorse effettivamente utilizzate, come Amazon EC2. In aggiunta, è importante tenere sotto controllo l'utilizzo delle risorse, per accertarsi che queste vengano utilizzate efficientemente e di interrompere l'utilizzo delle risorse quando non utilizzate, per evitare costi superflui.

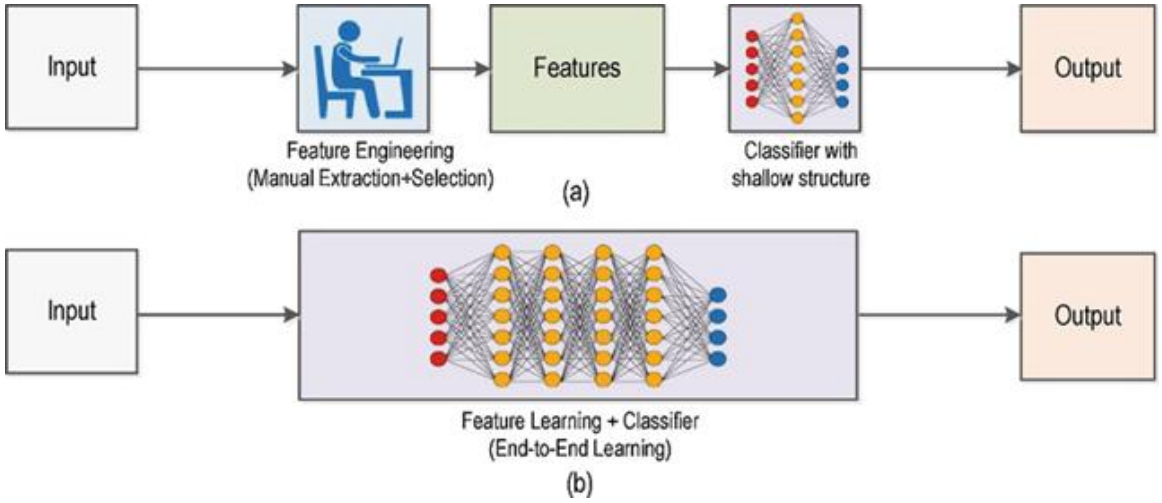

Figura 1.2: (a) Estrazione manuale delle feature vs. (b) estrazione automatica delle feature [13].

La Computer Vision, d'altra parte, è un campo dell'intelligenza artificiale che si focalizza sul come i computer riescono a capire ed interpretare informazioni visive. Questo include dei task come il riconoscimento in immagini e video, individuazione di oggetti, segmentazione d'immagini e generazione d'immagini.

Una delle principali applicazioni della Computer Vision la troviamo nell'ambito del riconoscimento in immagini e video; questo task implica l'allenamento di modelli per l'identificazione di oggetti, persone e altre caratteristiche, che possiamo estrapolare da un'immagine o da un video. Ad esempio, un modello può essere allenato per il riconoscimento del viso di una specifica persona, o per identificare un'auto all'interno di un video. Questa tecnologia ha una vasta gamma di applicazioni, dai sistemi di sicurezza per rilevare degli intrusi, alle auto con guida autonoma.

La segmentazione d'immagini è un altro utilizzo della Computer Vision. In questo specifico task si divide un'immagine in vari segmenti, ognuno dei quali corrisponde ad un oggetto o ad una regione diversa. É possibile sfruttare questa tecnica nel ramo delle immagini mediche, per individuare precise strutture in un'immagine o anche per rimuovere lo sfondo da un'immagine per estrapolare solo il soggetto in primo piano.

Una delle aree in cui il Deep Learning e la Computer Vision s'intersecano è lo sviluppo delle reti neurali convoluzionali (CNN), cioè un algoritmo di Deep Learning molto utilizzato in task di Computer Vision.

Il Deep Learning è stato applicato con successo in diversi ambiti; ad esempio le reti neurali convoluzionali, che abbiamo precedentemente citato, sono considerate come la tecnologia d'avanguardia attualmente nell'ambito del riconoscimento di pattern, come afferma Liu B [14]: "CNN è ancora considerato come uno dei migliori algoritmi per l'attività di riconoscimento di

pattern". Questa tipologia di rete può essere allenata nell'identificazione di specifici oggetti o su caratteristiche peculiari degli oggetti rappresentati all'interno di un'immagine con un tasso d'errore molto basso. Assieme, queste tecnologie hanno il potenziale di rivoluzionare il modo in cui ci approcciamo all'agricoltura smart.

1.3 Vitis vinifera

Le piante d'uva sono una coltura essenziale per molti coltivatori in giro per il mondo. Non solo producono uva deliziosa da consumare a tavola, ma ha anche svariati utilizzi nella produzione di vino, succo, frutta secca (uva passa), champagne e liquori distillati (brandy, cognac).

La vite è la pianta da frutto maggiormente coltivata nel mondo. L'Europa è il fulcro dell'industria per la produzione di uva e vino; menzione d'onore per L'Italia, che ricopre un quinto della produzione di vino nel mondo, superando anche Francia e Spagna per quantità di produzione e rilievo internazionale [15].

La vite, anche conosciuta come Vitis vinifera, è una specie di pianta che cresce in luoghi con una clima temperato, non sopporta inverni freddi e ha bisogno di estati calde per far si che i suoi frutti maturino. La pianta della vite offre i migliori risultati nelle aree che si trovano tra il 30° e i 50° di latitudine, in queste zone le piantagioni sono affette da diversi fattori ambientali come la presenza di montagne, correnti marine o particolari caratteristiche del suolo. Ad esempio, negli scavi archeologici di Pompei in Campania, ci sono dei vigneti la cui peculiare caratteristica è che sono coltivati su di un terreno vulcanico ricco di acido fosforico, che favorisce la qualità e l'abbondanza dei raccolti; inoltre, sono applicate le tecniche di più di duemila anni fa che rendevano il vino Pomepiano uno dei più pregiati e redditizi (Figura 1.3), come raccontato anche da Plinio il Vecchio nei suoi scritti [16].

Figura 1.3: Dipinto murale Villa Romana raffigurante uva [17].

1.3.1 Principali malattie del vigneto

Purtroppo, come tutte le piante, anche quelle d'uva sono soggette ad una vasta varietà di malattie che impattano negativamente sul loro benessere e sulla loro produttività. La malattia è molto deleteria per l'uva, perché questa si diffonde velocemente e i trattamenti sono generalmente inefficaci dopo che la malattia ha raggiunto un certo stadio. In questa sezione andremo ad illustrare le malattie fungine che maggiormente interessano i vigneti.

- **Peronospora (Downy Mildew)**
  La peronospora è causata da un fungo patogeno chiamato Plasmopara viticola, questo prospera in condizioni di elevata umidità. Probabilmente introdotta in Europa dall'America mediante il trasporto di vite Americana che, a quel tempo, era importata per effettuare le prime prove di resistenza alla Fillossera. Come illustra il sito Fitogest [18]: "è in grado di attaccare tutti gli organi verdi della pianta, principalmente le foglie, i germogli e i grappoli, causando ingenti danni se non gestita correttamente." Tuttavia, i danni provocati da queste infezioni non sono solo legate all'anno della loro comparsa, bensì possono anche compromettere gli anni successivi di raccolto; questo perché il fungo riduce le riserve nutritive della vite, portando ad una perdita di vigore che viene recuperata solo nel tempo, influenzando la produttività negli anni a venire [19].

  Sulle foglie la sintomatologia può essere duplice:

  - **Macchia d'olio**
    aspetto delle prime infezioni con umidità elevata. Nella pagina superiore delle foglie appaiono delle macchie piccole e gialle da cui deriva il nome "macchia d'olio", con conseguente avvizzimento e caduta di quelle maggiormente colpite; mentre nella pagina inferiore, il fungo produce della muffa bianca in corrispondenza delle macchie della pagina superiore. Nei casi peggiori, il fungo può infettare anche i frutti della pianta causandone la marcescenza (Figura 1.4a);
  - **Mosaico**
    sintomatologia tipica delle infezioni tardive. Si manifesta con delle macchie clorotiche, anche in questa casistica nella pagina inferiore compaiono delle macchie di muffa bianca in corrispondenza della mosaicatura (Figura 1.4b).

  Il fungo sverna sotto forma di oospore, che rimangono intrappolate sotto le foglie cadute a terra rimaste sotto le viti. Le oospore generano numerose zoospore, le quali propagano la malattia in tutto il vigneto attraverso il vento, insetti e schizzi d'acqua durante le piogge. Queste spore riescono ad infettare la pianta attraverso gli stomi, piccole aperture sulle foglie che permettono lo scambio di gas.

  Uno dei passaggi fondamentali per la prevenzione della peronospora è quello di rimuovere i residui di foglie infette alla fine della stagione, cosicché il numero di spore che potrebbero infettare la piantagione l'anno successivo diminuiscano.

In ultimo luogo, possono essere utilizzati anticrittogamici, ma solo come ultima risorsa. Un monitoraggio costante è essenziale, poiché se si esagerasse con le quantità di fungicida il fungo potrebbe sviluppare col tempo una resistenza a quest'ultimo.

(a) Parte inferiore foglia macchia d'olio

(b) Mosaico

Figura 1.4: Peronospora [20].

- **Oidio o Mal bianco (Powdery Mildew)**
  L'oidio è causato da un fungo denominato "Uncinula necator", che prospera in ambienti molto umidi, abbastanza caldi e in assenza di irrigazioni. La malattia è caratterizzata dalla presenza di un pulviscolo biancastro (Figura 1.5), simile come aspetto e consistenza alla polvere, sulle foglie, steli e frutto della vite. Col progredire della malattia, le macchie diventano più grandi e numerose, finendo col ricoprire l'interezza della foglia. Il fungo colpisce anche i frutti, facendoli diventare deformi ed invendibili.

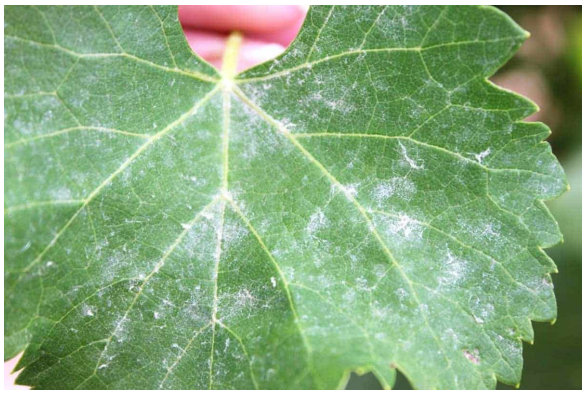

Figura 1.5: Oidio [21].

Come prevenzione per il Mal bianco [22], bisogna:

- monitorare l'umidità del luogo in cui cresce la vite;

  - scuotere le piante per rimuovere la brina notturna ed aiutare le piante ad asciugarsi velocemente;
  - rimuovere sempre l'eccesso di fogliame sia dalla pianta che dal suolo una volta cadute, perché proprio sotto le foglie cadute prospera il fungo.

- **Mal dell'esca (Black Measles)**
  L'Esca è causata da un gruppo di patogeni fungini:

  - Phaeomoniella chlamydospora;
  - Phaeoacremonium aleophilum;
  - Fomitiporia mediterranea.

  Come spiega il sito Istituto Agrario Sartor: "sulle foglie delle piante colpite dall'esca compaiono macchie leggermente decolorate o clorotiche rotondeggianti o irregolarmente circolari, localizzate fra le nervature (Figura 1.6) o lungo i margini della foglia. Gradualmente le macchie si espandono e confluiscono fra di loro per poi necrotizzare completamente o in parte" [23].

  La malattia può far formare delle fratture all'interno degli acini dell'uva, permettendo ad altri agenti patogeni dannosi di entrare all'interno del frutto e danneggiare il raccolto.

  Come tutti gli agenti fungini, questi prosperano in località umide ed abbastanza calde. Nel caso dell'Esca l'attacco di questa tipologia di funghi avviene a causa dei:

  - tagli di potatura;
  - ferite o altre lesioni provocate da agenti atmosferici.

  Purtroppo non esistono prodotti per la cura di questa malattia, ma è possibile adottare comportamenti mirati alla prevenzione. Un'accortezza è quella di cercare di non creare lesioni alle piante trattate e disinfettare sempre gli utensili utilizzati per la potatura delle piante malate, i quali fanno da tramite per la diffusione del fungo.

Figura 1.6: Esca [23].

- **Marciume nero (Black Rot)**
  Dal sito Aeb [24] possiamo comprendere come si manifesta la malattia: "inizialmente si manifesta con lesioni circolari di colore giallastro sulle foglie giovani. Queste, diffondendosi, diventano marroni (Figura 1.7) e originano corpi fungini simili a piccolissimi grani di pepe. Col progredire dell'infezione, le lesioni possono avvolgere il picciolo delle singole foglie, soffocandole. Da ultimo, il fungo colpisce i germogli causando grandi lesioni ellittiche di colore scuro."

  I sintomi del marciume si possono notare perlopiù sulle foglie della pianta, però i veri danni si trovano sui frutti; questi, ad un certo punto della crescita, marciscono ed iniziano ad avvizzirsi, diventando sempre più piccoli con delle pustole nere. Gli acini avvizziti fungono da trasportatori dell'infezione, grazie all'aiuto di umidità, pioggia ed innaffiatura.

Figura 1.7: Infezione da marciume nero [25].

Per combattere il marciume nero, come sempre è fondamentale la prevenzione, però in casi più avanzati di malattia bisogna:

- asportare i frutti marci;
- potare le piante e bruciare i resti malati;
- utilizzo di fungicidi appositi per combattere i segni di malattia rimanenti.

- **Ruggine della foglia (Leaf Blight)**
  La vite Europea, come si legge dall'articolo di Liang [26]: "molto vulnerabile a questa malattia e produce sintomi caratterizzati da macchie grandi ed irregolari, le quali sono di un colore che va dal rosso opaco al marrone, ma che diventano nere e fragili con il passare del tempo."

Figura 1.8: Ruggine della foglia [26].

Capitolo 2

Stato dell'arte

2.1 Evoluzione dei sistemi per l'identificazione delle malattie delle colture

Prima della tecnologizzazione delle pratiche agricole, l'identificazione delle malattie delle piante era condotta mediante l'aiuto di specialisti del settore; questi, monitoravano e studiavano lo stato salutare della pianta ad occhio nudo, basandosi unicamente sulle loro intuizioni ed le loro conoscenze. Questo metodo risultava laborioso e prendeva molto tempo, sopratutto in contesti di campi agricoli molto vasti. Inoltre è sempre possibile che l'esperto possa sbagliare la sua rilevazione, inducendo di conseguenza i coltivatori ad utilizzare trattamenti sbagliati sulle proprie piante, creando ulteriori danni.

Per superare le limitazioni delle pratiche tradizionali, con l'evoluzione della tecnologia sono state sviluppate tecniche per l'automazione delle operazioni di monitoraggio e previsione delle piantagioni. Con queste tecniche, si riesce a ridurre l'impatto ambientale usando la quantità adatta di pesticidi ed agenti chimici e al contempo ad abbattere i costi operativi e i tempi richiesti dai coltivatori per il monitoraggio del campo [27].

2.1.1 Modalità di acquisizione dati

Il primo passo è sicuramente il processo di acquisizione delle immagini di studio. Anche in questo campo la tecnologia ha fatto passi da gigante, sopratutto con l'avvento dell'Internet of Things (IoT), sensori remoti [28] e del cloud computing.

Inizialmente, le immagini era catturate con le fotocamere dei telefoni cellulari, o con l'aiuto di fotocamere specifiche di gran lunga più performanti. Come si può immaginare, in contesti con vasti campi agricoli da studiare, la cattura manuale delle immagini risulta quasi impossibile col solo lavoro umano. Oggi vengono utilizzati satelliti per scattare immagini dei campi, o anche con l'aiuto dei cosiddetti aeromobili a pilotaggio remoto, anche chiamati droni. I droni sono degli apparecchi caratterizzati dall'assenza di un pilota umano; questi possono essere pilotati da remoto da un operatore a terra, grazie ad uno schermo che mostra

il punto di vista del drone. Ciò permette ai piloti di attivare i vari strumenti equipaggiati dal veicolo, come fotocamere ad alta risoluzione o spray per la distribuzione di pesticidi (Figura 2.1); altrimenti, possono agire autonomamente se adeguatamente programmati e muoversi con l'aiuto dei sistemi di navigazione satellitare [29]. Come si può vedere nello studio di Albattah [30], gli autori utilizzano un quadricottero per la raccolta di immagini in tempo reale in condizioni di luce naturale, adoperando inoltre i fotogrammi contenuti nel dataset PlantVillage.

Figura 2.1: Droni utilizzati per la cura delle coltivazioni [31].

Se non si ha la possibilità di procurarsi immagini proprie, si possono sempre utilizzare dei motori di ricerca per trovare immagini su Internet, come nello studio condotto da Sladojevic [32]; però, come fa notare l'autore, sorge la necessità di applicare un processo di scrematura delle immagini e di far verificare individualmente ognuna di queste da un esperto del settore, per assicurarsi che la categoria affibbiata alla foto sia corretta.

Come spiega Domingues [33]: "le prestazioni di un modello di Machine Learning sono influenzate dalla qualità e dal tipo di immagini date in input al modello. Le immagini acquisite in un ambiente di laboratorio controllato e le immagini acquisite sul campo possono dare risultati completamente diversi tra loro."

Senza dubbio le immagini catturate in laboratorio avranno una qualità più alta rispetto a quelle scattate sul campo, le quali possono avere parti di altri piante, più foglie nella stessa immagine, diverse ombreggiature, diversi sfondi e differenti condizioni di luce. Ad esempio, i dataset pubblici più celebri come PlantVillage sono composti da foto scattate in un ambiente controllato e che quindi risultano di una qualità superiore [34]. Anche nello studio curato da Hernandez, possiamo notare un esempio di immagini acquisite in laboratorio, più precisamente immagini RGB di dischi di foglie inoculati con la malattia della Peronospora, successivamente inseriti all'interno di piastre di Petri per l'analisi [35].

Per ricevere e raccogliere i dati trasmessi dalle moderne apparecchiature utilizzate per il monitoraggio dei raccolti, l'Internet of Things si rivela una strategia rivoluzionaria; questa permette lo scambio di dati in modalità wireless tra i dispositivi di controllo degli agricoltori ed i sensori impiantati direttamente sui campi, o con i droni comandati da remoto che trasmettono in tempo reale foto e dati [28].

2.2 Machine Learning: algoritmi

Nelle seguenti sotto-sezioni saranno illustrati i vari algoritmi di Machine Learning utilizzati nell'ambito dell'Agricoltura di Precisione.

2.2.1 Macchina a Vettori di Supporto

Le SVM, cioè le Macchine a Vettori di Supporto, sono "modelli di classificazione il cui obiettivo è quello di trovare la retta di separazione delle classi che massimizza il margine tra le classi stesse, dove con margine si intende la distanza minima dalla retta ai punti delle due classi [36]."

Un vantaggio delle SVM è la loro abilità di gestire dati ad alta dimensionalità, dato che selezionano solo un sottoinsieme dei dati d'allenamento come vettori di supporto, il che riduce la complessità del modello e lo rende meno soggetto al fenomeno dell'overfitting (sovradattamento dei dati) rispetto ad altri modelli.

Nonostante le SVM siano utilizzate principalmente per studi di classificazione binaria, è possibile utilizzarle anche per classificazioni multi-classe.

Nell'ambito delle malattie della vite, Padol ha ottenuto un'accuratezza media del 88.89% nell'identificazione della Peronospora e dell'Oidio [37]. Mentre nell'individuazione di malattie della vite come: Marciume Nero, Mal dell'Esca e Ruggine della foglia, è stata raggiunta una precisione media del 90% nel lavoro di Agrawal [38].

Nello studio sulle malattie delle foglie di tè di Das [39], le SVM hanno registrato performance migliori rispetto ad altri modelli presi in considerazione, come la Regressione Logistica e la Foresta Casuale, ottenendo un'accuratezza del 87.6% per le SVM, del 67.3% per la Regressione Logistica e del 70.05% per la Foresta Casuale. Senza dubbio questo algoritmo presenta delle debolezze; ad esempio è molto sensibile rispetto alla qualità dei dati in input e ai parametri assegnati, che possono significativamente influenzare i risultati ottenuti.

Le SVM sono molto utilizzate anche grazie alla loro abilità di "generalizzare bene anche con campioni d'allenamento limitati" come asserisce Mountrakis [40] nel suo articolo.

2.2.2 Foresta Casuale

La Foresta Casuale è un algoritmo di Machine Learning che può essere utilizzato sia per problemi di classificazione che di regressione. Questa tecnica è basata sul concetto di "Apprendimento d'insieme", che consiste nel processo di "combinare molteplici classificatori per risolvere un problema complesso e migliorare le performance del modello [41]."

L'algoritmo dà in input ai vari alberi decisionali diversi sottoinsiemi del dataset, calcolando la media dei risultati intermedi appartenenti agli alberi per migliorare l'accuratezza della previsione. Naturalmente, usare un gran numero di alberi decisionali porterà a livelli di precisione maggiori e previene il problema dell'overfitting. Il modello si basa sul concetto di "saggezza delle folle", cioè un gran numero di modelli tra loro non correlati che operano come un insieme, surclasserà uno qualsiasi dei modelli presi individualmente. In un certo senso gli alberi si proteggono a vicenda, mentre alcuni sbaglieranno, molti altri daranno la risposta giusta (Figura 2.2).

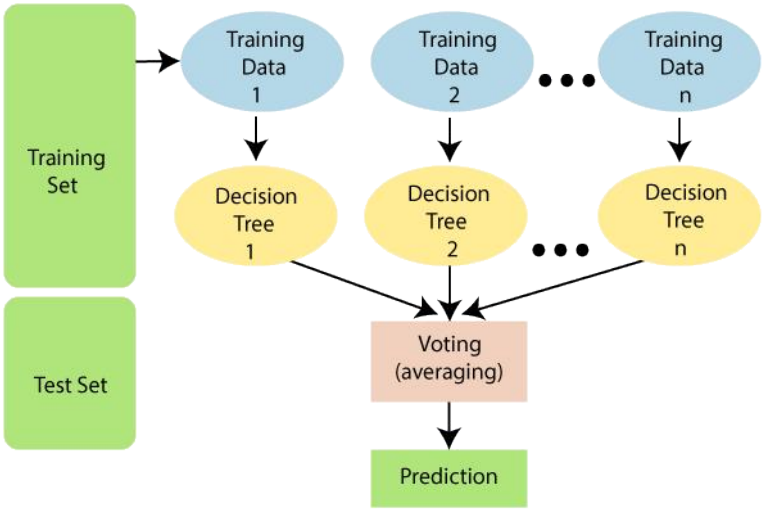

Figura 2.2: Funzionamento Foresta Casuale [41].

Come si può leggere nello studio di Ramesh sulle malattie delle foglie di papaya [42], la Foresta Casuale ha ottenuto il risultato migliore, se comparato ad altre tecniche di Machine Learning, con un risultato di accuratezza del 70%. Nonostante l'esiguo numero di foto utilizzate, la Foresta Casuale ha ottenuto buoni risultati, migliorabili andando ad utilizzare un più vasto numero di esemplari nel dataset.

2.2.3 K-nearest neighbors

L'algoritmo k-nearest neighbors (KNN) è un algoritmo di Machine Learning supervisionato utilizzato sia per operazioni di classificazione che di regressione, ma è utilizzato principalmente nel campo della classificazione. L'idea di base dietro il KNN è di stimare a che gruppo un insieme di dati appartiene, basandosi sui dati che li circondano, cioè i vicini più prossimi (Figura 2.3). Essenzialmente, il KNN esegue un meccanismo di votazione per determinare la classe dell'insieme di dati da classificare, la classe che otterrà la maggioranza dei voti diventerà la classe dell'insieme di dati considerato.

"É chiamato **algoritmo ad apprendimento pigro** o **apprendista svogliato**" perché non effettua nessun allenamento sui dati in ingresso. Invece, memorizza solo le informazioni durante il tempo di allenamento e non esegue alcun calcolo [43]."

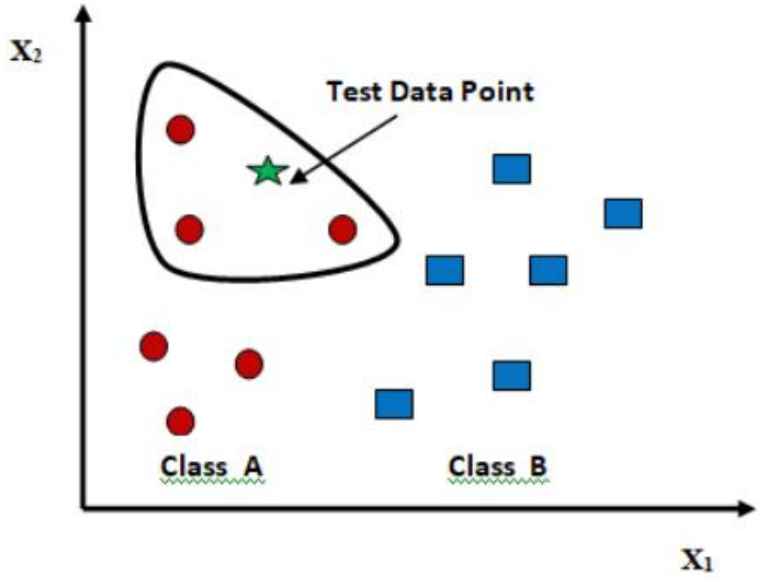

Figura 2.3: Funzionamento KNN [27].

Kaushal [44] ha proposto un algoritmo per l'identificazione delle malattie delle piante usando un classificatore KNN. Le prestazioni ottenute si sono rivelate migliori se comparate a quelle di un classificatore SVM.

2.2.4 Rete Neurale Artificiale

Una Rete Neurale Artificiale (ANN) è un modello computazionale che prende ispirazione dalla struttura di una rete neurale biologica. Una ANN si basa sulla simulazione di neuroni artificiali distribuiti nei tre layer: input (ingresso) , hidden (nascosti) e di output (uscita) (Figura 2.4). Una rete neurale con molteplici layer nascosti e svariati nodi in ognuno di questi è conosciuta come **sistema di Deep Learning** o come **rete neurale profonda**. Essenzialmente, ogni rete neurale con più di tre layer, inclusi quello di input e di output, è considerabile un modello di Deep Learning [45]. Con più layer nascosti, la rete è in grado di imparare relazioni complesse dalla combinazione gerarchica di più caratteristiche.

Esistono molteplici tipologie di reti neurali artificiali, ognuna progettata per eseguire un compito specifico e ottimizzata per un tipo particolare di dati.

- **Rete neurale con flusso in avanti**: è il tipo di rete neurale più semplice, dove il flusso di dati si muove in una sola direzione, dal layer d'ingresso a quello d'uscita, senza alcun ciclo;
- **Rete neurale ricorrente**: progettata per processare sequenze di dati, come ad esempio il linguaggio naturale. Contiene dei cicli, il che permette alle informazioni di durare nel tempo;

- **Rete neurale convoluzionale**: progettata per elaborare dati sotto forma di griglia, come immagini e video;
- **Deep Belief Network**: composta da più layer posto uno sopra l'altro tra loro connessi, ma non ci sono connessioni tra le unità all'interno di ogni layer;
- **Rete generativa avversaria**: composta da due reti, una rete generatrice ed una discriminante, le quali sono allenate insieme col fine di generare nuovi dati simili a quelli di un dataset dato in ingresso;
- **Autocodificatore**: è una rete neurale allenata per ricostruire i dati in ingresso, tipicamente per la riduzione della dimensionalità o per l'apprendimento di caratteristiche.

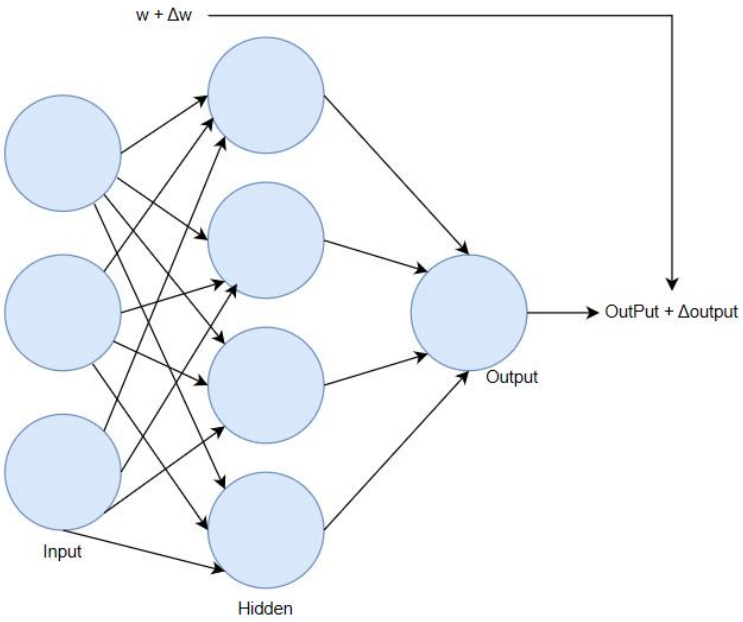

Figura 2.4: Struttura della rete neurale [33].

Nel suo studio Kutty ha proposto un metodo per classificare le malattie della Peronospora e dell'Antracnosi nelle foglie di anguria. Le foglie infette sono state differenziate da quelle sane tramite lo studio della gamma RGB. Per la classificazione è stata usata una rete neurale per l'identificazione di pattern ottenendo un'accuratezza del 75.9% [46].

2.2.5 Rete Neurale Convoluzionale

Nonostante sia il Deep Learning che gli algoritmi tradizionali di Machine Learning possono essere usati per risolvere le stesse tipologie di problemi, ad oggi gli algoritmi di Deep Learning sono diventati una scelta popolare tra gli studiosi nel campo dell'identificazione delle malattie delle piante grazie ai loro svariati vantaggi.

Tra gli algoritmi di Deep Learning più utilizzati nel campo dell'identificazione delle malattie delle piante troviamo le Reti Neurali Convoluzionali (CNN). Le CNN hanno ottenuto risultati ottimi nelle varie sfide per la classificazione di immagini, come in "Large Scale Visual Recognition Challenge". Col passare degli anni, le architetture si sono evolute e hanno ottenuto risultati sempre migliori, spingendo gli esperti all'applicazione di queste in vari domini applicativi.

Come si può intuire dallo studio della letteratura odierna in ambito agricolo, le CNN sono sempre più utilizzate. Un esempio è il lavoro di Lin [47], in cui compara le prestazioni delle architetture di CNN più celebri, con un modello di CNN chiamato "GrapeNet", creato ad hoc per lo studio dei sintomi delle varie fasi delle malattie della pianta della vite. I risultati si sono rivelati incoraggianti, dato che GrapeNet ha ottenuto le migliori prestazioni di classificazione quando confrontato con le architetture classiche, raggiungendo un'accuratezza del 86.29% ed un numero di parametri di solo 2.15 milioni.

Anche nel lavoro di Liu [14] è stato creato un modello totalmente nuovo per l'identificazione delle malattie del vigneto chiamato "DICNN", ottenendo un'accuratezza del 97.22% ed un incremento confrontando con GoogLeNet e ResNet-34 del 2.97% e 2.55% rispettivamente.

2.3 Conclusioni

Dall'analisi dei vari studi in letteratura, possiamo comprendere che la classificazione delle malattie delle piante è un ambito che sta ottenendo risultati promettenti. Abbiamo compreso la vastità di database utilizzati e la loro diversità, proprio per questo confrontare i risultati risulta un compito difficile e spesso inconcludente. Gli approcci basati sul Deep Learning hanno ottenuto i risultati migliori soprattutto nell'ambito della classificazione, in particolare utilizzando architetture CNN preesistenti di cui parleremo più approfonditamente nel prossimo capitolo, o creando dei modelli architetturali specifici per lo studio in esame.

Capitolo 3

Rete Neurale Convoluzionale

Come già anticipato nel capitolo precedente, le **Reti Neurali Convoluzionali** (CNN) hanno ottenuto risultati più che soddisfacenti nel compito di identificazione delle malattie delle piante; per questo motivo, sono state adoperate nel seguente studio ed illustrate approfonditamente nel capitolo che segue.

3.1 Architettura di una Rete Neurale Artificiale

Una Rete Neurale Artificiale (Artificial Neural Network, ANN), come già anticipato, è un modello computazionale del Machine Learning che simula la struttura ed il comportamento del cervello umano. Il cervello è l'unità di controllo della nostra rete neurale, questo è composto da **neuroni** interconnessi che sono gli elementi costitutivi del sistema nervoso centrale. Un neurone è costituito da tre parti fondamentali (Figura 3.1): **dendriti**, **sinapsi** ed **assoni**.

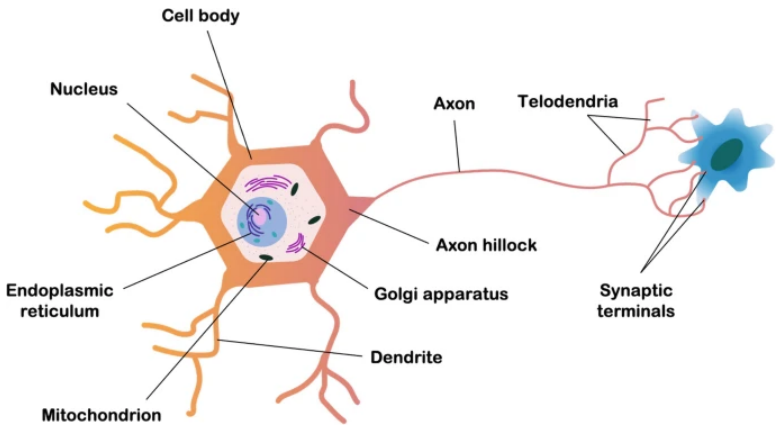

Figura 3.1: Struttura di un neurone [48].

I neuroni comunicano tra loro attraverso **segnali** elettrici e chimici, il che gli permette di processare informazioni e di farci reagire a stimoli esterni. Quando un neurone riceve un segnale da un altro neurone, genera un segnale elettrico conosciuto come **potenziale d'azione**

che viaggia attraverso l'**assone**, fino a raggiungere la sua parte terminale. Raggiunta la parte terminale, delle sostanze chimiche chiamate **neurotrasmettitori** vengono rilasciate nella **sinapsi**, cioè la struttura che si trova tra la parte finale dell'assone e i dendriti del neurone con cui si sta comunicando. I neurotrasmettitori si legano ai recettori dei **dendriti** nel neurone successivo, portando alla creazione di un nuovo segnale elettrico nel neurone ricevitore. Questo processo continua finché il segnale viaggia tra i neuroni, fino al raggiungimento della destinazione finale, ovvero l'azione richiesta in risposta allo stimolo esterno. In aggiunta, è possibile individuare svariate similitudini tra le reti neurali artificiali e quelle biologiche. Le reti artificiali sono composte da unità interconnesse che sono dei veri e propri **neuroni artificiali**, anche chiamati "nodi", il cui funzionamento si basa sul concetto di somma ponderata ed attivazione.

Per dare in input dei valori alla nostra rete artificiale, prima di tutto dobbiamo trasformare i dati in una forma che sia interpretabile da un elaboratore. Solitamente assegniamo dei valori numerici alle varie informazioni che intendiamo rappresentare, ottenendo un vettore numerico (Figura 3.2) in cui ogni elemento rappresenta una categoria o la tipologia di dato.

Figura 3.2: Rappresentazione dei dati in ingresso ad un neurone artificiale [49].

I segnali in ingresso ai neuroni artificiali vengono moltiplicati per dei valori detti **pesi** (Figura 3.3) e poi sommati da una **funzione di trasferimento**. La somma ottenuta dei valori in ingresso ponderati viene poi passata ad una funzione d'attivazione, la quale ha il compito di decidere se il neurone deve attivarsi oppure no, in base ad una soglia.

La funzione di attivazione, illustrata in Figura 3.4, è una componente fondamentale del neurone artificiale, perché determina i dati in uscita, basandosi sui dati in ingresso. Le funzioni d'attivazione più comuni sono:

- **Funzione sigmoidea**: mappa ogni input nell'intervallo 0-1, utilizzata per problemi di classificazione binaria, dove i dati in uscita devono essere interpretati come una probabilità;

- **Funzione rettificatore**: mappa tutti gli input negativi a 0 e lascia gli input positivi invariati. È una funzione computazionalmente efficiente, però può portare al problema della "ReLU morente", dove i neuroni diventano inattivi e smettono d'imparare;
- **Funzione softmax**: mappa l'output di un neurone in una distribuzione di probabilità su classi multiple, permettendo alla rete di produrre una previsione affidabile per la classe con la probabilità più alta. Particolarmente utilizzata nell'ultimo layer di una rete per problemi di classificazione multi-classe.

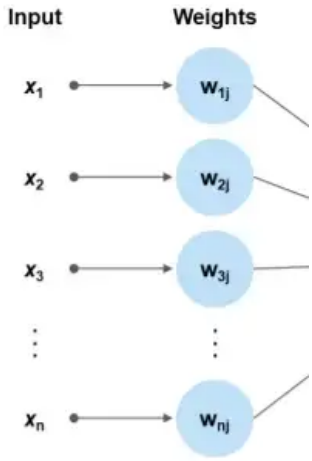

Figura 3.3: Rappresentazione dei dati ponderati in un nodo [49].

I pesi delle connessioni tra i nodi o altri parametri sono tipicamente aggiustati durante l'allenamento della rete per ottimizzare le sue prestazioni ed aumentare l'affidabilità delle previsioni.

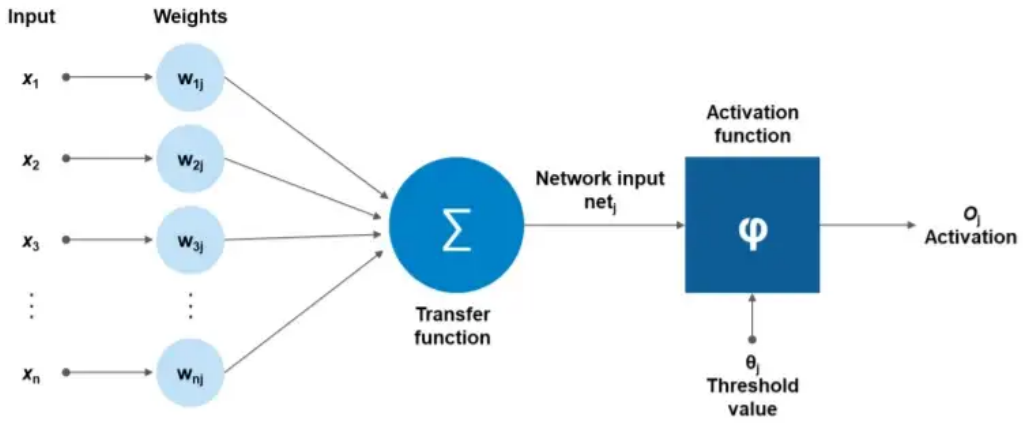

Figura 3.4: Rappresentazione della struttura di un neurone artificiale [49].

Una rete neurale risulta essere composta da più layer come si evince dalla Figura 2.4, cioè:

- **Layer d'ingresso**: layer in cui c'è l'ingresso dei dati la cui tipologia dipende dal problema affrontato;

- **Layer nascosti**: si trovano tra i livelli d'ingresso e d'uscita e dipendono dal tipo di rete implementata; il numero di livelli determina la profondità della rete e, come precisato nel capitolo due, una rete con più livelli nascosti è considerata una **rete neurale profonda**;
- **Layer di uscita**: layer che restituisce il risultato sotto forme diverse, dipendentemente dalla funziona di attivazione utilizzata.

Il Machine Learning ha diverse tecniche che permettono la soluzione di determinati tipi di problematiche e il loro utilizzo dipende dal tipo di dati disponibili:

- **Apprendimento Supervisionato**
  Nell'Apprendimento Supervisionato il modello è allenato con **dati di addestramento etichettati**, dove i segnali di output sono già noti. In altre parole, diamo in input un insieme di dati ed i corrispondenti dati di output, quindi il modello impara a predire i dati di output basandosi sulle caratteristiche dei dati in input. Questo tipo di apprendimento è tipicamente usato per problemi di regressione e di classificazione.
  - **Classificazione**
    La classificazione è una tecnica utilizzata nell'apprendimento supervisionato, dove i dati in input sono usati per predire un'etichetta discreta da un insieme finito di possibili etichette. L'algoritmo prova a trovare la corrispondenza tra l'input e l'output, per poi usarla per fare delle previsioni su dati mai visti prima;
  - **Regressione**
    Nella regressione non sono predette delle etichette classificate, bensì dà in output delle variabili continue, come dei valori numerici. Solitamente utilizzata per le previsioni finanziarie o ambiti simili.
- **Apprendimento non Supervisionato**
  Nell'Apprendimento Supervisionato il modello è allenato con **dati di addestramento non etichettati**. In questa tipologia di apprendimento, l'algoritmo cerca d'individuare pattern o strutture nei dati senza che gli venga detto quale dovrebbe essere l'output corretto. Con questo algoritmo possono essere scoperti dei pattern nascosti nei dati o estrarre delle caratteristiche dai dati utili per altri task di Machine Learning. È tipicamente usato per problemi di clustering e per la riduzione della dimensionalità dei dati.
  - **Clustering**
    Il Clustering è una tecnica per l'analisi dei dati che implica il raggruppamento di insiemi di dati in gruppi chiamati **cluster**. L'obiettivo del clustering è quello di trovare strutture o pattern all'interno dei dati che non sono immediatamente evidenti e di dividere i dati in gruppi basati sulle loro similarità;
  - **Riduzione della dimensionalità dei dati**
    La riduzione della dimensionalità è una tecnica nel Machine Learning che implica

la riduzione del numero di caratteristiche, chiamate **feature**, in un dataset. Il fine ultimo della riduzione della dimensionalità è quello di semplificare i dati e al contempo preservare quante più informazioni possibile. Il motivo per il quale si applica questa tecnica, è quello di affrontare le problematiche che sorgono con l'utilizzo di dataset ad alta dimensionalità, come l'inefficienza computazionale ed il sovradattamento.

- **Apprendimento Semi-Supervisionato**
  L'Apprendimento Semi-Supervisionato combina elementi di entrambe le tecniche precedentemente illustrate. In questo tipo di tecnica, il modello viene allenato su un mix di dati etichettati e non, permettendogli di utilizzare dati conosciuti e sconosciuti per produrre risultati più accurati, se confrontati con le tecniche con supervisione e senza supervisione prese singolarmente. Solitamente è utilizzata quando si dispone di un numero limitato di dati etichettati; in questi casi, l'algoritmo impara contemporaneamente dal gran numero di campioni non etichettati e fa anche uso dei pochi etichettati a disposizione.

La **Retropropagazione dell'errore** è uno degli algoritmi d'apprendimento supervisionato più usati per l'allenamento delle reti neurali artificiali. È un algoritmo iterativo per l'ottimizzazione, usato per regolare i pesi dei neuroni nella rete, e quindi per la minimizzazione dell'errore tra l'output predetto e l'output effettivo.

L'algoritmo di Retropropagazione si basa su un noto algoritmo di ottimizzazione [50], quello della **discesa del gradiente**, utilizzato per minimizzare la funzione di costo; questo è una misura della differenza tra i valori predetti e gli effettivi valori dei dati. L'obiettivo dell'allenamento è quello di trovare i parametri ottimali per il modello, che portano alla minimizzazione del costo calcolando il costo del gradiente, rispettando i parametri del modello e aggiornandoli nella direzione opposta al gradiente. Questo processo è reiterato finché l'errore raggiunge un livello accettabile o il numero massimo di iterazioni è stato raggiunto.

Quindi, dato un insieme di input X e i corrispondenti output Y desiderati, l'algoritmo regola i pesi della rete cosicché l'errore E della previsione possa essere minimizzato. La funzione di errore E può essere definita come:

$$E = \frac{1}{2}\sum_{i=1}^{n}[Y' - Y]^2 \tag{3.1}$$

dove Y' è l'output predetto ed Y è l'effettivo output.

Il termine "retropropagazione" è utilizzato perché l'algoritmo richiede il calcolo del gradiente solamente nel layer di output, il quale viene poi propagato all'indietro lungo l'intera rete per ricalcolare i pesi e ridurre l'errore.

È importante sottolineare che la funzione di attivazione, il learning rate e l'inizializzazione dei pesi possono influire sulle prestazioni della retropropagazione.

Dopo aver illustrato le basi del funzionamento e la struttura delle Reti Neurali, ora passeremo all'analisi delle reti usate nel presente elaborato: le **Reti Neurali Convoluzionali**.

Per utilizzare le reti neurali e gli algoritmi per i diversi compiti che vogliamo condurre, dobbiamo dare in input al nostro modello un insieme di immagini da analizzare. Per fare ciò, dobbiamo prima capire come i computer interpretano un'immagine e come la processano, dato che per noi umani la visione sembra un'abilità comune, o addirittura banale.

Dal punto di vista di un calcolatore, un'immagine è un grande insieme di pixel, ed ognuno di questi contiene informazioni sul colore, sull'intensità e sulla sua posizione all'interno dell'immagine. Partendo da questo presupposto, il computer vede l'immagine come una matrice di numeri con due dimensioni (Figura 3.5), in cui ogni pixel viene rappresentato da un singolo numero che va da 0 a 255, se parliamo di un'immagine in scala di grigio. Se invece abbiamo a che fare con delle immagini RGB, cioè colorate, avremo tre canali, uno per ogni colore; quindi uno per il rosso, uno per il verde ed uno per il blu, con altrettante matrici a due dimensioni.

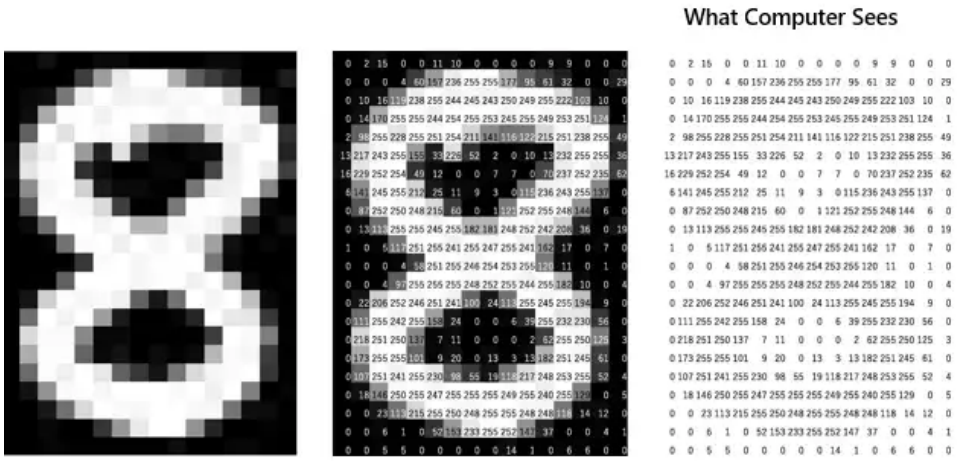

Figura 3.5: Rappresentazione di un'immagine vista dal computer [51].

3.2 La Convoluzione

Una Rete Neurale Convoluzionale (CNN) è un'architettura del Deep Learning specificatamente progettata per il riconoscimento d'immagini e compiti di visione artificiale. A differenza delle ANN, le CNN dispongono di un'architettura specializzata, con i layer di convoluzione e di pooling progettati appositamente per compiti di analisi delle immagini; inoltre, le CNN sono anche computazionalmente più efficienti grazie a questi layer, che permettono la riduzione del numero di parametri e della grandezza dell'output. Questa caratteristica rende questi modelli adatti per l'elaborazione di grandi volumi di dati, come immagini ad alta risoluzione e video.

Le CNN riescono a riconoscere delle caratteristiche all'interno delle immagini, chiamate **Feature**, in modo automatico, infatti non richiedono molto preprocessing. Il termine "Convoluzionali" nasce proprio dall'operazione matematica sfruttata, cioè la Convoluzione, che

consiste in un tipo particolare di operazione lineare usata per l'estrazione delle feature da un'immagine. Il layer convoluzionale è il layer di base di una CNN, in cui si svolge la maggior parte dell'elaborazione. Con ogni layer aggiunto, una CNN aumenta di profondità e, di conseguenza, incrementa anche la sua complessità, andando così ad identificare porzioni più grandi dell'immagine. I diversi layer possono essere rappresentati secondo un ordine gerarchico, infatti i primi in cui ci imbattiamo nel modello si focalizzano su delle feature di basso livello come colori e margini dell'immagine. Col passaggio dell'immagine lungo i diversi layer della rete, questa inizia a riconoscere feature di più alto livello fino al riconoscimento finale degli oggetti visualizzati.

Come abbiamo precedentemente spiegato, un'immagine RGB è rappresentata da tre matrici, una per ogni canale di colore, a loro volta da due dimensioni ciascuna. Delle immagini ad alta risoluzione, quindi con una maggiore densità di pixel, risulterebbero troppo complesse computazionalmente da analizzare; proprio qui entra in gioco la convoluzione, che si occupa di ridurre le immagini in una forma che sia più semplice da processare, preoccupandosi però di non perdere delle feature significative per la previsione finale.

3.2.1 Kernel

L'elemento che si occupa della riduzione si chiama **Kernel**, o filtro. Il filtro ha una sua **dimensione** che si può scegliere all'occorrenza, però la grandezza standard è posta a 3x3. Inoltre, possiamo scegliere la **falcata** del filtro usato, che consiste nel numero di pixel di cui il filtro si sposta sull'immagine. Quindi, se il filtro ha una grandezza di 3x3, questo prenderà in considerazione una matrice di pixel della stessa dimensione sull'immagine, effettuando un **prodotto di Hadamard** tra il filtro e la porzione d'immagine sulla quale il kernel si trova (Figura 3.6). Quest'operazione continua da sinistra a destra, finché il filtro non ha completato l'intera larghezza dell'immagine, in quel caso il filtro si sposterà alla riga successiva, sempre in base al parametro della falcata inserito.

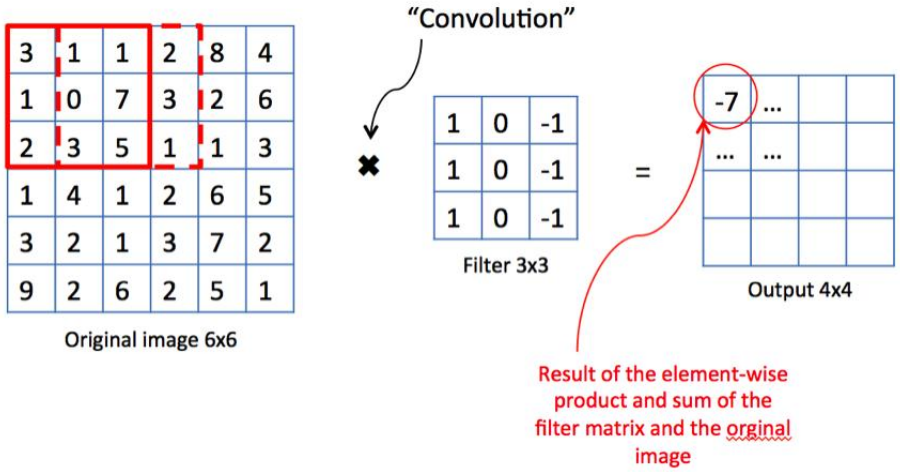

Figura 3.6: Esempio di convoluzione [52].

Nel caso in cui abbiamo a che fare con un'immagine RGB, quindi con più canali, il kernel avrà la stessa profondità dell'immagine in input, identificata dal numero di canali (Figura 3.7). La moltiplicazione tra matrici è eseguita e tutti i risultati sono sommati, così facendo otteniamo una singola mappa delle caratteristiche (Feature Map).

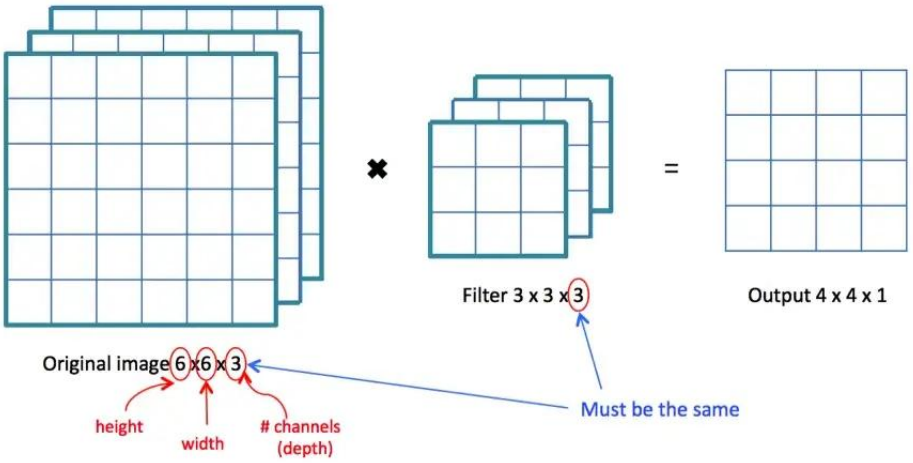

Figura 3.7: Convoluzione di un immagine RGB [52].

Naturalmente, è possibile aumentare il numero di filtri utilizzati durante la convoluzione, affinché aumenti il numero di feature rilevate dalla rete. Ogni filtro avrà un suo specifico output e questi possono essere sovrapposti tra loro.

3.2.2 Padding

Durante il passaggio dell'immagine di input attraverso i layer convoluzionali, le dimensioni di quest'ultima tendono a ridursi. La riduzione delle dimensioni spaziali dell'immagine porta ad una perdita di informazioni e ad una riduzione dell'abilità della rete nell'individuazione delle

feature nell'immagine.

Per affrontare questa problematica, si ricorre al **padding** [53]. Il padding è un processo tipicamente utilizzato per preservare le dimensioni spaziali dell'immagine durante il suo passaggio attraverso la CNN, permettendogli in tal modo di continuare ad individuare le feature non perdendo nessuna informazione.

Consiste nell'aggiunta di pixel extra ai bordi dell'immagine con valore 0, che quindi non andranno ad influire sulla moltiplicazione matriciale. Esistono due tipologie di padding: "valid padding" e "same padding". Il **valid padding** ci permette di non andare ad applicare il padding all'immagine, ottenendo così una dimensione in output minore di quella in input. D'altra parte, con il **same padding**, la dimensione in output sarà uguale a quella data in input, applicando il padding così come illustrato in Figura 3.8.

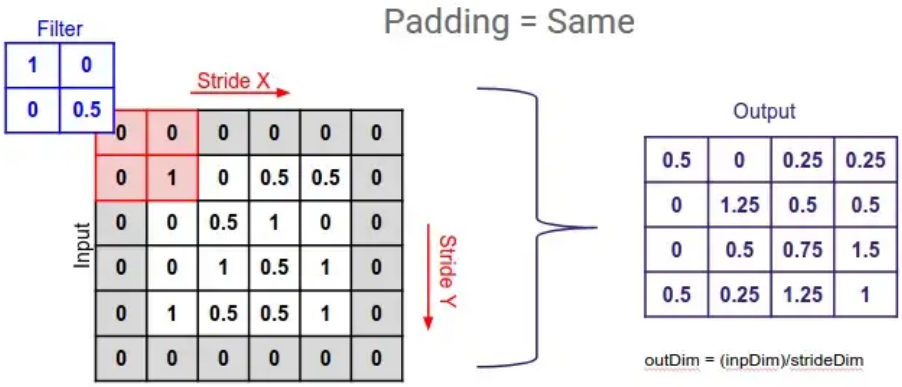

Figura 3.8: Esempio di convoluzione con padding [53].

Di seguito riporto la formula utile per il calcolo delle dimensioni di output dopo la convoluzione, conoscendo le dimensioni del filtro (f), la falcata (s), il padding (p) e le dimensioni dell'input [52]:

$$Outputsize = (\frac{n + 2p - f}{s} + 1) \times (\frac{n + 2p - f}{s} + 1) \tag{3.2}$$

Le reti neurali convoluzionali si basano su tre proprietà fondamentali: **l'interazione sparsa** (sparse interaction), **l'invarianza di traslazione** (equivariant representation) e la **condivisione dei parametri** (parameter sharing).

3.2.3 Interazione sparsa

L'interazione sparsa è una tecnica che si occupa della riduzione del numero di parametri nella CNN. L'idea dietro l'interazione sparsa è di incentivare la rete ad affidarsi solo ad un sottoinsieme dei suoi filtri, riducendo così il numero di parametri da imparare.

Processando un'immagine, si dispone di centinaia o migliaia di pixel, ma tecnicamente si possono riconoscere solo ridotte quantità di feature significative; questo sta a significare che è possibile ridurre il numero di parametri da salvare.

L'interazione sparsa permette quindi l'ottenimento di una CNN con meno parametri, costo computazionale ridotto e un miglioramento delle prestazioni di generalizzazione. Tutto questo grazie alla riduzione della "over-parameterization" e al miglioramento dell'interpretabilità della rete, dato che diventa più semplice capire quali filtri contribuiscono al risultato finale.

3.2.4 Condivisione dei parametri

Nella condivisione dei parametri, molteplici neuroni di un layer tendono ad utilizzare lo stesso insieme di pesi e biases per l'esecuzione delle loro computazioni. Questo approccio differisce dalle reti neurali tradizionali, dove ogni neurone possiede il proprio insieme di pesi.

Come nell'interazione sparsa, si segue lo stesso obiettivo di ridurre i parametri che la rete deve imparare, riducendo così il rischio di sovradattamento e migliorando l'efficienza della rete. Questa pratica si rivela particolarmente importante nei compiti di visione artificiale, dove si utilizzano tanti dati ed il numero di parametri può facilmente diventare proibitivo.

In un layer convoluzionale, la condivisione dei parametri si ottiene usando un filtro convoluzionale che scorre su tutta l'immagine data in input, svolgendo il prodotto scalare tra il filtro e una regione locale di questo. Usando lo stesso filtro in regioni diverse dell'immagine, la rete riesce ad imparare un singolo insieme di pesi che può essere usato per rilevare le stesse feature in diverse posizioni.

È importante però sottolineare che l'utilizzo di questa tecnica impone un limite alla capacità di apprendimento della rete. Questo sta a significare che la rete potrebbe riscontrare difficoltà ad imparare feature più complesse che variano nelle regioni dell'immagine. Per affrontare il problema, si utilizzano molteplici layer convoluzionali con diverse tipologie di filtro, per imparare a differenziare le varie feature; ogni layer quindi condividerà un insieme diverso di parametri.

3.2.5 Invarianza di traslazione

L'invarianza di traslazione si riferisce alla proprietà di un modello di far rimanere invariato l'output nonostante siano applicate delle trasformazioni ai dati in input. Nell'ambito della classificazione d'immagini, ad esempio, un modello è invariante alle rotazioni, traslazioni e altre trasformazioni, se l'output del modello rimane lo stesso dopo l'applicazione di queste trasformazioni all'immagine utilizzata come input.

Il vantaggio principale è che il modello riesce ad imparare delle informazioni che rendono il modello utilizzato più robusto e lo portano ad una migliore generalizzazione delle immagini.

3.3 Struttura di una CNN

Come detto più volte durante il seguente elaborato, una rete neurale convoluzionale si articola in molteplici layer alternati tra loro. Un'architettura tipica è formata da ripetizioni di uno stack di più "**convolutional layers**" e un "**pooling layer**", seguiti da uno o più "**fully connected layers**".

- **Input layer**
  L'input layer è il primo della rete e si occupa di prendere in ingresso l'immagine. L'immagine in input è rappresentata come una matrice multidimensionale con diverse profondità, in base al numero di canali di colore di questa;

- **Convolution layer**
  Come spiegato nella sezione precedente, i layer di convoluzione sono la parte fondamentale di una CNN, dove la rete esegue le operazioni di estrazione delle feature sull'immagine in input; queste consistono in una combinazione di operazioni lineari e non, cioè le operazioni di convoluzione e la funzione d'attivazione. La convoluzione utilizza una matrice di numeri, chiamata **kernel**, che applica lungo tutto l'input e che consiste anch'esso in una matrice di numeri, chiamato **tensore**. Il fine ultimo di questi layer è quello di ottenere delle feature map, che rappresentano diverse caratteristiche del tensore in input. Per fare ciò, molteplici filtri sono applicati all'immagine e possiamo considerare ognuno di questi come un estrattore di feature;

- **Funzione d'attivazione non lineare**
  I vari output di un'operazione lineare, quale è la convoluzione, sono poi passati ad una funzione d'attivazione non lineare. La funzione d'attivazione non lineare più comunemente utilizzata è **l'unità lineare rettificata**, **ReLU** (Figura 3.9) in inglese, la quale applica la seguente funzione:

$$f(x) = max(0, x) \tag{3.3}$$

  La funzione semplicemente rende tutti gli output maggiori o uguali di zero.

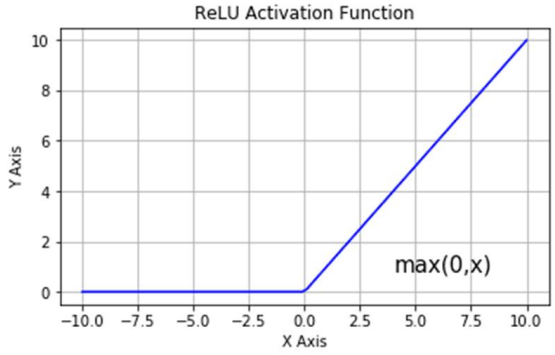

Figura 3.9: Funzione d'attivazione ReLU [54].

- **Pooling layer**
  Il pooling layer esegue delle operazioni di ridimensionamento sulle feature map generate dal convolution layer, affinché si introduca invarianza di traslazione rispetto a distorsioni o traslazioni, con conseguente riduzione del numero di parametri.

  Questo layer si preoccupa di effettuare il ridimensionamento, preservando però le informazioni più importanti, riducendo il carico computazionale e prevenendo il fenomeno del sovradattamento. Esistono diversi algoritmi di pooling, i quali illustreremo più avanti, ma senza dubbio il più utilizzato è il **Max pooling**.

  Di seguito è illustrata la formula [55] da utilizzare per calcolare la grandezza dell'output per una feature map con dimensioni $\boldsymbol{n_h} \times \boldsymbol{n_w} \times \boldsymbol{n_c}$, dove: $\boldsymbol{n_h}$ è l'altezza, $\boldsymbol{n_w}$ la larghezza e $\boldsymbol{n_c}$ il numero di canali della feature map. Inoltre useremo anche i simboli **f** per indicare le dimensioni del filtro e **s** per la lunghezza della falcata.

$$\frac{(n_h - f + 1)}{s} \times \frac{(n_w - f + 1)}{s} \times n_c \tag{3.4}$$

- **Fully connected layer**
  Le feature map del pooling layer sono tipicamente trasformate in un vettore di numeri, quindi con una sola dimensione, e quest'operazione in inglese è detta "Flattening". Successivamente, questo vettore è collegato ad uno o più **fully connected layer**, i quali sono anche conosciuti come **dense layer**, cioè una tradizionale rete neurale con flusso in avanti dove non si formano cicli ed ogni input è connesso ad ogni output attraverso un peso specifico. Dopo le trasformazioni avvenute nei convolution layer e i pooling layer, le feature estratte sono mappate da un gruppo di fully connected layer, ognuno seguito da una funzione non lineare come la ReLU sopra descritta, per la creazione degli output finali della rete.

- **Output layer**
  Come ultimo layer della nostra rete troviamo l'output layer. Questo particolare fully connected layer si occupa di produrre le previsioni di classificazione, le quali possono essere di diversi tipi a seconda del tipo di funzione di attivazione applicata. Questa deve essere scelta in modo appropriato, in base al tipo di task che si sta svolgendo.

  Le previsioni di classificazione si basano sulle feature imparate dai layer convoluzionali, di pooling e i pesi imparati dai fully connected layer. Nella Tabella 3.1 che potete osservare in basso, sono illustrate le funzioni di attivazione in base al tipo di task.

Task	Funzione di attivazione applicata all'output layer
Classificazione binaria	Sigmoide
Classificazione multi-classe	Softmax
Classificazione multi-etichetta	Sigmoide
Regressione a valori continui	Identità

Tabella 3.1: Funzioni d'attivazione output layer

3.4 Algoritmi di Pooling

Nell'architettura di una CNN è pratica comune inserire fra due o più layer convoluzionali un layer di Pooling, la cui funzione è quella di ridurre progressivamente la dimensione spaziale delle feature map in output, in modo da diminuire il numero di parametri e il carico computazionale. Di seguito saranno evidenziati alcuni tra gli algoritmi di pooling utilizzati e il loro ruolo nella costruzione delle reti convolutive.

3.4.1 Max Pooling

Come già affermato in precedenza, l'algoritmo di pooling più utilizzato è il **max pooling**. Questo seleziona il valore massimo in ogni regione della feature map (Figura 3.10) analizzata dal filtro e produce in output un'ulteriore feature map contenente le feature più significative della precedente.

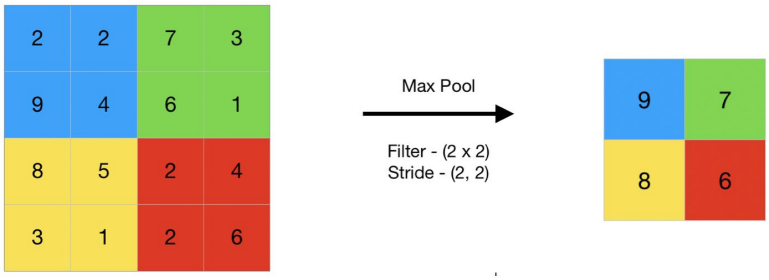

Figura 3.10: Esempio di Max Pooling [55].

3.4.2 Average Pooling

L'**average pooling** (Figura 3.11) è un'alternativa al max pooling, che effettua la media dei valori presenti nella regione di feature map analizzata dal filtro. Quindi, mentre il max pooling dà in output le feature più importanti, l'average pooling dà la media delle feature presenti nella feature map, però si rivela meno efficace a preservare le feature più importanti.

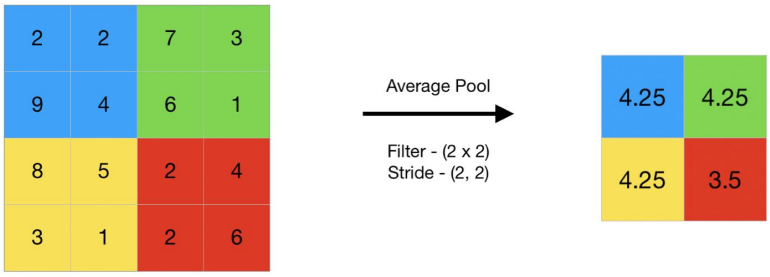

Figura 3.11: Esempio di Average Pooling [55].

3.4.3 Global Pooling

Il **global pooling** effettua un tipo estremo di riduzione aggregando l'intera feature map in un singolo valore, quindi trasformando una feature map di dimensioni $\boldsymbol{n_h} \times \boldsymbol{n_w} \times \boldsymbol{n_c}$ in una feature map di $\mathbf{1} \times \mathbf{1} \times \boldsymbol{n_c}$. Questa tipologia di pooling può essere sia usata come global max pooling, che come global average pooling.

3.4.4 Stochastic Pooling

Stochastic pooling (Figura [56]) è una tecnica usata per affrontare le problematiche di sovradattamento, infatti è basata sullo stesso concetto del dropout.

La tecnica consiste nello scegliere casualmente uno dei valori della feature map basandosi sulle probabilità create dalla distribuzione multinomiale, invece di scegliere sempre il valore massimo. Questo aiuta il modello ad imparare delle feature più robuste, dal momento che non fa affidamento su uno specifico valore d'attivazione.

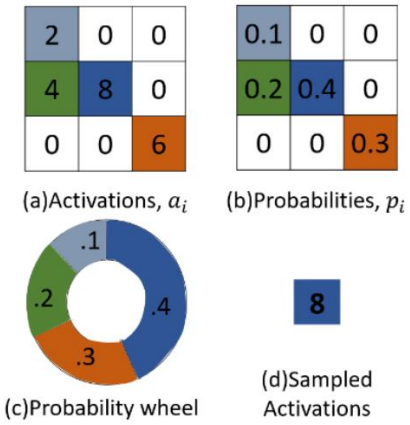

Figura 3.12: Esempio di Stochastic Pooling [56].

Capitolo 4

Soluzione proposta

L'obiettivo di questo elaborato è quello di rilevare cinque tipi di malattie che colpiscono le foglie della pianta della vite, ossia: Peronospora, Oidio, Mal dell'Esca, Marciume Nero e Ruggine della Foglia. Pertanto, è stato dapprima raccolto un insieme di immagini di foglie della vite che presentavano le malattie elencate, prese sia da dataset pubblici che da varie ricerche in rete. Data la scarsa qualità e quantità d'immagini raccolte, sono state applicate tecniche di segmentazione delle immagini, al fine di rimuovere lo sfondo da ciascun fotogramma e rendere il database più omogeneo. Per l'analisi delle immagini si è deciso di far uso delle Reti Neurali Convoluzionali, considerate l'avanguardia nell'ambito del riconoscimento d'immagini.

Nella prima sottosezione di questo capitolo sono descritti gli strumenti utilizzati durante questo lavoro di ricerca, come le varie librerie e le piattaforme utilizzate con annesso hardware; nella seconda sottosezione sono introdotte le quattro reti utilizzate per il caso di studio; nella terza sottosezione è descritto l'iter di ricerca delle immagini e le relative tecniche di preprocessing applicate; infine, nell'ultima sottosezione, è analizzata la configurazione dei modelli utilizzati e i risultati finali ottenuti.

4.1 Strumenti utilizzati

4.1.1 Google Colab

Google Colab è una piattaforma cloud che offre accesso gratuito alla piattaforma di elaborazione interattiva Jupyter Notebook e ad un ambiente in hosting per l'esecuzione e la sperimentazione con codice Python, insieme anche ad altri linguaggi di programmazione. È stata creata da Google come strumento di ricerca per il Machine Learning e la data science ed attualmente è largamente utilizzata da ricercatori e studenti nell'industria tecnologica.

Colab permette di scrivere, eseguire e condividere frammenti di codice; inoltre consente di collaborare con altre persone sui vari notebook, modificandoli in condivisione, e sperimentando modelli di Machine Learning, tutto all'interno del browser. I notebook sono dei documenti

interattivi nei quali è possibile scrivere (e quindi eseguire) il codice sviluppato.

Il motivo principale per il quale si utilizza Google Colab è senza dubbio l'accesso ad elevate risorse computazionali, come la GPU (NVIDIA Tesla T4 da 16 GB per l'utilizzo gratuito) e/o la TPU, rendendo possibile l'esecuzione di modelli computazionali complessi e l'allenamento di grandi modelli di Deep Learning.

Un altro beneficio di Colab è che integra Google Drive, permettendo di salvare e condividere facilmente i notebook, utilizzare dati salvati sul Drive o salvare i modelli allenati per futuri utilizzi. È possibile anche interfacciarsi con GitHub, il che rende possibile la clonazione e la modifica dei notebook presenti su GitHub, direttamente da Colab. Inoltre, Colab fornisce l'accesso a varie librerie di Machine Learning preinstallate, inclusi TensorFlow, Keras, PyTorch, OpenCV e una grande varietà di strumenti per l'analisi dei dati e la loro visualizzazione; questo rende la vita più facile ai fruitori del servizio, dato che non dovranno avere a che fare con eventuali conflitti tra le versioni delle librerie o installazioni corrotte.

4.1.2 TensorFlow

Figura 4.1: Logo TensorFlow [57].

TensorFlow è un libreria software per il dataflow e la programmazione differenziabile in una vasta gamma di task. È una libreria di algebra computazionale ed è soprattutto utilizzata per applicazioni di Machine Learning come le reti neurali. È stata sviluppata da Google ed è usata in molti dei suoi servizi, come Google Colab, grazie alla sua ampia capacità di comprendere e processare linguaggi naturali, classificazione di testi, immagini e riconoscimento di scrittura manuale.

TensorFlow permette agli utenti di sviluppare modelli di Machine Learning che siano computazionalmente complessi e ad alta scalabilità, fornendo strumenti per la costruzione, l'allenamento e la distribuzione dei progetti. Include strumenti per la scrittura e l'esecuzione di elaborazioni su matrici di dati, chiamate "tensori", oltre ad API ad alto livello per la costruzione e l'allenamento dei modelli e API di basso livello per la personalizzazione e l'estensione di questi ultimi. Il linguaggio di programmazione di base è Python, inoltre dispone di una

ampia comunità di ricercatori che sviluppano continuamente API aggiornate e migliorate in modo da mantenere sempre al passo la piattaforma.

4.1.3 Keras

Figura 4.2: Logo Keras [58].

Keras è una libreria software di alto livello che fornisce un'interfaccia Python per le reti neurali artificiali. È stata sviluppata per consentire una sperimentazione veloce e non per modelli complessi su larga scala. Questa si appoggia su librerie di più basso livello come ad esempio TensorFlow.

Keras offre un'interfaccia semplice e modulare per la creazione e l'allenamento di reti neurali; astrae la complessità della creazione di un modello, permettendo agli utenti di focalizzarsi sulle scelte di design e sulla valutazione del loro modello, invece che sulle difficoltà della programmazione. Con Keras è possibile costruire molte tipologie di architetture, come: reti con flusso in avanti, reti neurali convoluzionali, reti neurali ricorrenti e molte altre.

Una delle funzionalità più interessanti è sicuramente l'uso di Keras per il Transfer Learning; infatti, consente ai consumatori di utilizzare una vasta gamma di reti neurali convoluzionali preallenate sul celebre dataset "ImageNet", permettendo così un utilizzo celere dei modelli architetturali più famosi con l'uso di una semplice linea di codice. Inoltre, ogni rete è accompagnata da una vasta documentazione, comprendente anche i vari tipi di preprocessing utilizzati per allenarla, le dimensioni delle immagini e la struttura della rete stessa.

4.1.4 Scikit-learn

Scikit-learn è una libreria di Python per il Machine Learning, che offre un grande varietà di algoritmi per la classificazione, regressione, clustering, riduzione della dimensionalità e preprocessing. È costruita appoggiandosi su altre librerie famose come NumPy, Matplotlib e Pandas, rendendolo uno strumento potente per la data analysis.

In questo studio, è stata sfruttata la possibilità offerta da Scikit-learn di creare una **matrice di confusione**. Una matrice di confusione (Tabella 4.3) è ampiamente utilizzata per la valutazione delle prestazioni di un modello di classificazione. Si tratta di una sintesi dei risultati della previsione della rete su di un problema di classificazione multi-classe, dove i veri valori sono confrontati con i valori predetti dal modello. La matrice di confusione offre una visualizzazione chiara dell'accuratezza del modello sui valori di test e la sua abilità nell'identificare correttamente o predire le classi obiettivo.

Una matrice di confusione [59] è tipicamente composta da quattro voci:

- **Vero positivo (True Positive, TP)**
  Il modello ha correttamente classificato la malattia. Ciò significa che la foglia presente nel fotogramma è realmente affetta dalla malattia predetta;
- **Vero negativo (True Negative, TN)**
  Il modello ha correttamente escluso le altre malattie. Vale a dire che la foglia presente nel fotogramma non è affetta dalle malattie escluse dalla previsione;
- **Falso positivo (False Positive, FP)**
  Il modello ha erroneamente classificato la malattia. Questo sta a significare che la foglia nel fotogramma non è effettivamente affetta dalla malattia predetta;
- **Falso negativo (False Negative, FN)**
  Il modello ha erroneamente escluso la malattia giusta. Ad esempio quando il modello afferma che la foglia non è affetta da una determinata malattia, ma quest'ultima ne è realmente affetta.

Basandosi sui valori della matrice di confusione, è possibile estrapolare numerose metriche. Nel seguente elaborato, è stata utilizzata per analizzare l'accuratezza in fase di test, cioè quante volte il modello ha classificato correttamente i fotogrammi di test in input. L'accuratezza è calcolabile con la seguente formula:

$$ACC = \frac{TP + TN}{TP + TN + FP + FN} \tag{4.1}$$

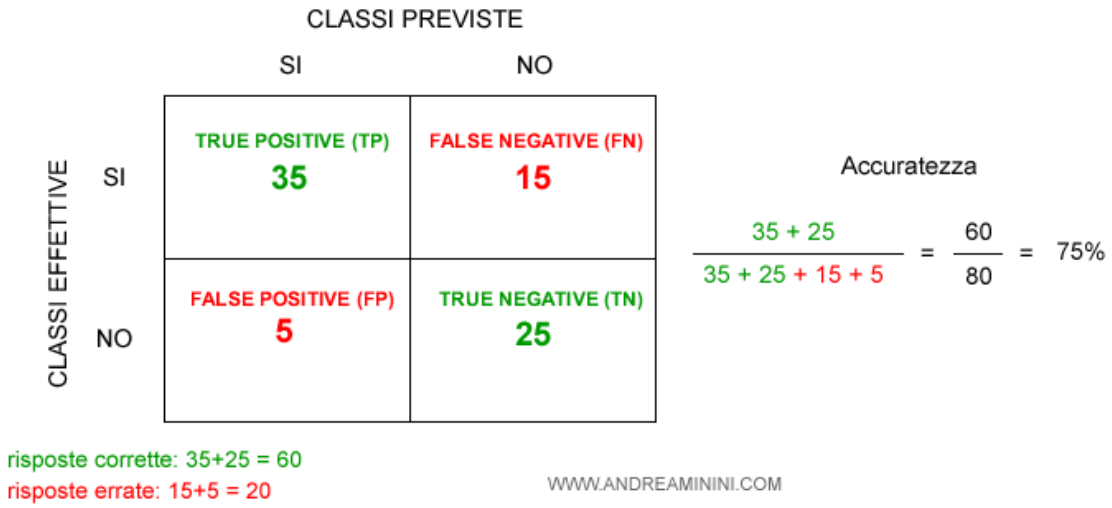

Figura 4.3: Matrice di confusione e calcolo dell'accuratezza [59].

4.2 Architetture dei modelli CNN

4.2.1 MobileNetV2

L'architettura MobileNetV2 è stata sviluppata da Google nel 2018 come miglioramento dell'architettura originale MobileNet. È progettata per applicazioni mobile o embedded, dove l'efficienza computazionale e l'uso di modelli di dimensioni contenute sono cruciali.

MobileNetV2 (Figura 4.4) è caratterizzata da un'architettura composta da **convoluzioni separabili in profondità** [60], per ridurre il numero di parametri e di conseguenza la computazione richiesta, se confrontati con i layer convoluzionali tradizionali. Questo permette la creazione di modelli leggeri, che possono essere eseguiti su dispositivi mobili con risorse di calcolo limitate.

Una delle innovazioni portate da MobileNetV2 è l'uso dei **blocchi residuali invertiti** [61], progettati per preservare la potenza rappresentativa della rete e al contempo ridurre i costi di calcolo. Questi blocchi prendono in input una rappresentazione compressa a bassa dimensione, che viene poi filtrata con una convoluzione separabile in profondità leggera. Questa struttura è appunto invertita rispetto ai blocchi residuali classici, consentendo una maggiore flessibilità nel controllo della complessità del modello.

Un altra caratteristica fondamentale è l'uso dei "**linear bottlenecks**", i quali sono layer lineari inseriti tra i blocchi residuali per ridurre la mole di calcolo. In questo modo, è possibile creare modelli più profondi e con più canali, mantenendo comunque un basso numero di parametri.

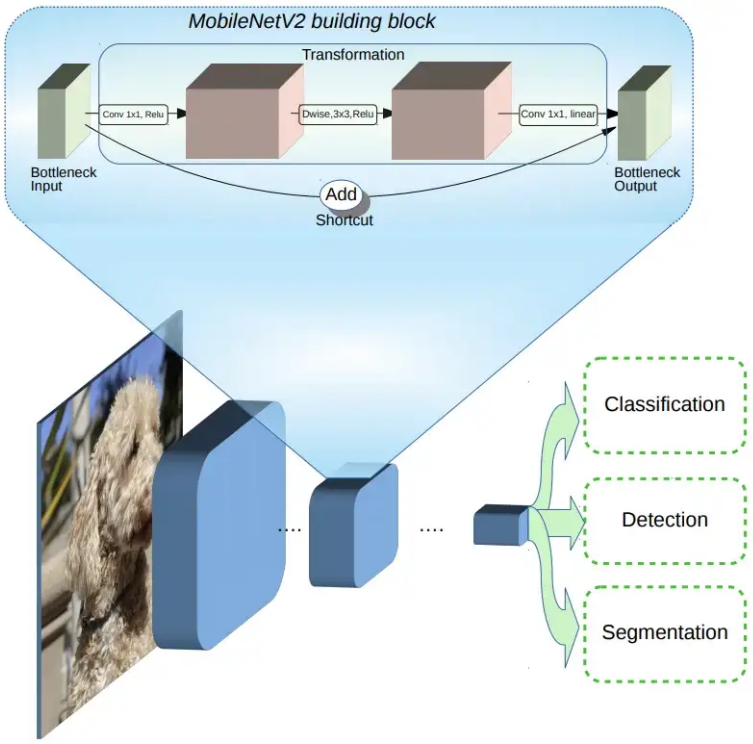

Figura 4.4: Blocco architetturale MobileNetV2 [62].

4.2.2 InceptionV3

InceptionV3 [63] è stata sviluppata da Google e rilasciata nel 2015. Una delle parti più interessanti del modello è la sua architettura, la quale è costruita su un concetto chiamato **moduli Inception**. Questi moduli riducono la dimensionalità attraverso convoluzioni consecutive e consentono alla rete di imparare diverse scale spaziali di feature in parallelo, concatenandole in una rappresentazione finale di feature. In questo modello, i layer convoluzionali più grandi sono stati scomposti in più layer consecutivi di dimensione inferiore per ridurre il costo computazionale (Figura 4.5).

InceptionV3 fa uso di molte altre tecniche per migliorare le sue prestazioni, in particolare utilizza funzioni di attivazione ReLU, regolarizzazione dropout e global average pooling. La funzione di attivazione ReLU consente alla rete di imparare più efficacemente andando a rettificare i valori negativi dell'attivazione, mentre la regolarizzazione dropout aiuta a prevenire il sovradattamento, andando a disattivare casualmente dei neuroni durante l'allenamento. Inoltre, è utilizzato un global average pooling layer che riduce il numero di parametri e la mole di calcolo.

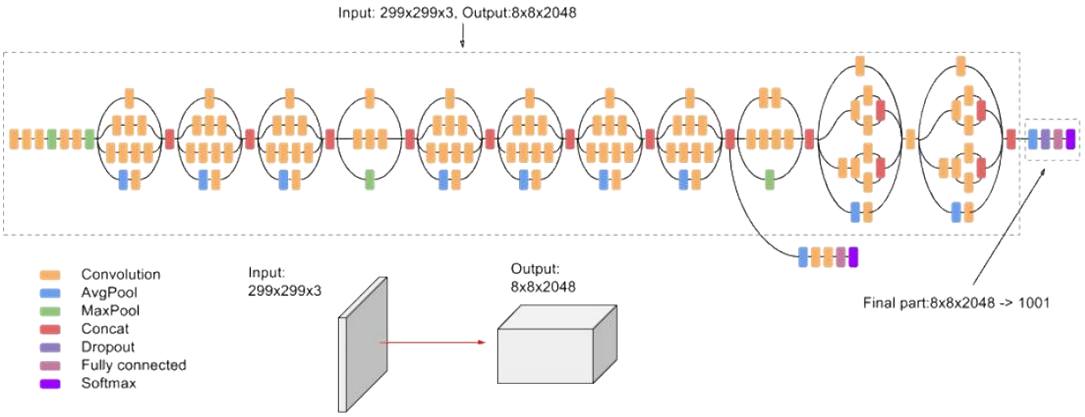

Figura 4.5: Diagramma del modello InceptionV3 [64].

4.2.3 VGG19

VGG19 [65] è stata sviluppata nel 2014 dal Visual Geometry Group (VGG) dell'università di Oxford. Si compone di un'architettura di 19 layer di cui 16 di convoluzione, 3 fully connected layer, 5 max pooling layer ed il layer softmax finale. Usa piccoli filtri 3x3 per estrarre le feature locali dall'immagine di input, inoltre utilizza il max pooling e il padding spaziale per preservare la risoluzione spaziale dell'immagine (Figura 4.6). La funzione di attivazione ReLU è usata per introdurre non linearità, e quindi per facilitare il compito di classificazione al modello, migliorando al contempo il tempo di elaborazione; infatti, i modelli precedenti facevano uso di funzioni tangente o sigmoide, che si sono rivelate inefficienti rispetto alla ReLU.

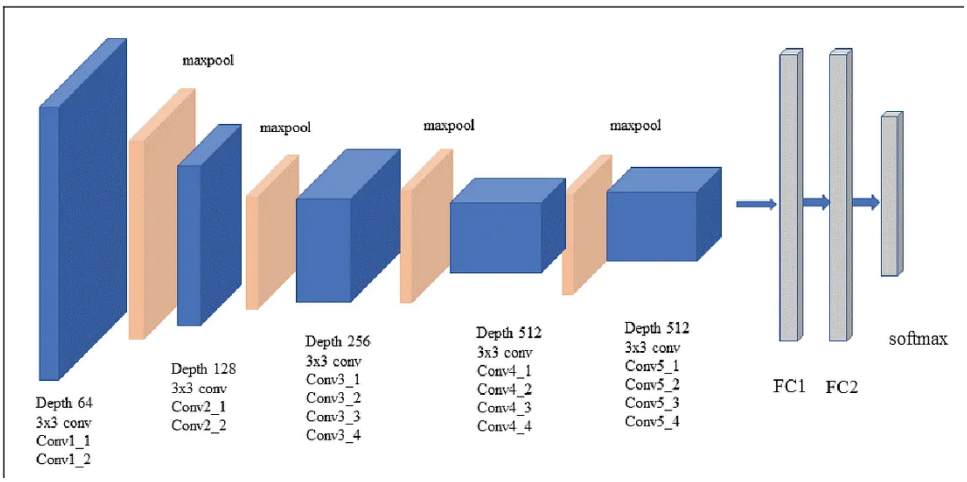

Figura 4.6: Architettura VGG19 [66].

4.2.4 ResNet50

ResNet50 [67] è stata sviluppata dalla Microsoft Research ed introdotta nel 2015. Una delle novità del modello è la sua architettura, progettata per affrontare il problema della scomparsa del gradiente nelle reti neurali. Nelle reti neurali profonde, il segnale del gradiente usato

per aggiornare i pesi può diventare molto piccolo durante la sua propagazione all'indietro attraverso molteplici layer, portando ad una convergenza lenta e prestazioni inadeguate. ResNet50 affronta il problema con l'uso delle shortcut connections, che permettono al segnale del gradiente di aggirare uno o più layer e raggiungerne direttamente altri all'interno della rete, trasformando una rete regolare in una residuale (Figura 4.7).

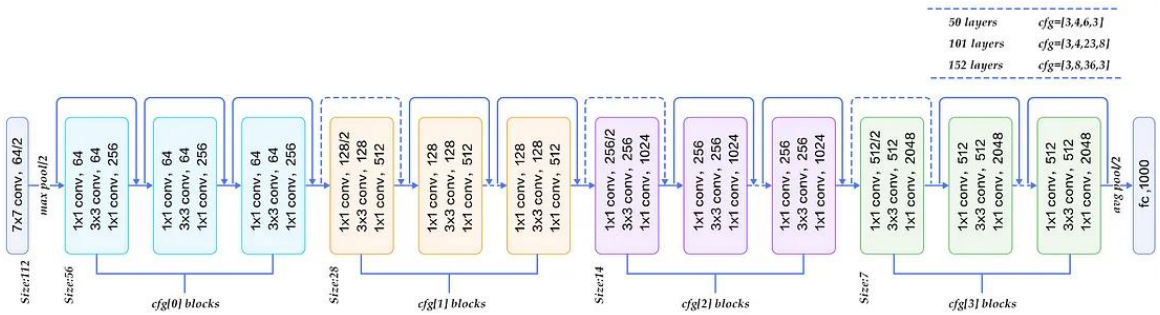

Figura 4.7: Architettura ResNet50 [68].

4.3 Dataset

Dataset PlantVillage

Il primo dataset individuato è stato PlantVillage. PlantVillage è un dataset pubblico creato dal Penn State College of Agricultural Sciences con la collaborazione di altri istituti internazionali. Il dataset è composto da 54,309 immagini divise in 38 classi contenenti 14 tipi diversi di piante come pomodoro, mela e vite, che sono state verificate da esperti di patologie delle piante. Le immagini sono state raccolte da varie fonti, tra cui esperimenti, osservazioni sul campo ed iniziative scientifiche rivolte ai cittadini. Ogni immagine è annotata da una sua etichetta che indica la presenza di una malattia specifica da cui è stata colpita; proprio questo lavoro di etichettatura rende il dataset molto utile per l'allenamento di reti adoperabili per l'identificazione automatica delle malattie.

Da questo dataset sono state utilizzate le immagini delle classi di malattie riguardanti la foglia della piante della vite, in particolare le seguenti malattie: marciume nero, ruggine della foglia e mal dell'esca. Oltre alle immagini delle foglie infette, è stata usata anche la classe contenente le immagini di foglie della vite sane.

Le foglie sono state rimosse dalla pianta, posizionate su di uno sfondo grigio o nero, e fotografate all'aria aperta con una fotocamera digitale (Sony DSC - Rx100/13 20.2 megapixels) in condizioni meteorologiche soleggiate o nuvolose, per rendere la cattura più vicina al contesto reale di acquisizione delle immagini da parte degli agricoltori (Figura 4.8).

Dataset Hermos

Data la mancanza di immagini di foglie della vite infette da Peronospora ed Oidio all'interno del dataset PlantVillage, le ricerche sul web sono state approfondite per trovare ulteriori dataset pubblici. Un dataset trovato è stato Hermos, dal quale abbiamo raccolto solo le immagini di nostro interesse, anche se in un numero limitato; questo include tre classi di immagini di foglie della vite: foglie con peronospora, foglie con oidio e foglie non affette da malattie.

Come si apprende dall'articolo di Ozacar [69], questo dataset si compone di 320 foglie con Oidio, 100 con Peronospora e 116 foglie sane. Le foto sono state scattate con due tipi di fotocamere digitali: Canon EOS 250D e Sony Alpha A5000, in condizioni di luce variabile, condizioni climatiche, sfondi ed angoli diversi nella regione della Manisa.

Dataset PDDB

Il database "Digipathos" contiene un vasto set di immagini, denominato "PDDB", per la diagnosi di malattie delle piante in Brasile [70]. Le immagini sono state acquisite con vari dispositivi, tra cui fotocamere e telefoni cellulari; inoltre, 715 immagini sono state scattate direttamente sul campo, mentre altre 1611 in condizioni controllate. Da questo dataset sono state raccolte altre foto di Peronospora ed Oidio della vite.

Figura 4.8: Esempi di immagini dal dataset PlantVillage. Ciascuna colonna appartiene ad una classe [71].

4.3.1 Preprocessing e segmentazione

Vista la differenza di qualità e quantità tra le immagini raccolte dal dataset PlantVillage e le restanti immagini di Peronospora ed Oidio della vite, si è deciso di intraprendere delle strategie di preprocessing per rendere omogeneo il dataset nella sua interezza, attraverso l'eliminazione dello sfondo di ciascuna foto. Di seguito un esempio di immagine con sfondo rimosso (Figura 4.9):

Figura 4.9: Foglia infetta da marciume nero segmentata.

Per questo compito è stato usato **rembg**, uno strumento apposito per la rimozione dello sfondo delle immagini. Si tratta di una soluzione popolare per l'editing di immagini, ed una delle tecnologie dietro rembg è l'uso delle reti neurali, che consentono allo strumento di rimuovere lo sfondo accuratamente.

La rete neurale usata da rembg è denominata $\boldsymbol{U^2Net}$, una rete preallenata specificatamente per compiti di riconoscimento di oggetti salienti in un'immagine e segmentazione di questi. L'architettura di U2net è stata introdotta nel 2020 da Qin et al. come nuovo approccio nei confronti della segmentazione semantica. $\boldsymbol{U^2Net}$ (Figura 4.10) sta per "U-Net con struttura U-Net nidificata su due livelli", infatti questa è costruita sulla celebre architettura U-Net. "Ha due vantaggi principali: riesce a catturare informazioni contestualizzate da diversi livelli grazie ai campi ricettivi di diverse dimensioni negli U-Block; incrementa la profondità dell'intera architettura senza incrementare particolarmente il costo dell'elaborazione grazie alle operazioni di pooling usate negli U-Block [72]."

U-Net è un'architettura ampiamente utilizzata per il compito di segmentazione d'immagini grazie alla sua abilità di estrapolare feature di alto e basso livello. La struttura si compone di una rete encoder che estrapola le feature di alto livello, e una rete decoder che fonde queste

feature con quelle di basso livello ottenute per generare il risultato finale. Tuttavia, la U-Net è limitata dalla sua abilità di gestire piccoli oggetti. La struttura U-Net nidificata nella U^2Net aiuta a superare questa limitazione aggiungendo un livello di astrazione addizionale alla rete. Questa struttura è costituita tre parti: un encoder a sei stadi, un decoder a cinque stadi e un modulo di fusione della mappa di salienza collegato agli stadi del decoder e all'ultimo stadio dell'encoder.

U^2Net introduce anche un nuove modulo di estrazione delle feature, chiamato "U-Block". Questo modulo è composto da tre parti: un layer convoluzionale di input, una struttura encoder-decoder come nella rete U-Net ed una connessione residuale che fonde le feature locali e le feature multi-scala. Questi "U-Block" aiutano a migliorare l'efficienza della rete riducendo il numero di parametri e l'elaborazione richiesta.

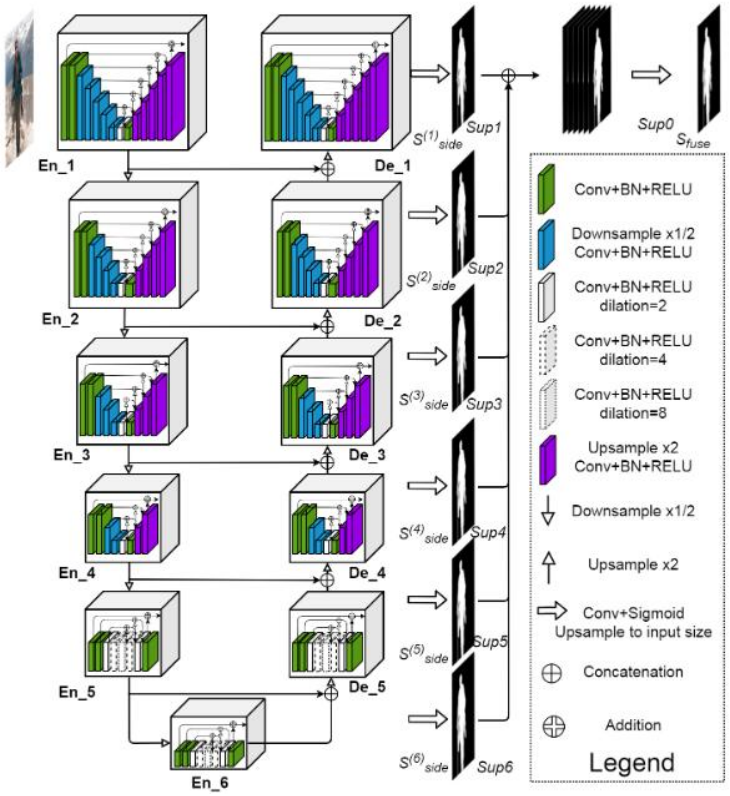

Figura 4.10: Struttura gerarchica U^2Net [72].

4.3.2 Preparazione del dataset

Una volta terminata la fase di raccolta delle immagini, si passa alla fase di preparazione del dataset per l'allenamento della rete.

Dataset split

Durante la fase di raccolta le immagini sono state suddivise in sei cartelle, ognuna delle quali contenente le immagini delle foglie di ogni malattia considerata ed una cartella contenente le

immagini delle foglie non infette. In totale è stato collezionato il seguente numero di immagini per ogni classe:

- 500 di marciume nero;
- 500 di mal dell'esca;
- 500 di ruggine della foglia;
- 432 di Peronospora;
- 294 di Oidio;
- 423 di foglie in salute.

In seguito all'applicazione delle tecniche di segmentazione delle immagini, è stata usata una libreria apposita per lo split dei dati in tre cartelle: **allenamento**, **validazione** e **test**.

Il nome della libreria in questione è **split-folders**, creata appositamente per la suddivisione di dataset e perfetta per lo stato di suddivisione delle nostre cartelle. La cartella data in input alla libreria deve avere il formato mostrato nella Figura 4.11:

```
input/
    class1/
        img1.jpg
        img2.jpg
        ...
    class2/
        imgWhatever.jpg
        ...
    ...
```

Figura 4.11: Input split-folders [73].

Affinché si abbiano in output tre cartelle formate come nella Figura 4.12:

```
output/
    train/
        class1/
            img1.jpg
            ...
        class2/
            imga.jpg
            ...
    val/
        class1/
            img2.jpg
            ...
        class2/
            imgb.jpg
            ...
    test/
        class1/
            img3.jpg
            ...
        class2/
            imgc.jpg
            ...
```

Figura 4.12: Output split-folders [73].

Attraverso il seguente codice è stata installata la libreria all'interno del nostro ambiente di sviluppo Colab:

```
pip install split-folders
```

È stata importata la libreria nell'ambiente ed è stata utilizzata la funzione per lo split:

```
import splitfolders

input_folder = '/content/ImagesGrapeSegmented'
splitfolders.ratio(input_folder, output = '/content/gdrive/MyDrive/DatasetSplit',
                   seed = 42, ratio = (.8, .1, .1),
                   group_prefix = None)
```

È possibile modificare diversi parametri della funzione, cioè:

- Input: cartella di input;
- Output: percorso della cartella di output;
- Seed: impostiamo un seme per avere sempre lo stesso split di immagini ad ogni esecuzione della funzione (nel caso presentato il seme è "42");
- Ratio: il rapporto per lo split, in questo caso si è optato per una suddivisione 80, 10, 10, cioè 80% delle immagini per l'allenamento, 10% per la validazione e 10% per il testing.

Per comodità, è stata sfruttata l'integrazione all'interno di Google Colab di Google Drive, il quale è stato usato come percorso di output della funzione di split; questa strategia permette di caricare all'interno del runtime di Colab la cartella di split ad ogni esecuzione dell'ambiente, dato che quando scade la sessione i dati caricati nel runtime vengono eliminati.

```
from google.colab import drive
drive.mount('/content/gdrive')
```

In seguito, sono state create delle variabili contenenti i valori degli iperparametri da usare per l'allenamento e rendere la modifica di questi per le varie prove più flessibile.

Di seguito riporto il codice per il caricamento dal drive delle varie immagini, suddivise in allenamento, validazione e testing, all'interno delle corrispettive variabili. Come si può notare, viene usata la funzione di Keras "**image dataset from directory**", alla quale passiamo il percorso della cartella da cui reperire i dati; scegliamo se mescolare le immagini per rendere l'allenamento più omogeneo possibile; la dimensione della batch (impostata a 32) e la dimensione delle immagini che verranno ridimensionate automaticamente dalla funzione, in base ai valori da noi impostati. Si noti che nel testing scegliamo di non mescolare le immagini; questo perché utilizziamo la matrice di confusione di Scikit-learn, la quale richiede i dati in ordine di classe per rappresentarli all'interno della matrice.

```
train_path = '/content/gdrive/MyDrive/DatasetSplit/train'
valid_path = '/content/gdrive/MyDrive/DatasetSplit/val'
test_path = '/content/gdrive/MyDrive/DatasetSplit/test'

train_dataset = tf.keras.utils.image_dataset_from_directory(train_path,
                                                            shuffle=True,
                                                            batch_size=BATCH_SIZE,
                                                            image_size=IMAGE_SIZE)

validation_dataset = tf.keras.utils.image_dataset_from_directory(valid_path,
                                                                 shuffle=True,
                                                                 batch_size=
                                                                     BATCH_SIZE,
                                                                 image_size=
                                                                     IMAGE_SIZE)

test_dataset = tf.keras.utils.image_dataset_from_directory(test_path,
                                                           shuffle=False,
                                                           batch_size=BATCH_SIZE,
                                                           image_size=IMAGE_SIZE)

class_names = train_dataset.class_names
```

4.3.3 Data augmentation

Dato l'esiguo numero di immagini reperite tra dataset ed internet, è stata utilizzata una tecnica chiamata **Data augmentation**. Questa tecnica è usata per incrementare artificialmente la dimensione di un dataset, andando a generare nuovi dati da dati esistenti. L'idea è quella di applicare trasformazioni minori come rotazione dell'immagine, ridimensionamento, capovolgimento o aggiunta di rumore. Il risultato finale è un dataset più grande e diversificato per aumentare l'accuratezza del modello.

Ci sono diversi benefici nell'utilizzo della data augmentation. Il primo è che riesce a prevenire il **sovradattamento dei dati**, il quale è un problema comune nel Deep Learning, dove un modello impara i dati di allenamento troppo meticolosamente e non riesce a generalizzare

bene sui nuovi dati di test. Attraverso l'incremento artificiale della dimensione del dataset con la data augmentation, il modello è esposto a campioni di dati sempre diversi e risulta meno propenso al sovradattamento dei dati di allenamento.

In secondo luogo, la data augmentation riesce anche ad irrobustire il modello. Ad esempio, andando a ruotare, capovolgere o ridimensionare le immagini, i nuovi dati creati per il modello riflettono la variabilità dei dati che riscontriamo nel mondo reale. Tutto ciò aiuta il modello a gestire variazioni nei dati e a comportarsi bene anche con esempi nuovi che differiscono dai dati di allenamento.

Infine, data l'indole del Deep Learning di aver bisogno di una grande molte di dati, la data augmentation aiuta col numero di dati quando questo è limitato, rendendo l'allenamento più efficiente.

Nel seguente elaborato, sono stati utilizzati sei tipi di data augmentation: rotazione, capovolgimento, traslazione, zoom, luminosità e contrasto (Figura 4.13). Sono stati scelti in base ai diversi contesti in cui è possibile scattare una foto in un campo agricolo; ad esempio, il fattore della luminosità è fondamentale in un campo, a causa dei fattori meteorologici variabili che possono influenzare la luminosità di una foto; oppure le rotazioni, traslazioni o zoom per rappresentare le diverse prospettive da cui è possibile scattare una foto. Come si vede dal seguente codice, si utilizza un range di valori, quindi "l'intensità" dello specifico layer di data augmentation può variare nell'intervallo specificato.

```
data_augmentation = tf.keras.Sequential([
  tf.keras.layers.RandomFlip('horizontal_and_vertical'),
  tf.keras.layers.RandomRotation(0.2),
  tf.keras.layers.RandomTranslation(height_factor=0.2, width_factor=0.2),
  tf.keras.layers.RandomZoom(0.2),
  tf.keras.layers.RandomBrightness(0.3),
  tf.keras.layers.RandomContrast(0.2),
])
```

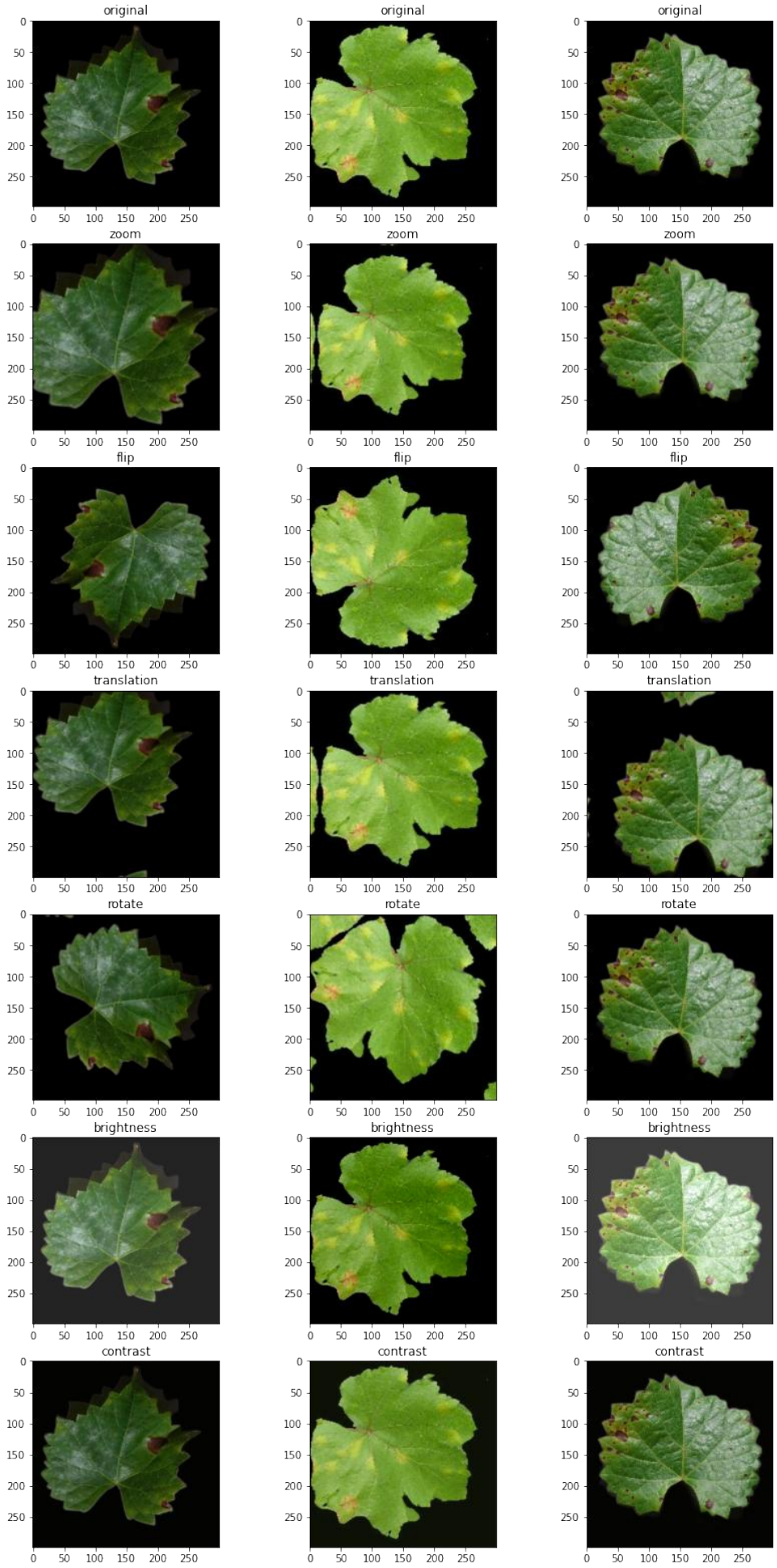

Figura 4.13: Data augmentation applicata a tre foglie diverse.

Il concetto fondamentale su cui si basa il funzionamento della data augmentation nelle reti neurali convoluzionali è l'invarianza di traslazione. Con l'utilizzo dei layer per la data augmentation di Keras, "l'augmentation" dei dati avviene in tempo reale durante la fase di addestramento della rete; inoltre, ha senso chiarire cosa si intende con il termine "augmentation". In primo luogo, prima dell'addestramento bisogna scegliere il numero di **epoche**, cioè il numero di volte in cui il modello analizzerà l'intero insieme dei dati di allenamento, il quale è suddiviso nel nostro caso in batch di 32 immagini ciascuno. La data augmentation non aumenta il numero di immagini da far analizzare al modello durante ogni epoca, bensì viene utilizzata una diversa trasformazione di queste durante le varie iterazioni. Utilizzando questa strategia, evitiamo che il nostro modello sia allenato su immagini sempre uguali, il che lo porta ad "imparare a memoria" le feature analizzate, con conseguenti scarse prestazioni di generalizzazione.

4.4 Configurazione modello e sperimentazione

Dopo aver preparato i dati di studio, è il momento di descrivere i modelli e le tecniche di allenamento utilizzate in questo elaborato.

4.4.1 Transfer learning

Nonostante l'applicazione della data augmentation, i dati a disposizione non erano sufficienti per allenare un modello da noi creato, quindi la scelta è ricaduta sull'utilizzo di **reti neurali convoluzionali preallenate**, cioè reti precedentemente allenate su un grande dataset, tipicamente ImageNet. Il dataset ImageNet contiene 14,197,122 immagini etichettate in più di 20.000 categorie, ed un suo sottoinsieme è utilizzato dal 2010 nel "ImageNet Large Scale Visual Recognition Challenge", una sfida tenuta per la ricerca di tecniche di visione artificiale e Deep Learning. Questa sfida ha cadenza annuale ed è diventata un campo di prova per la valutazione degli algoritmi di classificazione di immagini ed individuazione di oggetti.

Una rete preallenata può essere usata direttamente allenandola sul proprio dataset, oppure è possibile usare il transfer learning per personalizzare il modello allo specifico compito da realizzare. Dato il numero limitato di dati, non potremmo allenare reti così grandi e complesse sul nostro dataset, perciò è stata utilizzata la tecnica del **transfer learning**.

Nel Transfer Learning, un modello pre-addestrato su di uno specifico compito viene utilizzato come punto di partenza per la creazione di un secondo modello, il quale è specificatamente progettato per risolvere un task differente dall'originale, ma correlato. Ad esempio, un modello di rete neurale profonda allenata sul dataset ImageNet, può essere usato come estrattore di feature per un nuovo dataset.

Esistono due strategie di transfer learning:

- **Feature Extraction**
  Si utilizzano le feature imparate dalla rete preallenata per estrapolare feature significative da un nuovo dataset. In questa tecnica, si congelano tutti i layer della rete neurale scelta, cosicché questi non possano essere allenati e quindi modificati, il che porterebbe ad una perdita delle conoscenze del dataset precedentemente imparato dalla rete. Dopodiché, si sostituiscono gli ultimi layer della rete preallenata, adattandoli al nuovo compito di classificazione. Solitamente, l'ultimo layer è progettato per dare in output 1000 classi, dato che è stato allenato su ImageNet; nel nostro caso, questo sarà sostituito da un dense layer con 6 neuroni, uno per ogni classe da classificare, abbinato ad una funzione d'attivazione Softmax, ideale per le classificazioni multi-classe, come illustrato nel capitolo tre. Il modello è poi allenato, ma solo i layer aggiunti saranno allenabili, dato che l'intera rete è stata congelata; ciò consente di riproporre le feature map imparate dal dataset ImageNet, ma ottenendo in output la classificazione delle nostre sei classi;

- **Fine-Tuning**
  Nel Fine-Tuning, si congela solo parte della rete preallenata e si allena una piccola parte di questa, solitamente i layer più profondi. Quindi, verranno allenati al contempo sia i layer non congelati, che il nuovo layer classificatore da noi aggiunto alla fine della rete. Questo ci permette di "riprogrammare" le rappresentazioni delle feature di più alto livello del modello base, per renderle più pertinenti al nostro specifico compito. È opportuno sottolineare che in questa strategia bisogna utilizzare un learning rate più basso del solito, affinché le feature imparate dai layer della rete non congelati non vengano stravolte.

4.4.2 Reti preallenate

Per utilizzare le reti preallenate, ci siamo affidati alla libreria Keras. Keras fornisce l'accesso ad una vasta quantità di modelli preallenati sul dataset ImageNet, di seguito inserisco un'immagine che mostra una parte di quelli utilizzabili:

Available models

Model	Size (MB)	Top-1 Accuracy	Top-5 Accuracy	Parameters	Depth	Time (ms) per inference step (CPU)	Time (ms) per inference step (GPU)
Xception	88	79.0%	94.5%	22.9M	81	109.4	8.1
VGG16	528	71.3%	90.1%	138.4M	16	69.5	4.2
VGG19	549	71.3%	90.0%	143.7M	19	84.8	4.4
ResNet50	98	74.9%	92.1%	25.6M	107	58.2	4.6
ResNet50V2	98	76.0%	93.0%	25.6M	103	45.6	4.4
ResNet101	171	76.4%	92.8%	44.7M	209	89.6	5.2
ResNet101V2	171	77.2%	93.8%	44.7M	205	72.7	5.4
ResNet152	232	76.6%	93.1%	60.4M	311	127.4	6.5
ResNet152V2	232	78.0%	94.2%	60.4M	307	107.5	6.6
InceptionV3	92	77.9%	93.7%	23.9M	189	42.2	6.9
InceptionResNetV2	215	80.3%	95.3%	55.9M	449	130.2	10.0
MobileNet	16	70.4%	89.5%	4.3M	55	22.6	3.4
MobileNetV2	14	71.3%	90.1%	3.5M	105	25.9	3.8
DenseNet121	33	75.0%	92.3%	8.1M	242	77.1	5.4
DenseNet169	57	76.2%	93.2%	14.3M	338	96.4	6.3
DenseNet201	80	77.3%	93.6%	20.2M	402	127.2	6.7
NASNetMobile	23	74.4%	91.9%	5.3M	389	27.0	6.7
NASNetLarge	343	82.5%	96.0%	88.9M	533	344.5	20.0

Figura 4.14: Parte dei modelli offerti dalla libreria Keras [58].

Per ogni rete sono descritte delle caratteristiche, come lo spazio occupato dalla rete in MB, il numero di parametri e la sua profondità. Cliccando su ogni rete, si apre una pagina con le informazioni riguardanti la rete specifica. Ad esempio, è possibile reperire la funzione da utilizzare per scaricare la rete e i relativi argomenti. Tra gli argomenti possiamo scegliere:

- **include_top**: valore booleano per scegliere se utilizzare il fully connected layer alla fine della rete;
- **weights**: possiamo scegliere se avere una rete non allenata, 'imagenet' per utilizzare la rete allenata sul dataset ImageNet o possiamo inserire il percorso verso il file contente i pesi da caricare;
- **input_tensor**: tensore opzionale di Keras;

- **input_shape**: tupla opzionale per specificare le dimensioni delle immagini in input alla rete;
- **pooling**: si può scegliere tra average pooling o max pooling come output per l'ultimo layer convoluzionale;
- **classes**: da specificare solo se include_top = True e se non sono specificati dei pesi per scegliere il numero di classi da classificare;
- **classifier_activation**: la funzione di attivazione da usare nel layer di classificazione.

Inoltre, nella pagina di descrizione di una rete, viene specificata la dimensione delle immagini date in input durante il suo allenamento originale e la tipologia di preprocessing specifica della rete applicata ai dati. Di seguito, come esempio, lascio la linea di codice per la funzione di preprocessing delle rete InceptionV3, che si limita a ridimensionare i pixel di input tra -1 e 1. Dei dati in input che hanno un grande intervallo di valori, possono rendere instabile il processo di apprendimento della rete, quindi il loro ridimensionamento è necessario per affrontare questa problematica e per rendere più celere la fase di allenamento.

```
preprocess_input = tf.keras.applications.inception_v3.preprocess_input
```

Per scaricare la rete di nostra scelta e quindi utilizzarla, è stata usata la funzione apposita con i relativi parametri, come nell'esempio in cui mostro la funzione di InceptionV3 con i pesi di ImageNet e senza il layer classificatore finale originale.

```
IMG_SHAPE = IMAGE_SIZE + (3,)
base_model = tf.keras.applications.inception_v3.InceptionV3(include_top=False,
    weights="imagenet", input_shape=IMG_SHAPE)
```

4.4.3 Costruzione della rete

A questo punto è sorta l'esigenza di scegliere quale tecnica utilizzare tra **Feature Extraction** e **Fine-Tuning** per allenare la rete.

Per la **Feature Extraction** è stato utilizzato il seguente codice per congelare l'intera rete:

```
base_model.trainable = False
```

Mentre per il **Fine-Tuning** è stata congelata solo parte della rete, nel nostro caso il 90%, percentuale scelta dopo varie sperimentazioni condotte, con questo codice:

```
layer_num = len(base_model.layers)

for layer in base_model.layers[:int(layer_num * 0.9)]:
    layer.trainable = False

for layer in base_model.layers[int(layer_num * 0.9):]:
    layer.trainable = True
```

Dato che per i nostri scopi avevamo bisogno di sostituire il layer classificatore per entrambe le tecniche, sono stati aggiunti alla nostra rete due layer allenabili: un global average pooling layer e un dense layer con 6 neuroni, uno per ogni classe da classificare, con annessa funzione d'attivazione Softmax.

```
global_average_layer = tf.keras.layers.GlobalAveragePooling2D()
prediction_layer = tf.keras.layers.Dense(CLASSES, activation='softmax')
```

Fatto ciò, la rete è stata progressivamente costruita, aggiungendo i layer di data augmentation, la funzione di preprocessing e i due layer sopracitati.

```
inputs = tf.keras.Input(shape=(299, 299, 3))
x = data_augmentation(inputs)
x = preprocess_input(x)
x = base_model(x, training=False)
x = global_average_layer(x)
outputs = prediction_layer(x)
model = tf.keras.Model(inputs, outputs)
```

4.4.4 Allenamento della rete

Dopo la costruzione della rete, il modello è stato compilato scegliendo diversi parametri tra cui: **l'ottimizzatore** da utilizzare associato al learning rate scelto, la **funzione di perdita** (loss function in inglese) e la **metrica** di nostro interesse.

Le **funzioni di perdita** sono rappresentazioni matematiche usate per quantificare l'errore tra l'output predetto dal modello ed l'output corretto. Per modelli di classificazione usiamo la funzione di **Cross-entropia**, tre sono quelle più importanti:

- **Binary Crossentropy**: utilizzata quando l'output di un modello è una probabilità tra 0 e 1 e quindi si tratta di classificazione binaria;
- **Categorical Crossentropy**: utilizzata quando l'output di un modello è una distribuzione di probabilità su più classi e quindi usata per la classificazione multi-classe; i valori sono visualizzati con una rappresentazione **one-hot encoded**, cioè ogni classe è rappresentata da un vettore con tutti zero, ad eccezione di un elemento indicato con 1 che indica la classe;
- **Sparse Categorical Crossentropy**: anche questa utilizzata per la classificazione multi-classe, differisce dalla precedente per la rappresentazione; in questa rappresentazione infatti, ogni classe è rappresentata da un singolo valore, andando così ad efficientare l'uso della memoria.

Nel nostro caso abbiamo sfruttato la Sparse Categorical Crossentropy come funzione di perdita scelta:

```
model.compile(optimizer=tf.keras.optimizers.Adam(learning_rate=LEARNING_RATE),
              loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=False
                  ),
              metrics=['accuracy'])
```

Compilato il modello, lo abbiamo allenato attraverso l'uso della funzione fit alla quale abbiamo passato come parametri il dataset contenente i dati per l'allenamento, il numero di epoche e il dataset con i dati per la validazione:

```
history = model.fit(train_dataset,
                    epochs=EPOCHS,
                    validation_data=validation_dataset)
```

Finito l'allenamento della nostra rete, abbiamo valutato l'accuratezza sul dataset di test attraverso la funzione di valutazione apposita, la quale dà come risultato una percentuale di accuratezza del modello sui dati di test:

```
loss, acc = model.evaluate(test_dataset, verbose = 2)
print('Model␣accuracy:␣{:5.2f}%'.format(100 * acc))
```

Inoltre, è stata anche sfruttata la matrice di confusione per la valutazione del modello, descritta nella sottosezione dedicata a Scikit-learn, attraverso l'utilizzo del codice da loro fornito che si può trovare sul loro sito web, e da me appositamente modificato per questo caso di studio. Di seguito il codice:

```
predictions = model.predict(test_dataset, verbose = 0)

classes = np.concatenate([y for x, y in test_dataset], axis=0)

%matplotlib inline
from sklearn.metrics import confusion_matrix
import itertools
import matplotlib.pyplot as plt

cm = confusion_matrix(y_true=classes, y_pred=np.argmax(predictions, axis=-1))
def plot_confusion_matrix(cm, classes,
                          normalize=False,
                          title='Confusion␣matrix',
                          cmap=plt.cm.Blues):
    """
    This function prints and plots the confusion matrix.
    Normalization can be applied by setting `normalize=True`.
    """
    plt.imshow(cm, interpolation='nearest', cmap=cmap)
    plt.title(title)
    plt.colorbar()
    tick_marks = np.arange(len(classes))
    plt.xticks(tick_marks, classes, rotation=45)
    plt.yticks(tick_marks, classes)

    if normalize:
        cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]
        print("Normalized␣confusion␣matrix")
    else:
        print('Confusion␣matrix,␣without␣normalization')

    print(cm)

    thresh = cm.max() / 2.
    for i, j in itertools.product(range(cm.shape[0]), range(cm.shape[1])):
        plt.text(j, i, cm[i, j],
            horizontalalignment="center",
            color="white" if cm[i, j] > thresh else "black")

    plt.tight_layout()
    plt.ylabel('True␣label')
    plt.xlabel('Predicted␣label')

cm_plot_labels = ['blackRot', 'esca','leafBlight', 'healthy','downy','powdery']
plot_confusion_matrix(cm = cm, classes = cm_plot_labels, title = 'Confusion␣Matrix
    ')
```

4.4.5 Risultati

Come detto, le reti neurali convoluzionali utilizzate sono state: **MobileNetV2**, **InceptionV3**, **VGG19** e **ResNet50**; queste sono anche state analizzate approfonditamente nella sezione delle architetture dei modelli CNN.

Per ognuna di queste reti sono state condotte svariate sperimentazioni, sia con tecniche di Feature Extraction che di Fine-Tuning; inoltre, sono stati saggiati diversi tipi di iperparametri per trovare i valori ottimali, a partire da diversi learning rate, varie prove per gli ottimizzatori e un numero variabile di epoche.

Dal punto di vista degli ottimizzatori, sono stati testati tre tipi: **Adam**, **RMSProp** e **SGD**. Tra questi il migliore in ogni esperimneto si è rivelato **Adam**, ottenendo sempre valori di accuratezza maggiori rispetto agli altri ottimizzatori.

Per ogni rete neurale, è stata utilizzata la dimensione delle immagini in input originalmente applicata per l'allenamento sul dataset ImageNet; questa è stata reperita dalla pagina di descrizione di Keras specifica per ogni rete. Ad esempio, per MobileNetV2 la tupla utilizzata per le dimensioni è stata **(160, 160, 3)**, dove l'ultimo valore rappresenta il numero di canali di colore, in questo caso tre canali perché si tratta di immagini RGB. Parlando degli altri modelli, per InceptionV3 è stata usata una dimensione delle immagini di (299, 299, 3), per VGG19 di (224, 224, 3) ed infine per ResNet50 di (224, 224, 3).

Come risultato finale delle sperimentazioni, le reti che hanno ottenuto i risultati migliori sono state **MobileNetV2** ed **InceptionV3**, utilizzando due tipologie differenti di allenamento.

MobileNetV2

Per MobileNetV2, il risultato migliore è stato ottenuto con la tecnica di Feature Extraction. Prima di tutto è stata applicata la funzione di preprocessing della rete sui dati del dataset, la quale ridimensiona i pixel in input in un intervallo tra -1 ed 1, con il seguente codice:

```
preprocess_input = tf.keras.applications.mobilenet_v2.preprocess_input
```

Dopodiché, è stata scaricata la rete coi pesi di ImageNet; il classificatore finale è stato sostituito con un global average pooling layer e un dense layer con 6 neuroni, con funzione di attivazione Softmax. L'intera rete è stata congelata prima dell'allenamento, ma non i layer appena aggiunti, i quali sono stato gli unici ad essere allenati. Come detto, l'ottimizzatore usato è stato **Adam** ed il learning rate applicato è stato di **0.0001** attraverso **80 epoche**.

La Figura 4.15 mostra la matrice di confusione prodotta dalla rete, contenente le previsioni del modello in fase di test. Come si può notare, otto immagini del mal dell'esca sono state predette erroneamente come immagini di oidio dal modello, e anche cinque immagini di oidio sono state scambiate per immagini del mal dell'esca, facendoci intendere che il modello non riesce a distinguere al meglio queste due malattie.

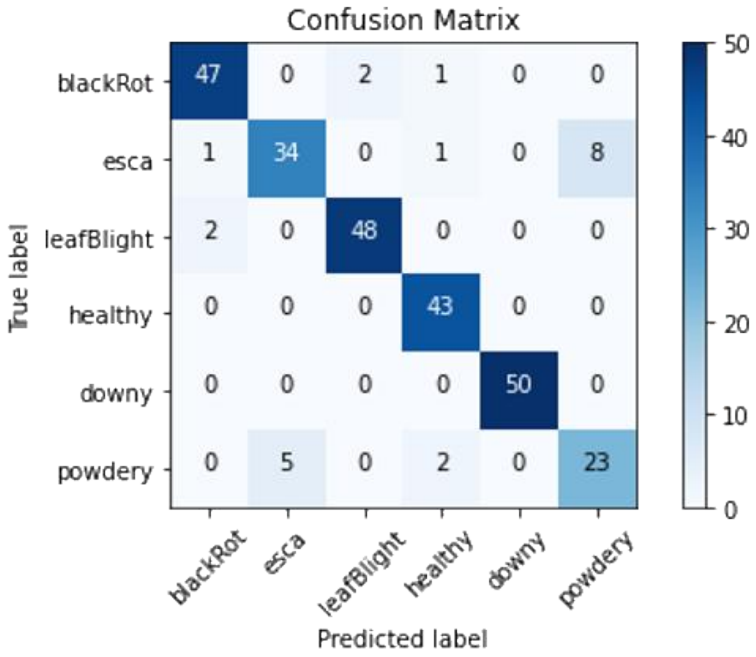

Figura 4.15: Matrice di confusione MobileNetV2.

InceptionV3

Per InceptionV3, il risultato migliore è stato ottenuto con la tecnica di Fine-Tuning. Prima di tutto è stata applicata la funzione di preprocessing della rete sui dati del dataset, la quale ridimensiona i pixel in input in un intervallo tra -1 ed 1, con il seguente codice:

```
preprocess_input = tf.keras.applications.inception_v3.preprocess_input
```

In seguito, allo stesso modo del modello MobileNetV2, è stata scaricata la rete coi pesi di ImageNet; il classificatore finale è stato sostituito con un global average pooling layer e un dense layer con 6 neuroni, con annessa funzione di attivazione Softmax. Prima di aggiungere questi ultimi due layer, per mettere in pratica la tecnica del Fine-Tuning, esattamente il 90% della rete è stata congelata prima dell'allenamento (codice nella sottosezione sulla costruzione della rete), andando così ad allenare i layer più profondi della rete per riadattare i pesi da una generica feature map, a delle feature associate specificatamente col nostro dataset, in contemporanea al nuovo classificatore inserito appositamente per il nostro compito.

Dato che stiamo riadattando parte dei pesi della rete, come già citato, necessitiamo di un learning rate più basso del solito; questo perché altrimenti la magnitudine degli aggiornamenti del gradiente sarebbe troppo grande ed il nostro modello preallenato dimenticherebbe cosa ha imparato in precedenza. Come detto, l'ottimizzatore che ha ottenuto risultati migliori è stato **Adam** ed il learning rate applicato è stato più basso rispetto a quello usato per la Feature Extraction, precisamente è stato usato un learning rate dello **0.00001** attraverso **50 epoche**.

La Figura 4.16 riporta la matrice di confusione prodotta dalla rete, contenente le previsioni del modello in fase di test. Anche in questo caso (come per MobileNetV2), è possibile notare

la difficoltà della rete nella distinzione tra foglie infette da mal dell'esca e da oidio.

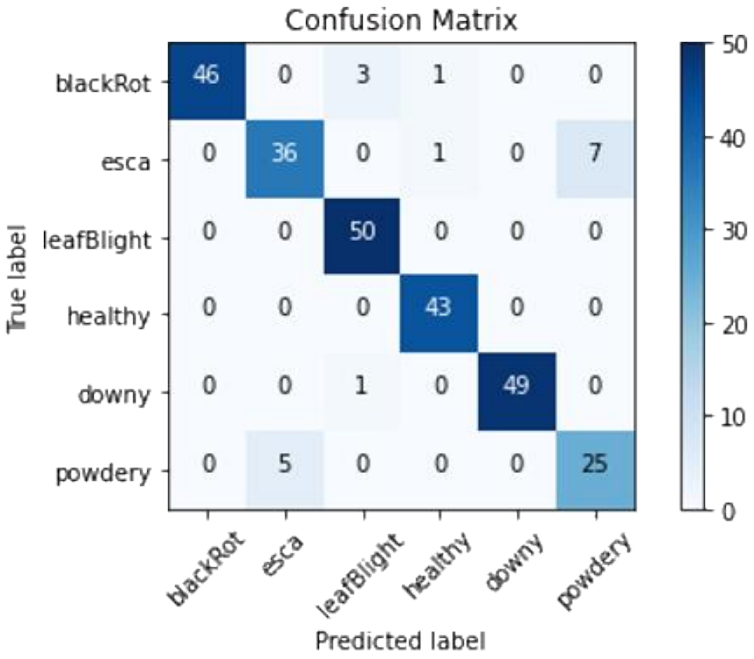

Figura 4.16: Matrice di confusione InceptionV3.

In generale, dai risultati si deduce che tra le quattro reti utilizzate in questo elaborato, quelle che hanno prodotto risultati migliori sono state quelle con un numero di parametri minore rispetto alle altre due. Di seguito inserisco una Tabella riassuntiva 4.1 dei valori delle due reti:

Modello	Dimensione (MB)	Parametri	Profondità	Epoche	Learning Rate	Modello base congelato	Ottimizzatore	Dimensione immagini	Accuratezza test
MobileNetV2	14	3.5M	105	80	0.0001	Tutto	Adam	(160, 160)	91.76%
InceptionV3	92	23.9M	189	50	0.00001	90%	Adam	(299, 299)	93.26%

Tabella 4.1: Tabella riassuntiva dei valori delle reti migliori

Capitolo 5

Conclusioni

L'obiettivo di questo percorso di ricerca è stato quello di automatizzare il processo di rilevamento di alcune delle più note malattie della vite mediante l'utilizzo di noti modelli di Deep Learning. Per raggiungere lo scopo prefissato, è stato costruito un dataset ad hoc con il maggior numero di immagini di foglie della vite infette.

Mediante il Transfer Learning, è stato possibile addestrare le reti ed ottenere risultati preliminari incoraggianti, con un'accuratezza pari al 91.76% per MobileNetV2 e al 93.26% per InceptionV3.

Le prestazioni raggiunte fanno presagire che le tecnologie utilizzate possono produrre risultati promettenti, valutando le sue applicazioni anche in altri contesti strettamente correlati all'agricoltura di precisione. Ad esempio, si potrebbe esaminare l'intensità della malattia che ha infettato una foglia attraverso la segmentazione specifica della regione infetta e/o sviluppare un sistema per rilevare l'insorgenza di una malattia prima che questa comporti un danneggiamento cospicuo.

Attualmente, la problematica principale è strettamente correlata all'esigenza di creare e/o rendere disponibili banche dati con un numero di immagini sufficienti; difatti, il numero di fotogrammi raccolti è di vitale importanza per l'addestramento del modello di Deep Learning. Inoltre, anche il contesto di acquisizione risulta una questione fondamentale, considerando i risultati ottenuti in letteratura utilizzando immagini catturate in un ambiente controllato e scattate direttamente sul campo (le quali trovano un riscontro diretto con la variabilità delle condizioni del mondo reale). Per risolvere le questioni sopracitate, si potrebbe instaurare una collaborazione diretta con aziende agricole del territorio, reperendo così campioni direttamente sul campo.

Bibliografia

[1] Mohamed I Alshelmani et al.
"Nontraditional feedstuffs as an alternative in poultry feed".
In: *Advances in Poultry Nutrition Research*. IntechOpen, 2021.

[2] MP Rico-Fernández et al.
"A contextualized approach for segmentation of foliage in different crop species".
In: *Computers and Electronics in Agriculture* 156 (2019), pp. 378–386.

[3] VG Dhanya et al.
"Deep learning based computer vision approaches for smart agricultural applications".
In: *Artificial Intelligence in Agriculture* (2022).

[4] Nicoleta Tantalaki, Stavros Souravlas e Manos Roumeliotis. "Data-driven decision making in precision agriculture: the rise of big data in agricultural systems".
In: *Journal of agricultural & food information* 20.4 (2019), pp. 344–380.

[5] Yonis Gulzar et al. "A convolution neural network-based seed classification system".
In: *Symmetry* 12.12 (2020), p. 2018.

[6] Stan Matthews et al.
"Evaluation of seed quality: from physiology to international standardization".
In: *Seed Science Research* 22.S1 (2012), S69–S73.

[7] F Lahoche et al. "An innovative approach based on neural networks for predicting soil component variability".
In: *Proceedings of the 6th International Conference on Precision Agriculture and Other Precision Resources Management, Minneapolis, MN, USA*. 2002, pp. 14–17.

[8] Kristine M Larson et al.
"Use of GPS receivers as a soil moisture network for water cycle studies".
In: *Geophysical Research Letters* 35.24 (2008).

[9] Khadijeh Alibabaei, Pedro D Gaspar e Tânia M Lima. "Crop yield estimation using deep learning based on climate big data and irrigation scheduling".
In: *Energies* 14.11 (2021), p. 3004.

[10] David Rolnick et al. "Tackling climate change with machine learning".
In: *ACM Computing Surveys (CSUR)* 55.2 (2022), pp. 1–96.

[11] Niall O'Mahony et al. "Deep learning vs. traditional computer vision".
In: *Science and information conference*. Springer. 2019, pp. 128–144.

[12] Jiasi Chen e Xukan Ran. "Deep learning with edge computing: A review".
In: *Proceedings of the IEEE* 107.8 (2019), pp. 1655–1674.

[13] Jinjiang Wang et al.
"Deep learning for smart manufacturing: Methods and applications".
In: *Journal of manufacturing systems* 48 (2018), pp. 144–156.

[14] Bin Liu et al.
"Grape leaf disease identification using improved deep convolutional neural networks".
In: *Frontiers in Plant Science* 11 (2020), p. 1082.

[15] Rai - Radiotelevisione Italiana Spa. *L'Italia è il maggior produttore di vino al mondo.* 2022. URL: https://www.rainews.it/articoli/2022/01/litalia--il-maggior-produttore-di-vino-al-mondo-0c835c3c-f01b-4f00-b31a-7b55bf9d489a.html.

[16] Alessandra Randazzo. *Il vino pregiato di Pompei, apprezzato sin dall'antichità.* 2018. URL: https://www.classicult.it/vino-pregiato-pompei-antichita/.

[17] Parco archeologico di Pompei. URL: http://pompeiisites.org/.

[18] Fitogest. *Peronospora della Vite.*
URL: https://fitogest.imagelinenetwork.com/it/malattie-piante/malattie-parassiti/funghi/peronospore/peronospora-della-vite/2272.

[19] Adama. *Malattie della vite e rimedi: la linea tecnica di Adama.*
URL: https://www.adama.com/italia/it/approfondimenti-dal-blog/malattie-della-vite-e-rimedi-la-linea-tecnica-di-adama.

[20] istitutoagrariosartor. *Peronospora della vite.*
URL: https://www.istitutoagrariosartor.edu.it/wp-content/uploads/2016/10/PERONOSPORA-VITE-testo.pdf.

[21] Valente. *Oidio della vite, ecco tutti i rimedi.*
URL: https://valentepali.com/oidio-nella-vite-ecco-tutti-i-rimedi/.

[22] Tuttogreen. *Oidio: che cos'è e come si manifesta questo fungo e quali metodi naturali per combatterlo.*
URL: https://www.tuttogreen.it/oidio-cose-fungo-delle-piante/.

[23] Istitutoagrariosartor. *Mal dell'esca.*
URL: https://www.istitutoagrariosartor.edu.it/wp-content/uploads/2013/11/Mal-dellesca.pdf.

[24] Aeb. *Come curare il Black-rot della vite.*
URL: https://www.aeb-group.com/it/come-curare-il-black-rot-della-vite.

[25] CronerCALS. *Managing Black Rot.*
URL: https://grapesandwine.cals.cornell.edu/newsletters/appellation-cornell/2014-newsletters/issue-17/managing-black-rot/.

[26] Chunhao Liang et al. "Identification and characterization of Pseudocercospora species causing grapevine leaf spot in China".
In: *Journal of Phytopathology* 164.2 (2016), pp. 75–85.

[27] Punitha Kartikeyan e Gyanesh Shrivastava. "Review on emerging trends in detection of plant diseases using image processing with machine learning".
In: *International Journal of Computer Applications* 975.8887 (2021).

[28] Achilles D Boursianis et al. "Internet of things (IoT) and agricultural unmanned aerial vehicles (UAVs) in smart farming: a comprehensive review".
In: *Internet of Things* 18 (2022), p. 100187.

[29] Treccani. *UAV*. URL: https://www.treccani.it/enciclopedia/uav.

[30] Waleed Albattah et al. "Artificial intelligence-based drone system for multiclass plant disease detection using an improved efficient convolutional neural network".
In: *Frontiers in Plant Science* 13 (2022).

[31] UAV training Ausatralia. *How Drones Are Used in Agriculture*. URL: https://www.uavtrainingaustralia.com.au/how-drones-are-used-in-agriculture/.

[32] Srdjan Sladojevic et al.
"Deep neural networks based recognition of plant diseases by leaf image classification".
In: *Computational intelligence and neuroscience* 2016 (2016).

[33] Tiago Domingues, Tomás Brandão e João C Ferreira. "Machine Learning for Detection and Prediction of Crop Diseases and Pests: A Comprehensive Survey".
In: *Agriculture* 12.9 (2022), p. 1350.

[34] David Hughes, Marcel Salathé et al. "An open access repository of images on plant health to enable the development of mobile disease diagnostics".
In: *arXiv preprint arXiv:1511.08060* (2015).

[35] Inés Hernández et al. "Assessment of downy mildew in grapevine using computer vision and fuzzy logic. Development and validation of a new method". In: (2022).

[36] Developers Maggioli. *Support-Vector Machine*. 2019.
URL: https://www.developersmaggioli.it/blog/support-vector-machine/.

[37] Pranjali B Padol e Anjali A Yadav.
"SVM classifier based grape leaf disease detection".
In: *2016 Conference on advances in signal processing (CASP)*. IEEE. 2016, pp. 175–179.

[38] Nitesh Agrawal, Jyoti Singhai e Dheeraj K Agarwal. "Grape leaf disease detection and classification using multi-class support vector machine". In: *2017 International Conference on Recent Innovations in Signal processing and Embedded Systems (RISE)*. IEEE. 2017, pp. 238–244.

[39] Debasish Das et al. "Leaf disease detection using support vector machine".
In: *2020 International Conference on Communication and Signal Processing (ICCSP)*. IEEE. 2020, pp. 1036–1040.

[40] Giorgos Mountrakis, Jungho Im e Caesar Ogole.
"Support vector machines in remote sensing: A review".
In: *ISPRS journal of photogrammetry and remote sensing* 66.3 (2011), pp. 247–259.

[41] Javatpoint. *Random Forest Algorithm.*
URL: https://www.javatpoint.com/machine-learning-random-forest-algorithm.

[42] Shima Ramesh et al. "Plant disease detection using machine learning".
In: *2018 International conference on design innovations for 3Cs compute communicate control (ICDI3C)*. IEEE. 2018, pp. 41–45.

[43] G2. *What Is K-Nearest Neighbor? An ML Algorithm to Classify Data.*
URL: https://learn.g2.com/k-nearest-neighbor.

[44] Gautam Kaushal e Rajni Bala.
"GLCM and KNN based algorithm for plant disease detection".
In: *International Journal of Advanced Research in Electrical, Electronics and Instrumentation Engineering* 6.7 (2017), pp. 5845–5852.

[45] Geeksforgeeks. *Difference between a Neural Network and a Deep Learning System.*
URL: https://www.geeksforgeeks.org/difference-between-a-neural-network-and-a-deep-learning-system/.

[46] Suhaili Beeran Kutty et al.
"Classification of watermelon leaf diseases using neural network analysis".
In: *2013 IEEE Business Engineering and Industrial Applications Colloquium (BEIAC)*. IEEE. 2013, pp. 459–464.

[47] Jianwu Lin et al. "GrapeNet: A Lightweight Convolutional Neural Network Model for Identification of Grape Leaf Diseases". In: *Agriculture* 12.6 (2022), p. 887.

[48] Jong Chul Ye. "Biological Neural Networks". In:
Geometry of Deep Learning: A Signal Processing Perspective.
Singapore: Springer Nature Singapore, 2022, pp. 79–90. ISBN: 978-981-16-6046-7.
DOI: 10.1007/978-981-16-6046-7_5.
URL: https://doi.org/10.1007/978-981-16-6046-7_5.

[49] Towardsdatascience. *Explain like I'm five: Artificial neurons.*
URL: https://towardsdatascience.com/explain-like-im-five-artificial-neurons-b7c475b56189.

[50] Medium. *A Quick Tour to Cost Function, Gradient Descent, and Back-Propagation.*
URL: https://medium.com/analytics-steps/a-quick-tour-to-cost-function-gradient-descent-and-back-propagation-ac17e1463c19.

[51] Medium. *CNN Series Part 1: How do computers see images?*
URL: https://medium.com/analytics-vidhya/cnn-series-part-1-how-do-computers-see-images-32462a0b33ca.

[52] Michele Cavaioni. *DeepLearning series: Convolutional Neural Networks.*
URL: https://medium.com/machine-learning-bites/deeplearning-series-convolutional-neural-networks-a9c2f2ee1524.

[53] Ayeshmantha Perera.
What is Padding in Convolutional Neural Network's(CNN's) padding. URL: https://ayeshmanthaperera.medium.com/what-is-padding-in-cnns-71b21fb0dd7.

[54] Naveen. *What is ReLU and Sigmoid activation function?* 2022.
URL: https://www.nomidl.com/deep-learning/what-is-relu-and-sigmoid-activation-function/.

[55] Geeksforgeeks. *CNN | Introduction to Pooling Layer*. 2023.
URL: https://www.geeksforgeeks.org/cnn-introduction-to-pooling-layer/.

[56] Hossein Gholamalinezhad e Hossein Khosravi.
"Pooling methods in deep neural networks, a review".
In: *arXiv preprint arXiv:2009.07485* (2020).

[57] TensorFlow. *Building the Future of TensorFlow*. URL: https://blog.tensorflow.org/2022/10/building-the-future-of-tensorflow.html.

[58] URL: https://keras.io/.

[59] Andrea Mini. *Matrice di confusione*. URL: https://www.andreaminini.com/ai/machine-learning/matrice-di-confusione.

[60] François Chollet. "Xception: Deep learning with depthwise separable convolutions".
In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2017, pp. 1251–1258.

[61] Mark Sandler et al. "Mobilenetv2: Inverted residuals and linear bottlenecks".
In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2018, pp. 4510–4520.

[62] Sik-Ho Tsang. *Review: MobileNetV2 — Light Weight Model (Image Classification)*.
URL: https://towardsdatascience.com/review-mobilenetv2-light-weight-model-image-classification-8febb490e61c.

[63] Christian Szegedy et al. "Rethinking the inception architecture for computer vision".
In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016, pp. 2818–2826.

[64] Google cloud. *Advanced Guide to Inception v3*.
URL: https://cloud.google.com/tpu/docs/inception-v3-advanced?hl=it.

[65] Opengenus. *Understanding the VGG19 Architecture*.
URL: https://iq.opengenus.org/vgg19-architecture/.

[66] Roland Hewage. *Extract Features, Visualize Filters and Feature Maps in VGG16 and VGG19 CNN Models*.
URL: https://towardsdatascience.com/extract-features-visualize-filters-and-feature-maps-in-vgg16-and-vgg19-cnn-models-d2da6333edd0.

[67] Datagen. *ResNet-50 Architecture*.
URL: https://datagen.tech/guides/computer-vision/resnet-50/.

[68] Aditi Rastogi. *ResNet50*.
URL: https://blog.devgenius.io/resnet50-6b42934db431.

[69] Tuğba Özacar, Övünç Öztürk e Nurdan Güngör Savaş.
"Hermos: An annotated image dataset for visual detection of grape leaf diseases".
In: *Journal of Information Science* (2022), p. 01655515221091892.

[70] Jayme Garcia Arnal Barbedo et al. "Annotated plant pathology databases for image-based detection and recognition of diseases".
In: *IEEE Latin America Transactions* 16.6 (2018), pp. 1749–1757.

[71] Mehmet Alican Noyan. "Uncovering bias in the PlantVillage dataset".
In: *arXiv preprint arXiv:2206.04374* (2022).

[72] Xuebin Qin et al.
"U2-Net: Going deeper with nested U-structure for salient object detection".
In: *Pattern recognition* 106 (2020), p. 107404.

[73] split-folders. *Project description*. URL: https://pypi.org/project/split-folders/.