SAPIENZA
UNIVERSITÀ DI ROMA

Anomaly detection su immagini d'uva mediante Rete Convoluzionale

Facoltà di Ingegneria dell'informazione, informatica e statistica

Corso di Laurea in Ingegneria Informatica e Automatica

Candidato

Andrea De Carlo
Matricola 1943362

Relatore

Prof. Thomas Alessandro Ciarfuglia

Anno Accademico 2022/2023

Anomaly detection su immagini d'uva mediante Rete Convoluzionale
Tesi di Laurea. Sapienza – Università di Roma

Questa tesi è stata composta con LaTeX e la classe Sapthesis.

Email dell'autore: andrea001.adc@gmail.com

Ai miei genitori,
per avermi supportato e sopportato.

Ai miei nonni,
che mi hanno accudito come un figlio.

A mio fratello,
affinché possa prendere esempio.

A mio zio Antonio,
fonte di ispirazione.

Alle mie zie Anna e Italia,
seconde madri.

A Nico ed Ezio,
miei idoli e mentori.

A Grazia, Lella, e Carmelina,
sorelle maggiori.

Ai miei amici di sempre,
compagni di viaggio.

Al Bar Italia e a Pietraroja,
scuole di vita e rifugi.

A Roma,
giungla in cui crescere.

Alle persone incontrate in questi anni,
per avermi donato una parte di voi.

Sommario

La differenziazione automatica di chicchi d'uva tra sani e danneggiati è di rilevante importanza nel progetto europeo Canopies [1], che punta a sviluppare un nuovo paradigma di collaborazione tra uomo e robot nel campo dell'agricoltura di precisione, sfruttando robot dotati di bracci per la potatura e telecamere in grado di catturare immagini per identificare i grappoli d'uva nell'ambiente in cui si muovono.

L'obiettivo di questa relazione è fornire una delle soluzioni alla differenziazione dei grappoli d'uva in sani e danneggiati, mediante una rete neurale convoluzionale basata su LeNet [8], confrontando i risultati sul nostro dataset con un paper della comunità scientifica [2] che ha già risolto il problema con ottimi risultati.

La tesi è suddivisa in capitoli:

- inizialmente esaminerò lo stato dell'arte attuale, analizzando il paper a cui ho fatto riferimento durante lo studio, con le relative metriche di giudizio e le conclusioni che ne derivano analizzandole;
- successivamente passerò all'implementazione del mio modello, spiegando nel dettaglio il codice con le scelte costruttive che hanno portato ai risultati dello studio;
- infine verranno valutati i risultati ottenuti dal mio modello, sia in modo qualitativo che quantitativo, traendone quindi le conclusioni finali.

Indice

Capitolo 1

Introduzione

1.1 Descrizione del Task

Il progetto di machine learning per l'anomaly detection in oggetto è parte di un'iniziativa europea volta a migliorare l'efficienza e la precisione delle operazioni di potatura in viticoltura [1]. L'obiettivo principale è distinguere automaticamente i grappoli d'uva sani da quelli rovinati o malati. Questo processo è fondamentale per garantire una produzione di uva di alta qualità e per ottimizzare le risorse umane impiegate, sostituendo il lavoro manuale dei potatori con robot autonomi.

Figura 1.1. Collaborazione tra uomo e macchina

La rilevazione automatica di anomalie è cruciale in una varietà di settori, inclusa l'agricoltura. Nel nostro caso specifico, l'identificazione dei grappoli d'uva danneggiati consente di:

1. Aumentare la qualità del raccolto: rimuovendo grappoli rovinati si può garantire una produzione di vino di alta qualità.

2. Ottimizzare le risorse: l'automazione della potatura basata sull'AI può ridurre

i costi e migliorare l'efficienza.

3. Monitorare la salute delle piante: l'identificazione tempestiva dell'uva malate può contribuire a prevenire la diffusione di malattie nelle vigne.

In tale direzione, si sta a lungo studiando per migliorare i risultati della classificazione, e negli ultimi anni i risultati maggiori sono stati ottenuti con techine di Deep Learning, più specificatamente grazie alle reti neurali convoluzionali (CNN).

Il loro impiego permette l'estrazione automatica delle features spaziali dalle immagini mediante l'operazione di convoluzione:

data un'immagine H ed un kernel K, il valore puntuale del pixel *G[i, j]* dell'immagine G ottenuta come output è:

$$G[i,j] = \sum_m \sum_n H[m,n] \cdot F[i-m, j-n] \tag{1.1}$$

Oggetto di studio da parte dei ricercatori è minimizzare il numero di parametri, dato che l'impiego di tali architetture può diventare molto oneroso in termini di costi computazionali ed economici.

L'obiettivo di questa relazione è mostrare come l'impiego di una di tali architetture, in particolare la LeNet, sia in grado di risolvere il problema della classificazione tra immagini di uva sana e danneggiata, mediante uno studio sperimentale per ottimizzare i suoi parametri al fine di ottenere i migliori risultati possibili una volta applicata al dataset in nostro possesso.

1.2 Metodologia di ricerca

La metodologia di ricerca adottata per il presente studio si è articolata in diverse fasi chiave, al fine di garantire una rigorosa valutazione dell'efficacia dell'approccio di rilevazione delle anomalie nell'ambito della viticoltura.

La fase iniziale della ricerca ha comportato un'approfondita analisi dello stato dell'arte nel campo della rilevazione delle anomalie. Questa revisione bibliografica ha permesso di identificare i principali approcci, metodi, e tecnologie utilizzati in contesti analoghi, fornendo un contesto teorico solido per lo sviluppo e la valutazione del nostro modello.

Un aspetto cruciale del nostro studio è stato la raccolta e la preparazione del dataset. Le immagini di grappoli d'uva, acquisite in condizioni reali di un vigneto, sono state sottoposte a un processo di pre-processing per garantire la qualità e la coerenza dei dati. Questo ha incluso la normalizzazione dei colori e l'aumento del dataset per migliorare i risultati ottenuti.

Successivamente, il dataset è stato suddiviso in tre insiemi distinti: il train set, utilizzato per addestrare il modello, il validation set, per valutare il modello durante

il training, e il test set, riservato per la valutazione delle prestazioni del modello, inteso come immagini che non ha mai visto durante l'allenamento.

Nel cuore della nostra metodologia di ricerca si trova l'implementazione della rete neurale convoluzionale LeNet, basata sul modello descritto nel paper originale di LeCun et al. (1998) [8]. Questa rete è stata scelta per la sua efficienza nella classificazione di immagini e la sua adattabilità alle dimensioni limitate delle immagini dei grappoli d'uva.

La rete LeNet è stata configurata e addestrata utilizzando il train set precedentemente preparato. Durante il processo di addestramento, sono stati monitorati gli indicatori di prestazione chiave, tra cui l'andamento della loss function e l'accuratezza del modello, nonchè valutati in base alla precision e alla recall ottenuti.

1.3 Strumenti utilizzati

Nell'ambito di questa ricerca, la selezione degli strumenti è stata di fondamentale importanza per garantire l'efficienza dell'addestramento del modello di deep learning e per la visualizzazione dei dati.

Data la necessità di disporre di una potenza di calcolo significativa per l'addestramento di un modello di deep learning, è stato scelto di sfruttare Google Colab (Colaboratory) [4]. Google Colab è una piattaforma di Google Cloud che offre un ambiente di sviluppo basato su cloud per l'esecuzione di codice Python, inclusi progetti di machine learning e deep learning. Questo ambiente ha fornito accesso a risorse di calcolo elevate, tra cui unità di elaborazione grafica (GPU) e unità di elaborazione tensoriale (TPU), senza la necessità di disporre di hardware locale avanzato.

Il linguaggio di programmazione Python è stato scelto come linguaggio principale per lo sviluppo dell'intero progetto. Python è ampiamente utilizzato nell'ambito dell'intelligenza artificiale e del machine learning grazie alla sua vasta libreria di moduli e framework specifici, rendendolo una scelta naturale per la creazione di modelli di deep learning.

La base scelta per poter allenare il modello è PyTorch [5], una libreria di machine learning open source ampiamente adottata per la creazione e l'addestramento di reti neurali. La flessibilità e la facilità d'uso di PyTorch lo rendono ideale per lo sviluppo di modelli complessi, come la rete neurale convoluzionale LeNet utilizzata in questo studio. PyTorch ha permesso di definire e addestrare il modello in modo efficace, con il supporto nativo per il calcolo su GPU per accelerare l'addestramento.

Al fine di monitorare gli andamenti e visualizzare le immagini sono state usate le librerie Matplotlib [6] e Seaborn [7]. Queste offrono una vasta gamma di opzioni per la generazione di grafici chiari e informativi, che sono essenziali per l'analisi dei risultati dell'addestramento del modello. Matplotlib è particolarmente utile per la creazione di grafici personalizzati, mentre Seaborn fornisce uno strato di astrazione aggiuntivo per semplificare la creazione di grafici statistici complessi.

L'uso combinato di questi strumenti mi ha consentito di creare un ambiente di sviluppo efficiente e performante per il nostro studio.

Capitolo 2

Dataset

2.1 Struttura del dataset

Il dataset in nostro possesso è composto da patches di diverse scale di zoom, in particolare 3, derivanti da immagini a colori (RGB) di grappoli d'uva interi. Questa suddivisione ha l'obiettivo di catturare dettagli specifici delle immagini dei grappoli d'uva in diverse prospettive e risoluzioni. Le tre categorie di scala di zoom consentono di considerare variazioni significative nelle immagini, dalla visione generale dei grappoli fino ai dettagli più ravvicinati.

Figura 2.1. Scala 2

Figura 2.2. Scala 1.5

Figura 2.3. Scala 1

È stato scelto di utilizzare la scala di zoom 1.5, in modo da catturare in una singola patch un chicco d'uva. Questa decisione è stata presa in conformità con il paper di riferimento con cui ci confrontiamo [2], il quale ha adottato una metodologia simile per l'allenamento del loro modello. In particolare, nel paper vengono utilizzate patches di dimensione 120 x 120 pixels, che assicurano per il loro dataset di immagini l'inquadramento di un singolo chicco. Nel nostro caso invece le patches sono di dimensioni 450 x 450, derivanti da immagini con grandezze non omogenee. Il totale delle patches quindi è di 506.

La concentrazione su un singolo chicco d'uva per patch può migliorare l'accuratezza della rilevazione delle anomalie, in quanto riduce la complessità dovuta alla presen-

za di più elementi nell'immagine. Questo rende più chiare le caratteristiche delle anomalie sui chicchi stessi, facilitando l'identificazione da parte del modello.

2.2 Disposizione files

(a) Divisione in cartelle

(b) Nomenclatura dei files

Figura 2.4. Disposizione dei files

Il dataset fornitomi è stato suddiviso in modo efficiente in diverse cartelle, ciascuna delle quali svolge un ruolo chiave nell'ambito del mio studio.

Le prime due cartelle, denominate *"good"* (buone) e *"bad"* (danneggiate), hanno il compito di categorizzare le immagini in base alle condizioni dei bersagli oggetto del mio problema di ricerca, i chicchi d'uva. Questa suddivisione iniziale è fondamentale, poiché mi consente di distinguere tra immagini chicchi d'uva in condizioni ottimali e immagini di chicchi danneggiati, il che è l'obiettivo principale del mio studio.

Successivamente, all'interno del dataset, trovo una serie di immagini RGB che sono state collocate nella cartella *"scales"* (scale). Queste immagini sono nel formato .jpg e accompagnate da file .txt che, però, non sono rilevanti per il mio studio e sono stati ignorati.

Per sviluppare il mio approccio sperimentale mi sono basato sul lavoro eseguito nel paper di riferimento [2]. In questo, gli autori hanno utilizzato patches di dimensione 120x120 pixel. Questa scelta è stata motivata dal fatto che le immagini utilizzate nel loro lavoro avevano la particolarità di avere una scala di 0.3 mm per pixel, il che consentiva di ottenere una rappresentazione estremamente dettagliata. Questa dimensione consentiva loro di centrare in modo accurato gli oggetti d'interesse, nel nostro caso i chicchi d'uva, nelle immagini.

Dopo un'attenta analisi dei dati a mia disposizione, ho deciso di adottare un approccio simile per garantire una coerenza tra i miei risultati e quelli presentati nel paper di riferimento. Pertanto, ho scelto di utilizzare la scala 1.5, in cui i chicchi d'uva sono completamente visibili nelle immagini, pur presentando alcuni difetti

strutturali che affronterò e discuterò in seguito. Questa scelta di scala è stata drastica per consentire una valutazione accurata delle prestazioni del mio algoritmo in confronto al lavoro di riferimento.

2.3 Implementazione classe dataset

Per gestire il dataset ho ritenuto essenziale definire una classe personalizzata chiamata *"CanopiesDataset"*. Questa classe eredita le funzionalità della classe predefinita *"Dataset"* di PyTorch, offrendomi così un controllo granulare e flessibilità nell'elaborazione dei dati.

```
from PIL import Image
from torch.utils.data import Dataset
import numpy as np

class CanopiesDataset(Dataset):
    def __init__(self, image_names, transform):
        self.image_names = image_names
        self.transform = transform

    def __len__(self):
        return len(self.image_names)

    def __getitem__(self, idx):
        image = np.array(
            Image.open(self.image_names[idx]).convert("RGB")
        )

        if self.transform:
            image = self.transform(image)
        if "good" in self.image_names[idx]:
            label = 0
        else:
            label = 1

        return image, label
```

Prima di tutto, ho dovuto importare le librerie necessarie per il mio lavoro, come *"PIL"* per la gestione delle immagini, *"torch.utils.data"*per le funzionalità del dataset, e *"numpy"* per la manipolazione di array numerici.

All'interno della classe *"CanopiesDataset"*, il costruttore (init) accetta due argomenti principali: *"image_names"* e *"transform"*. *"image_names"* rappresenta una lista di nomi di immagini, che raffigurano le immagini nel mio dataset. *"transform"* rappresenta una serie di trasformazioni delle immagini che verranno applicate alle immagini durante il caricamento, consentendomi di adattare le immagini in base alle mie esigenze, come la normalizzazione o il ridimensionamento.

Il metodo *"len"* restituisce la lunghezza totale del dataset, necessaria per iterare correttamente attraverso di esso durante l'addestramento del modello.

Il metodo *"getitem"* ci permette di accedere all'immagine effettiva. Iniziamo caricando l'immagine corrispondente all'indice specifico da *"image_names"* e la convertiamo in un array numpy nel formato RGB. Questa è una fase fondamentale, poiché ci permette di rappresentare l'immagine in un formato compatibile con *PyTorch*.

Successivamente, verifichiamo se è stata specificata una trasformazione da applicare all'immagine. Se la trasformazione è stata definita, la applichiamo all'immagine. Ciò verrà usato successivamente per normalizzare in modo coerente e ampliare il dataset.

Infine, stabiliamo l'etichetta associata all'immagine. Nel mio caso, sto lavorando con immagini etichettate come *"good"* (buone) o *"bad"* (danneggiate), quindi assegniamo un valore di 0 se l'immagine è *"good"* e 1 se è *"bad"*.

Per accedere alle immagini nelle cartelle ho definito la seguente funzione:

```
def get_image_paths(data_dir):
    return [
        os.path.join(data_dir, image_name)
            for image_name in os.listdir(data_dir)
                if image_name.endswith(".jpg")
        ]
```

All'interno della funzione, viene utilizzata una list comprehension per generare una lista di percorsi completi alle immagini. Questo viene fatto iterando attraverso i nomi dei file presenti nella directory *"data_dir"* utilizzando la funzione *"os.listdir(data_dir)"*. Durante questo processo di iterazione, la condizione *"if image_name.endswith(".jpg")"* viene applicata per assicurarsi che vengano selezionate solo le immagini con estensione *".jpg"*, escludendo così qualsiasi altro tipo di file presente nella directory.

Ogni percorso completo viene quindi creato utilizzando *"os.path.join(data_dir, image_name)"* per concatenare il percorso della directory principale *"data_dir"* con il nome del file dell'immagine e successivamente inserito in un array.

"data_dir" sarà a turno il percorso assoluto che punta alle cartelle *"bad"* e *"good"* nella directory *"scale_1.5"*. Gli array quindi ottenuti verranno passati come parametri alle classi *DataLoader* predefinite di *Pytorch* insieme alla relativa trasformazione per poter essere iterate durante i processi di training e valutazione.

2.3.1 Normalizzazione e Data Augmentation

Per ottenere risultati migliori ho deciso di normalizzare il dataset in mio possesso, nonostante il paper a cui facessi riferimento [2] non ne citi dettagli:

```
def batch_mean_and_sd(loader):
    cnt = 0
```

```
    fst_moment = torch.empty(3)
    snd_moment = torch.empty(3)

    for images, _ in loader:
        b, c, h, w = images.shape
        nb_pixels = b * h * w
        sum_ = torch.sum(images, dim=[0, 2, 3])
        sum_of_square = torch.sum(images ** 2,
                                  dim=[0, 2, 3])
        fst_moment = (cnt * fst_moment + sum_) / (
                     cnt + nb_pixels)
        snd_moment = (cnt * snd_moment + sum_of_square) / (
                             cnt + nb_pixels)
        cnt += nb_pixels

    mean, std = fst_moment, torch.sqrt(
      snd_moment - fst_moment ** 2)
    return mean, std
```

Inizialmente, vengono inizializzate alcune variabili cruciali, tra cui *"cnt"* (un contatore), *"fst_moment"* (il primo momento statistico), e *"snd_moment"* (il secondo momento statistico).

Successivamente, un ciclo for viene utilizzato per iterare attraverso i batch di dati forniti dal *"loader"*, che nel nostro caso sarà il train_loader. Durante ogni iterazione, vengono effettuati calcoli dettagliati, tra cui il calcolo del numero di pixel nel batch, la somma degli elementi dell'immagine, e la somma dei quadrati degli elementi dell'immagine. Tutti questi calcoli sono cruciali per ottenere le statistiche aggregate necessarie per la normalizzazione.

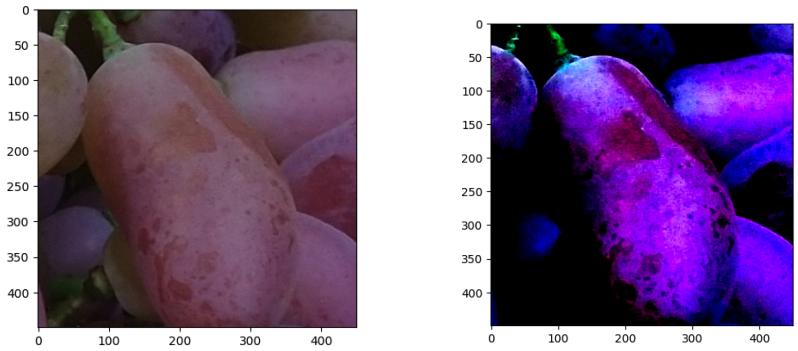
Le variabili *"fst_moment"* e *"snd_moment"* vengono aggiornate durante ogni iterazione per tener conto dei momenti statistici cumulativi. Questo approccio garantisce una stima accurata della media e della deviazione standard per l'intero set di dati, anche se i dati vengono elaborati a lotti.

Alla fine del ciclo, le stime di media e deviazione standard vengono calcolate utilizzando le variabili accumulate.

Infine, definisco una trasformazione chiamata *"transform_NORMALIZED"* che utilizza le stime di media e deviazione standard calcolate in precedenza per normalizzare le immagini durante il processo di elaborazione, che verrà passata come parametro nella creazione dei vari DataLoader.

```
transform_NORMALIZED = torchvision.transforms.Compose(
        [torchvision.transforms.Normalize(
            mean = mean, std = std)]
    )
```

Il risultato sulle immagini è il seguente:



(a) Originale **(b)** Normalizzata

Figura 2.5. Risultato della normalizzazione

Una sfida importante è stata quella legata allo sbilanciamento del mio dataset, un problema che ho anche riscontrato nel paper di riferimento. In particolare, nel loro caso le immagini categorizzate come "danneggiate" erano significativamente inferiori in quantità rispetto alle immagini "intatte" (in proporzione 1/3). Il problema è stato risolto aggiungendo alle "danneggiate" le stesse ruotate di 120° e 240°.

Per affrontare questa situazione e garantire una rappresentazione più equilibrata delle classi nel mio dataset, ho adottato una strategia simile a quella descritta nel paper di riferimento. Tuttavia, ho dovuto fare alcune modifiche per adattare questa strategia alle caratteristiche specifiche del mio dataset. Nel mio caso, la proporzione tra il *"bad_dataset"* e il *"good_dataset"* era 1/4.

Per mantenere l'equilibrio tra le classi nel mio dataset, ho deciso di ruotare le immagini nel *"bad_dataset"* di 90°, 180° e 270°. Questo approccio è stato scelto per evitare di dover ritagliare ulteriori patch dalle immagini, come era stato fatto nel paper di riferimento per rimuovere gli angoli neri che si formavano a causa delle rotazioni non allineate agli assi cartesiani.

Capitolo 3

Lo stato dell'arte

3.1 L'architettura del paper originario

Il problema da risolvere è una classificazione binaria, e il paper [2], fonte fondamentale di confronto per il mio lavoro, ha concentrato il suo studio su una rete basata su LeNet. Questa architettura, sebbene possa sembrare relativamente semplice, ha dimostrato la sua efficacia nel trattare problemi di visione artificiale. LeNet, infatti, è stata una delle prime reti neurali convoluzionali sviluppate, e la sua struttura stratificata, che include strati convoluzionali e di pooling, ha dimostrato di essere altamente adatta per estrarre e apprendere le caratteristiche chiave dalle immagini. L'uso di LeNet in questo contesto rappresenta quindi un punto di partenza interessante per affrontare la classificazione binaria.

La rete basata su LeNet è stata poi confrontata con un secondo approccio basato su una rete pre-trained utilizzando l'architettura ResNet. ResNet è una rete più profonda e complessa che ha rivoluzionato il campo del deep learning grazie all'introduzione di skip connections o "shortcut connections." Queste connessioni consentono al modello di apprendere meglio le rappresentazioni gerarchiche delle immagini, evitando il problema della scomparsa del gradiente. L'utilizzo di una rete pre-trained basata su ResNet può portare a risultati notevolmente migliori grazie alla sua capacità di catturare caratteristiche più complesse nelle immagini.

3.2 Introduzione alle reti

3.2.1 LeNet: architettura

La prima e principale rete dello studio è la rete neurale convoluzionale LeNet, sviluppata da Yann LeCun nel 1998 [8]. Questa architettura è stata una delle prime CNN di successo ed è stata concepita principalmente per l'elaborazione delle immagini, in particolare per la classificazione di cifre scritte a mano, come il riconoscimento di cifre nei codici postali.

L'architettura di LenNet è composta da diverse strati che lavorano insieme per

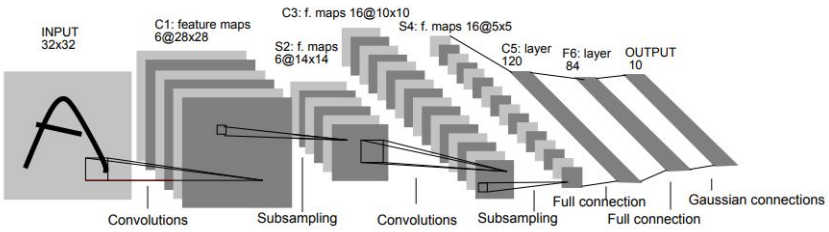


Figura 3.1. Struttura di LeNet

estrarre progressivamente caratteristiche sempre più complesse dalle immagini in ingresso. Il funzionamento di LeNet è il seguente:

Strato di convoluzione : il primo strato di LeNet è un layer di convoluzione. Questo strato applica una serie di filtri (kernel) alle regioni locali dell'immagine di input. I filtri catturano diverse caratteristiche, come linee o bordi, nelle diverse parti dell'immagine. Durante questa operazione, l'immagine viene convoluta con i filtri, producendo una mappa di caratteristiche (feature map).

Strato di pooling : dopo la convoluzione, LeNet utilizza uno strato di pooling (di solito MaxPooling) per ridurre la dimensionalità delle feature map. Questo strato prende la massima valore da regioni sovrapposte della mappa delle caratteristiche, riducendo così la dimensione spaziale. Questo aiuta a ridurre il numero di parametri e a rendere il modello più robusto alle variazioni di posizione delle caratteristiche.

Strato completamente connesso : dopo uno o più strati di convoluzione e pooling, ci sono uno o più strati completamente connessi (o densi). Questi strati sono simili a quelli di una rete neurale tradizionale e sono responsabili di combinare le caratteristiche estratte dalle mappe delle caratteristiche in una rappresentazione complessa dell'immagine. Queste rappresentazioni vengono quindi utilizzate per effettuare una classificazione finale.

Funzione di attivazione : tra ogni strato, di solito viene applicata una funzione di attivazione non lineare, come la funzione ReLU (Rectified Linear Unit). Questa funzione introduce non linearità nel modello, consentendo alla rete di apprendere relazioni complesse nei dati.

Classificazione finale : l'ultimo strato completamente connesso di solito ha un numero di unità di output corrispondente al numero di classi in cui si desidera effettuare la classificazione. Spesso si utilizza una funzione di attivazione come Softmax per ottenere delle probabilità associate a ciascuna classe, indicando la probabilità che l'input appartenga a ciascuna classe.

3.2.2 ResNet: architettura

Le performances della rete LeNet sono poi state confrontate con una ResNet, acronimo di *"Residual Network"*, la quale rappresenta una significativa innovazione nel campo delle reti neurali convoluzionali (CNN) ed è un modello fondamentale nell'ambito della computer vision e del deep learning [9]. È stato proposto per affrontare un problema comune noto come *"degradazione del gradiente"*, che può verificarsi quando si addestrano reti neurali molto profonde.

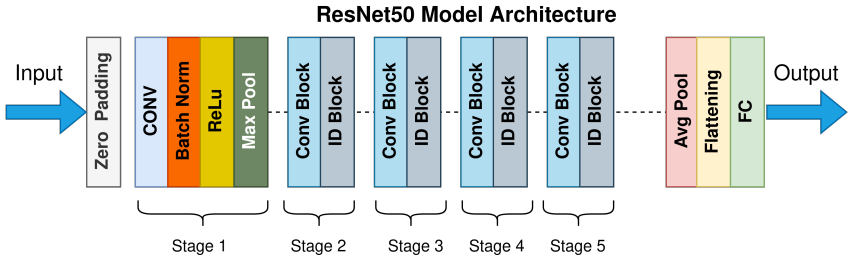


Figura 3.2. Struttura di ResNet50

Per comprenderne il funzionamento, è importante iniziare con una panoramica delle reti neurali tradizionali. In una CNN classica, l'input fluisce attraverso una serie di strati convoluzionali e di pooling, seguiti da strati completamente connessi (fully connected) per la classificazione. Ogni strato applica una trasformazione ai dati e passa l'output al successivo strato, come succede anche in LeNet. Questo approccio funziona bene per molte applicazioni, ma ha il limite di diventare inefficace quando la rete diventa molto profonda. L'aggiunta di strati ulteriori può comportare un peggioramento delle prestazioni invece di un miglioramento, a causa del problema della scomparsa del gradiente.

ResNet affronta questo problema introducendo il concetto di *"skip connection"* o *"shortcut connection"*. Invece di passare l'output da uno strato direttamente al successivo, ResNet aggiunge un collegamento diretto tra gli strati intermedi, noti come *"residual blocks"* o *"blocchi residuali"*. Questi collegamenti consentono all'informazione di "saltare" uno o più strati intermedi, passando direttamente al di là di essi. Questo è possibile grazie alla somma degli output degli strati intermedi con l'output dell'ultimo strato del blocco residuale.

Questo approccio offre diversi vantaggi:

- risolve il problema della degradazione del gradiente: i collegamenti residui consentono al gradiente di fluire più agevolmente attraverso la rete durante la retropropagazione, rendendo possibile l'addestramento di reti neurali molto più profonde senza problemi di degradazione del gradiente.

- facilità di apprendimento di funzioni residue: ResNet rende più semplice per la rete apprendere le differenze tra l'output desiderato e l'output corrente, in quanto il collegamento residuo rappresenta proprio l'errore residuo da correggere.

- miglioramento delle prestazioni: ResNet ha dimostrato di ottenere prestazioni superiori rispetto a modelli precedenti in molte applicazioni di visione artificiale, inclusa la classificazione di immagini e il rilevamento di oggetti.

I blocchi residuali possono essere impilati per creare reti neurali profonde, spesso con centinaia di strati. Questo ha portato a varianti di ResNet, come ResNet-50 e ResNet-101, utilizzate comunemente in varie applicazioni.

3.3 Operazioni effettuate dalle reti

Di seguito vengono spiegate le principali operazioni effettuate dalle reti in oggetto durante lo studio:

Convoluzione : è un'operazione fondamentale nell'elaborazione delle immagini e nel deep learning. Essa coinvolge due segnali, tipicamente una matrice di input e un filtro (kernel) di dimensioni definite. L'operazione di convoluzione tra il filtro e l'input produce una mappa delle caratteristiche (feature map) che evidenzia le informazioni rilevanti presenti nell'input.

Formalmente, la convoluzione discreta tra una matrice di input I e un filtro K in una posizione i, j può essere espressa come segue:

$$(I * K)(i, j) = \sum_m \sum_n I(i - m, j - n) \cdot K(m, n)$$

dove (i, j) è la posizione di output e m, n sono indici che scorrono il filtro K. L'operazione di convoluzione viene applicata a diverse posizioni dell'input per creare la feature map.

Max Pooling : è una tecnica utilizzata per ridurre la dimensione delle feature map generate dalla convoluzione. Si tratta di un'operazione di sottocampionamento che seleziona il valore massimo da una regione rettangolare dell'input e lo assegna alla posizione corrispondente nella feature map di output.

Formalmente, l'operazione di max pooling in una regione R dell'input I può essere espressa come segue:

$$\text{Max Pooling}(R) = \max_{(i,j) \in R} I(i, j)$$

Il max pooling aiuta a ridurre il numero di parametri e a rendere le feature map più robuste alle piccole variazioni nella posizione delle caratteristiche.

ReLU (Rectified Linear Unit) : è una funzione di attivazione comunemente utilizzata nelle reti neurali. Essa introduce la non-linearità nell'output di un neurone o di una rete. La funzione ReLU è definita come:

$$\text{ReLU}(x) = \max(0, x)$$

In altre parole, se l'input x è positivo o zero, la funzione ReLU restituisce x; altrimenti, restituisce zero. Questa semplice funzione di attivazione è ampiamente utilizzata per introdurre non-linearità nei modelli di deep learning.

Softmax : la funzione softmax è utilizzata per convertire un vettore di valori in una distribuzione di probabilità. È spesso utilizzata nell'ultimo strato di una rete neurale per ottenere le probabilità associate alle diverse classi in un problema di classificazione.

Data una serie di valori $z_1, z_2, \ldots, z_n$, la funzione softmax calcola le probabilità p_i per ogni valore z_i come segue:

$$p_i = \frac{e^{z_i}}{\sum_{j=1}^{n} e^{z_j}}$$

dove e è il numero di Nepero (2.71828). La somma dei p_i sarà sempre uguale a 1, trasformando così il vettore di valori in una distribuzione di probabilità.

3.4 Metriche di giudizio

Le performance del paper, e quindi per confronto anche le performance del nostro studio, sono state valutate secondo le seguenti metriche di giudizio:

Confusion Matrix (Matrice di confusione): è una tabella utilizzata per valutare le prestazioni di un modello di classificazione. Essa mostra il confronto tra le predizioni del modello e le classi reali dei dati di test. La matrice è composta da quattro elementi principali:

- **True Positive (TP)**: i casi in cui il modello ha previsto correttamente una classe positiva.
- **True Negative (TN)**: i casi in cui il modello ha previsto correttamente una classe negativa.
- **False Positive (FP)**: i casi in cui il modello ha previsto erroneamente una classe positiva quando in realtà era negativa (errore di tipo I).
- **False Negative (FN)**: i casi in cui il modello ha previsto erroneamente una classe negativa quando in realtà era positiva (errore di tipo II).

La matrice di confusione formalmente è rappresentata nel seguente modo:

$$\begin{matrix} TP & FP \\ FN & TN \end{matrix}$$

Nel nostro studio *Negativo* e *Positivo* corrisponderanno rispettivamente a *Sano* e *Danneggiato*.

Precision (Precisione): è una metrica di valutazione della classificazione che misura la proporzione di predizioni positive corrette rispetto al totale delle predizioni positive effettuate dal modello. È calcolata come:

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

Una precisione alta indica che il modello ha una bassa probabilità di fare predizioni positive erronee.

Recall (Recupero o Sensibilità): è una metrica di valutazione della classificazione che, a differenza della precisione, misura la proporzione di casi positivi correttamente previsti dal modello rispetto al totale dei casi positivi reali. È calcolato come:

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

Un recall alto indica che il modello è in grado di individuare la maggior parte dei casi positivi.

3.5 Osservazioni sulle prestazioni del paper

N = 14 321		prediction		
		unharmed	damaged	
actual	unharmed	6827	487	7314
	damaged	265	6742	7007
		7092	7229	Total error: 5.25 %

Figura 3.3. Confusion Matrix della LeNet

Il comportamento della rete neurale nel modello del paper è notevolmente soddisfacente, come evidenziato dai risultati ottenuti. L'accuracy registrata sul training set del 96.4% e sul validation set del 97% dimostra chiaramente l'efficacia del modello nell'apprendimento dei dati e nella generalizzazione delle previsioni. Questi risultati suggeriscono una capacità di classificazione molto robusta e coerente.

Un ulteriore indicatore delle prestazioni del modello è rappresentato dalla confusion matrix, la quale fornisce una panoramica dettagliata delle previsioni effettuate dal modello rispetto alle classi target. Anche qui, i risultati sono promettenti. La precisione, calcolata al 96.26%, indica che la rete ha una capacità elevata di classificare correttamente i dati appartenenti a ciascuna classe. La recall, con un valore del 93.34%, suggerisce che il modello riesce a identificare la maggior parte dei casi positivi con buona precisione.

Tuttavia, è importante sottolineare che, sebbene il modello abbia dimostrato un'eccellente capacità di classificazione, ci sono alcune considerazioni da fare. In particolare, i tassi di errore di tipo uno e tipo due meritano un'analisi più approfondita. L'errore di tipo due, con un valore del 6.66%, è leggermente superiore all'errore di tipo uno, che si attesta al 3.74%. Questo significa che la rete tende a classificare erroneamente un numero maggiore di patch contenenti bacche intatte come danneggiate, rispetto al caso opposto.

Nonostante queste discrepanze tra i tipi di errore, è fondamentale notare che l'ammontare complessivo di misclassificazioni rimane limitato. Questi risultati indicano che il modello ha dimostrato una notevole abilità nel riconoscere le differenze tra le classi di interesse, anche se con alcune lievi tendenze a classificare erroneamente alcune istanze.

Successivamente, è stata condotta una comparazione dettagliata tra la rete neurale basata su LeNet e un modello ResNet50. Sono state effettuate due fasi di addestramento del ResNet50: una volta utilizzando pesi pre-addestrati su ImageNet e una seconda volta partendo da zero (addestramento *"from scratch"*). Entrambi i modelli sono stati addestrati per oltre 60 epoche al fine di garantire una convergenza adeguata e di valutare le loro prestazioni in modo completo.

N = 14 3210		prediction		
		unharmed	damaged	
actual	unharmed	6019	1295	7314
	damaged	242	6765	7007
		6261	8060	Total error: 10.73 %

(a) ResNet50 pre-addestrato

N = 14 3210		prediction		
		unharmed	damaged	
actual	unharmed	6192	1121	7314
	damaged	262	6745	7007
		6455	7866	Total error: 9.66 %

(b) ResNet50 addestrato "from scratch"

Figura 3.4. Confuzion matrix delle ResNet50

L'analisi delle prestazioni del modello ResNet50 pre-addestrato ha rivelato una precisione del 96.13% e una recall dell'82.29%. È interessante notare che il tasso di errore di tipo uno è stato calcolato al 3.87%, mentre il tasso di errore di tipo due è notevolmente più elevato, pari al 17.71%.

Il secondo ResNet50, addestrato *"from scratch"*, ha prodotto risultati comparabili.

La precisione ottenuta è del 95.94%, leggermente inferiore rispetto al modello pre-addestrato. Tuttavia, la recall è leggermente superiore, attestandosi all'84.67%. Anche in questo caso, i tassi di errore di tipo uno (4.06%) e di tipo due (15.33%) presentano caratteristiche simili a quelli del modello pre-addestrato.

Un'osservazione rilevante emersa dall'analisi è che entrambe le versioni del ResNet50 mostrano una tendenza a misclassificare le patch contenenti bacche intatte come danneggiate (falsi negativi), il che si traduce in un tasso di errore di tipo due elevato. Inoltre, il numero di patch che mostrano bacche danneggiate ma vengono classificate come intatte (falsi positivi) è comparabile a quello della nostra rete. In generale, è interessante notare che l'errore totale per entrambi i modelli ResNet50 è quasi il doppio rispetto alla nostra struttura di rete più semplice.

Questi risultati mettono in evidenza l'efficacia della rete neurale basata su LeNet rispetto a modelli più complessi come ResNet50. Considerando le differenze nelle prestazioni tra i modelli, l'approccio più leggero della LeNet mostra una maggiore stabilità nella classificazione delle patch, con un errore totale inferiore. Questi risultati sottolineano l'importanza di considerare la complessità del modello in relazione alle specifiche esigenze dell'applicazione e dimostrano che, in alcuni casi, strutture di rete più semplici possono offrire prestazioni altrettanto valide, se non superiori, rispetto a modelli più profondi e complessi.

Capitolo 4

Implementazione e Sperimentazione

Di seguito verranno discussi i metodi di implementazioni con le varie ragioni e i rispettivi vantaggi nell'ambito del mio studio. Questa analisi riveste un ruolo fondamentale nel mio percorso di ricerca, in quanto influenzerà direttamente la qualità dei risultati ottenuti e la validità delle conclusioni raggiunte. È possibile visionare il codice al seguente link: [3].

4.1 Inizializzazione dei pesi

Nell'ambito della mia ricerca, uno degli aspetti critici che richiede un'attenzione particolare è l'inizializzazione dei pesi all'interno della nostra rete neurale. La scelta dell'inizializzazione dei pesi gioca un ruolo determinante nella convergenza dell'addestramento della rete e, di conseguenza, nelle prestazioni del modello.

Dopo una fase di sperimentazione accurata e analisi empirica, ho optato per l'inizializzazione dei pesi utilizzando il metodo di Xavier, noto anche come inizializzazione *"Glorot"* [10]. Questa scelta è stata motivata da diverse considerazioni teoriche e pratiche.

Inizialmente ho sperimentato l'inizializzazione dei pesi con valori costanti o uniformi. Tuttavia, queste strategie spesso portano a problemi di convergenza lenta o instabile, in quanto possono causare gradienti troppo piccoli o troppo grandi all'inizio dell'addestramento. Questo può rallentare notevolmente il processo di apprendimento o persino portare a divergenza.

L'inizializzazione di Xavier, al contrario, è stata progettata per affrontare specificamente questo problema. Questo metodo imposta i pesi in modo da avere una varianza adeguata in modo che il segnale fluisca attraverso la rete in modo coerente durante la retropropagazione dei gradienti. In particolare, Xavier utilizza una varianza calcolata in base al numero di input e output di ciascun strato, ottimizzando

la propagazione dei gradienti.

```
for m in self.modules():
    if isinstance(m, nn.Linear):
        nn.init.xavier_uniform_(m.weight)
        if m.bias is not None:
            nn.init.zeros_(m.bias)
```

Per l'implementazione di questa inizializzazione iteriamo attraverso i moduli della nostra rete neurale. Per gli strati lineari (o fully connected) all'interno della rete, utilizziamo la funzione *nn.init.xavier_uniform_* per inizializzare i pesi con il metodo Xavier. Inoltre, se uno strato ha anche un termine bias, lo inizializziamo a zero utilizzando *nn.init.zeros_*.

4.2 Funzione di Loss

Fondamentale è esaminare attentamente la scelta della funzione di loss utilizzata durante l'addestramento della nostra rete neurale. La selezione di una funzione di loss adeguata riveste un ruolo cruciale nell'ottimizzazione dei parametri del modello e nell'acquisizione di risultati accurati e coerenti.

```
criterion = nn.BCEWithLogitsLoss()
```

La funzione di loss adottata nel mio studio è la *"BCEwithLogitsLoss"*, abbreviazione di *"Binary Cross-Entropy Loss with Logits"*. Questa scelta è stata motivata da una serie di considerazioni teoriche e pratiche.

In primo luogo, la BCEwithLogitsLoss è una scelta appropriata quando si affrontano problemi di classificazione binaria, come nel nostro caso. Questa funzione di loss è specificamente progettata per gestire output di tipo *logit*, cioè valori reali che rappresentano la *log-odds*, ovvero probabilità, di appartenenza a una classe restituita come risultato dal modello mediante la funzione Softmax. Tramite la BCEwithLogitsLoss, è possibile calcolare la loss, ovvero la differenza tra valore predetto e valore reale, in modo efficiente e stabile, considerando sia la classe positiva che quella negativa, contribuendo così all'addestramento di un modello di classificazione binaria coerente e accurato.

In secondo luogo, la BCEwithLogitsLoss è ampiamente utilizzata nella comunità di machine learning per problemi di classificazione binaria, ed è stata oggetto di numerosi studi e ricerche. La sua validità e la sua efficacia sono ben supportate dalla letteratura scientifica, confermandola come scelta affidabile per il mio studio.

Altro aspetto importante da considerare è che la BCEwithLogitsLoss è in grado di gestire la *sigmoidalizzazione* degli output, il che significa che non è necessario applicare manualmente una funzione di attivazione sigmoidale all'output del modello, semplificando di fatto il processo di addestramento e riducendo la possibilità di errori di implementazione.

4.3 Scelta degli ottimizzatori

Altro aspetto di fondamentale importanza nel contesto del mio studio è la selezione dell'ottimizzatore da utilizzare durante l'addestramento della nostra rete neurale. La scelta dell'ottimizzatore è cruciale, in quanto influisce direttamente sulla velocità di convergenza del modello, sulla stabilità dell'addestramento e sulle prestazioni finali del sistema.

Per decidere quale usare, ho condotto una rigorosa analisi comparativa tra due ottimizzatori di rilievo: lo "Stochastic Gradient Descent" (SGD) e "Adam."

Inizialmente, è stato preso in considerazione SGD, un ottimizzatore tradizionale ampiamente utilizzato nel campo del deep learning. Questo metodo di ottimizzazione si basa sulla stima del gradiente della loss function utilizzando un sottoinsieme casuale dei dati di addestramento a ciascuna iterazione. La sua semplicità e trasparenza lo rendono un'opzione attraente per molte applicazioni. Tuttavia, SGD è noto per richiedere una messa a punto accurata degli iperparametri, come il tasso di apprendimento, al fine di ottenere risultati ottimali.

D'altro canto, Adam è un ottimizzatore più recente e complesso che combina elementi di SGD con adattività ai gradienti e stime del secondo momento. Questa metodologia mira a fornire un tasso di apprendimento adattivo per ciascun parametro, il che può accelerare la convergenza e semplificare la selezione degli iperparametri. Adam è noto per la sua efficacia in molte situazioni e per la sua capacità di adattarsi a varie configurazioni di problemi.

Tuttavia, la selezione tra questi due ottimizzatori non è stata una decisione superficiale. Ho effettuato una serie di esperimenti empirici esaminando diversi set di iperparametri per ciascun ottimizzatore. Questo processo mi ha permesso di valutare le prestazioni relative e confrontare le curve di apprendimento, la stabilità e la velocità di convergenza tra i due metodi.

Alla luce dei risultati ottenuti, alla fine è prevalso SGD, ed ho scelto lui come ottimizzatore principale nel mio studio. Questa decisione è stata basata sulla sua robustezza e sulla possibilità di messa a punto accurata degli iperparametri, che mi ha permesso di ottenere prestazioni accettabili senza compromettere la stabilità dell'addestramento.

4.4 Ottimizzazione degli iperparametri

Nel processo di sperimentazione e ottimizzazione del nostro modello, ho condotto una meticolosa ricerca degli iperparametri ottimali. Questa fase è di cruciale importanza nell'ambito dell'apprendimento automatico, in quanto gli iperparametri influenzano direttamente il comportamento e le prestazioni del modello.

Per effettuare questa ricerca, ho adottato un approccio sistematico noto come "grid search". In questa metodologia, ho inizialmente definito diverse griglie di valori per ciascun iperparametro di interesse e ho eseguito un processo di addestramento del

modello per ogni combinazione possibile di questi valori. Questo mi ha permesso di esplorare un ampio spettro di configurazioni iperparametriche e identificare quelle che hanno prodotto i risultati più promettenti.

La selezione dei valori ottimali degli iperparametri è stata guidata da un'analisi attenta delle metriche di valutazione delle prestazioni del modello. In particolare, ho monitorato l'andamento della loss function, l'accuratezza (accuracy) e la matrice di confusione (confusion matrix) con particolare attenzione alla recall e alla precision. La loss function mi ha fornito informazioni sulla convergenza del modello e sulla sua capacità di minimizzare l'errore, mentre l'accuratezza mi ha indicato la bontà con cui il modello sia in grado di classificare correttamente le istanze. La matrice di confusione, insieme a recall e precision, mi ha permesso di valutare la capacità del modello di distinguere tra le classi di interesse, soprattutto nel nostro contesto di classificazione binaria.

Dopo un'attenta analisi dei risultati ottenuti attraverso la grid search, sono giunto alla selezione degli iperparametri ottimali per l'ottimizzatore SGD. I valori ottimali che ho trovato sono i seguenti:

- Tasso di apprendimento (lr): 0.001
- Momentum: 0
- Weight decay: 0.001

Questi valori sono emersi come quelli che massimizzano le prestazioni del nostro modello in base alle metriche considerate. La scelta di un tasso di apprendimento moderato, un momentum nullo e un basso weight decay ha portato ad ottenere una convergenza stabile e una buona generalizzazione del modello, con performance soddisfacenti che verranno discusse in seguito.

4.5 Salvataggio del modello

Una delle sfide che ho affrontato è stata la gestione del limite di utilizzo del piano gratuito di Colab, che comporta restrizioni temporali per le sessioni di calcolo. Al fine di superare questo ostacolo e garantire la continuità delle mie attività di ricerca, ho implementato una soluzione che prevede il salvataggio del modello addestrato.

La procedura di salvataggio del modello mi consente di archiviare in modo sicuro e persistente lo stato del modello e tutti i relativi parametri. Questo processo è stato implementato attraverso l'utilizzo di PyTorch e la funzione *torch.save()*. Il codice in questione è il seguente:

```
torch.save(model.state_dict(), save_path)
```

Questa istruzione registra lo stato attuale del modello, compresi i pesi dei parametri, in un file specificato da *save_path*. Questa azione è estremamente utile, in quanto consente di conservare i risultati del nostro addestramento e di preservare il progresso ottenuto nel caso in cui la sessione di calcolo dovesse interrompersi in

modo improvviso o se desideriamo analizzare ulteriormente il modello in un secondo momento.

Il salvataggio del modello mi assicura che il lavoro svolto durante l'addestramento del modello non vada perso e mi consente di riprendere l'addestramento da uno stato precedente, qualora fosse necessario continuare il processo di ottimizzazione.

Capitolo 5

Risultati

Ricapitolando, il modello è stato addestrato seguendo le seguenti specifiche:

- Training set: 459 patches
- Validation set: 61 patches
- Epoche di addestramento: 120
- Dimensione del batch: 40
- Funzione di loss: Binary Cross Entropy con Logits Loss
- Ottimizzatore: Stochastic Gradient Descent con
  - Tasso di apprendimento: 0.001
  - Momentum: 0
  - Weight decay: 0.001

5.1 Valutazione metrica dell'addestramento

L'osservazione del grafico mette in evidenza un punto di interesse cruciale nell'addestramento del modello in esame. La curva di validazione della perdita dimostra una chiara tendenza alla stabilizzazione a partire dalla 70esima epoca, suggerendo un possibile punto di sosta nell'iterazione del training. Questo fenomeno è indicativo dell'efficacia del modello nell'apprendere dai dati di addestramento e nella sua capacità di generalizzare le conoscenze acquisite ai dati di validazione con una velocità moderata.

La valutazione delle prestazioni del modello sul set di validazione rivela altresì un notevole risultato. L'accuracy, ossia l'accuratezza, si mantiene costante a un livello apprezzabile, che è circa dell'80%, suggerendo che il modello è in grado di classificare correttamente una percentuale significativa dei campioni di validazione. Nel

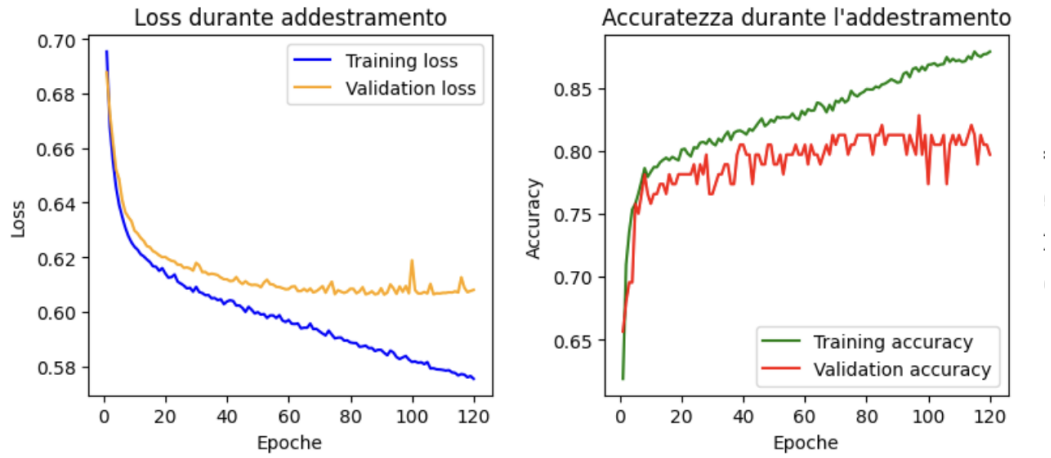


Figura 5.1. Loss e Accuracy

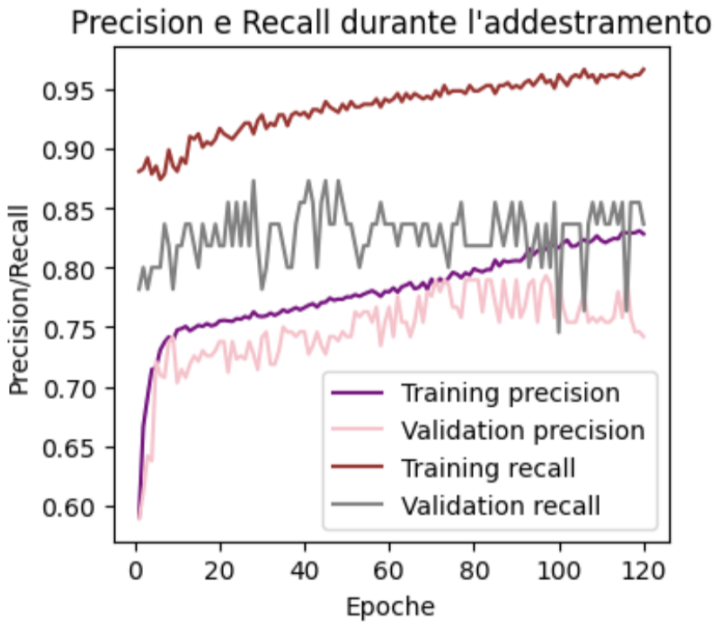


Figura 5.2. Precision e Recall

contempo, è importante notare che precision e recall, metriche fondamentali per la valutazione delle performance di un classificatore, raggiungono rispettivamente il 75% e l'85%. Questo dimostra la capacità del modello non solo di individuare correttamente la maggior parte delle istanze positive, nel nostro caso i chicchi d'uva danneggiati (recall elevato) ma anche di farlo con una percentuale considerevole di accuratezza (precision elevata).

Questi risultati sottolineano la rilevanza della decisione di interrompere il training alla 70esima epoca, poiché ulteriori iterazioni potrebbero non apportare miglioramenti significativi alle prestazioni del modello sul set di validazione, comportando un aumento del rischio di overfitting.

Esamineremo ora come tali valori si traducono concretamente nella qualità della classificazione, illustrando le immagini insieme ai corrispondenti valori reali e predetti, con l'obiettivo di identificare le istanze che la rete riconosce in modo accurato e quelle che occasionalmente confonde.

5.2 Valutazione visiva dei test: punti di forza

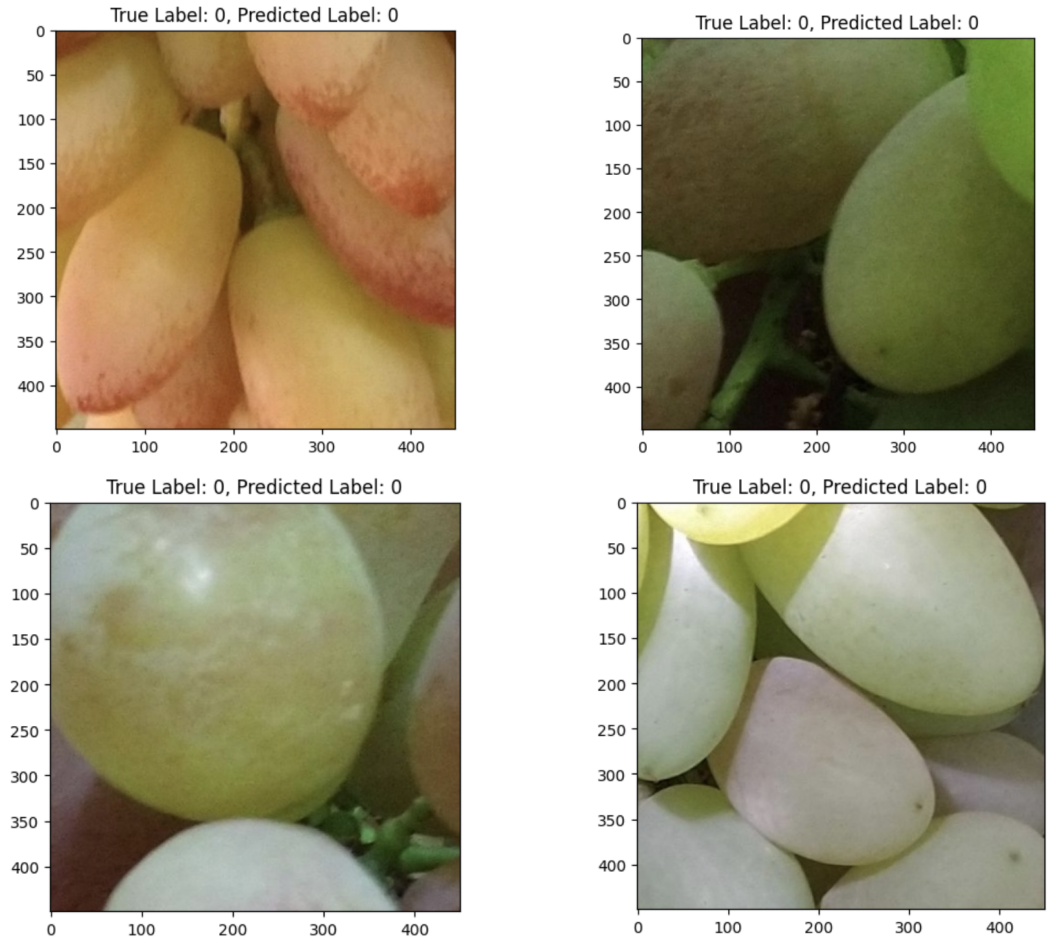


Figura 5.3. Risultati sui chicchi sani

Riguardo alle porzioni di immagine predette con precisione dalla rete neurale, con particolare attenzione alla categoria "sana", un'analisi visiva delle predizioni mostra un insieme di caratteristiche comuni. Questi attributi condivisi tra le patch correttamente classificate includono la percezione del colore predominante, la presenza di una texture uniformemente liscia sulla superficie dei chicchi e, notevolmente, la distribuzione quasi omogenea di chicchi sani in tutta l'area dell'immagine.

Queste osservazioni indicano che la rete neurale è in grado di riconoscere e associare correttamente questi tratti distintivi alle regioni dell'immagine che sono effettivamente costituite da chicchi di uva in buona salute. Tale capacità di identificazione si traduce in una precisione più che soddisfacente nella classificazione di campioni

sani, suggerendo un appropriato apprendimento delle caratteristiche discriminative rilevanti per questa categoria specifica.

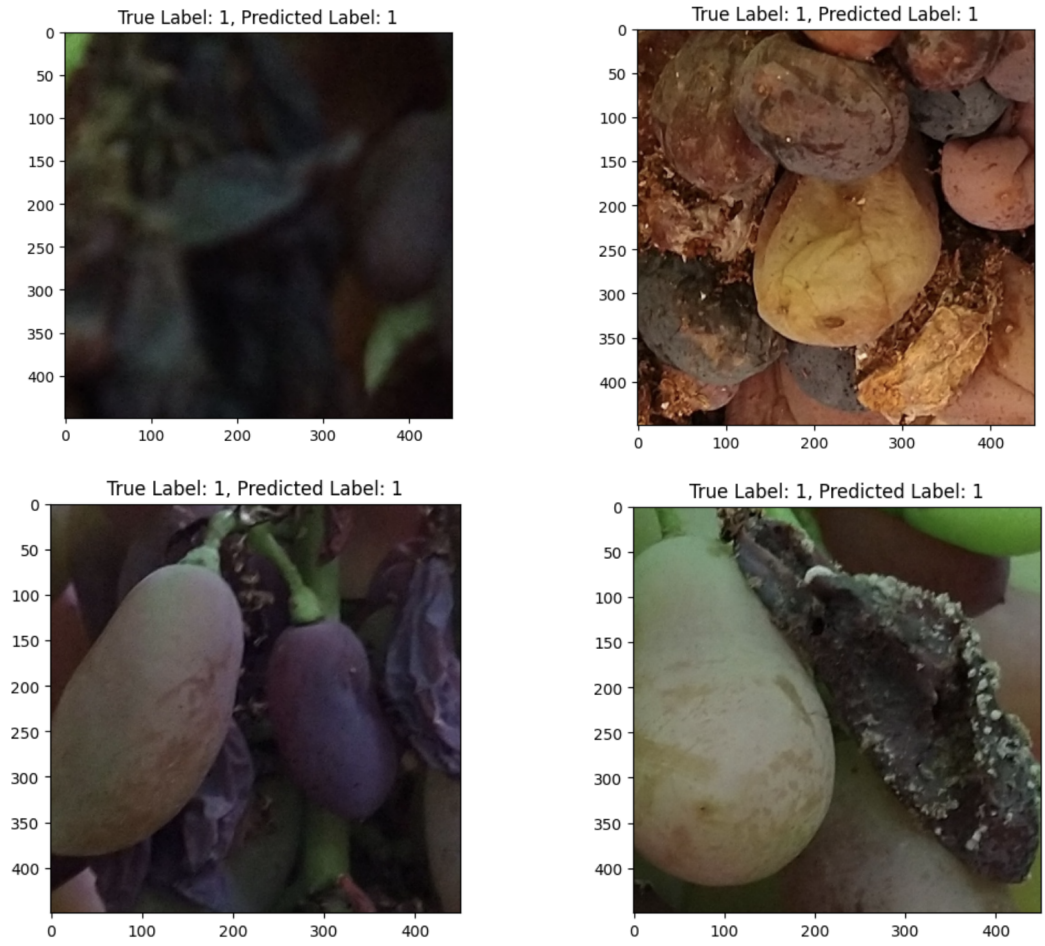


Figura 5.4. Risultati sui chicchi danneggiati

Relativamente ai chicchi d'uva danneggiati che sono stati correttamente identificati e catalogati dal nostro modello, è possibile osservare che condividono un andamento cromatico che tende a presentare colorazioni più scure, spesso inclinate verso tonalità violacee. Questa variazione cromatica può essere interpretata un avanzamento eccessivo della maturazione, insieme alla manifestazione visiva delle lesioni o delle alterazioni presenti sulla superficie dei chicchi, che si riflettono in una diversa riflettanza della luce incidente.

Inoltre, una caratteristica distintiva dei chicchi danneggiati è una texture più ruvida, che si manifesta attraverso la presenza di molte asperità superficiali e la formazione di ombre e discrepanze. Queste irregolarità sulla superficie dei chicchi possono derivare da danni fisici, patologie o difetti di crescita, e si rivelano pertanto come indicatori visivi cruciali per il riconoscimento accurato di chicchi danneggiati.

5.3 Valutazione visiva dei test: punti di debolezza

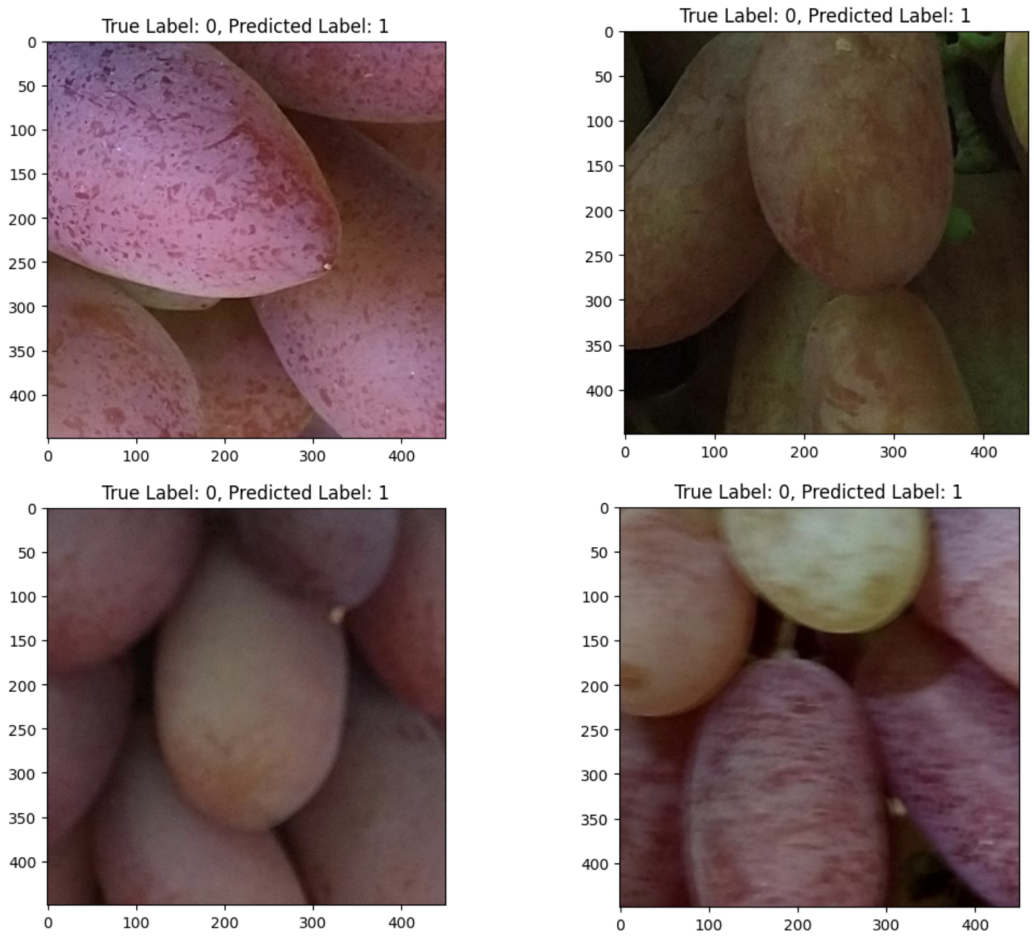


Figura 5.5. Esempi di falsi positivi

Rivolgendo l'attenzione sui falsi positivi, vale a dire i chicchi d'uva sani erroneamente identificati come danneggiati dal nostro modello, emergono alcune caratteristiche visive che destano particolare interesse. In questa categoria di campioni, osserviamo che, sotto il punto di vista cromatico, presentano una connotazione simile a quella dei chicchi danneggiati, manifestando colorazioni più scure con una notevole inclinazione verso tonalità violacee. Questa somiglianza cromatica tra i chicchi danneggiati e i falsi positivi potrebbe costituire una fonte di confusione per il modello, poiché la variazione cromatica è una delle caratteristiche determinanti per la classificazione.

Inoltre, va sottolineato che spesso le immagini corrispondenti ai falsi positivi presentano una notevole sfocatura o mosso, il che induce inevitabilmente una texture più frastagliata a livello dei pixel. Questo fenomeno, in cui le immagini sono affette da un'acquisizione difettosa o da un movimento durante la cattura, contribuisce a rendere la texture dei chicchi visivamente più ruvida e disomogenea. Tale aspetto può rappresentare una sfida significativa per il modello, in quanto potrebbe indurlo a interpretare erroneamente le irregolarità causate dalla sfocatura o dal movimento come segni di danneggiamento effettivo.

Figura 5.6. Esempi di falsi negativi

Per quanto concerne i falsi negativi, ossia situazioni in cui chicchi danneggiati vengono erroneamente classificati come sani, è essenziale osservare alcuni tratti distintivi nei dati visivi. In particolare, si può notare la mancanza di striature decise e contrasti accentuati tipici dei chicchi danneggiati nelle immagini soggette a tale errore di classificazione. Questi segni visivi, che normalmente caratterizzano i chicchi danneggiati, risultano essere poco evidenti o assenti nelle immagini dei chicchi catalogati erroneamente come sani.

Inoltre, un'altra situazione che contribuisce ai falsi positivi è la presenza di chicchi danneggiati di dimensioni relativamente ridotte, i quali possono trovarsi inglobati all'interno di un contesto di chicchi sani predominanti nell'immagine. Questa circostanza può ingannare il sistema di classificazione, portando all'identificazione errata di tali chicchi danneggiati come sani a causa della loro presenza limitata e della difficoltà nell'individuarli in mezzo alla maggioranza di chicchi in condizioni ottimali.

5.4 Indagine quantitativa sulle prestazioni

È essenziale sottolineare che tali osservazioni sono basate su un'analisi visiva delle immagini e, quindi, rappresentano solamente una prima fase nell'indagine delle prestazioni del modello. Per confermare l'efficacia delle caratteristiche identificate e valutare in modo completo la capacità di classificazione del modello rispetto alla categoria dei chicchi danneggiati, aggiungo anche dei dati quantitativi:

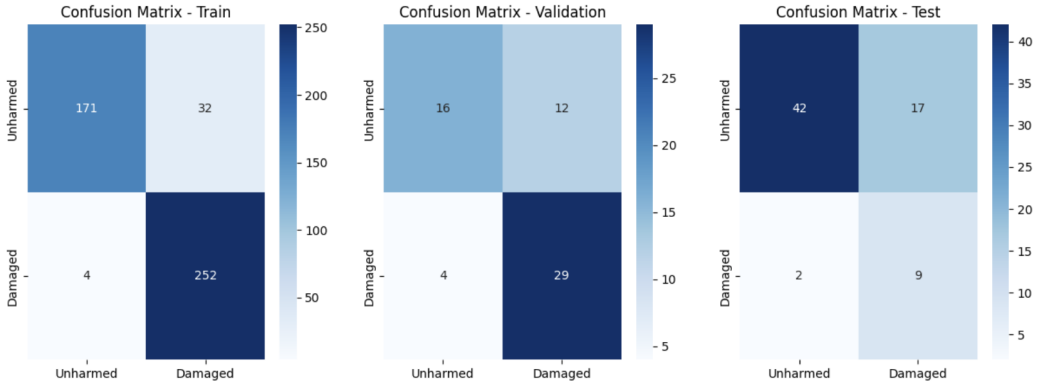


Figura 5.7. Risultati - Confusion matrix

Osservando le matrici di confusione, in particolare con riferimento al set di convalida, si evidenzia un quadro interessante che richiede un'analisi più approfondita. Inizialmente, è importante notare che il modello sembra incontrare una maggiore difficoltà nel processo di classificazione dei chicchi sani. Ciò è in parte attribuibile ai falsi positivi, in cui chicchi danneggiati vengono erroneamente identificati come sani. Questi falsi positivi possono essere ricondotti, come descritto precedentemente, alla mancanza di striature decise e contrasti accentuati, caratteristiche comuni dei chicchi danneggiati, nelle immagini dei chicchi catalogati erroneamente come sani. Questo fenomeno evidenzia la complessità intrinseca nella distinzione tra chicchi sani e danneggiati, soprattutto quando i tratti distintivi sono meno evidenti.

Tuttavia, è degno di nota che il modello dimostra una buona capacità di classificazione dei chicchi danneggiati, incluso il riconoscimento efficace di tali chicchi in immagini non precedentemente osservate. Questo suggerisce che il modello è in grado di generalizzare le caratteristiche dei chicchi danneggiati apprese durante l'addestramento e applicarle con successo a nuovi dati.

Capitolo 6

Conclusioni

I risultati ottenuti dalla mia ricerca rivelano una discrepanza significativa tra le prestazioni del modello utilizzato nel mio studio e il modello di riferimento descritto nel paper di riferimento. Nonostante l'adozione dello stesso modello, l'obiettivo, nonché difficoltà principale, del mio studio è stato comprendere le cause di questa differenza nell'efficacia del modello. Durante l'analisi, ho condotto una serie di esperimenti volti a migliorare le prestazioni, tra cui l'applicazione di tecniche di data augmentation e l'ottimizzazione dell'inizializzazione dei pesi del modello, e nonostante tali sforzi, le prestazioni del modello rimangono inferiori rispetto al modello di riferimento.

Dopo una meticolosa esplorazione delle cause di questa discrepanza, ho identificato il dataset come un fattore critico che influisce sulle prestazioni del modello. Nel mio studio, il dataset fornito presenta caratteristiche lievemente diverse ma significative rispetto a quello utilizzato nel paper di riferimento. In particolare, ho notato che le patches di immagini nel mio dataset spesso contengono entrambe le classi di chicchi di uva studiate, cioè sia chicchi sani che danneggiati, e ciò è dovuto al non preciso centramento del chicco preso in considerazione. Questo contrasta con il dataset utilizzato nel paper di riferimento, in cui ciascuna immagine conteneva un singolo chicco, centrato, senza altre distrazioni visive.

La presenza di patch contenenti entrambe le classi di chicchi rende il processo di addestramento del modello più complesso e suscettibile a una maggiore variabilità. Questa variabilità può comportare una minore precisione nelle previsioni del modello, in quanto deve imparare a discriminare tra due classi diverse all'interno della stessa immagine. La natura aleatoria di questa disposizione può comportare una maggiore difficoltà nel catturare le caratteristiche discriminanti dei chicchi sani e danneggiati, contribuendo così alla discrepanza osservata nelle prestazioni rispetto al modello di riferimento.

In conclusione, i risultati del mio studio indicano che la differenza nelle prestazioni tra il modello utilizzato e il modello di riferimento è in gran parte attribuibile alla natura del dataset e alla qualità delle patches fornite con le relative etichette. Que-

sta comprensione è fondamentale per l'ulteriore sviluppo del modello e suggerisce la necessità di riconsiderare la raccolta dei dati e la loro preparazione al fine di migliorare le prestazioni del modello nelle future iterazioni della ricerca.

Bibliografia

[1] A Collaborative Paradigm for Human Workers and Multi-Robot Teams in Precision Agriculture Systems,
https://cordis.europa.eu/project/id/101016906/it

[2] Automatic Differentiation of Damaged and Unharmed Grapes Using RGB Images and Convolutional Neural Networks,
https://link.springer.com/chapter/10.1007/978-3-030-65414-6_24

[3] Anomaly Detection on Wine Grapes using CNN,
https://github.com/Dr3dre/Bachelor-Degree/blob/main/Anomaly_Detection_on_Wine_Grapes.ipynb

[4] Google Colaboratory,
https://colab.google/

[5] Pytorch,
https://pytorch.org/

[6] Matplotlib: Visualization with Python,
https://matplotlib.org/

[7] Seaborn: statistical data visualization,
https://seaborn.pydata.org/

[8] GradientBased Learning Applied to Document Recognition,
http://vision.stanford.edu/cs598_spring07/papers/Lecun98.pdf

[9] Deep Residual Learning for Image Recognition,
https://arxiv.org/abs/1512.03385

[10] Xavier Initialization,
https://proceedings.mlr.press/v9/glorot10a/glorot10a.pdf