UNIVERSITÀ POLITECNICA DELLE MARCHE

FACOLTÀ DI INGEGNERIA

Corso di Laurea Triennale in
Ingegneria Informatica e dell'Automazione

Sviluppo di un algoritmo di deep learning per il riconoscimento della septoria nell'agricoltura di precisione

Development of a deep learning algorithm for septoria recognition in a precision agriculture context

Relatore:
Dott. Mancini Adriano

Laureando:
Massimo Ciaffoni

Anno Accademico 2019-2020

Indice

Capitolo 1

Introduzione

La **septoriosi** è una malattia che colpisce il frumento causata da due diversi agenti fungini: *Septoria tritici* e *Stagonospora nodorum*; la prima predilige le foglie mentre la seconda anche i nodi e le spighe [1]. La **sintomatologia** della malattia è rappresentata dalla formazione di lesioni sull'apparato fogliare che successivamente necrotizzano e determinano il disseccamento della lamina fogliare. Sulle lesioni fogliari successivamente appaiono i picnidi (figura 1.1), corpi fruttiferi del fungo, di forma rotondeggiante e colore nero, contenenti i conidi che rappresentano gli elementi di propagazione dell'infezione [2]. La presenza dei picnidi costituisce un importante **elemento diagnostico** della malattia poiché i sintomi relativi al complesso della septoriosi possono essere facilmente confusi con quelli relativi ad altre patologie fungine.

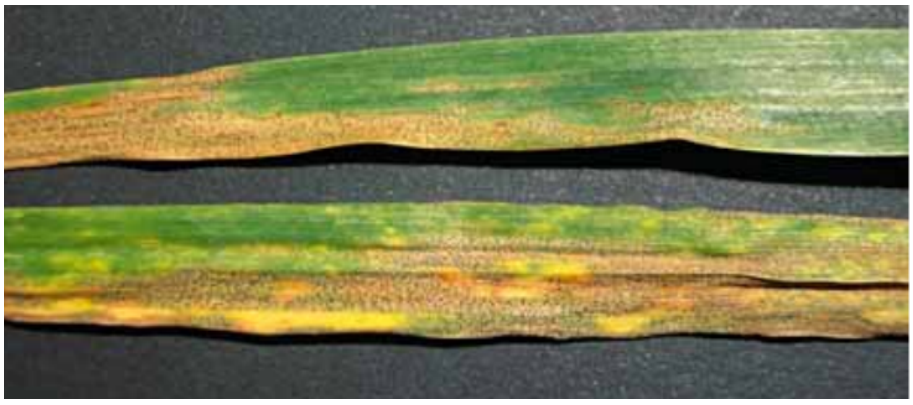

Figura 1.1 – Lesioni di Septoria tritici su foglia e picnidi del patogeno [2]

Nella coltivazione dei cereali, attacchi di malattie fogliari come septoriosi possono determinare significative perdite di produzione **fino al 50 %** [3]. È pertanto essenziale mantenere le foglie sane quanto più a lungo possibile al fine di massimizzare lo sviluppo delle colture e per poter sfruttare il potenziale produttivo della varietà prescelta.

1.1 Artificial Intelligence nell'agricoltura di precisione

Negli ultimi anni l'applicazione dell'analisi ad apprendimento automatico in ambito agricolo sta portando enormi giovamenti alle prime grandi aziende del settore che le stanno adottando. L'evoluzione tecnologica all'interno del settore è iniziata intorno agli anni 90 grazie a tecnologie satellitari, GPS (Global Positioning System) e software sui macchinari che hanno permesso il diffondersi della cosiddetta **agricoltura di precisione**, intesa come "*una strategia che usa le tecnologie d'informazione per integrare dati provenienti da più strati informativi ai fini decisionali per la gestione dei sistemi agricoli*" (National Research Council americano, 1997) [4]. Ad oggi, questa sfida, ha subito uno sviluppo ed un miglioramento grazie all'AI (Artificial Intelligence). Essa è intesa come la possibilità di utilizzare l'analisi incrociata di fattori ambientali, climatici e colturali per stabilire il fabbisogno irriguo e nutritivo delle coltivazioni, prevenendo patologie ed identificando possibili infestazioni prima che proliferino [5]. Ad esempio è possibile realizzare algoritmi in grado di riconoscere il livello di maturazione dei pomodori (figura 1.2).

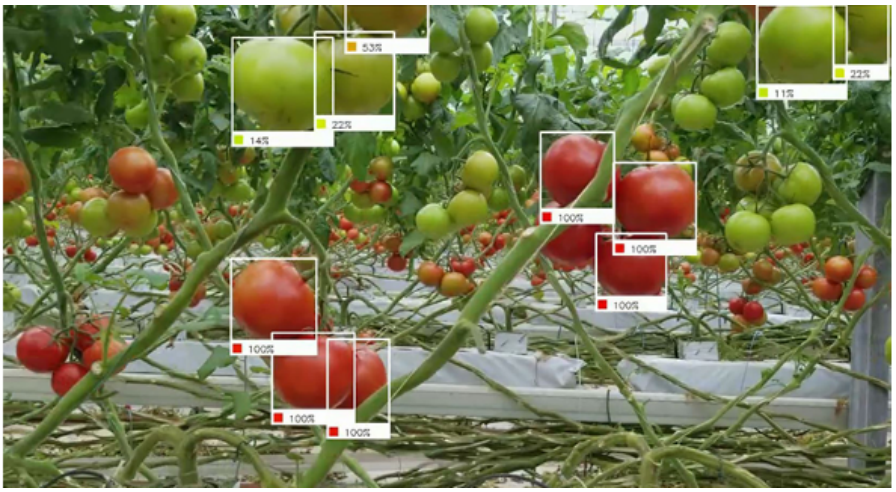

Figura 1.2 – Esempio di rilevazione della maturazione di pomodori [5]

Tutto ciò è possibile grazie allo sviluppo di nuove tecniche di **deep learning** che permettono di realizzare sensori e software in grado di gestire ed analizzare in tempo reale le colture. I moderni **sensori** di telerilevamento, innestati nel terreno possono identificare le erbacce e le altre piante infestanti, consentendo di individuare quali specifici prodotti applicare in ogni singola zona interessata. Questo trattamento a zone, di fatto, serve per minimizzare e ottimizzare l'utilizzo di prodotti, non sempre benefici per il nostro organismo. Altre tecniche di monitoraggio, invece, fanno uso di **droni** muniti di telecamere integrate con software in grado di analizzare quasi in tempo reale le colture di un azienda agricola ed individuare le eventuali aree problematiche.

1.2 Il Progetto

Sulla base di quanto detto fin ora, l'obiettivo di questa tesi è di realizzare due diversi modelli di **rete neurale** che siano in grado di riconoscere, da una serie di immagini del frumento, la presenza di septoriosi. L'addestramento dei modelli è stato effettuato mediante due reti pre-addestrate (che saranno descritte nello specifico più avanti).

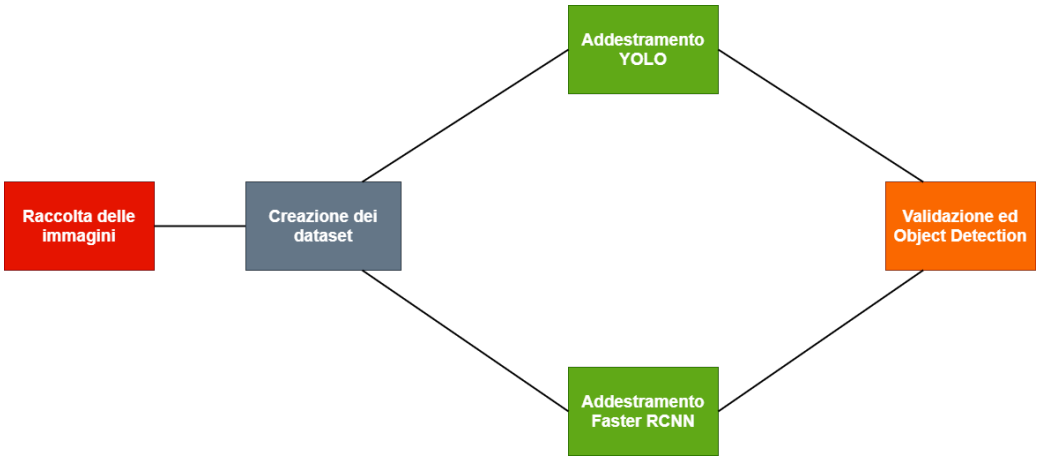

Figura 1.3 – Schema della progettazione delle reti neurali

Lo sviluppo e la progettazione delle reti è avvenuta in diverse fasi:

1. **Raccolta delle immagini**: Nella prima fase di progetto si è cercato di raccogliere il maggior numero di immagini di septoriosi del grano, le quali successivamente sono state divise in tre distinti dataset (train, validation, test).

2. **Creazione dei dataset**: Uno degli elementi fondamentali per il *machine learning* è la disponibilità di dataset adeguati. Maggiore è la loro dimensione e qualità, migliore è l'accuratezza dei modelli risultanti. Per questo motivo il numero delle immagini raccolte è stato ulteriormente aumentato grazie a tecniche di *data augmentation*. Una volta ottenuti i dataset desiderati, le immagini dei rispettivi dataset di training e validation sono state etichettate tramite un' applicazione di labellizzazione.

3. **Addestramento**: Una volta etichettato il dataset di training ed esportato le etichette in formato JSON (JavaScript Object Notation) le immagini sono state utilizzate come dati di input a due diverse reti neurali:

   - **YOLO** [6]
   - **Faster RCNN** [7]

4. **Test e Validation**: una volta terminata la fase di addestramento ed ottenuto i due modelli, essi sono stati utilizzati per effettuare delle previsioni sulle immagini del dataset di test, dove sono presenti anche immagini di

piante sane. Successivamente le previsioni sono state effettuate anche nel dataset di validation in modo da confrontare le etichette previste con le etichette effettive in modo da valutare le prestazioni dei modelli. Una volta terminata la fase di validation viene generato un grafico precision/recall per entrambi.

1.3 Struttura della Tesi

La tesi è strutturata in modo da ripercorrere punto per punto tutti i vari passaggi effettuati durante la realizzazione dei due modelli di rete neurale:

- nel secondo capitolo sono descritte nello specifico tutte le tecnologie utilizzate nella realizzazione dei modelli di rete.
- Nel terzo capitolo viene descritto in generale il *deep learning* ed in particolare il funzionamento delle reti neurali. Successivamente sono introdotti e descritti nello specifico i due modelli di rete e le principali metriche utilizzate per valutarne le prestazioni.
- Nel quarto capitolo sono ripercorse le varie fasi della realizzazione dei modelli del progetto.
- Nel quinto capitolo sono analizzati i risultati ottenuti, le varie limitazioni dei modelli addestrati e sono proposti diversi sviluppi futuri del software.

Capitolo 2

Tecnologie utilizzate nel progetto

2.1 Python

Figura 2.1 – Logo del linguaggio di programmazione Python [8]

Python è un linguaggio di programmazione di alto livello orientato agli oggetti e multi-paradigma. Sebbene Python venga in genere considerato un linguaggio interpretato, il codice sorgente non viene convertito direttamente in linguaggio macchina. Infatti passa prima da una fase di pre-compilazione in bytecode evitando così di reinterpretare ogni volta il sorgente e migliorando le prestazioni. Inoltre è possibile distribuire programmi Python direttamente in bytecode, saltando totalmente la fase di interpretazione da parte dell'utilizzatore finale e ottenendo programmi Python a sorgente chiuso [8]. La sua semplicità di sintassi e modularità hanno reso Python ad oggi uno dei linguaggi di programmazione più utilizzato dagli sviluppatori di tutto il mondo (figura 2.2).

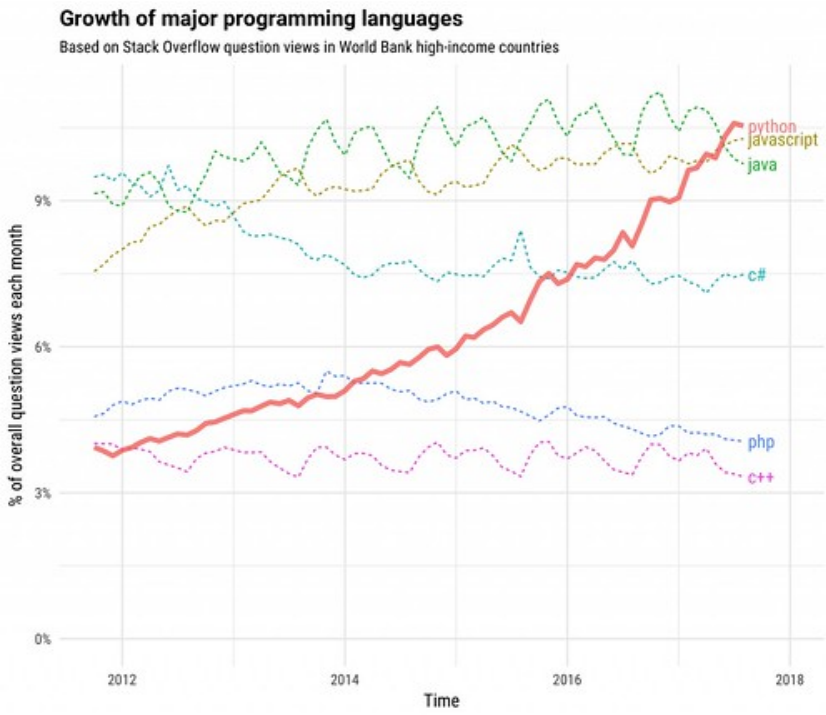

Figura 2.2 – Andamento Python [9]

Python, infatti rappresenta spesso la lingua preferita dagli sviluppatori e data scientist che hanno bisogno di applicare tecniche statistiche o di analisi dei dati nel loro lavoro. La combinazione di sintassi coerente, i tempi di sviluppo più brevi e la flessibilità rendono tale linguaggio adatto allo sviluppo di modelli di previsione sofisticati che possono essere collegati direttamente ai sistemi di produzione ed apprendimento automatico. Questo è possibile anche grazie all'ampio set di librerie, ossia insiemi di routine e funzioni scritte che svolgono un determinato compito, che lo sviluppatore può richiamare a seconda delle necessità.

2.2 Tensorflow

Tensorflow è una libreria open-source ed un'interfaccia di programmazione scalabile e multi-piattaforma per l'implementazione e l'esecuzione di algoritmi di apprendimento automatico sviluppata da Google Brain, team di Google specializzato nel Machine Learning e Intelligenza Artificiale [10]. La libreria è composta da una serie di API (Application Programming Interface) di alto livello scritte in linguaggio Python progettate con l'obbiettivo di realizzare ed unificare in maniera semplice modelli di apprendimento automatico. L'architettura di TensorFlow [11] è strutturata su diversi livelli di astrazione (figura 2.3):

- La maggior parte dei calcoli di deep learning sono codificati in C ++. Per eseguire operazioni sulla GPU, TensorFlow utilizza una libreria sviluppata da NVIDIA chiamata CUDA.

- L'API di basso livello Python quindi utilizza il codice sorgente C ++. Quando un metodo Python viene utilizzato in TensorFlow, di solito richiama il codice sorgente. Questo strato wrapper consente agli utenti di lavorare più velocemente in quanto Python è considerato più facile da usare rispetto a C ++ e non lo fa richiedendo la compilazione.
- L'API di alto livello presenta due componenti: Keras e l'API Estimator. Keras è un wrapper intuitivo, modulare ed estensibile per TensorFlow. L'API Estimator invece contiene diversi componenti predefiniti che consentono di creare facilmente modelli di machine learning.

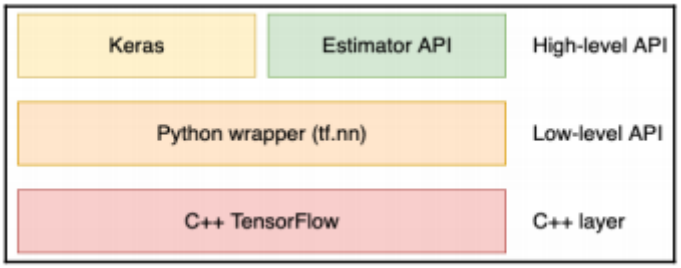

Figura 2.3 – Diagramma dell'architettura di Tensorflow [11]

Tensorflow si basa sul concetto di **tensore**. Un tensore è un oggetto matematico utilizzato per rappresentare un array N-dimensionale. Matematicamente, i tensori possono essere intesi come una generalizzazione di scalari, vettori, matrici (figura 2.4). Più concretamente, uno scalare può essere definito come un tensore di rango 0, un vettore può essere definito come un tensore di rango 1, una matrice può essere definita come un tensore di rango 2 e così via [12].

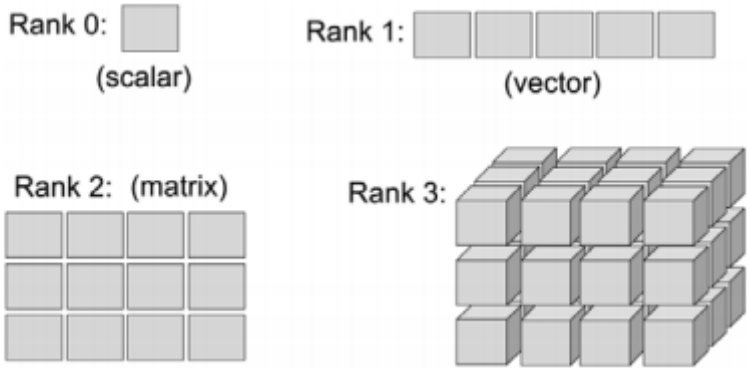

Figura 2.4 – Rappresentazione di tensori dal rango 0 al 3 [12]

TensorFlow utilizza i tensori sia come input che come output. Le componenti di input sono trasformate in output attraverso delle **operazioni**, le quali sono rappresentate da un **grafo aciclico diretto** (DAC), un grafo di calcolo composto da un insieme di nodi (figura 2.5). Ogni nodo rappresenta un'operazione che può avere zero o più input o output.

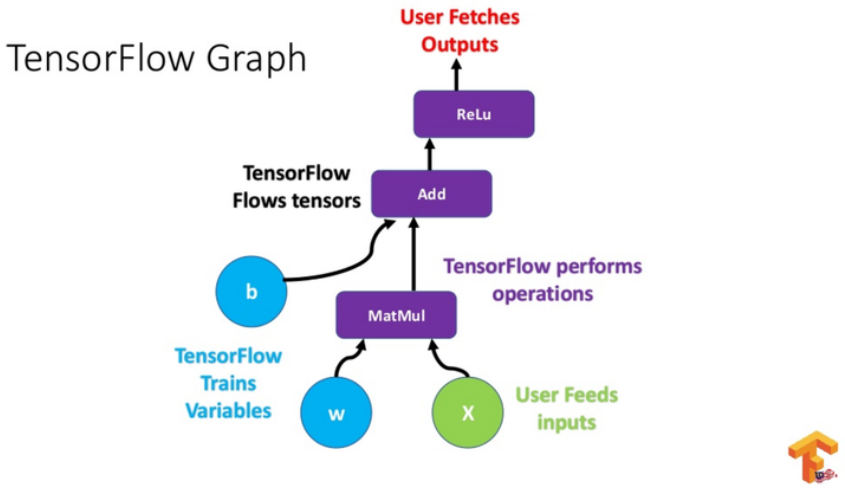

Figura 2.5 – Esempio di un grafo Tensorflow [13]

2.2.1 Tensorboard

Tensorboard è un toolkit di visualizzazione sviluppato per Tensorflow. Esso offre uno strumento grafico in grado di fornire le misurazioni e le visualizzazioni necessarie durante il flusso di lavoro di machine learning. Consente il monitoraggio delle metriche come la perdita e l'accuratezza, la visualizzazione del grafico del modello, la proiezione di incorporamenti in uno spazio dimensionale inferiore e molto altro ancora [14].

2.3 JSON

JSON (JavaScript Object Notation) è un formato testuale utilizzato per la strutturazione di dati. Esso è completamente indipendente dal linguaggio di programmazione, ma utilizza diverse loro convenzioni in modo da aggregare dati (stringhe, numeri, etc.) per creare informazioni di più alto livello. Questa caratteristica fa di JSON un linguaggio ideale per lo scambio di dati [15].

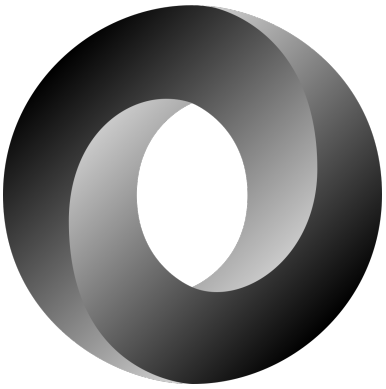

Figura 2.6 – Logo del formato JSON [15]

JSON è basato su **due strutture**:

- un insieme di coppie nome/valore. In diversi linguaggi, questo è realizzato come un oggetto, un record, uno struct, un dizionario, una tabella hash, un elenco di chiavi o un array associativo.
- Un elenco ordinato di valori realizzati mediante array, elenchi o sequenze.

Virtualmente tutti i linguaggi di programmazione moderni supportano le due strutture in entrambe le forme. É sensato che un formato di dati che è interscambiabile con linguaggi di programmazione debba essere basato su queste strutture. In JSON, assumono queste forme:

- Un oggetto è una serie non ordinata di nome/valore racchiuso da delle parentesi graffe. Ogni nome è seguito da due punti mentre le coppie nome/valore sono separate da una virgola.
- Un array è una raccolta ordinata di valori racchiusi dentro delle parentesi quadre e separate da virgola.
- Un valore può essere una stringa tra virgolette, un numero, un valore booleano, un valore nullo, un oggetto o un array. Queste strutture a loro volta possono essere annidate.
- Una stringa è una raccolta di zero o più caratteri UNICODE, tra virgolette.

2.4 XML

XML (eXtensible Markup Language) è un metalinguaggio sviluppato da World Wide Web Consorsium (WC3) per la definizione di linguaggi di markup. Rispetto all'HTML, l'XML ha uno scopo ben diverso: mentre il primo definisce una grammatica per la descrizione e la formattazione di pagine web (layout) e, in generale, di ipertesti, il secondo è un metalinguaggio utilizzato per creare nuovi linguaggi, atti a descrivere documenti strutturati [16]. Concretamente, un documento XML è un file di testo che contiene una serie di tag, attributi e testo secondo regole sintattiche ben definite.
Il linguaggio XML è inoltre indipendente dalla piattaforma, ossia qualsiasi applicazione progettata per usarlo consente di leggere ed elaborare dati XML, indipendentemente dall'hardware o dal sistema operativo. Con i tag XML appropriati è ad esempio possibile usare un'applicazione per PC desktop per aprire e gestire i dati provenienti da un computer mainframe. Indipendentemente dall'autore di un testo di dati XML, è inoltre possibile usare gli stessi dati in diverse applicazioni.

2.5 Labelbox

Molti modelli di apprendimento automatico richiedono dati di addestramento i quali devono essere etichettati. Immagini e testo non elaborati possono essere etichettati utilizzando piattaforme come Labelbox [17]. **Labelbox** è un sistema di strumenti di etichettatura che consente di creare dataset pronti per essere utilizzati dagli algoritmi di apprendimento automatico.

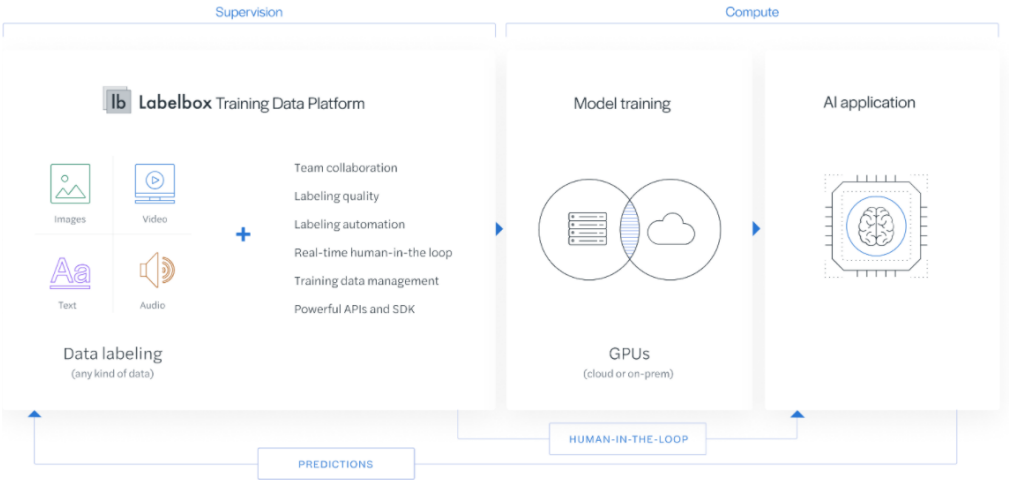

Figura 2.7 – Architettura Labelbox [17]

Labelbox è uno strumento ideale per gli sviluppatori di modelli di machine learning che vogliono etichettare testo, audio, immagini o video. Per quanto riguarda le immagini esse possono essere etichettate con diversi strumenti [18]:

- **Segmentation**: viene utilizzato per etichettare immagini che saranno utilizzate per creare applicazioni con elevata precisione.

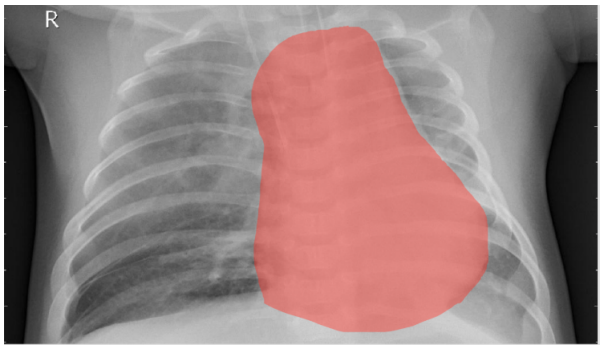

Figura 2.8 – Esempio di segmentation [17]

- **Bounding Box**: viene utilizzato per etichettare le immagini con annotazioni in 2D.
- **Polygon**: viene utilizzato per creare etichette geometriche disegnando delle linee fra più punti.

- **Polyline**: viene utilizzato per creare annotazioni continue costituite da uno o più segmenti.
- **Point**: viene utilizzato per annotare una singola coordinata x y dell'immagine.

Figura 2.9 – Esempi di Bounding Box, Polygon, Polyline e Point [17]

2.6 Google Colab

Figura 2.10 – Logo di Google Colab [19]

Nello sviluppo di applicazioni di machine learning molta potenza di calcolo è utilizzata per l'implementazione ed addestramento dei modelli di rete i quali devono risultare robusti ed efficienti. Per le prime fasi di test può essere sufficiente lavorare sulla propria macchina, ma con il crescere delle dimensioni del dataset la potenza necessaria per il training dei modelli può aumentare a dismisura ed essere proibitiva. Per aiutare data scientist e data analyst in questa attività esistono numerosi servizi cloud che offrono potenza di calcolo sul mercato. **Google Colab** è una piattaforma che ci permette di eseguire codice direttamente sul Cloud in maniera gratuita [19]. Per eseguire codice, Google Colab sfrutta i cosiddetti **Jupyter Notebook**, dei documenti interattivi dove è possibile suddividere ed eseguire il codice in celle le quali possono contenere anche testo informativo in formato Markdown [20]. Il notebook Jupiter verrà poi eseguito su macchine virtuali di server Google. Ciò consente agli sviluppatori di svincolarsi dalla parte hardware e di concentrarsi solamente sul codice Python e sui contenuti che si vogliono integrare nel notebook. Le macchine virtuali messe a

disposizione in Google Colab ospitano un ambiente configurato che consente di concentrarsi sin da subito sui progetti di Data Science: sono presenti numerose librerie Python, tra cui moltissime di Data Science come Keras e Tensorflow. É possibile configurare la macchina in cui verrà eseguito il nostro codice Python abilitando il supporto all'uso della GPU. Sotto la voce Hardware acceleration, possiamo infatti selezionare GPU, sfruttando la potenza di calcolo parallelo fornita da questo tipo di hardware o in alternativa potremmo decidere di sfruttare le Tensor Processing Unit (TPU), uno strumento hardware special purpose creato da Google proprio per rendere più efficiente e veloce l'addestramento di reti neurali (figura 2.11).

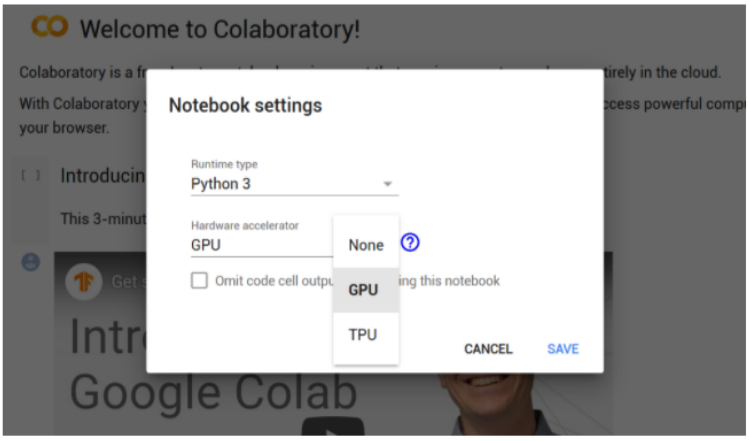

Figura 2.11 – Configurazione accelerazione hardware Google Colab

Importare i dati nel notebook è molto semplice: è possibile fare un upload dei dati manualmente oppure è possibile utilizzare i file presenti sul nostro Google Drive. In Google Colab le risorse che vengono messe a disposizione agli utenti sono limitate e variano a seconda delle fluttuazioni nella domanda. Se si ha bisogno di maggior stabilità, di macchine più performanti o di accedere a GPU e TPU più potenti, è possibile utilizzare la licenza a pagamento disponibile nella versione Pro.

Capitolo 3

Machine Learning e Deep Learning

Il **Machine Learning** è una branca dell'Intelligenza Artificiale (figura 3.4) che si basa su un nuovo approccio di programmazione. Nella programmazione classica i programmatori inseriscono delle regole (algoritmi) e i dati di input da elaborare in base a queste regole, e si ottengono dei dati di output. Con il machine learning questo concetto si ribalta, i programmatori inseriscono i dati di input (*features*) e le risposte attese in base a questi dati (*labels*), ed il calcolatore individua le regole. Queste regole poi possono essere applicate a nuovi dati per produrre altre risposte, originali [21].

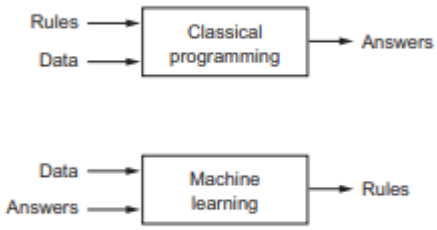

Figura 3.1 – Approccio di programmazione del Machine learning [21]

Un sistema di machine learning dunque non viene programmato ma bensì **addestrato**. É possibile classificare i sistemi di machine learning in base alle proprie funzionalità [22] :

1. **Apprendimento supervisionato**: il sistema viene costruito utilizzando dei dati di addestramento etichettati, i quali saranno utilizzati per fare previsioni su nuovi dati non disponibili nel dataset iniziale. La supervisione è data dal fatto che ad ogni campione del nostro dataset di addestramento è associato un output desiderato tramite un' etichetta.
   Questo tipo di apprendimento è principalmente utilizzato in due diverse tecniche:

   - *Classificazione*: è una tecnica in cui l'obiettivo è quello di prevedere, in base all'analisi dei dati precedentemente etichettati, l'etichettatu-

ra di nuovi dati. Le etichette sono valori discreti non ordinati che possono essere considerati appartenenti ad una determinata classe. Ad esempio nella figura 3.2 è rappresentata una classificazione binaria dove si possono distinguere due tipologie di classi. Utilizzando un algoritmo d'apprendimento supervisionato saremo in grado di separare le due classi con un confine decisionale e ad associare i dati, sulla base dei loro valori, alle due diverse categorie.

- *Regressione*: è una tecnica in cui si dispone di un numero di variabili predittive e una variabile target continua. L'algoritmo di apprendimento automatico cerca di trovare una relazione tra queste due variabili al fine di prevedere un risultato. Ad esempio nella figura 3.2 data una variabile predittiva x e una di risposta y, viene disegnata una retta al fine di diminuire la distanza far i punti e la retta stessa.

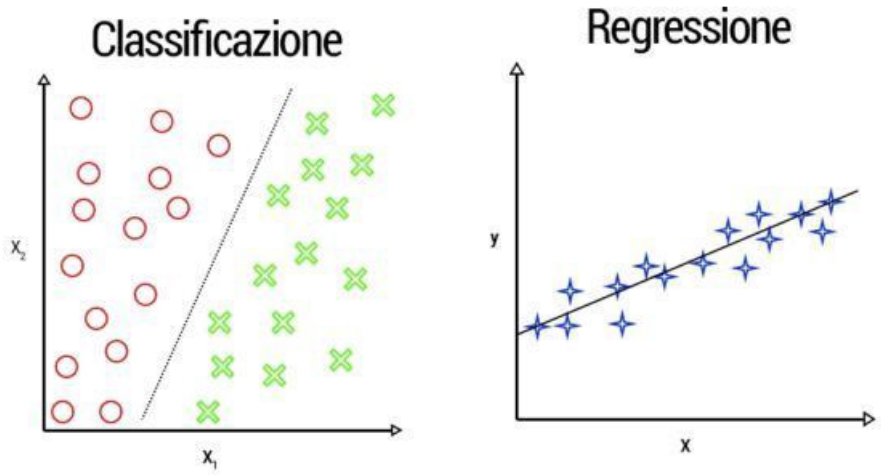

Figura 3.2 – Esempi di classificazione e regressione [22]

2. **Apprendimento non supervisionato**: al contrario dell'apprendimento supervisionato i dati in questo caso non sono etichettati. L'algoritmo osservando la struttura dei dati è in grado di estrapolare delle determinate caratteristiche significative che li descrivono. L'apprendimento non supervisionato è spesso utilizzato in due tecniche:

   - *Clustering*: è una tecnica esplorativa che consente di aggregare dei dati di cui non abbiamo alcuna conoscenza in determinati gruppi (detti cluster). Dunque partendo da un dataset di grandi dimensioni è possibile raggruppare i campioni in cluster in base a degli elementi simili fra di loro. All'interno di ogni cluster, infatti troveremo quei dati che hanno caratteristiche simili tra di loro (figura 3.3). Il clustering è dunque un' ottima tecnica che permette di scovare delle relazioni tra i dati.
   - *Riduzione dimensionalità dei dati*: è un approccio utilizzato nella preelaborazione delle caratteristiche dei dati con l'obiettivo di eliminare

il "rumore". Grosse quantita di dati possono generare un problema di memorizzazione e una diminuzione delle prestazioni a livello computazionale. La riduzione dei dati rende lo spazio dimensionale più compatto al fine di mantenere le informazioni più rilevanti. La riduzione della dimensionalità può essere utile anche per la rappresentazione dei dati, come ad esempio all'interno di un feature space (spazio delle caratteristiche) a elevata dimensionalità, che possono essere così proiettati su uno spazio di minore dimensione (figura 3.3).

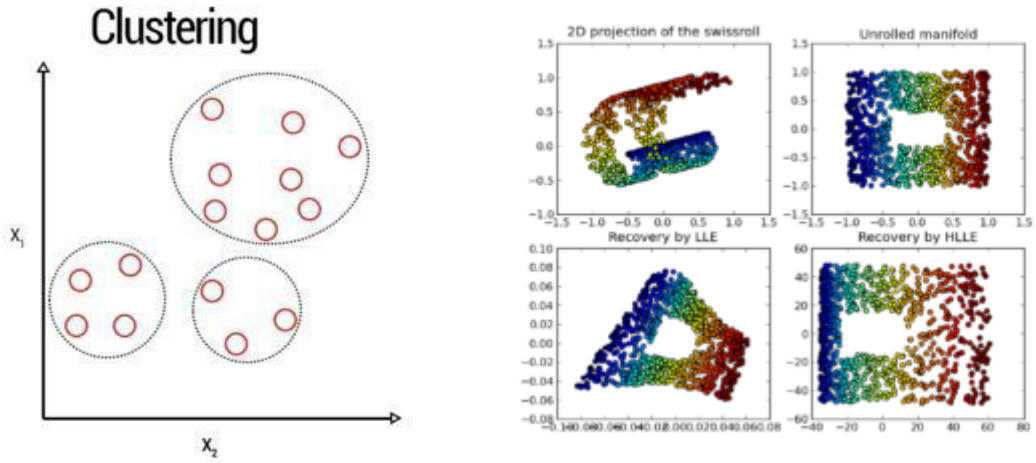

Figura 3.3 – Esempio di clustering e di riduzione dimensionalità dei dati [22]

3. **Apprendimento semi-supervisionato**: è un modello "ibrido" dove al calcolatore viene fornito un set di dati incompleti per l'addestramento; alcuni di questi input presentano i rispettivi esempi di output (come nell'apprendimento supervisionato), altri invece ne sono privi (come nell'apprendimento non supervisionato). L'obiettivo, di fondo, è sempre lo stesso: identificare regole e funzioni per la risoluzione dei problemi, nonché modelli e strutture di dati utili a raggiungere determinati obiettivi.

4. **Apprendimento per rinforzo**: L'obiettivo di questo tipo di apprendimento è quello di costruire un sistema (agente) che attraverso le interazioni con l'ambiente migliori le proprie performance. Per poter migliorare le funzionalità del sistema vengono introdotti dei rinforzi, ovvero dei segnali di ricompensa. Questo rinforzo non è dato dalle label (etichette) o dai valori corretti di verità, ma è una misurazione della qualità delle azioni intraprese dal sistema. Per questo motivo tale apprendimento non può essere assimilato ad un apprendimento supervisionato.

All'interno del machine learning distinguiamo una sua sotto-branca chiamata **Deep Learning** (figura 3.4) la quale rappresenta e trasforma i dati attraverso l'utilizzo di *layer* successivi, con rappresentazioni sempre più significative.

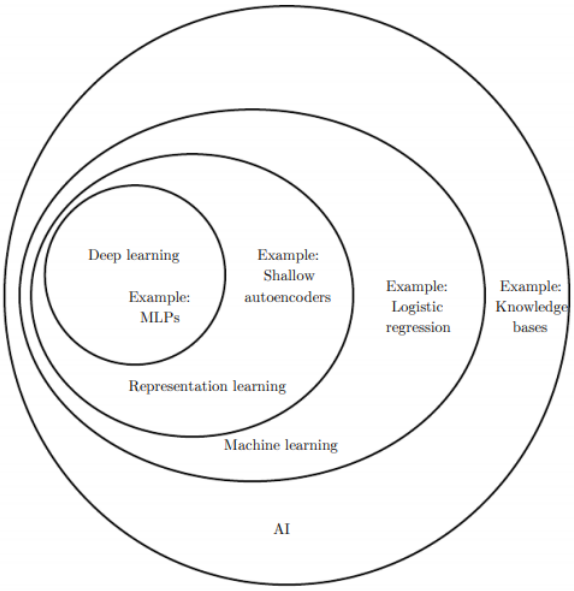

Figura 3.4 – Classificazione dell'Intelligenza Artificiale [23]

3.1 Le Reti Neurali

Nel Deep Learning le rappresentazioni a layer sono apprese tramite modelli chiamati **reti neurali artificiali**, strutturate attraverso livelli sovrapposti l'uno all'altro, come strati. Questi layer imitano il modello di un neurone biologico e sono alla base dell'apprendimento di un modello ML.

3.1.1 Struttura di una rete neurale

Una rete neurale può essere immaginata come composta da diversi layer (strati) di nodi, ciascuno dei quali è collegato ai nodi del layer successivo. Possiamo considerare la rete neurale come una scatola nera con degli input, degli strati intermedi e degli output [24] . Tutte le reti neurali infatti sono composte da almeno tre strati (figura 3.5) :

- Un **input layer**, ovvero i dati di ingresso.
- Uno o più **hidden layer**, dove i dati sono elaborati.
- Un **output layer**, il quale contiene il risultato finale.

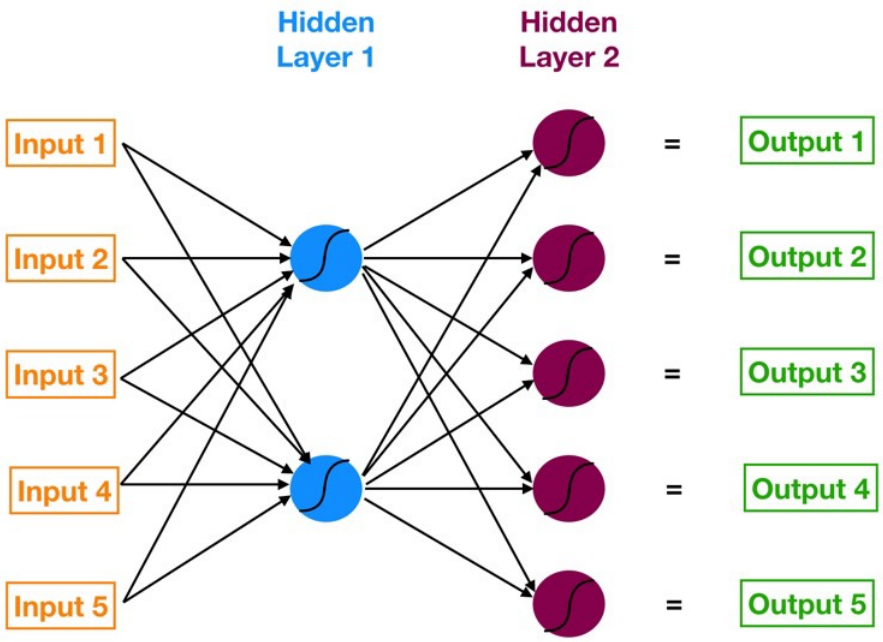

Figura 3.5 – Esempio di rete neurale [25]

La rete è composta da unità chiamate neuroni (figura 3.6) i quali tipicamente sono collegati a tutti i neuroni dello strato successivo tramite connessioni pesate. Le specifiche di ciò che un layer fa ai suoi dati di input, infatti, sono conservate nei ***weights***. Successivamente ciascun neurone somma tutti i valori pesati dei neuroni ad esso collegati ed aggiunge un ***bias***. A questo risultato viene infine applicata una ***funzione di attivazione*** che trasforma matematicamente il valore calcolato prima di passarlo allo strato successivo.

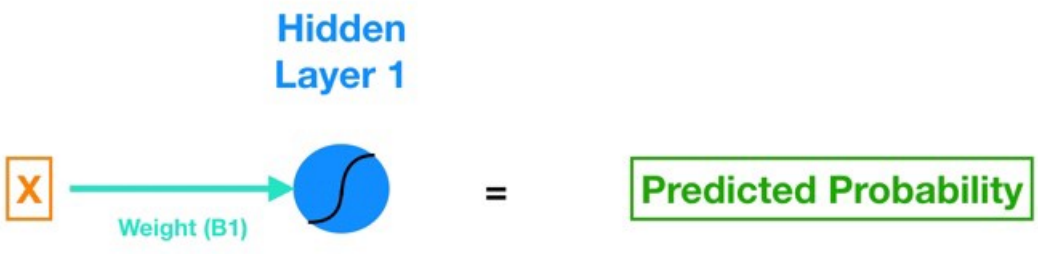

Figura 3.6 – Esempio di un singolo neurone artificiale [25]

Dunque matematicamente le operazioni svolte dal neurone sono semplici funzioni con un ingresso ed un'uscita (figura 3.7):

- **X** è l'input, che può essere un elemento del dataset o un uscita di un layer precedente.
- **B1** è il weight, il quale indica quanto il parametro di input "pesa" sulla predizione di uscita.

- **B0** è il bias, il quale indica quanto il neurone "pesa" rispetto a tutti gli altri della rete.
- **Sigmoid** è la funzione di attivazione.

Figura 3.7 – Operazioni matematiche svolte da un neurone artificiale [25]

3.1.2 Funzioni di attivazione

L'obiettivo del Deep Learning è quello di regolare pesi e bias in modo da ottenere il risultato desiderato; per questo motivo la funzione di attivazione deve essere **non lineare**, in modo che il modello sia in grado di approssimare qualsiasi funzione. Una rete neurale senza funzione di attivazione infatti equivale ad un semplice modello di regressione il quale approssima le distribuzioni dei dati con una retta. Con questo modello praticamente ogni layer si comporterebbe allo stesso modo del precedente ed il risultato sarebbe sempre lineare (figura 3.8).

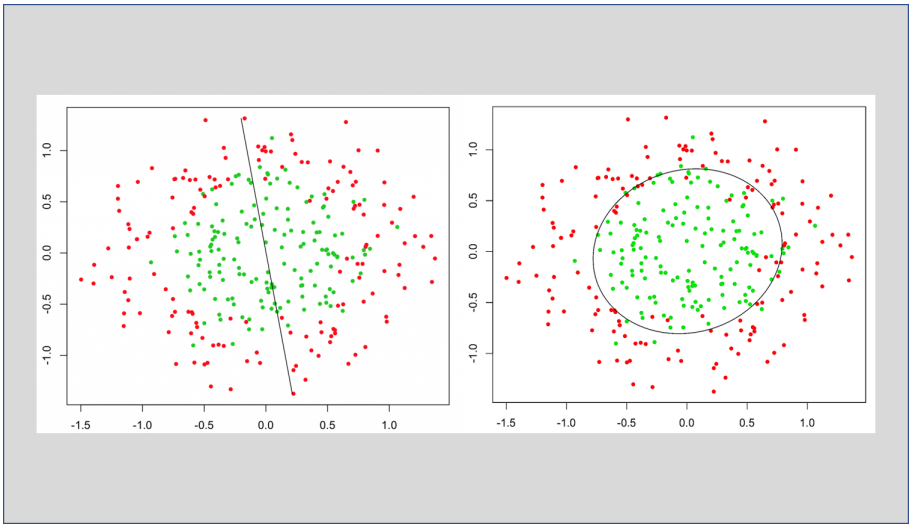

Figura 3.8 – Differenza tra un' approssimazione lineare e non lineare [24]

3.1.3 Backpropagation

Per controllare l'output di una rete neurale, occorre essere in grado di misurare quanto quel determinato output si avvicini al risultato previsto. Per questo motivo nella rete è presente una funzione obiettivo chiamata ***loss function***. Il compito della loss function è quello di analizzare e confrontare le predizioni della rete ed il target (ciò che preferibilmente si vuole che la rete produca in output)

ed assegnare un punteggio di distanza che valuti le prestazioni della rete. Questa valutazione grazie agli algoritmi di ML può essere impiegata come segnale di feedback per regolare il valore dei weights in una direzione che riduca la distanza dal target. Tale ottimizzazione è implementata da un **algoritmo di backpropagation**. Ciascun neurone della rete, infatti trasforma i suoi input in tal modo:

output= sigmoid(input*W + b)

In questa espressione W e b sono dei tensori e rappresentano i parametri addestrabili del neurone in quanto contengono le informazioni apprese dalla rete a causa della sua esposizione ai dati di addestramento. Inizialmente, ai weights della rete sono assegnati dei valori casuali, ai quali la rete applica semplicemente una serie di trasformazioni. In tal modo l'output non sarà ideale e la valutazione fornirà una distanza elevata. Ad ogni esempio elaborato dalla rete, i weigths saranno modificati in modo da orientarsi nella direzione corretta. Questa fase è detta ciclo di addestramento, il quale ripetuto un numero sufficente di volte fornisce dei valori dei weigths, i quali minimizzano la funzione obiettivo. Una rete neurale è detta addestrata quando fornisce un valore minimo della funzione obiettivo tale per cui gli output sono molto vicini ai target.

3.2 Reti neurali convoluzionali

Una **Convolutional Neural Network (CNN)** è un architettura di rete neurale ampiamente utilizzata per il riconoscimento e la classificazione di immagini. Una rete convoluzionale è formata da diversi strati [26]:

- Conv
- ReLU
- Pool
- FC

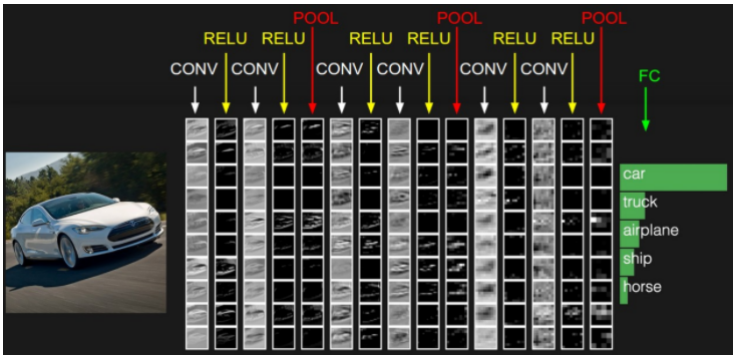

Figura 3.9 – Architettura di una CNN [27]

3.2.1 Conv

Il livello convoluzionale è il livello principale della rete. Lo scopo del livello è quello di individuare schemi (curve, angoli, circonferenze) e caratteristiche delle immagini di input con elevata precisione. Esso prende il nome dall'operazione di convoluzione che viene effettuata tra la matrice di input (matrice a tre dimensioni nel caso di immagini) ed un **filtro** che produce come risultato una **feature map**, o mappa delle caratteristiche. Per filtro si intende generalmente una matrice che rappresenta una caratteristica (feature) che il livello convoluzionale vuole identificare. Inizialmente per i primi livelli convoluzionali si dice che il filtro rappresenti delle caratteristiche di basso livello poiché identifica semplici oggetti come linee e curve. Negli ultimi livelli invece il filtro identifica caratteristiche di alto livello come circonferenze ed oggetti complessi. Individuata la caratteristica che si vuole identificare nel livello convoluzionale, si decide la dimensione del filtro e il numero di filtri da utilizzare nel livello. Scelta la dimensione del filtro si va ad analizzare nella matrice di input un **campo ricettivo** della stessa dimensione. In ogni posizione dunque, viene eseguita una moltiplicazione puntuale tra la regione di input selezionata dal filtro (il campo ricettivo) e il filtro stesso il cui risultato sarà posizionato nella feature map. L'operazione viene ripetuta e di conseguenza il campo ricettivo viene fatto spostare di un determinato **passo**. Man mano che continuiamo ad applicare i livelli convoluzionali, la dimensione del volume diminuirà più velocemente di quanto vorremmo. Nei primi strati della rete, però, è bene conservare il maggior numero di informazioni sul volume di input originale in modo da poter estrarre tali caratteristiche di basso livello, che altrimenti andrebbero perse. Per poter mantenere la stessa dimensionalità della matrice di ingresso, è possibile effettuare l'operazione di **zero padding** che identifica uno strato da apporre al volume di input iniziale al fine di evitare di perdere alcune informazioni nel passaggio da un livello a un altro.

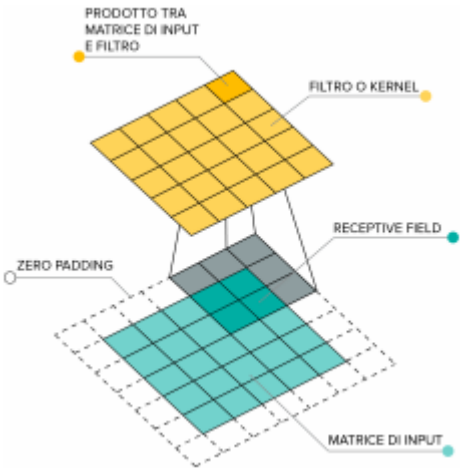

Figura 3.10 – Esempio di convoluzione tra matrice di input e filtro [26]

3.2.2 ReLU

Il livello ReLU rappresenta un livello non lineare, il cui scopo è quello di introdurre la non linearità a un sistema che sostanzialmente sta calcolando operazioni lineari durante i livelli convoluzionali (tramite il prodotto scalare tra il filtro e il campo ricettivo). Tramite questi livelli la rete è in grado di allenarsi molto più velocemente (a causa dell'efficienza computazionale) senza impattare significativamente sull'accuratezza dei risultati. Il livello ReLU applica la funzione **f(x)=max (0, x)** a tutti i valori nel volume di input. In parole semplici, questo livello annulla tutti i valori negativi, aumentando le proprietà non lineari del modello e della rete globale senza influenzare i campi ricettivi del livello convoluzionale.

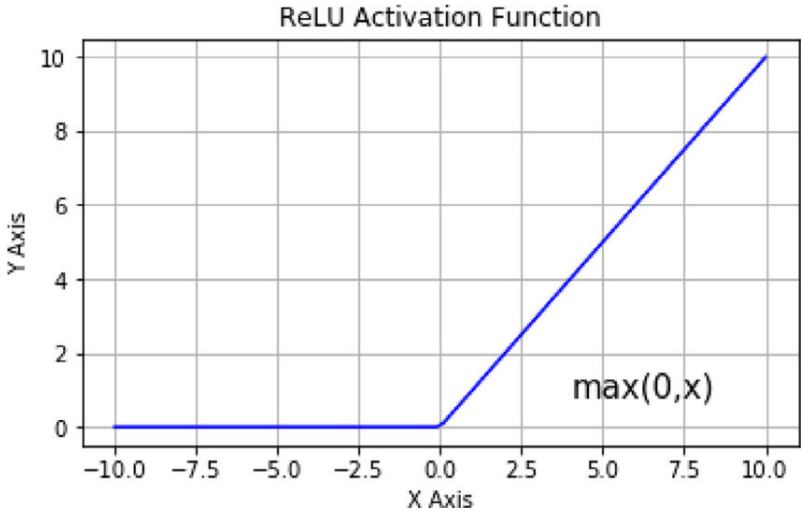

Figura 3.11 – Funzione ReLU [28]

3.2.3 Pool

Il livello di pooling è responsabile della riduzione della dimensionalità spaziale dei dati. Tale riduzione è effettuata in modo da diminuire i requisiti computazionali necessari per processare i dati e per ricavare le caratteristiche dominanti dell'immagine che si sta analizzando. Esistono due tipi di pooling: **Max-Pooling** e **Average-Pooling**. Il primo restituisce il valore massimo della sezione dell'immagine coperta dal filtro mentre il secondo il suo valore medio.

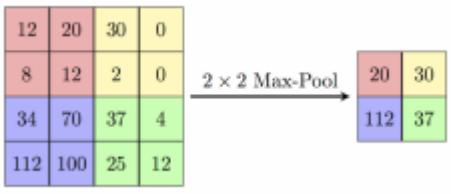

Figura 3.12 – Esempio di *max-pooling* [26]

3.2.4 FC

Lo strato full connection è l'ultimo livello della rete il quale prende un volume di input (i quali possono essere gli output di uno strato precedente di convoluzione, relu o pool) e genera un vettore N dimensionale il quale sarà classificato in una delle N classi definite. Infatti, prima di essere processato la matrice delle caratteristiche viene trasformata in un vettore da un livello di **flatten**. Fondamentalmente, un livello FC guarda quali caratteristiche di alto livello sono maggiormente correlate ad una particolare classe e calcola i prodotti tra i pesi e il livello precedente per ottenere le probabilità corrette per le diverse classi. Ad esempio dato un classificatore multiclasse con N pari a 10, supponiamo che il vettore risultante sia:

$$[0\ 0\ 15\ 10\ 0\ 0\ 0\ 65\ 0\ 10]$$

Allora si avrà una probabilità del 15% che l'immagine rappresenti la classe 3, una probabilità del 10% che l'immagine rappresenti la classe 4, una probabilità del 65% che l'immagine rappresenti la classe 8 e una probabilità del 10% che l'immagine sia la classe 10 (tutte le altre classi hanno probabilità nulla di essere scelte).

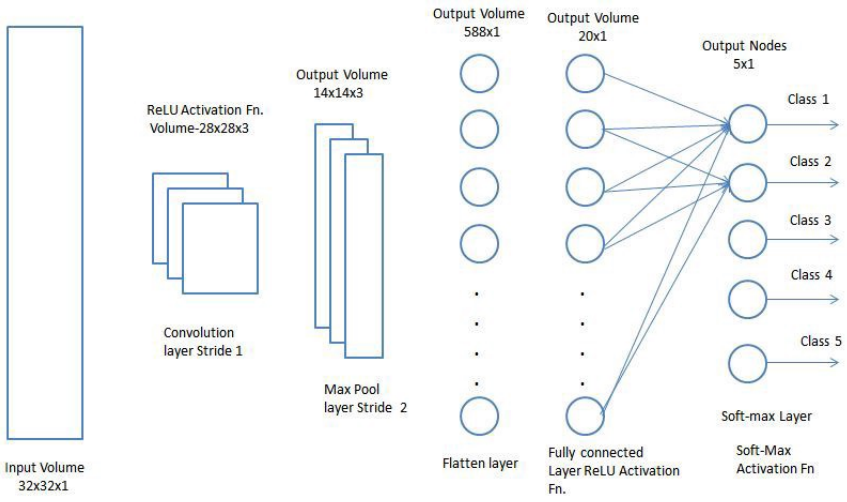

Figura 3.13 – Livello FC [26]

3.3 Metriche per la valutazione dei modelli

Generalmente un classificatore binario può essere visto come un classificatore di istanze *positive* o *negative*. Le istanze positive sono tutte quelle che sono classificate come membro della classe che si sta cercando di identificare, mentre quelle negative sono l'istanze classificate come non appartenenti alla classe [29]. Dato un classificatore binario e T=P+N pattern da classificare (P positivi e N negativi), il risultato di ciascuno dei tentativi di classificazione può essere:

- **True Positive (TP)**: un pattern positivo è stato correttamente assegnato ai positivi.
- **False Positive (FP)**: un pattern negativo è stato erroneamente assegnato ai positivi.
- **True Negative (TN)**: un pattern negativo è stato correttamente assegnato ai negativi.
- **False Negative (FN)**: un pattern negativo è stato erroneamente assegnato ai positivi.

Tali valori spesso sono utilizzati all'interno di una **matrice di confusione** (figura 3.14) la quale illustra le prestazioni del classificatore. Le considerazioni fatte possono essere estese anche a classificatori multi-classe, in tal caso la matrice di confusione è utilizzata per comprendere in che modo sono distribuiti gli errori.

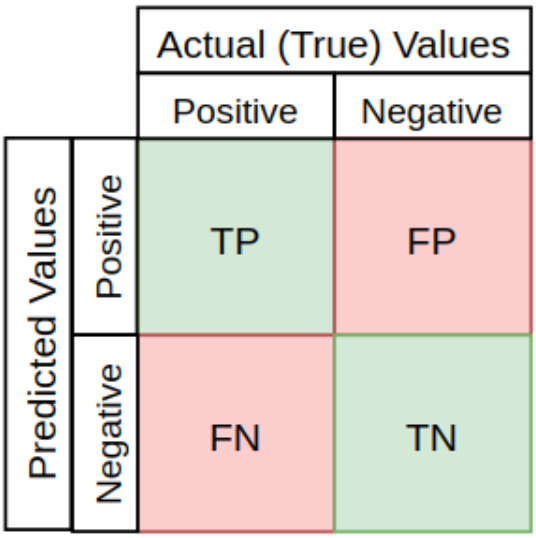

Figura 3.14 – Esempio di Confusion Matrix [30]

Alla base di questi concetti è possibile definire diverse metriche per la valutazione di un modello di rete neurale [30]:

- **Accuracy**: Descrive il numero di previsioni corrette sul totale di previsioni effettuate. Nel caso di un classificatore binario potremmo calcolare l'accuratezza tramite la seguente formula:

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN}$$

- **Precision**: Descrive quanto è accurato il sistema indicando quanti elementi selezionati sono rilevanti. Il suo valore è dato dal numero di predizioni positive sul totale di predizioni corrette.

$$Precision = \frac{TP}{TP + TF}$$

- **Recall**: Indica quanto il sistema è selettivo. Il suo valore sarà dunque il numero di elementi correttamente selezionati sul numero di elementi rilevanti (cioè quelli che sono positivi).

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

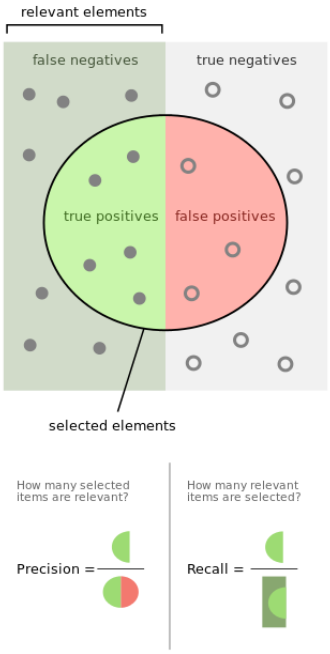

Figura 3.15 – Precision e Recall [29]

- **F1 score**: Nell'analisi statistica della classificazione binaria, l'F1 score (noto anche come F-score o F-measure) è una misura dell'accuratezza di un test. Statisticamente è calcolato come la media armonica di precision e recall.

$$F1 = 2 \cdot \frac{\text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}}$$

- **IoU (Intersetion over Union)**: Nell'apprendimento supervisionato i dati di input sono classificati mediante delle etichette. La IoU misura la sovrapposizione tra due aree, in particolare misura la sovrapposizione tra l'area dell'etichetta dell'immagine di input e l'etichetta che il modello di rete ha predetto per quella determinata immagine (figura 3.16). Data l'etichetta A (Prediction) predetta dalla rete e l'etichetta B (Ground truth) il target della immagine che si vuole classificare, l'IoU sarà dato dal rapporto tra l'intersezione e l'unione di questi due elementi [31].

$$IoU = \frac{|A \cap B|}{|A \cup B|} = \frac{|I|}{|U|}$$

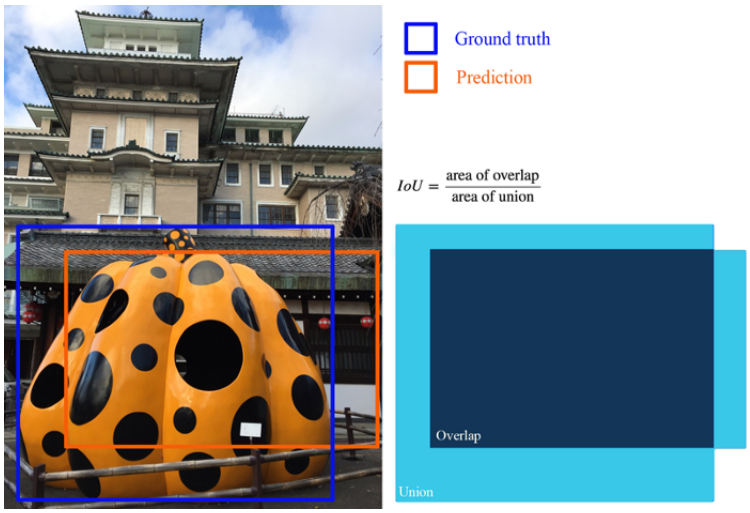

Figura 3.16 – Esempio di calcolo di IoU [31]

- **GIoU (Generalized Intersection over Union)**: Nel riconoscimento di oggetti l'IoU è una metrica utilizzata per indicare quanto una previsione si avvicina all'output desiderato. Nonostante ciò IoU può darci una stima della bontà della previsione solo se l'area dell'etichetta prevista si interseca con l'area dell'etichetta effettiva, in tutti gli altri casi il suo valore assume sempre zero. Infatti supponiamo di avere due previsioni le quali non intersecano la ground truth; per poter misurare quale delle due si avvicina di più all'etichetta desiderata non è possibile utilizzare la IoU (che avrà valore pari a 0 per entrambe le previsioni), ma introduciamo una nuova matrica chiamata GIoU [32]. Data la previsione A e la ground truth B, sia C il più piccolo involucro convesso che li racchiude, la GIoU può essere calcolata tramite la seguente forumula:

$$GIoU = \frac{|A \cap B|}{|A \cup B|} - \frac{|C \backslash (A \cup B)|}{|C|} = IoU - \frac{|C \backslash (A \cup B)|}{|C|}$$

Come si nota dalla formula e dal grafico (figura 3.17), la GIoU assume valori che variano da -1 a 1. Si hanno valori negativi quando l'area che racchiude entrambe le etichette (l'involucro convesso C) è maggiore dell'IoU. All'aumentare di quest'ultimo, il valore della GIoU converge sempre di più al valore dell'IoU.

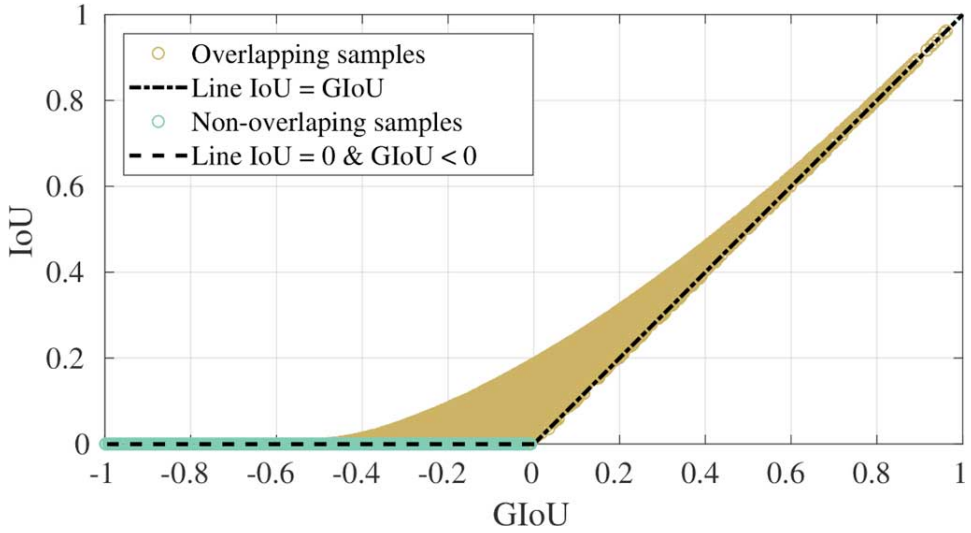

Figura 3.17 – Correlazione tra GIoU e IoU [32]

- **AP (Average Precision)**: L'IoU può essere utilizzata per determinare se un'etichetta prevista è TP, FP o FN (il caso TN non è valutato in quanto si presume che ogni immagine contenga almeno un oggetto).

  1. *True Positive*: Tradizionalmente definiamo una previsione TP se il valore della IoU>0.5
  2. *False Positive*: Una previsione è definita FP se IoU<0.5 o se le etichette previste sono duplicate.
  3. *False Negative*: Il modello è considerato FN se sbaglia la sua previsione. L'errore può avvenire se il modello non riconosce nessun oggetto o se il modello ha previsto un' etichetta con IoU>0.5, ma di una classe errata.

  Una volta definiti formalmente TP, FP ed FN possiamo calcolare la precision e la recall per quella determinata classe che vogliamo riconoscere e tracciare il grafico (figura 3.18). Generalmente l'AP è definita come l'area sottesa della curva precision/recall:

$$\int_0^1 p(r)\ dr$$

  Poichè sia la precision che la recall assumono un valore compreso tra 0 e 1 anche il valore dell'AP cadrà in questo intervallo.

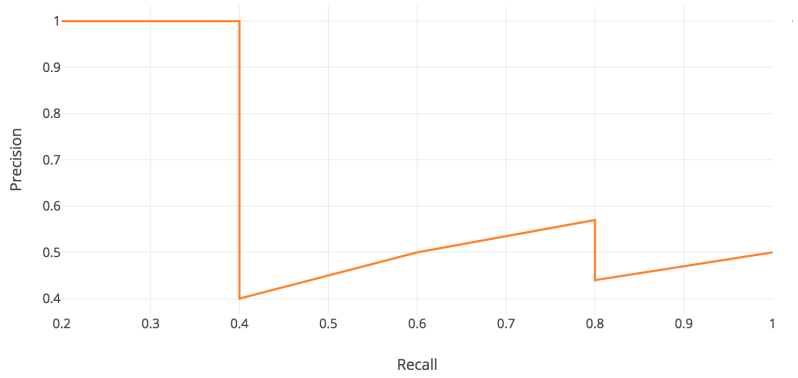

Figura 3.18 – Esempio di grafico precision/recall di una determinata classe [31]

- **mAP(Mean Average Precision)**: Molti algoritmi di object detection utilizzano un ulteriore metrica chiamata mAP. Essa è definita come l'AP medio su tutte le classi o su tutte le soglie di IoU complessive. Ad esempio in PASCAL VOC2007, l'AP per una classe di oggetti viene calcolato per una soglia IoU di 0,5. Quindi la mAP viene calcolato come media su tutte le classi di oggetti. Per COCO 2017, invece, il mAP viene calcolato come media su tutte le categorie di oggetti e su 10 soglie IoU. Supponiamo di voler calcolare la media dell'average precision utilizzando 11 soglie.

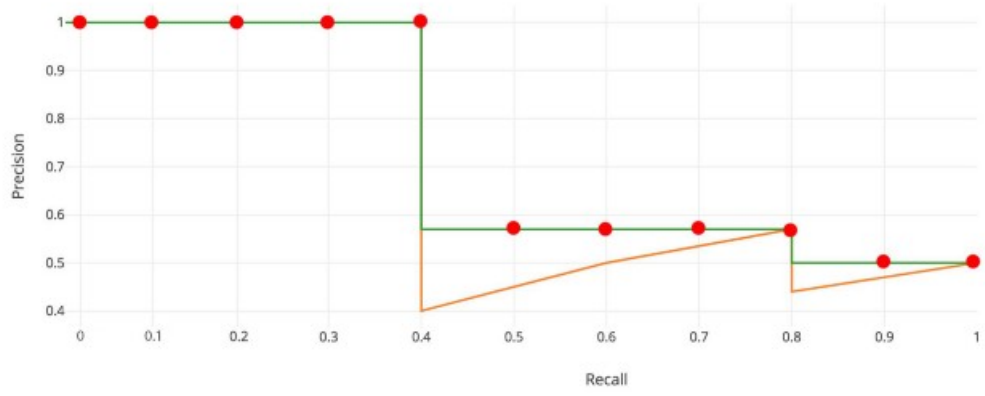

Figura 3.19 – Esempio di grafico precision/recall con 11 soglie [31]

Il valore di recall (che varia da 0 a 1,0) viene diviso in 11 punti: 0; 0,1; 0,2 ... 0,9 e 1,0. Successivamente viene calcolata la media del massimo valore di precision per ogni valore di recall.

$$AP = \frac{1}{11} \sum_{Recall_i} AP_r$$

3.4 Modelli utilizzati

Di seguito sono descritte le architetture e le funzionalità delle due reti neurali utilizzate nel progetto.

3.4.1 YOLO

"You Only Look Once" (YOLO) [6] è un algoritmo che utilizza le reti neurali convoluzionali per riconoscere gli oggetti. Sebbene non sia uno degli algoritmi più accurati è un' ottima scelta per il rilevamento di oggetti in tempo reale essendo uno dei più veloci in circolazione.

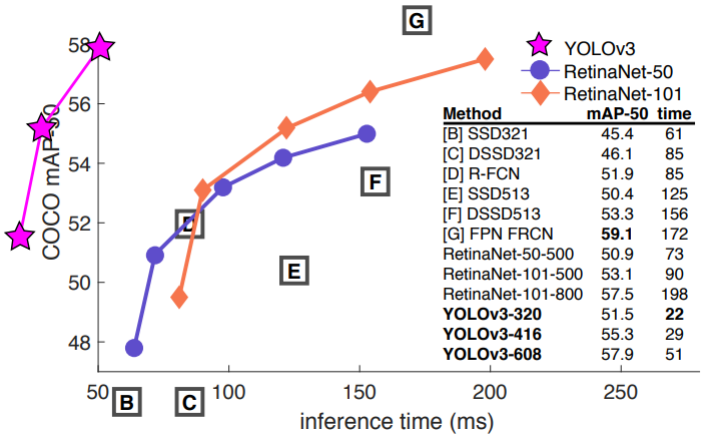

Figura 3.20 – Confronto velocità algoritmi di object recognition [6]

YOLO utilizza solo livelli convoluzionali costruendo una rete FCN (full convolutional network). L'architettura utilizzata chiamata **Darknet-53** (figura 3.21), infatti contiene 53 livelli convoluzionali ognuno dei quali è seguito da un livello di normalizzazione e da una funzione di attivazione ReLu. Dunque all'interno della rete non esiste nessuna forma di pooling, ma spesso viene utilizzato uno strato convoluzionale con passo pari a 2 per eseguire il sotto-campionamento (*downsampling*) delle feature map. Ciò aiuta a prevenire la perdita di caratteristiche di basso livello spesso attribuite ai livelli di pool. YOLO è invariante alla dimensione dell'immagine di input, ma tuttavia per evitare problemi nell'implementazione dell'algoritmo è preferibile mantenere una dimensione costante. Se vogliamo elaborare le nostre immagini in **batch** (gruppi di minor dimensioni del volume di input le quali possono essere elaborate in parallelo nelle GPU) dobbiamo utilizzare immagini di altezza e larghezza fisse. Ciò è necessario per concatenare più immagini in batch di grandi dimensioni. La rete dunque esegue il down-sampling dell'immagine di un fattore pari al passo. Ad esempio, se il passo della rete è 32, un'immagine di input di dimensione 416 x 416 produrrà un output di dimensione 13 x 13. Descriviamo in che modo la rete riesca a classificare un' immagine in una determinata classe (figura 3.22):

	Type	Filters	Size	Output
	Convolutional	32	3 × 3	256 × 256
	Convolutional	64	3 × 3 / 2	128 × 128
1×	Convolutional	32	1 × 1	
	Convolutional	64	3 × 3	
	Residual			128 × 128
	Convolutional	128	3 × 3 / 2	64 × 64
2×	Convolutional	64	1 × 1	
	Convolutional	128	3 × 3	
	Residual			64 × 64
	Convolutional	256	3 × 3 / 2	32 × 32
8×	Convolutional	128	1 × 1	
	Convolutional	256	3 × 3	
	Residual			32 × 32
	Convolutional	512	3 × 3 / 2	16 × 16
8×	Convolutional	256	1 × 1	
	Convolutional	512	3 × 3	
	Residual			16 × 16
	Convolutional	1024	3 × 3 / 2	8 × 8
4×	Convolutional	512	1 × 1	
	Convolutional	1024	3 × 3	
	Residual			8 × 8
	Avgpool		Global	
	Connected		1000	
	Softmax			

Figura 3.21 – Modello Darknet-53 [6]

YOLO [33] prende in input un batch di forma (m,416,416,3) dove m è il numero di immagini e restituisce in output una lista di **bounding box** ognuno identificato dalla propria classe di appartenenza. Ogni bounding box è rappresentato da 6 valori (pc,bx,by,bh,bw,c) dove c è il vettore delle classi il quale può avere una dimensione massima di 80 (YOLO infatti può classificare fino ad un massimo di 80 classi). Come in tutti i rilevatori di oggetti, le caratteristiche apprese dagli strati convoluzionali vengono trasferite a un classificatore / regressore che effettua la previsione. In Yolo la previsione viene eseguita da uno strato convoluzionale 1x1 in modo che l'output (chiamato ***prevision map***) sia della stessa dimensione della feature map. La previsione sarà effettuata dividendo la prevision map in celle ognuna delle quali conterrà un insieme di bounding boxes. Supponiamo di avere (B x (5+C)) elementi nella mappa delle caratteristiche, dove B rappresenta il numero di bounding box previsti per ogni cella ognuna delle quali è specializzata nel rilevamento di un oggetto specifico. Ogni bounding box presenta 5+C attributi i quali descrivono le coordinate, un punteggio di *objectness* (il quale rappresenta la probabilità che un oggetto sia contenuto all'interno del bounding box considerato) e le relative C probabilità che un oggetto appartenga ad una determinata classe. La rete Yolo in particolare utilizza 3 bounding box per cella e analizza solo le celle della feature map il cui campo recettivo è posto al centro dell'oggetto che si vuole classificare. Per identificare tale cella la rete divide l'immagine di input in un griglia di dimensioni uguali alla mappa delle caratteristiche finale.

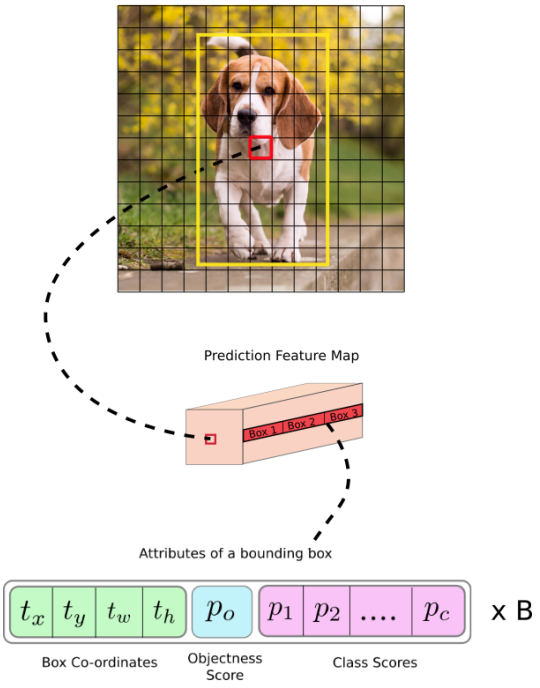

Figura 3.22 – Esempio di object recognition in YOLO [33]

Per ogni bounding box di una cella sarà calcolato un **vettore di score** i cui elementi saranno dati dal prodotto dell'objectness (la quale sarà pari a 1 per la cella posta al centro dell'oggetto e vicina allo 0 nelle celle poste ai lati) e la probabilità che quel determinato box sia di classe i-esima.

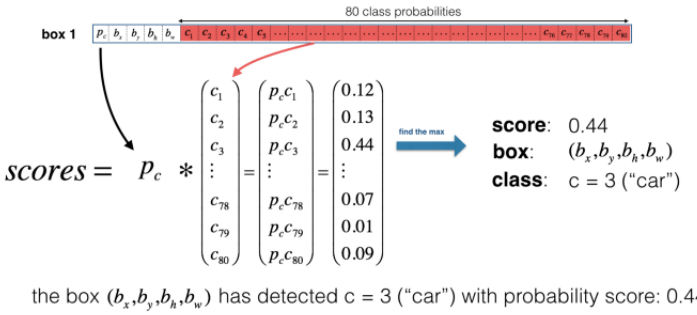

Figura 3.23 – Esempio di calcolo delle probabilità di un bounding box [33]

Yolo inoltre effettua la previsione su tre differenti **scale** (figura 3.24); il layer di detection viene utilizzato per effettuare il rilevamento di oggetti in mappe delle caratteristiche di tre dimensioni, con passi pari a 32, 16 o 8. La rete esegue il downsampling dell'immagine di input fino al primo livello dove la detection viene effettuata utilizzando feature map con passo pari a 32. Inoltre, i livelli sono sovracampionati di un fattore 2 e concatenati con mappe delle caratteristiche dei livelli precedenti aventi stesse dimensioni. La stessa procedura di sovracampionamento viene effettuata nei due livelli di detect con passo pari a 16 ed 8. Alla fine della detection ogni cella prevederà 9 bounding box (3 per ogni scala).

La detection su più scale viene principalmente utilizzata in quanto l'upscaling può aiutare la rete ad apprendere caratteristiche fondamentali per la rilevazione di piccoli oggetti.

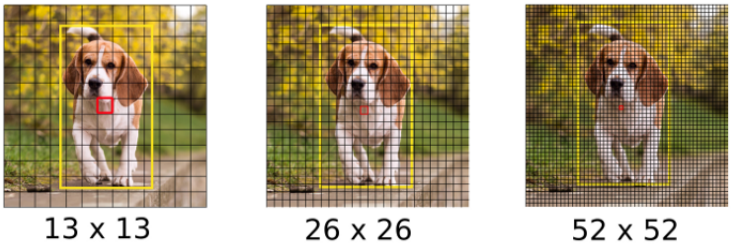

Figura 3.24 – Esempio di detection su differenti scale [33]

Dunque data una immagine di dimensioni 416 x 416 Yolo prevede 10647 bounding box, i quali sono ovviamente troppi nel caso in cui dovessimo rilevare un singolo oggetto. Per questo motivo i bounding box previsti sono filtrati in base all'objectness score. In genere la procedura effettuata è la seguente:

1. Tutti i box inferiori ad una certa **soglia** (generalmente pari a 0,5) di objectness sono ignorati.
2. Quando più box si sovrappongo (ad esempio i tre bounding box della cella posta al centro dell'oggetto e le celle adiacenti possono rilevare lo stesso oggetto) solo uno ne viene selezionato attraverso l'utilizzo di una tecnica chiamata **NMS (Non-maximum Suppression)**, la quale fa uso dell'IoU. La tecnica può essere descritta nei seguenti passi:
   - Inizialmente viene selezionato il box con lo score più alto.
   - Viene calcolata la sua sovrapposizione con tutte le restanti caselle. Le caselle con un IoU maggiore di quello di soglia (cioè quelle che maggiormente si sovrappongono) sono eliminate.
   - La procedura viene nuovamente effettuata finchè non esistono più box con uno score minore di quello correntemente selezionato.

Figura 3.25 – Esempio di NMS [33]

3.4.2 RCNN Networks

Tra i modelli più utilizzati nel riconoscimento delle immagini esiste la famiglia delle **RCNN networks**, una serie di modelli sviluppati da Ross Girshick dal 2014. Il primo modello sviluppato chiamato RCNN (Region based Convolutional Network) (figura 3.26) si basa sull'utilizzo dell'algoritmo di **ricerca selettiva**, il quale è utilizzato per "proporre" un insieme di regioni (chiamate "region of interests" o "RoI") sul quale successivamente identificare un insieme di caratteristiche [34].

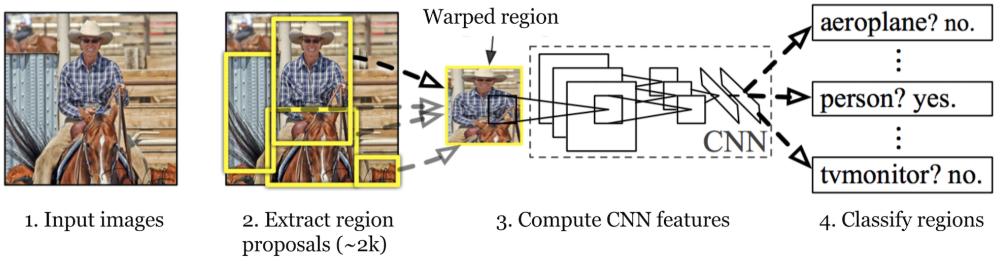

Figura 3.26 – Architettura di una RCNN [34]

Il workflow del modello può essere sintetizzato in tre principali moduli [35]:

1. ***Region Proposal***: Nel primo modulo l'algoritmo di ricerca selettiva raggruppa e produce regioni con caratteristiche simili delimitate all'interno dell'immagine di input [36]. Il raggruppamento delle regioni avviene attraverso un approccio iterativo in cui l'algoritmo trova diverse somiglianze tra i vari pixel dell'immagine basandosi sul colore, la texture e la taglia (figura 3.27). In questo modo il metodo di region proposal genera attraverso l'algoritmo di ricerca selettiva 2000 regioni indipendenti di diverse dimensioni.

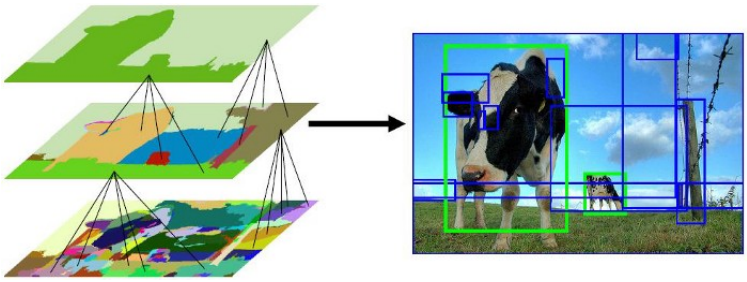

Figura 3.27 – Algoritmo di ricerca selettivo utilizzato per identificare delle regioni [35]

2. ***Feature extration***: In questa fase le RoI sono utilizzate per estrarre le principali caratteristiche tramite un rete CNN. Inizialmente le regioni candidate sono deformate e ridimensionate in modo tale da avere una dimensione fissa come richiesto dalla CNN. Successivamente per ogni singola regione la rete genera un vettore delle caratteristiche di dimensione 4096. Il processo viene ripetuto per ogni singola immagine disponibile nel dataset, in modo da generare in output un vettore 2000x4096 per ogni immagine.

3. ***Classification and Localization***: Una volta che abbiamo generato un vettore dimensionale 4096 per ogni oggetto candidato in ciascuna immagine, possiamo eseguire un modello SVM lineare (support vector machine, modelli di apprendimento supervisionato molto utilizzati per la classificazione) addestrato per classificare quell'oggetto in una delle tante classi disponibili.

Esaminando i vari passi effettuati per l'addestramento di una rete RCNN è chiaro che il modello richieda un elevato costo computazionale in quanto:

- L'algoritmo di ricerca selettivo deve essere utilizzato per ogni immagine in modo da generare le 2000 proposal region
- Per ogni regione dell'immagine viene generato un vettore delle caratteristiche (Nimmagini*2000)
- L'intero processo coinvolge tre modelli separatamente senza una computazione condivisa: la rete neurale convoluzionale per la classificazione delle immagini e l'estrazione delle caratteristiche; il classificatore SVM lineare per identificare gli oggetti in una determinata classe.

Per questo motivo nel 2015 Ross Girshick ha sviluppato un nuovo modello chiamato **Fast RCNN** [37]. A differenza dell'algoritmo precedente la Fast RCNN analizza inizialmente l'immagine di input tramite la CNN in modo da generare una feature map. La mappa delle caratteristiche generata sarà poi successivamente utilizzata per identificare l'insieme delle regioni proposte le quali saranno deformate da uno strato di *pooling RoI*. Lo strato di RoI pooling proietta le caratteristiche della regione considerata in una finestra di dimensione fissa HxW, la quale sarà divisa in griglie ad ognuna delle quali sarà applicata la funzione di max-pooling. Una volta terminata la fase di RoI pooling e generato il vettore delle caratteristiche l'ultimo livello FC della rete convoluzionale classifica gli oggetti dell'immagine in una determinata classe. A differenza della RCNN la sua versione Fast effettua l'operazione di convoluzione una sola volta per immagine e non per ogni regione proposta, in questo modo il costo computazionale è ridotto in maniera significativa (figura 3.30).

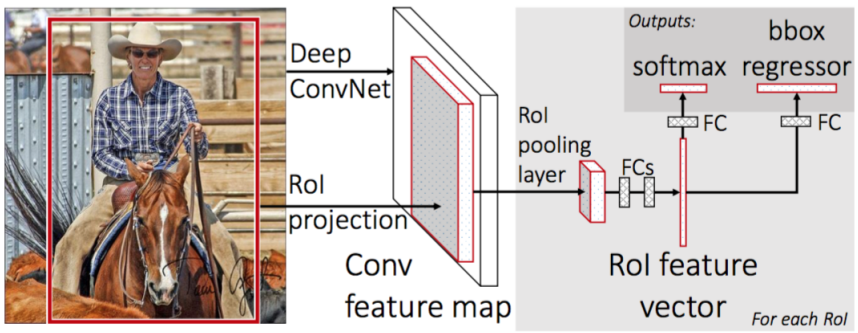

Figura 3.28 – Architettura della Fast RCNN [37]

Entrambi i modelli discussi finora utilizzano l'algoritmo di ricerca selettiva il quale propone possibili regioni di interesse e richiede di classificarle tutte. Inoltre, durante il processo di selezione della regione non è coinvolto alcun apprendimento, limitando l'accuratezza effettiva del modello. Per questo motivo nel 2016 è stato proposto un algoritmo migliorato chiamato **Faster RCNN** [7], che elimina del tutto la ricerca selettiva e consente alla rete di apprendere direttamente le proposte regionali. Questo nuovo modello utilizza una rete CNN preaddestrata chiamata **Region Prediction Network (RPN)** posta dopo l'ultimo strato convoluzionale. La RPN è una rete addestrata la quale è in grado di generare le proposte regionali senza la necessità di alcun meccanismo esterno come la ricerca selettiva. Dunque la RPN produce un insieme di proposte, ciascuna delle quali ha un punteggio relativo alla sua probabilità di essere un oggetto o di essere la classe /etichetta dell'oggetto stesso. Queste proposte sono successivamente perfezionate tramite l'utilizzo di due livelli FC: il primo per la regressione, il secondo per la classificazione. Dopo aver generato le RoI la rete si comporta in maniera simile alla sua versione Fast attraverso l'uso di uno strato di RoI pooling ed di un classificatore.

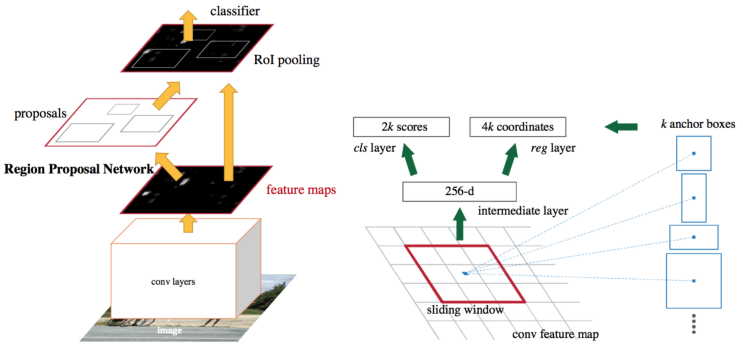

Figura 3.29 – Architettura della Faster RCNN [7]

In conclusione è possibile vedere come il modello Faster RCNN diminuisce ulteriolmente la complessità computazionale necessaria all'addestramento con un aumento delle prestazioni pari al 245% rispetto alla prima versione e dell' 11% rispetto alla versione Fast (figura 3.30).

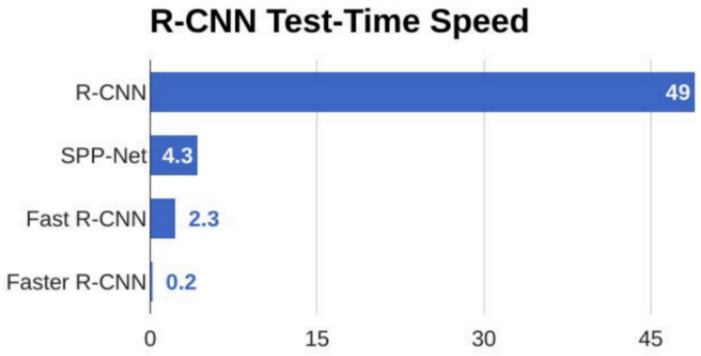

Figura 3.30 – Confronto della velocità degli algoritmi RCNN [38]

Capitolo 4

Sviluppo del progetto

4.1 Preparazione dei dataset

Nella prima fase del progetto sono state raccolte delle immagini, nel caso specifico immagini del frumento affetto da septoriosi, e divise in tre distinti dataset: train, validation, test. In particolare i dataset di train e validation sono stati etichettati mediante lo strumento bounding-box ed esportati in formato JSON.

4.1.1 Raccolta delle immagini

Le raccolta è stata effettuata utilizzando uno script ***image_ downloader.py*** il quale una volta specificata una determinata keyword permette di scaricare le immagini da due diverse fonti ed eventualmente ridimensionarle. Lo script viene eseguito lanciando il seguente comando:

```
!python3 image_downloader.py [-keyword "_"] [-destination <dest_folder>]
                             [-limit <num_imm>] [-file_prefix <prefix>]
                             [-resize <w>x<h>] [-sources <sorgente>]
```

Gli argomenti richiesti in input prima dell'esecuzione dello script sono:

- **keyword**: Specifica la parola chiave che sarà ricercata nel browser per scaricare le immagini.
- **destination**: Specifica la cartella in cui le immagini scaricate saranno salvate.
- **limit**: Specifica il numero di immagini che si vogliono scaricare.
- **file_prefix**: Permette di specificare il prefisso del nome di tutti i file che saranno salvati.
- **resize**: Permette di specificare *width* ed *height* utilizzati per ridimensionare le immagini.
- **sources**: Specifica le fonti (Google o Bing) sulle quali si andranno a scaricare le immagini.

Di seguito sono descritte tutti i moduli utilizzati dallo script. Le funzioni sviluppate utilizzano sia moduli di base sia moduli realizzati appositamente per lo script(GrabbedImage, BingGrabber, GoogleGrabber)

```
import base64
import math
import time
import urllib.request
from typing import NoReturn, Tuple
from skimage import io as _io
from skimage.transform import resize
from image_grabber.grabbed_image import GrabbedImage
from utils.utils import *
from .bing_grabber import BingGrabber
from .google_grabber import GoogleGrabber
```

La funzione principale eseguita dello script ***download_images*** una volta specificata la keyword e le eventuali fonti, scarica le immagini dal browser e le salva all'interno della destination folder.

```
def download_images(self, keyword: str) -> NoReturn:
  start_time = time.time()
  if not keyword:
    raise Exception('No keyword to search')
  self.keyword = keyword
  self.__set_default_file_prefix()
  all_sources = [e.value for e in GrabSourceType]
  selected_sources = all_sources if ALL_SOURCE in self.sources else self.
    sources
  images = []
  if GrabSourceType.GOOGLE.value in selected_sources:
    google_grabber = GoogleGrabber()
    google_grabber.full_image = self.full_image
    images.extend(google_grabber.get_images_url(self.keyword, self.limit))
  if GrabSourceType.BING.value in selected_sources:
    bing_grabber = BingGrabber()
    bing_grabber.full_image = self.full_image
    images.extend(bing_grabber.get_images_url(self.keyword, self.limit))
  if ALL_SOURCE in self.sources or len(selected_sources) > 1:
    images = self.__repart_between_image_sources(selected_sources, images)
  if not images:
    raise (NoImageFoundException("No images found on sources %s" % ",".join(
    list(self.sources))))
  else:
    sub_folder_name = self.__create_destination_folder()
    print("\n %s images found on %s, limit to download: %s \n" % (len(images),
     self.sources, self.limit))
    nb_downloaded = self.__download_files(images, sub_folder_name)
    end_time = time.time()
    print("\n\n %s images downloaded in %s sec" % (nb_downloaded, round(
    end_time - start_time, 2)))
```

Tale funzione utilizza altre funzioni che eseguono compiti più specifici:

- La funzione ***set_default_file_prefix*** genera un prefisso da assegnare ai file scaricati basandosi sulla keyword scelta. Se il valore dell'argomento prefisso non è specificato (valore di default), la funzione cambia il prefisso assegnandogli come valore la keyword.

```
def __set_default_file_prefix(self) -> NoReturn:
  if self.file_prefix is None:
    self.file_prefix = StringUtil.underscore_and_lowercase(self.keyword)
```

- La funzione ***create_destination_folder*** crea una sotto-cartella, con il nome della keyword scelta, all'interno della cartella di destinazione.

```
def __create_destination_folder(self) -> str:
  if self.destination is None:
    self.destination = 'images'
  if not os.path.exists(self.destination):
    os.mkdir(self.destination)
  sub_folder = os.path.join(self.destination, StringUtil.
    underscore_and_lowercase(self.keyword))
  if not os.path.exists(sub_folder):
    os.mkdir(sub_folder)
  return sub_folder
```

- La funzione ***download_files*** prende in input una lista di oggetti di tipo GrabbedImage (insieme di url delle immagini) ed effettua il download.

```
def __download_files(self, images: List[GrabbedImage], folder_path: str)
     -> int:
    nb_downloaded = 0
    for i, image in enumerate(images):
       if nb_downloaded == self.limit:
          break
       try:
          full_destination = FileUtil.generate_next_file_path(
    folder_path, self.file_prefix)
          if self.resize is not None:
              self.__resize_and_save(image, self.resize,
    full_destination)
          else:
              self.__save_image(image, full_destination)
          nb_downloaded = nb_downloaded + 1
          ProgressBarUtil.update(nb_downloaded, self.limit)
       except Exception as e:
          ExceptionUtil.print(e)
          pass
       return nb_downloaded
```

- La funzione ***save_image*** salva un immagine di tipo base64 o un file da un url all'interno del disco.

```
def __save_image(self, image: GrabbedImage, full_destination: str):
    image_to_write = None
    if image.base64 is not None:
        image_to_write = self.__decode_base64(image.base64)
    elif image.url is not None:
        image_to_write = urllib.request.urlopen(image.url).read()
    f = open(full_destination, 'wb')
    f.write(image_to_write)
    f.close()
```

- La funzione ***resize_and_save*** permette (se specificati width ed heigth) di ridimensionare le immagini e salvarle sul disco.

```
def __resize_and_save(self, image: GrabbedImage, size: Tuple[int], dest:
     str):
    image_url = None
    if image.url is not None:
        image_url = image.url
    elif image.base64 is not None:
        f = open(dest, 'wb')
        f.write(self.__decode_base64(image.base64))
        f.close()
        image_url = dest
```

```
10          image_array = _io.imread(image_url)
11          _io.imsave(dest, resize(image_array, size))
```

La funzione **download_images**, specificata la fonte sulla quale si vogliono cercare le immagini, utilizza due moduli diversi a seconda della sorgente scelta. Di seguito è descritto il modulo **google_grabber** (l'altro modulo è molto simile a quest'ultimo).

```
import json
import time
from selenium import webdriver
from selenium.webdriver.common.keys import Keys
from typing import List

class GrabbedImage:
    url = None
    extension = None
    base64 = None
    source = None

class GoogleGrabber(AbstractGrabber):
    full_image = True
    GOOGLE_URL = "https://www.google.co.in/search?q=%s&source=lnms&tbm=isch"

def __init__(self):
    pass

def get_images_url(self, keyword: str, nb_images: int) -> List[GrabbedImage]:
    query = keyword.split()
    query = '+'.join(query)
    url = self.GOOGLE_URL % query
    print('> searching image on Google : ' + url)
    options = webdriver.ChromeOptions()
    browser = webdriver.Chrome(chrome_options=options)
    browser.get(url)
    time.sleep(2)
    elem = browser.find_element_by_tag_name("body")
    no_of_pages_down = 20 if nb_images < 300 else 100
    while no_of_pages_down:
        elem.send_keys(Keys.PAGE_DOWN)
        time.sleep(0.2)
        no_of_pages_down -= 1
        try:
            show_more_btn = browser.find_element_by_id("smb")
            if show_more_btn.is_displayed():
                show_more_btn.click()
        except Exception as e:
            pass

    images_objects = []
    if self.full_image:
        images = browser.find_elements_by_class_name("rg_meta")
        for image in images:
            image_obj = GrabbedImage()
            image_obj.source = GrabSourceType.GOOGLE.value
            json_content = image.get_attribute('innerHTML')
            image_obj.url = json.loads(json_content)["ou"]
            image_obj.extension = json.loads(json_content)["ity"]
            images_objects.append(image_obj)
    else:
        images = browser.find_elements_by_class_name("rg_ic")
        for image in images:
            image_obj = GrabbedImage()
            image_obj.source = GrabSourceType.GOOGLE.value
            src = image.get_attribute('src')
```

```
58              if StringUtil.is_http_url(src):
59                  image_obj.url = src
60              else:
61                  image_obj.base64 = src
62              images_objects.append(image_obj)
63      browser.close()
64      return images_objects
```

La classe ***GoogleGrabber*** analizza il browser ricercando le possibili immagini e restituisce un array composto dagli URL(Uniform Resource Locator) delle immagini che si vogliono salvare.

4.1.2 Data Augmentation

La **data augmentation** è una tecnica utilizzata nell'analisi dei dati per aumentare la quantità degli stessi attraverso l'uso di trasformazioni casuali. É un metodo conveniente e frequentemente utilizzato per la generazione di un maggior numero di dati di addestramento con poco sforzo o quando l'accumulo di nuovi campioni non è più fattibile. Le trasformazioni che possono essere applicate possono essere diverse:

- **Flip**: l'immagine viene specchiata in orizzontale o verticale.
- **Rotation**: l'immagine viene ruotata verso destra o sinistra fino ad un massimo quantitativo di gradi definito.
- **Resize**: l'immagine viene ridimensionata.
- **Blur**: l'immagine viene sfocata.
- **Noise**: viene aggiunto un rumore di tipo gaussiano all'immagine.

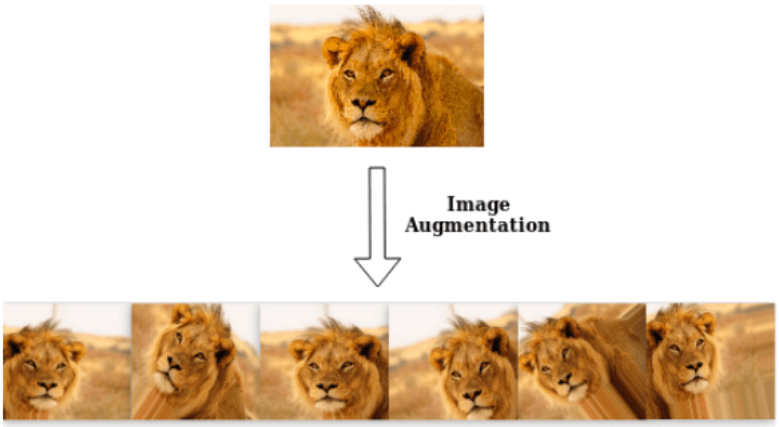

Figura 4.1 – Esempio di data augmentation

Solitamente la tecnica di data augmentation viene realizzata definendo una **pipline** di operazioni le quali possono essere applicate all'immagine di input [39] con una determinata probabilità. Durante il progetto è stato utilizzato un script

di augmentation il quale utilizza una pipeline di operazioni configurabili [40]. Lo script viene eseguito tramite il seguente comando:

```
!python3 augmentation.py [-folder_path <path_images>] [-limit <num_trasf>]
                         [destination_folder <path_aug>]
```

- **folder_path**: indica la cartella contenente le immagini a cui si vogliono effettuare le trasformazioni.
- **limit**: specifica il numero di operazioni che si vogliono effettuare (e quindi il numero delle nuove immagini che si vogliono generare).
- **destination_folder**: indica la cartella in cui saranno salvate le immagini trasformate.

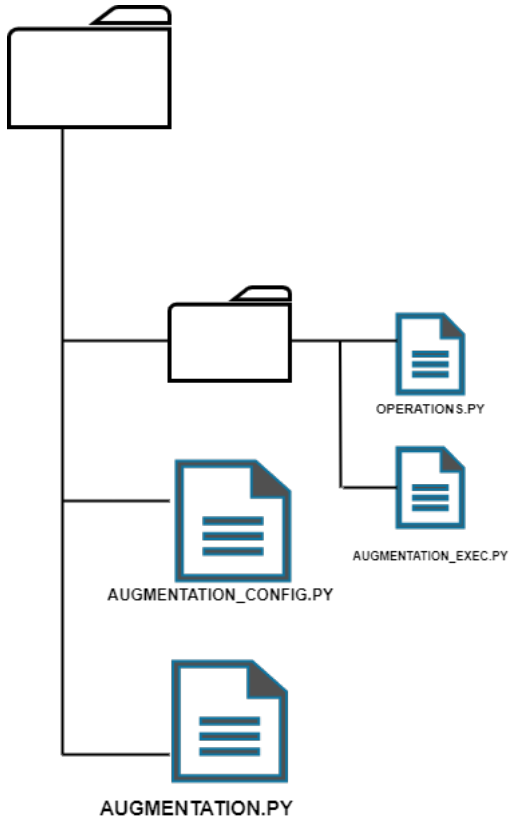

Figura 4.2 – Organizzazione del file system per lo script di augmentation

La definizione delle operazioni e le relative probabilità sono memorizzate nel file ***augmentation_config.py***. In tal modo è possibile modificare in qualisasi momento le probabilità che una determinata trasformazione sia effettuata sull'immagine di input.

```
1 DEFAULT_OPERATIONS = [
2     'rotate',
3     'blur',
4     'random_noise',
5     'horizontal_flip',
```

```
    'vertical_flip'
]

# Rotation configuration
DEFAULT_ROTATE_PROBABILITY = 0.5
DEFAULT_ROTATE_MAX_LEFT_DEGREE = 25
DEFAULT_ROTATE_MAX_RIGHT_DEGREE = 25

# Blur configuration
DEFAULT_BLUR_PROBABILITY = 0.1

# Random noise configuration
DEFAULT_RANDOM_NOISE_PROBABILITY = 0.5

# Horizontal flip configuration
DEFAULT_HORIZONTAL_FLIP_PROBABILITY = 0.3

# Horizontal flip configuration
DEFAULT_VERTICAL_FLIP_PROBABILITY = 0.3
```

Nello script ***operations.py*** sono definite e realizzate nello specifico tutte operazioni di data augmentation (per il codice delle operazioni e la definizione della pipeline si rimanda all'appendice). Una volta configurata la pipeline di operazioni lo script **augmentation_exec.py** le esegue in maniera casuale e salva le trasformazioni in una cartella.

```
from augmentation.operations import OperationPipeline
from image_grabber.grab_settings import DEBUG_MODE
from utils.utils import FileUtil, ProgressBarUtil, NoImageFoundException,
    ExceptionUtil


class DatasetGenerator(OperationPipeline):
    folder_path = None
    num_files = None
    save_to_disk = True
    folder_destination = "result"

    ...

    def execute(self):
        start_time = time.time()
        images_in_folder = FileUtil.get_images_file_path_array(self.
    folder_path)

        if not images_in_folder:
            raise (NoImageFoundException("No images found in %s folder" % self
    .folder_path))

        images_to_transform = []

        while len(images_to_transform) < self.num_files:
            images_to_transform.append(random.choice(images_in_folder))

        i = 0
        for file_path in images_to_transform:
            try:
                augmented_image = FileUtil.open(file_path)
                for operation in self.operations:
                    random_num = random.uniform(0, 1)
                    do_operation = random_num <= operation.probability
                    if do_operation:
                        augmented_image = operation.execute(augmented_image)
                if self.save_to_disk:
```

```
36                      FileUtil.save_file(augmented_image, self.
    folder_destination, "aug")
37              except Exception as e:
38                  ExceptionUtil.print(e)
39                  pass
40              finally:
41                  i = i + 1
42                  ProgressBarUtil.update(i, self.num_files)
43
44          end_time = time.time()
45          print('\n\n %s images generated in the folder %s in %s sec' % (self.
    num_files, self.folder_destination, round(end_time - start_time, 2)))
```

4.1.3 Etichettatura mediante LabelBox

Una volta raccolte le immagini e divise nei tre dataset, l'apprendimento automatico richiede che le immagini dei dataset di training e validation siano **etichettate**. L'etichettatura è stata effettuata tramite lo strumento Bounding-Box disponibile su Labelbox.

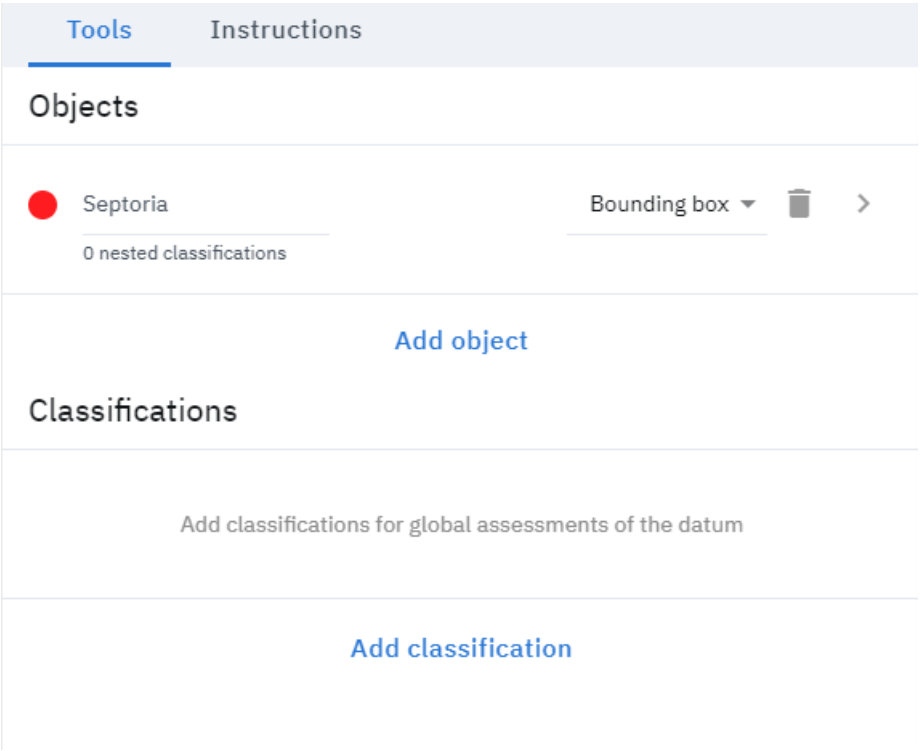

Figura 4.3 – Definizione dello strumento bounding box in Labelbox

Una volta caricato il dataset, infatti, Labelbox permette di definire diversi strumenti utili per il riconoscimento e la classificazione di oggetti definendo il valore e l'eventuale colore delle etichette(figura 4.3). Di seguito è rappresentato un esempio di pianta del grano affetta da septoriosi correttamente etichettata.

Figura 4.4 – Esempio di etichettatura di septoriosi del grano

4.1.4 Esportazione dei dataset in formato JSON

Una volta terminata l'etichettatura delle immagini ogni dataset può essere esportato in diversi formati tra cui il formato JSON [41]. Il formato JSON esportato sarà formato da diverse chiavi che saranno ripetute per ogni immagine del dataset:

- **Identificatori**: Per ogni immagine del dataset saranno assegnati un ID e un DataRowID.
- **Labeled Data**: Definisce l'url contente l'immagine non etichettata.
- **Label**: Per ogni immagine esportata, la chiave Label contiene tutte le informazioni sulle annotazioni. Le annotazioni sono divise in due categorie:
  1. *Objects*: Un vettore contenente tutte le etichette realizzate utilizzando gli strumenti Segmentation mask, Bounding box, Polygon, Polyline, Point.
  2. *Classification*: Un vettore contenente tutte le etichette realizzate utilizzando gli strumenti Radio, Checklist, Dropdown, e Text (solitamente utilizzati per classificazioni senza informazioni geometriche).
  3. *InstanceURI*: Riporta l'url contentente l'etichetta.
- **Dataset Name**: In questa chiave viene riportato il nome del dataset esportato.
- **View Label**: Riporta l'url di labelbox contente l'immagine etichettata (la quale può essere eventualmente modificata spostando l'etichetta)

Di seguito viene riportato un esempio di esportazione in formato JSON dei dataset realizzati. Avendo utilizzato lo strumento bounding box le informazioni sulle etichette saranno memorizzate nel vettore object il quale contiene diverse

chiavi. La più importante è la chiave bbox la quale attraverso i 4 campi *top, left, height e width* individua la posizione dell'etichetta all'interno dell'immagine.

```
"ID": "ckdivbsvr00003r5s68iaw3yu",
"DataRow ID": "ckdiv7usbihbw0aow2w5r0990",
"Labeled Data": "https://storage.labelbox.com/ckaf3r6..."
"Label": {
   "objects": [
      {
        "featureId": "ckdivbpco0lxj0y2t7oeadmka",
        "schemaId": "ckdivbiq60k5q0y7l97xyh7ji",
        "title": "Septoria",
        "value": "septoria",
        "color": "#006bff",
        "bbox": {
            "top": 186,
            "left": 192,
            "height": 300,
            "width": 519
        },
        "instanceURI": "https://api.labelbox.com/masks/feature/ckb..."
      }
    ],
    "classifications": []
}
...
"Dataset Name": "Septoria_training",
"View Label":"https://editor.labelbox.com?project=ck..."
```

4.2 Addestramento della rete

Di seguito sono riportati tutti i passaggi effettuati durante l'addestramento dei due modelli di rete. L'addestramento è avvenuto tramite l'utilizzo di **Jupyter Notebooks** di Colaboratory in modo da sfruttare la potenza di calcolo offerta da Google.

4.2.1 YOLO

Il modello YOLO utilizzato è disponibile in una repository github [42]. Per poter accedere ai suoi contenuti è sufficente utilizzare i seguenti comandi all'interno del notebook:

```
$ git clone https://github.com/ultralytics/yolov3
$ cd yolov3
$ pip install -r requirements.txt
```

4.2.1.1 Trasformazione dataset in formato Darknet

Prima di essere addestrato il dataset di training deve essere esportato in formato **Darknet**(figura 4.5), il quale prevede un file *.txt per ogni immagine (se non è presente nessuna annotazione nell'immagine il file non è richiesto). Le specifiche del file sono:

- Una **riga** per ogni oggetto etichettato.
- Ogni riga è costituita da quattro valori *xcenter, ycenter, width, height* i quali rappresentano le coordinate dell'etichetta. Le coordinate devono

essere **normalizzate** (con valore compreso da 0 a 1). Se le coordinate sono espresse in pixel, è sufficiente dividere *xcenter* e *width* per la lunghezza dell'immagine ed *ycenter* e *heigth* per l'altezza dell'immagine.

- Le classi da riconoscere sono **indicizzate** (a partire da 0).

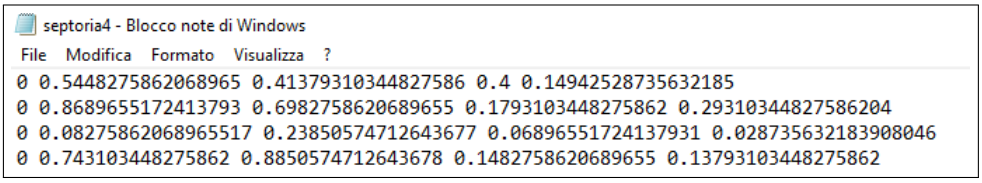

Figura 4.5 – Esempio di label in formato Darknet

Per poter trasformare il dataset dal formato JSON in formato Darknet è stato utilizzato uno script, il quale una volta scaricato il file JSON, effettua la conversione normalizzando i valori espressi in pixel nella chiave bbox ed associando a ciascuna immagine (le quali saranno salvate in una cartella images) la sua etichetta (le quali saranno salvate nella cartella labels). Lo script è strutturato in questo modo:

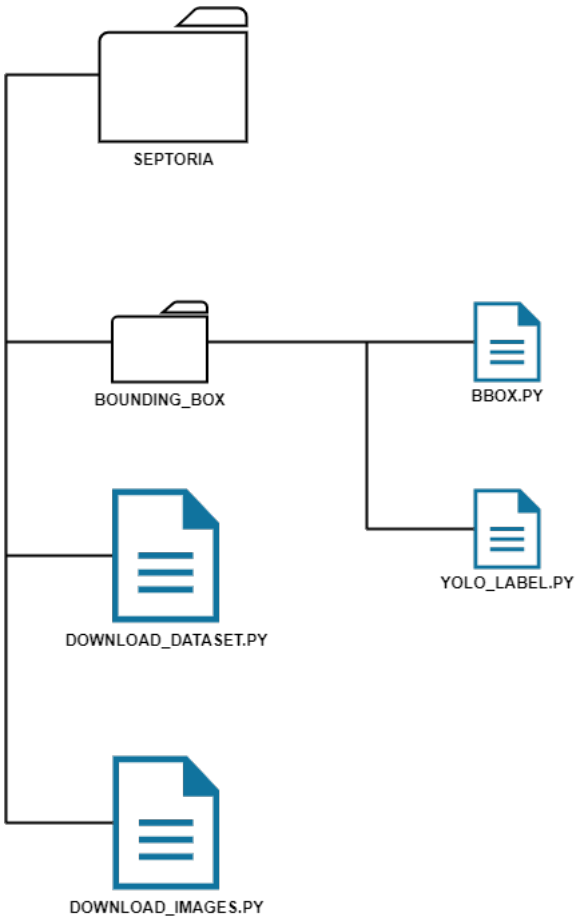

Figura 4.6 – Struttura dello script per la conversione delle etichette in formato Darknet

- Il modulo **download_dataset.py** effettua il download del file JSON contenente le varie informazioni riguardanti le etichette, create mediante Labelbox. Una volta scaricato la funzione genera inoltre le due sottocartelle images e labels in cui saranno salvate rispettivamente le immagini del dataset e le loro rispettive etichette tradotte in formato Darknet.

```
import os,urllib.request,json,shutil
import download_images as im
import bounding_box.bbox as bbox

def download_dataset(path,url):
  if os.path.exists(path):
    shutil.rmtree(path):
  os.makedirs(path)
    try:
        urllib.request.urlretrieve(url,path+"/dataset.json")
        dataset=json_parse(path,"dataset.json")
    except Exception:
        print("Errore durante il download\nControllare la connessione!")
        dataset=[]
  return dataset

...


def dataset(path):
  url="https://storage.googleapis.com/labelbox-exports/ck..."
    dataset=download_dataset(path+"/dataset",url)
  if len(dataset)>0:
    images_dir="images"
    labels_dir="labels"
    create_directory(path+"/dataset",images_dir,labels_dir)
    im.download_images(path+"/dataset",dataset,images_dir,labels_dir)
    bbox.label_images(path+"/dataset",images_dir)
```

- Il modulo **download_images.py** prende in input il file JSON scaricato e salva tutte le immagini appartenenti al dataset nella cartella images (il nome scelto avrà un prefisso il quale sarà quello della classe). Il download delle immagini è effettuato leggendo all'interno del file JSON tutti i valori contenuti dalla chiave LabeledData.

```
import urllib.request
...
import bounding_box.yolo_labels as lb

def download_images(path,dataset,images_dir,labels_dir):
    print("\nDownload immagini in corso... ")
    name=list(dataset[0]["Label"].keys())[0]
    j=0
    for data in dataset:
        date=data["Updated At"]
        if confronta_data(date) and len(data["Label"])>0:
            nome_file="septoria"+str(j)
            try:
                urllib.request.urlretrieve(data["LabeledData"],path+"/"+
    images_dir+"/"+nome_file+".jpg")
            except Exception:
                print("Errore durante il download\nControllare la
    connessione!")
            file=open(path+"/"+labels_dir+"/"+nome_file+utils.EXTENSION,
    "w")
            file.write(utils.labels_to_file(data["Label"][name],path,
    images_dir,nome_file+".jpg"))
```

```
            file.close()
            j=j+1
    print("\nDownload terminato\n")
```

- Il modulo **bbox.py** permette di salvare le immagini del dataset all'interno della cartella labeled_images disegnando per ciascuna le relative etichette.

```
import os,shutil
...
import bounding_box.yolo_labels as lb
from PIL import Image, ImageDraw

def save_image(path,nome_file,mode):
    image_extension="."+nome_file.split(".")[1]
    nome_file=nome_file.split(".")[0]
    if os.path.exists(path+"/images") and os.path.exists(path+"/labels")
     and os.path.isfile(path+"/images/"+nome_file+image_extension):
        image=Image.open(path+"/images/"+nome_file+image_extension)
        utils=lb
        data={"size":image.size}
        file=open(path+"/labels/"+nome_file+utils.EXTENSION,"r")
        lines=file.readlines()
        file.close()
        labels=[]
        for line in lines:
            data["field"]=line
            data=utils.add_bbox(data,labels)
        image=draw_bbox(labels,image,utils)
        image.save(path+"/labeled_images/"+nome_file+".jpg","JPEG")
        image.close()


...


def label_images(path,images_dir):
    if os.path.exists(path+"/labeld_images"):
        shutil.rmtree(path+"/labeled_images")
    os.makedirs(path+"/labeled_images")
    files=os.listdir(path+"/"+images_dir)
    for nome_file in files:
        save_image(path,nome_file)
```

- Il modulo **yolo_label.py** genera la stringa contenente le coordinate delle etichette normalizzate come previsto dal formato Darknet.

```
def labels_to_file(labels,path,cartella,nome_file):
    image_path=path+"/"+cartella+"/"+nome_file
    picture=Image.open(image_path)
    (picture_width,picture_height)=picture.size
    new_size=(800,600)
    picture=picture.convert('RGB')
    picture=picture.resize(new_size)
    picture.save(image_path)
    picture.close()
    s=""
    parse=parsing_top_left
    for bbox in labels:
        (x_center,y_center,width,height)=parse(bbox)
        x_center=x_center/picture_width
        y_center=y_center/picture_height
        width=width/picture_width
        height=height/picture_height
        s+="0"+str(x_center)+" "+str(y_center)+" "+str(width)+" "+str(
    height)+"\n"
```

```
    return s

def parsing_top_left(bbox):
    top=bbox["bbox"]["top"]
    left=bbox["bbox"]["left"]
    height=bbox["bbox"]["height"]
    width=bbox["bbox"]["width"]
    x_center=left+width/2
    y_center=top+height/2
    return (x_center,y_center,width,height)
```

4.2.1.2 Configurazione

Prima di effettuare l'addestramento la rete necessita di diversi file di configurazione i quali sono stati creati tramite le seguenti funzioni:

- La rete utilizza un file di **configurazione** il quale indica il numero di filtri da utilizzare durante l'addestramento ed il numero di classi da individuare (fino ad un massimo di 80). La repository possiede un file config standard utilizzato per la classificazione del dataset COCO, un dataset contenente migliaia di immagini da classificare in 80 classi. Tale file sarà modificato per il nostro dataset monoclasse diminuendo i filtri da 255 a 18 e modificando il numero di classi ad una sola. La funzione **create_cfg** modifica il file esistente yolov3.cfg con i parametri richiesti creando un nuovo file di configurazione septoria.cfg.

```
def create_cfg(path):
    if not path.endswith("/yolo"):
        path+="/yolo"
    file=open(path+"/cfg/yolov3.cfg","r")
    lines=file.readlines()
    file.close()
    i=0
    while i<len(lines):
        start=lines[i].find("filters=255")
        if start!=-1:
            end=start+len("filters=255")
            lines[i]="filters=18"+lines[i][end:]
        else:
            start=lines[i].find("classes=80")
            if start!=-1:
                end=len("classes=80")
                lines[i]="classes=1"+lines[i][end:]
        i=i+1
    file=open(path+"/cfg/septoria.cfg","w")
    s=""
    for line in lines:
        s+=line
    file.write(s)

```

- Un ulteriore file di configurazione prevede la creazione di un file .txt contente tutti i path delle immagini che la rete dovrà addestrare. La funzione **create_septoria_txt** prende in input la directory contenente le immagini e ne salva i percorsi in una stringa.

```
def create_septoria_txt(path):
    pictures_name=os.listdir(path)
    s=""
```

```
4     for picture in pictures_name:
5         s+=path+"/"+picture+"\n"
6     return s
```

- Il nome delle classi che si vogliono individuare viene salvato in un file di testo con estensione .names; per ogni riga del file sarà associato un **indice** (a partire da 0) con il quale è indicata il valore dell'etichetta nelle label. Nel nostro caso il file sarà solo composto da una stringa "septoria".

```
1 def create_septoria_names():
2 return "septoria"
```

- L'insieme delle informazioni riguardanti la configurazione della rete (numero di classi, il loro nome, il path delle immagini) sono salavati in un file di estensione .data il quale sarà creato dalla funzione **create_septoria_data**.

```
1 def create_septoria_data():
2 s=("classes=1\n")
3 s+=("train=data/septoria.txt\n")
4 s+=("valid=data/septoria.txt\n")
5 s+=("names=data/septoria.names\n")
6 return s
```

4.2.1.3 Addestramento

Una volta terminata la configurazione è possibile avviare l'addestramento tramite il seguente comando:

```
!python3 train.py --data septoria.data --cfg septoria.cfg
                  --weights yolov3-spp-ultralytics.pt --batch=8
                  --epochs=300 --nosave --name septoria
```

- **Data**: specifica il file di configurazione data con cui vogliamo addestrare il modello.
- **Cfg**: indica il file di configurazione da utilizzare.
- **Weights**: indica il file contenente i valore dei pesi della rete preaddestrata.
- **Batch**: i diversi pattern non sono analizzati dalla rete tutti insieme ma sono divisi in batch. Dunque l'argomento batch specifica il numero di immagini per ogni gruppo analizzato.
- **Epochs**: l'addestramento prevede che il dataset sia addestrato per un numero n di volte. Per un epoca si intende la presentazione alla rete di tutti i pattern del training dataset. Dunque ad ogni epoca, gli n pattern del training set sono ordinati in modo casuale e poi suddivisi in gruppi (batch).
- **Nosave**: se specificato la rete salverà il modello solo ad addestramento terminato.
- **Name**: specifica il nome dei pesi generati una volta addestrato il modello.

Nella seguente immagine è riportata una porzione dell'addestramento effettuato ed il file test_batch0 generato dalla rete:

```
     Epoch   gpu_mem       GIoU        obj        cls      total    targets  img_size
260/299         9G       1.13       1.12          0       2.25         42        384: 100% 19/19 [00:23<00:00,  1.25s/it]
               Class     Images    Targets          P          R    mAP@0.5         F1: 100% 19/19 [00:11<00:00,  1.60it/s]
                 all        147    1.1e+03      0.852      0.979      0.985      0.911

     Epoch   gpu_mem       GIoU        obj        cls      total    targets  img_size
261/299         9G       1.17       1.34          0       2.51         34        352: 100% 19/19 [00:23<00:00,  1.26s/it]
               Class     Images    Targets          P          R    mAP@0.5         F1: 100% 19/19 [00:11<00:00,  1.61it/s]
                 all        147    1.1e+03      0.842      0.977      0.984      0.904

     Epoch   gpu_mem       GIoU        obj        cls      total    targets  img_size
262/299         9G       1.12      0.945          0       2.07         39        608: 100% 19/19 [00:27<00:00,  1.45s/it]
               Class     Images    Targets          P          R    mAP@0.5         F1: 100% 19/19 [00:11<00:00,  1.61it/s]
                 all        147    1.1e+03      0.844      0.978      0.984      0.906

     Epoch   gpu_mem       GIoU        obj        cls      total    targets  img_size
263/299         9G       1.07        1.2          0       2.27         24        384: 100% 19/19 [00:21<00:00,  1.12s/it]
               Class     Images    Targets          P          R    mAP@0.5         F1: 100% 19/19 [00:11<00:00,  1.61it/s]
                 all        147    1.1e+03      0.852      0.978      0.984      0.911

     Epoch   gpu_mem       GIoU        obj        cls      total    targets  img_size
264/299         9G       1.09        1.1          0       2.19         18        384: 100% 19/19 [00:19<00:00,  1.03s/it]
               Class     Images    Targets          P          R    mAP@0.5         F1: 100% 19/19 [00:11<00:00,  1.61it/s]
                 all        147    1.1e+03      0.852      0.978      0.984      0.911
```

Figura 4.7 – Esempio di addestramento della rete Yolo

Figura 4.8 – Etichette generate dalla rete per il batch 0

Una volta terminato l'addestramento i nuovi pesi aggiornati sono salvati in un file chiamato **last_septoria.pt**, il quale sarà salvato nel drive e successivamente utilizzato nella fase di validation e test. Yolo inoltre registra gli **indici di prestazione** della rete su Tensorboard e in un file di log personalizzato chiamato result.txt il quale può essere trasformato in un grafico (figura 4.9) tramite il comando:

```
from utils import utils;
utils.plot_results()
```

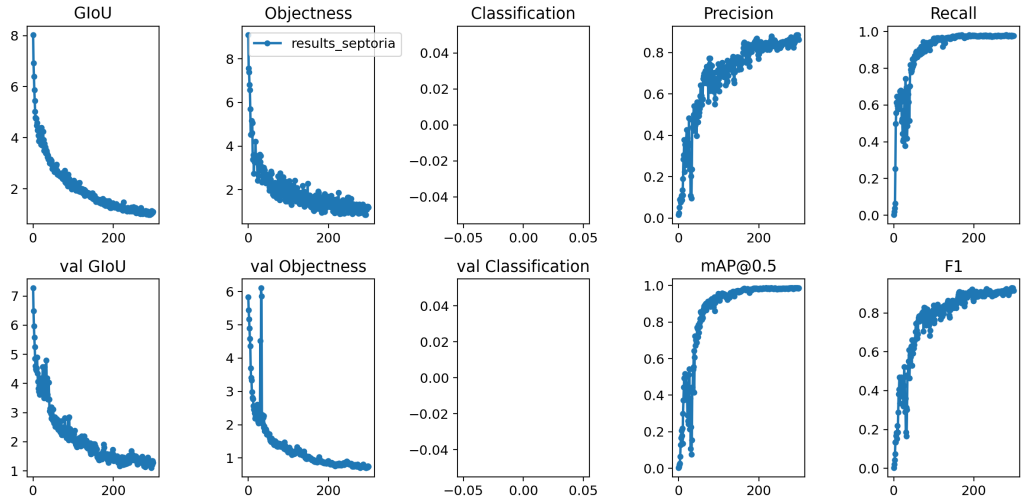

Figura 4.9 – Grafico generato da Tensorboard dopo l'addestramento

4.2.2 Faster RCNN

Il modello Faster RCNN è disponibile presso la repository ufficiale di Tensorflow [43]. Per accedere ai contenuti della repository è sufficiente eseguire un semplice script:

```
def download_neural_network(path):
    url="https://github.com/tensorflow/models/archive/master.zip"
    detection_path=path+"/utils/neural_network/object_detection"
    path+="/faster_RCNN"
    print("\nDownload Tensorflow in corso...")
    if os.path.exists(path):
        shutil.rmtree(path)
    os.makedirs(path)
    if not os.path.exists(path+"/CP"):
        os.makedirs(path+"/CP")
    if not os.path.exists(path+"/weights"):
        os.makedirs(path+"/weights")
    try:
        urllib.request.urlretrieve(url,path+"/tensorflow.zip")
    except Exception:
        print("Errore durante il download\nControllare la connessione!")
        path="NULL"
    file=zipfile.ZipFile(path+"/tensorflow.zip")
    file.extractall(path)
    file.close()
    os.remove(path+"/tensorflow.zip")
    return path
```

Una volta scaricata la repository è necessario installare le librerie tf_slim e tensorflow-gpu utilizzando il comando pip:

```
!pip install tf_slim
!pip install tensorflow-gpu==1.15
```

4.2.2.1 Configurazione

Prima di effettuare l'addestramento, è necessario scaricare un modello preaddestrato e configurare alcuni file. La configurazione della rete è stata eseguita dal

seguente script:

```
def config_network(path):
    create_label_map(path+"/dataset")
    nome_modello="faster_rcnn_resnet101_coco"
    url="http://download.tensorflow.org/models/object_detection/faster_rcnn
    ....tar.gz"
    if not os.path.exists(path+"/pre_trained_models"):
        os.makedirs(path+"/pre_trained_models")
    try:
        urllib.request.urlretrieve(url,path+"/pre_trained_models/model.tar.gz"
    )
    except Exception:
        print("Errore durante il download\nControllare la connessione!")
    file=tarfile.open(path+"/pre_trained_models/model.tar.gz")
    file.extractall(path+"/pre_trained_models")
    file.close()
    os.remove(path+"/pre_trained_models/model.tar.gz")
    file=open(path+"/models-master/research/object_detection/samples/configs/"
    +nome_modello+".config","r")
    data=file.readlines()
    file.close()
    model_path=path+"/pre_trained_models/"+os.listdir(path+"/
    pre_trained_models")[0]
    i=0
    while i<len(data):
        if data[i].find("PATH_TO_BE_CONFIGURED")!=-1:
            if data[i].find("fine_tune_checkpoint")!=-1:
                data[i]="fine_tune_checkpoint: '"+model_path+"/model.ckpt'\n"
            else:
                if data[i].find("input_path")!=-1:
                    data[i]="input_path: '"+path+"/dataset/train.record'\n"
                else:
                    if data[i].find("label_map_path")!=-1:
                        data[i]="label_map_path: '"+path+"/dataset/label.pbtxt
    '\n"
        i=i+1
    file=open(model_path+"/"+nome_modello+".config","w")
    for line in data:
        file.write(line)
    file.close()
```

- Tensorflow utilizza una mappa delle labels la quale assegna ad ogni etichetta un valore intero. Tale mappa sarà utilizzata sia in fase di training che di object detection.

```
item {
    id: 1
    name: 'cat'
}

item {
    id: 2
    name: 'dog'
}
```

Figura 4.10 – Esempio di label-map

Nel nostro caso la mappa è stata creata dalla funzione **create_label_map**:

```
def create_label_map(path):
    s="item {\n\tid: 1\n\tname: 'septoria'\n}"
```

```
3     file=open(path+"/label.pbtxt","w")
4     file.write(s)
5     file.close()
```

- Una volta scaricato il modello preaddestrato desiderato (nel nostro caso il modello scelto è il faster_rcnn_resnet101_coco), è possibile configurare un file .config in base alle proprie necessità. Nel file infatti sono presenti delle sezioni sotto il nome di **PATH_TO_BE_CONFIGURED** in cui è possibile specificare i file riguardanti la mappa delle etichette, i record di input ed i chekpoint (file i quali acquisiscono i valori esatti di tutti i parametri utilizzati) del modello.

4.2.2.2 Trasformazione dataset in formato Pascal VOC

Prima di iniziare l'addestramento vero e proprio il dataset di training deve essere trasformato in formato **Pascal VOC** e successivamente trasformato in formato tf record. Il formato Pascal VOC richiede che ad ogni immagine del dataset sia associato un file xml il quale riporta le informazioni relative alle annotazioni. Il file è formato da diversi campi racchiusi nei vari tag:

```
<annotation>
        <folder>GeneratedData_Train</folder>
        <filename>000001.png</filename>
        <path>/my/path/GeneratedData_Train/000001.png</path>
        <source>
                <database>Unknown</database>
        </source>
        <size>
                <width>224</width>
                <height>224</height>
                <depth>3</depth>
        </size>
        <segmented>0</segmented>
        <object>
                <name>21</name>
                <pose>Frontal</pose>
                <truncated>0</truncated>
                <difficult>0</difficult>
                <occluded>0</occluded>
                <bndbox>
                        <xmin>82</xmin>
                        <xmax>172</xmax>
                        <ymin>88</ymin>
                        <ymax>146</ymax>
                </bndbox>
        </object>
</annotation>
```

Figura 4.11 – Esempio di file in formato Pascal VOC

- **Folder**: Specifica la cartella dove è salvata l'immagine.
- **Filename**: Specifica il nome del file.

- **Path**: Indica l'indirizzo dell'immagine nel file system.
- **Size**: Specifica le dimensioni dell'immagine.
- **Object**: Permette di riportare le informazioni relative alle etichette come il nome e la sua posizione in termini di limiti superiori ed inferiori.

In maniera analoga a quanto accaduto con il modello precedente il dataset di training in formato JSON è stato trasformato in formato Pascal VOC tramite lo script **labels_to_file** modificato in maniera adeguata:

```
def labels_to_file(labels,path,cartella,nome_file):
    image_path=path+"/"+cartella+"/"+nome_file
    picture=Image.open(image_path)
    if picture.mode !='RGB':
        picture=picture.convert('RGB')
    (picture_width,picture_height)=picture.size
    new_size=(400,300)
    resize_factor=(new_size[0]/picture_width,new_size[1]/picture_height)
    picture=picture.resize(new_size)
    picture.save(image_path)
    picture.close()
    s="<annotation>\n"
    s+="\t<folder>"+cartella+"</folder>\n"
    s+="\t<filename>"+nome_file+"</filename>\n"
    s+="\t<path>"+image_path+"</path>\n"
    s+="\t<source>\n\t\t<database>Septoria</database>\n\t</source>\n"
    s+="\t<size>\n\t\t<width>"+str(new_size[0])+"</width>\n\t\t<height>"+str(
    new_size[1])+"</height>\n\t\t<depth>3</depth>\n\t</size>\n"
    s+="\t<segmented>0</segmented>\n"
    for bbox in labels:
        ymin=resize_factor[1]*bbox["bbox"]["top"]
        xmin=resize_factor[0]*bbox["bbox"]["left"]
        height=resize_factor[1]*bbox["bbox"]["height"]
        width=resize_factor[0]*bbox["bbox"]["width"]
        ymax=ymin+height
        xmax=xmin+width
        temp="\t\t\t<xmin>"+str(round(xmin))+"</xmin>\n\t\t\t<ymin>"+str(round
    (ymin))+"</ymin>\n\t\t\t<xmax>"+str(round(xmax))+"</xmax>\n\t\t\t<ymax>"+
    str(round(ymax))+"</ymax>\n"
        s+="\t<object>\n\t\t<name>septoria</name>\n\t\t<pose>Unspecified</pose
    >\n\t\t<truncated>0</truncated>\n\t\t<difficult>0</difficult>\n\t\t<bndbox
    >\n"+temp+"\t\t</bndbox>\n\t</object>\n"
    s+="</annotation>"
    return s
```

4.2.2.3 Creazione dei TFRecords

La rete Faster RCNN richiede il formato tf record il quale può essere generato mediante lo script **create_pascal_tf_record.py**. Lo script prende in input i dati in formato Pascal VOC e li trasforma in formato tf record effettuando anche la normalizzazione delle coordinate del bounding box.

```
def dict_to_tf_example(data,
                       dataset_directory,
                       label_map_dict,
                       ignore_difficult_instances=False):

  ...

  width = int(data['size']['width'])
```

```
  height = int(data['size']['height'])
  xmin = []
  ymin = []
  xmax = []
  ymax = []
  classes = []
  classes_text = []
  truncated = []
  poses = []
  difficult_obj = []
  if 'object' in data:
    for obj in data['object']:
      difficult = bool(int(obj['difficult']))
      if ignore_difficult_instances and difficult:
        continue
      difficult_obj.append(int(difficult))
      xmin.append(float(obj['bndbox']['xmin']) / width)
      ymin.append(float(obj['bndbox']['ymin']) / height)
      xmax.append(float(obj['bndbox']['xmax']) / width)
      ymax.append(float(obj['bndbox']['ymax']) / height)
      classes_text.append(obj['name'].encode('utf8'))
      classes.append(label_map_dict[obj['name']])
      truncated.append(int(obj['truncated']))
      poses.append(obj['pose'].encode('utf8'))
  example = tf.train.Example(features=tf.train.Features(feature={
      'image/height': dataset_util.int64_feature(height),
      'image/width': dataset_util.int64_feature(width),
      'image/filename': dataset_util.bytes_feature(
          data['filename'].encode('utf8')),
      'image/source_id': dataset_util.bytes_feature(
          data['filename'].encode('utf8')),
      'image/key/sha256': dataset_util.bytes_feature(key.encode('utf8')),
      'image/encoded': dataset_util.bytes_feature(encoded_jpg),
      'image/format': dataset_util.bytes_feature('jpeg'.encode('utf8')),
      'image/object/bbox/xmin': dataset_util.float_list_feature(xmin),
      'image/object/bbox/xmax': dataset_util.float_list_feature(xmax),
      'image/object/bbox/ymin': dataset_util.float_list_feature(ymin),
      'image/object/bbox/ymax': dataset_util.float_list_feature(ymax),
      'image/object/class/text': dataset_util.bytes_list_feature(classes_text)
    ,
      'image/object/class/label': dataset_util.int64_list_feature(classes),
      'image/object/difficult': dataset_util.int64_list_feature(difficult_obj)
    ,
      'image/object/truncated': dataset_util.int64_list_feature(truncated),
      'image/object/view': dataset_util.bytes_list_feature(poses),
  }))
  return example
```

La trasformazione viene effettuata lanciando il seguente comando:

```
!python3 /content/septoria/dataset/create_pascal_tf_record.py \
    --data_dir={dataset_path} \
    --annotations_dir={labels} \
    --output_path={output_path} \
    --label_map_path={label_map_path}
```

- **data_dir**: indica la cartella del dataset in formato Pascal VOC che si vuole trasformare.
- **annotations**: specifica la cartella dove sono salavate le annotazioni xml.
- **output_path**: indica la cartella in cui sarà salvato il file record.
- **label_map_path**: specifica l'indirizzo della label map che si vuole utilizzare.

4.2.2.4 Addestramento

Una volta generato il file record è possibile effettuare l'addestramento vero e proprio lanciando il seguente comando:

```
!python3 object_detection/legacy/train.py \
    --train_dir={train_path} \
    --pipeline_config_path={pipeline_config_path} \
    --num_train_steps={num_train_steps} \
```

- **train_dir**: specifica la cartella in cui saranno salvati i file chkp e meta relativi al modello.
- **pipeline_config_path**: specifica il file di configurazione da utilizzare durante l'addestramento.
- **num_train_steps**: indica il numero di passi in cui la rete andrà a modificare i parametri addestrabili.

```
I0820 08:02:54.920987 140394427864960 learning.py:512] global step 10670: loss = 0.1333 (0.657 sec/step)
INFO:tensorflow:global step 10671: loss = 0.0431 (1.188 sec/step)
I0820 08:02:56.276692 140394427864960 learning.py:512] global step 10671: loss = 0.0431 (1.188 sec/step)
INFO:tensorflow:global step 10672: loss = 0.2828 (1.268 sec/step)
I0820 08:02:57.622917 140394427864960 learning.py:512] global step 10672: loss = 0.2828 (1.268 sec/step)
INFO:tensorflow:global step 10673: loss = 0.1786 (0.941 sec/step)
I0820 08:02:58.580170 140394427864960 learning.py:512] global step 10673: loss = 0.1786 (0.941 sec/step)
INFO:tensorflow:global step 10674: loss = 0.0981 (0.980 sec/step)
I0820 08:02:59.669995 140394427864960 learning.py:512] global step 10674: loss = 0.0981 (0.980 sec/step)
INFO:tensorflow:Recording summary at step 10674.
I0820 08:02:59.793692 140391035754240 supervisor.py:1050] Recording summary at step 10674.
INFO:tensorflow:global step 10675: loss = 0.0749 (0.743 sec/step)
I0820 08:03:00.416523 140394427864960 learning.py:512] global step 10675: loss = 0.0749 (0.743 sec/step)
INFO:tensorflow:global step 10676: loss = 0.0513 (0.660 sec/step)
I0820 08:03:01.078304 140394427864960 learning.py:512] global step 10676: loss = 0.0513 (0.660 sec/step)
INFO:tensorflow:global step 10677: loss = 0.1395 (0.750 sec/step)
I0820 08:03:01.830467 140394427864960 learning.py:512] global step 10677: loss = 0.1395 (0.750 sec/step)
INFO:tensorflow:global step 10678: loss = 0.1070 (0.660 sec/step)
I0820 08:03:02.491855 140394427864960 learning.py:512] global step 10678: loss = 0.1070 (0.660 sec/step)
INFO:tensorflow:global step 10679: loss = 0.0873 (0.657 sec/step)
I0820 08:03:03.150187 140394427864960 learning.py:512] global step 10679: loss = 0.0873 (0.657 sec/step)
INFO:tensorflow:global step 10680: loss = 0.1261 (0.666 sec/step)
I0820 08:03:03.818120 140394427864960 learning.py:512] global step 10680: loss = 0.1261 (0.666 sec/step)
```

Figura 4.12 – Esempio di addestramento della rete Faster RCNN

Una volta terminato l'addestramento il modello ed i suoi pesi sono salvati in un **inference graph** mediante il comando:

```
import re
import numpy as np

output_path="/content/septoria/faster_RCNN/weights"
model_dir="/content/septoria/faster_RCNN/CP"
pipeline_fname="/content/septoria/faster_RCNN/pre_trained_models/
    faster_rcnn_resnet101_coco_2018_01_28/faster_rcnn_resnet101_coco.config"
lst = os.listdir(model_dir)
lst = [l for l in lst if 'model.ckpt-' in l and '.meta' in l]
steps=np.array([int(re.findall('\d+', l)[0]) for l in lst])
last_model = lst[steps.argmax()].replace('.meta', '')
last_model_path = os.path.join(model_dir, last_model)
!python3 /content/septoria/faster_RCNN/models-master/research/object_detection
    /export_inference_graph.py \
```

```
--input_type=image_tensor \
--pipeline_config_path={pipeline_fname} \
--output_directory={output_path} \
--trained_checkpoint_prefix={last_model_path}
```

- **pipeline_config_path**: specifica il file di configurazione da utilizzare.
- **output_directory**: specifica la cartella dove sarà salvato il grafo di inferenza.
- **trained_chekpoint_prefix**: specifica quale file ckpt del modello utilizzare per la generazione del grafo.

4.3 Object recognition

La fase di object recognition prevede che i modelli addestrati siano utilizzati per classificare le immagini del dataset di test il quale contiene anche immagini di piante sane non affette da septoriosi. Le immagini del dataset di training sono state inserite in una cartella samples la quale sarà utilizzata come input dai due modelli. I risultati oltre ad essere salvati come immagini sono salvati in un file txt.

4.3.1 Yolo

Una volta riconfigurato la rete Yolo e scaricato i pesi del nostro modello dal drive (last_septoria.pt), è possibile effettuare l'object detection lanciando il seguende comando:

```
!python3 detect.py --source /samples --weights /weights/last_septoria.pt
                   --out yolov3/results --cfg cfg/septoria.cfg
                   --names data/septoria.names
```

- **source**: specifica la cartella contenente le immagini che si vogliono classificare.
- **weights**: specificano i pesi del modello che si vogliono utilizzare per effettuare l'inferenza.
- **out**: indica la cartella dove saranno salvati i risultati.
- **cfg**: specifica il file di configurazione da utilizzare.
- **names**: specifica le classi che si vogliono individuare.

```
Model Summary: 225 layers, 6.25733e+07 parameters, 6.25733e+07 gradients
image 1/50 /content/septoria/sample/0a029eead_15.jpg: 512x512 Done. (2.099s)
image 2/50 /content/septoria/sample/0a73f8a0f.jpg: 512x512 1 septorias, Done. (1.947s)
image 3/50 /content/septoria/sample/0a73f8a0f_0.jpg: 512x512 Done. (1.937s)
image 4/50 /content/septoria/sample/0a73f8a0f_1.jpg: 512x512 Done. (1.938s)
image 5/50 /content/septoria/sample/0a73f8a0f_10.jpg: 512x512 Done. (1.935s)
image 6/50 /content/septoria/sample/0a73f8a0f_11.jpg: 512x512 Done. (1.934s)
image 7/50 /content/septoria/sample/0a73f8a0f_12.jpg: 512x512 Done. (1.968s)
image 8/50 /content/septoria/sample/0a73f8a0f_13.jpg: 512x512 Done. (1.930s)
image 9/50 /content/septoria/sample/0a73f8a0f_14.jpg: 512x512 Done. (1.954s)
image 10/50 /content/septoria/sample/0a73f8a0f_15.jpg: 512x512 Done. (1.934s)
image 11/50 /content/septoria/sample/0a73f8a0f_2.jpg: 512x512 Done. (1.939s)
image 12/50 /content/septoria/sample/0a73f8a0f_3.jpg: 512x512 Done. (1.946s)
image 13/50 /content/septoria/sample/0a73f8a0f_4.jpg: 512x512 Done. (1.938s)
image 14/50 /content/septoria/sample/0a73f8a0f_5.jpg: 512x512 Done. (1.932s)
image 15/50 /content/septoria/sample/0a73f8a0f_6.jpg: 512x512 Done. (1.937s)
image 16/50 /content/septoria/sample/0a73f8a0f_7.jpg: 512x512 Done. (1.951s)
image 17/50 /content/septoria/sample/0a73f8a0f_8.jpg: 512x512 Done. (1.949s)
image 18/50 /content/septoria/sample/0a73f8a0f_9.jpg: 512x512 1 septorias, Done. (1.926s)
image 19/50 /content/septoria/sample/0a97d54ae.jpg: 512x512 Done. (1.929s)
image 20/50 /content/septoria/sample/0a97d54ae_0.jpg: 512x512 Done. (1.951s)
image 21/50 /content/septoria/sample/0a97d54ae_1.jpg: 512x512 Done. (1.939s)
image 22/50 /content/septoria/sample/0a97d54ae_2.jpg: 512x512 Done. (1.951s)
image 23/50 /content/septoria/sample/0a97d54ae_3.jpg: 512x512 Done. (1.938s)
image 24/50 /content/septoria/sample/0a97d54ae_4.jpg: 512x512 Done. (1.958s)
image 25/50 /content/septoria/sample/111.jpg: 512x384 Done. (1.492s)
image 26/50 /content/septoria/sample/112.jpg: 384x512 24 septorias, Done. (1.493s)
```

Figura 4.13 – Esempio di detect in Yolo

Una volta terminata la fase di detect è possibile visualizzare i risultati mediante un semplice script:

```
from IPython.display import Image, display
path="/content/septoria/yolo/results"
for image in os.listdir(path):
    if not image.endswith(".txt"):
      display(Image(filename=path+"/"+image))
```

Figura 4.14 – Detect Yolo su una pianta malata ed una sana

4.3.2 Faster RCNN

In maniera simile a quanto avvenuto per il modello precedente, anche nella rete Faset RCNN l'object recoginition viene eseguita con il comando:

```
!python3 detect.py --source /samples --out /faster_RCNN/results
                   --path_to_frozen_graph weights/frozen_inference_graph.pb
                   --path_to_labels dataset/labels.pbtxt
```

- **source**: indica la cartella contenente le immagini del dataset di test.
- **out**: indica la cartella sulle quali saranno salvate le immagini etichettate.
- **path_to_frozen_graph**: specifica il grafo d'inferenza da utilizzare.
- **path_to_labels**: specifica la label map da utilizzare.

```
2/50 /content/septoria/sample/466.jpg

2020-08-20 08:13:23.367031: W tensorflow/core/framework/cpu_allocator_impl.cc:81] Allocation of 240844800 exceeds 10% of system memory.

3/50 /content/septoria/sample/241.jpg

2020-08-20 08:13:38.830414: W tensorflow/core/framework/cpu_allocator_impl.cc:81] Allocation of 240844800 exceeds 10% of system memory.

4/50 /content/septoria/sample/0a73f8a0f_6.jpg

2020-08-20 08:13:53.808259: W tensorflow/core/framework/cpu_allocator_impl.cc:81] Allocation of 240844800 exceeds 10% of system memory.

5/50 /content/septoria/sample/0a97d54ae_2.jpg

2020-08-20 08:14:08.863897: W tensorflow/core/framework/cpu_allocator_impl.cc:81] Allocation of 240844800 exceeds 10% of system memory.

6/50 /content/septoria/sample/85.jpeg


7/50 /content/septoria/sample/0a73f8a0f_5.jpg


8/50 /content/septoria/sample/0a73f8a0f_15.jpg
```

Figura 4.15 – Esempio di detect in Fast RCNN

Anche in questo caso è possibile visualizzare i risultati con lo script mostrato precedentemente.

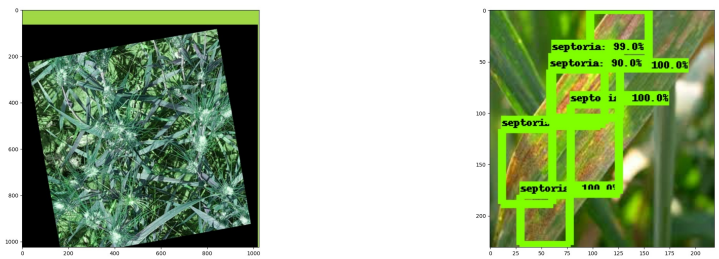

Figura 4.16 – Detect Faster RCNN su pianta sana e malata

4.4 Validation

Durante la fase di validation, i due modelli addestrati vengono valutati effettuando l'inferenza sul dataset di validation il quale presenta le sue etichette. Una volta effettuata l'object detection le etichette predette sono confrontate con quelle reali del dataset in modo da calcolare l'IoU e classificare quella determinata prediction in uno dei 3 casi (TP, FP, FN). Una volta classificate le prediction viene calcolato l'mAP, la precision/recall (generandone anche il grafico per ognuno dei modelli) ed un model score. L'intera fase di validation è stata eseguita attraverso un script:

- La funzione **intersect_over_union** calcola l'IoU dei bounding box passati per parametro. La funzione richiede come argomenti le coordinate e le dimensioni dei bounding box reali e di quelli predetti nella fase di inferenza e ne calcola prima l'intersezione, poi successivamente l'unione e ne fa il rapporto. Se i bounding box non si intersecano viene restituito 0.

```
def intersect_over_union(bbox,pred_bbox):
  left_top_corner=[0,0]
  left_down_corner=[0,0]
  right_top_corner=[0,0]
  d_xcenter=abs(bbox["x_center"]-pred_bbox["x_center"])
  d_ycenter=abs(bbox["y_center"]-pred_bbox["y_center"])
  if d_xcenter>(bbox["width"]/2+pred_bbox["width"]/2) or d_ycenter>(bbox
    ["height"]/2+pred_bbox["height"]/2):
    iou=0
  else:
    if bbox["x_center"]>pred_bbox["x_center"]:
        left_top_corner[0]=bbox["x_center"]-bbox["width"]/2
        left_down_corner[0]=bbox["x_center"]-bbox["width"]/2
        right_top_corner[0]=pred_bbox["x_center"]+pred_bbox["width"]/2
    else:
        left_top_corner[0]=pred_bbox["x_center"]-pred_bbox["width"]/2
        left_down_corner[0]=pred_bbox["x_center"]-pred_bbox["width"]/2
        right_top_corner[0]=bbox["x_center"]+bbox["width"]/2
    if bbox["y_center"]>pred_bbox["y_center"]:
        left_top_corner[1]=bbox["y_center"]-bbox["height"]/2
        left_down_corner[1]=pred_bbox["y_center"]+pred_bbox["height"]/2
        right_top_corner[1]=bbox["y_center"]-bbox["height"]/2
    else:
        left_top_corner[1]=pred_bbox["y_center"]-pred_bbox["height"]/2
        left_down_corner[1]=bbox["y_center"]+bbox["height"]/2
        right_top_corner[1]=pred_bbox["y_center"]-pred_bbox["height"]/2
    intersection_height=left_down_corner[1]-left_top_corner[1]
    intersection_width=right_top_corner[0]-left_top_corner[0]
    intersection_area=intersection_height*intersection_width
    bbox_area=bbox["height"]*bbox["width"]
    pred_bbox_area=pred_bbox["height"]*pred_bbox["width"]
    union_area=bbox_area+pred_bbox_area-intersection_area
    iou=intersection_area/union_area
  return iou
```

- La funzione **classification** confronta i bounding box predetti con quelli reali rilevando i veri positivi, falsi negati ed i falsi positivi.

```
def classification(bbox,pred_bbox):
    def get_iou(elem):
        return elem.get("iou")
    index=[]
    bbox_match=[]
```

```
    pred_bbox_match=[]
    soglia=0.5
    for i in pred_bbox:
        for j in bbox:
            iou=intersect_over_union(j,i)
            if iou>soglia:
                index.append({"bbox":bbox.index(j),"pred_bbox":pred_bbox
    .index(i),"iou":iou})
    index.sort(key=get_iou,reverse=True)
    for elem in index:
        if(elem["bbox"] not in bbox_match and elem["pred_bbox"] not in
    pred_bbox_match):
            bbox_match.append(elem["bbox"])
            pred_bbox_match.append(elem["pred_bbox"])
    return {"tp":len(bbox_match),"fp":len(pred_bbox)-len(pred_bbox_match
    ),"fn":len(bbox)-len(bbox_match)}
```

- La funzione **precision_recall** prende in input il dizionario contenente i veri positivi, i falsi negativi ed i falsi positivi e ne calcola i valori di precision e recall.

```
def precision_recall(results):
    tp=0
    fp=0
    fn=0
    for elem in results.values():
        tp+=elem["tp"]
        fp+=elem["fp"]
        fn+=elem["fn"]
    if(tp==0):
        precision=0
        recall=0
    else:
        precision=tp/(tp+fp)
        recall=tp/(tp+fn)
    return (precision,recall)
```

- La funzione **calculate_mAP** calcola il mean average precision tra i dati relativi ai bounding box reali e quelli predetti. L'mAP è calcolato con l'interpolazione di 11 punti, e restituisce sotto forma di dizionario le liste relative alla precision, la recall ed ai valori di soglia.

```
def calculate_mAP(validation_data,detection_data):
    def get_confidence(elem):
        return elem.get("confidence")
    def get_index(data,nome_immagine):
        i=0
        while(data[i]["nome_immagine"]!=nome_immagine):
            i+=1
        return i
    data=detection_data[:]
    model_scores=get_model_scores(data)
    temp={}
    model_sorted=sorted(model_scores.keys())[:-1]
    first_elem=model_sorted[0]
    for k in model_sorted:
        temp[k]=model_scores[k]
    model_scores=temp
    for image in data:
        image["bbox"].sort(key=get_confidence)
    precisions=[]
    recalls=[]
    model_scores_th=[]
```

```
    results={}
    images=[]
    for image in data:
        images.append(image["nome_immagine"])
    for soglia in model_scores:
        if model_scores[first_elem]!=soglia:
            images=model_scores[soglia]
        for image in images:
            i=get_index(data,image)
            if len(data[i]["bbox"])>0:
                j=0
                while(j<len(data[i]["bbox"]) and data[i]["bbox"][j]["
    confidence"]<soglia):
                    j+=1
                data[i]["bbox"]=data[i]["bbox"][j:]
            results[image]=classification(validation_data[i]["bbox"],
    data[i]["bbox"])
        (precision,recall)=precision_recall(results)
        precisions.append(precision)
        recalls.append(recall)
        model_scores_th.append(soglia)
    precisions=numpy.array(precisions)
    recalls=numpy.array(recalls)
    PR=[]
    for recall_level in numpy.linspace(0,1,11):
        try:
            args=numpy.argwhere(recalls>recall_level).flatten()
            precision=max(precisions[args])
        except ValueError:
            precision=0.0
        PR.append(precision)
    mAP=numpy.mean(PR)
    return {'mAP': mAP,'precisions': precisions,'recalls': recalls,'
    model_thrs': model_scores_th}
```

- La funzione **salva** ci permette di visualizzare i risultati relativi al confronto dei bounding box predetti con quelli reali. In particolare salva le immagini del dataset disegnandone sia i bounding box reali (in nero) che quelli predetti (in rosso) utilizzando la funzione **detect**. Inoltre il file salva in un file di testo contenente le statistiche del confronto e un immagine contenente il grafico della precision/recall.

```
def salva(validation_data,data,path,nome_rete):
    def get_name(elem):
        return elem.get("nome_immagine")
    data.sort(key=get_name)
    if not os.path.exists(path+"/output"):
        os.makedirs(path+"/output")
    if os.path.exists(path+"/output/"+nome_rete):
        shutil.rmtree(path+"/output/"+nome_rete)
    os.makedirs(path+"/output/"+nome_rete)
    if len(validation_data)==len(data):
        detect(validation_data,data,path,nome_rete)
        results=calculate_mAP(validation_data,data)
        figure=matplotlib.pyplot.figure()
        matplotlib.pyplot.plot(results["recalls"],results["precisions"])
        matplotlib.pyplot.title("Precision at recall graph "+nome_rete)
        matplotlib.pyplot.xlabel("Recall")
        matplotlib.pyplot.ylabel("Precision")
        matplotlib.pyplot.show()
        figure.savefig(path+"/output/"+nome_rete+"_stats.jpg")
        matplotlib.pyplot.close()
        s=("\nnome rete: "+nome_rete)
        s+=("\nmAP: "+str(results["mAP"])+"\n\n")
```

```
23          for elem in zip(results["model_thrs"],results["precisions"],
    results["recalls"]):
24              s+=("\nmodel_score: "+str(elem[0]))
25              s+=("\nprecision: "+str(elem[1]))
26              s+=("\nrecall: "+str(elem[2]))
27              s+=("\n\n\n")
28          file=open(path+"/output/results.txt","w")
29          file.write(s)
30          file.close()
31          print(s)
32
33
34  def detect(validation_data,data,path,nome_cartella):
35      for image in zip(validation_data,data):
36          nome_file=image[0]["nome_immagine"]
37          im=Image.open(path+"/dataset/images/"+nome_file)
38          size=im.size
39          nome_file=nome_file.split(".")[0]
40          labels=ylb.denormalize_bbox(image[0]["bbox"],size)
41          bbox.draw_bbox(labels,im,ylb,0)
42          labels=ylb.denormalize_bbox(image[1]["bbox"],size)
43          bbox.draw_bbox(labels,im,ylb,"red")
44          im.save(path+"/output/"+nome_cartella+"/"+nome_file+"_"+
    nome_cartella+".jpg","JPEG")
45          im.close()
```

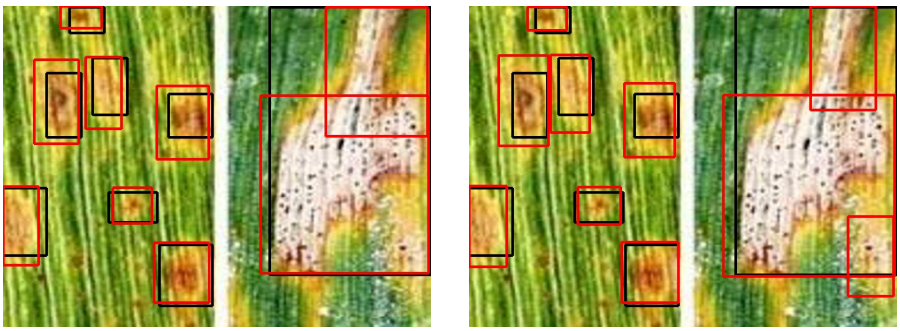

Figura 4.17 – Immagine contente le etichette previste e reali per Yolo e Fast RCNN

Capitolo 5

Conclusioni e Sviluppi Futuri

5.1 Conclusioni

Tramite l'utilizzo di sistemi di etichettatura come Labelbox, l'uso di reti neurali convoluzionali pre-addestrate, lo sviluppo dei due modelli di rete è stato completato con successo. La realizzazione del software tuttavia presenta alcune limitazioni:

- La scarsa disposizione dei campioni di addestramento generano **overfitting**, ovvero i modelli addestrati sono in grado di riconoscere molto bene i dati di addestramento, ma non i nuovi dati mai visti. In questo modo i modelli di rete non sono in grado di generalizzare perfettamente la classe che si vuole riconoscere. Tale problematica è stata limitata tramite l'uso di tecniche di *data augmentation* le quali sono in grado di aumentare il numero di campioni disponibili attraverso l'uso di trasformazioni casuali. Ciò aiuta a esporre il modello ad un maggior numero di aspetti dei dati, a vantaggio della sua capacità di generalizzazione. Tuttavia gli input che la rete utilizzerà durante la fase di training, saranno fortemente correlati in quanto provengono comunque da un numero relativamente piccolo di immagini originali. Per questo motivo tale tecnica potrebbe non essere del tutto efficace contro l'overfitting.

- La fase di "validation" effettuata permette di valutare le prestazione dei due modelli addestrati. Tuttavia tale fase utilizza un semplice approccio **Holdout** (figura 5.1) il quale consiste di mettere da parte alcuni dati etichettati per la valutazione del modello. Tale protocollo di validazione è il più semplice ed il più utilizzato, ma soffre di un difetto: se si hanno a disposizione pochi dati, i set di convalida e di test potrebbero contenere troppi pochi campioni per essere statisticamente rappresentativi dei dati disponibili. Tale problematica può essere individuata se mescolamenti casuali differenti dei dati etichettati utilizzati per la convalida forniscono misure molto differenti delle prestazioni del modello.

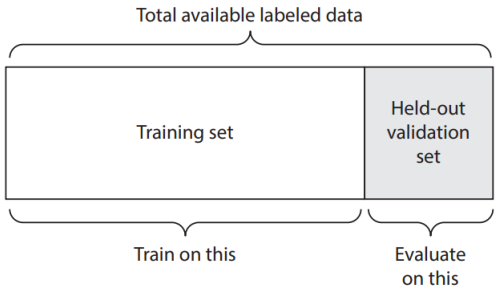

Figura 5.1 – Semplice suddivisione per la convalida hold-out [21]

Per evitare di perdere campioni significativi nel partizionamento dei dati nel dataset di training e validation si può ricorrere a tecniche di **cross validation**. Tale tecnica consiste nel ripetere l'addestramento diverse volte con diverse configurazioni del training set e considerare la prestazione media dei modelli addestrati. Esistono diverse tipologie di cross validation:

1. **Convalida K Fold**: In tale approccio i dati etichettati sono divisi in K partizioni di pari dimensioni (figura 5.2). Per ogni partizione *i-esima* viene addestrato un modello di rete sulle rimanenti K-1 partizioni e poi successivamente valutato sulla partizione *i*. L'addestramento viene ripetuto per ogni diversa partizione i ognuna delle quali genererà un diverso score. Il punteggio finale del modello sarà dato dalla media dei K punteggi ottenuti.

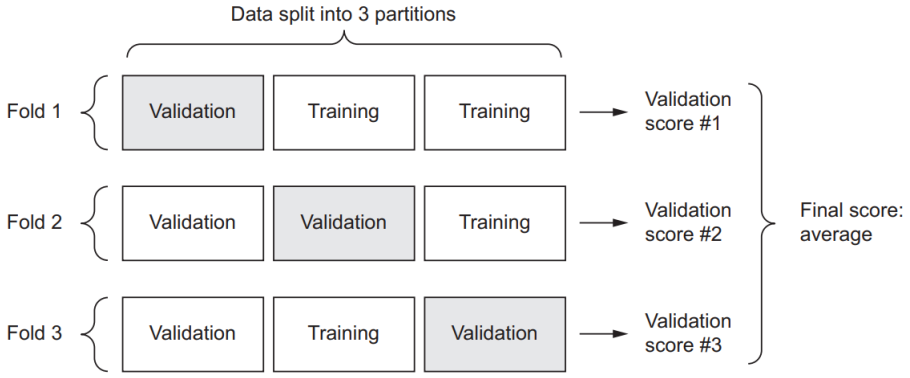

Figura 5.2 – Convalida k-fold con k=3 [21]

2. **Convalida K Fold iterata con shuffling**: Questo tipo di convalida è ottima nel caso in cui si abbiano a disposizione relativamente pochi dati e si debba valutare il modello il più precisamente possibile. Tale tecnica consiste nell'applicare più volte la convalida K-fold mescolando ogni volta i dati prima di dividerli nelle K partizioni. Il punteggio finale del modello dunque sarà dato dalla media dei punteggi ottenuti per ogni convalida K-fold.

5.2 Sviluppi Futuri

In futuro sarà possibile sfruttare ulteriori modelli di rete o utilizzare altre tecniche di addestramento. Nel paragrafo 1.1 si è descritto come sia possibile realizzare software i quali, grazie all'uso di droni siano in grado di analizzare le colture in tempo reale. In particolare la ItalDron ha sviluppato un drone chiamato **4HSE Nebula** (figura 5.3) il quale permette di diffondere nelle colture prodotti disinfettanti e disinfestanti. Grazie ad un serbatoio modulare integrato in una soluzione SAPR (Sistema Aeromobile a Pilotaggio Remoto) compatta e facilmente trasportabile è possibile creare missioni di volo autonomo e garantire il rilascio omogeneo e puntuale del prodotto per varie finalità di utilizzo [44]. Una possibilità, dunque, è quella di sfruttare la rete Yolo che grazie alla sua struttura, la sua velocità e tramite l'uso delle librerie **OpenCV** permette il riconoscimento di oggetti tramite videocamera. Il software sviluppato potrà essere successivamente integrato con il drone in modo tale da creare un sistema in grado di riconoscere preventivamente la presenza di septoriosi ed eventualmente rilasciare agenti disinfestanti. Inoltre, lo stesso software realizzato può essere modificato in modo tale da realizzare diversi modelli, anche multi-classe in grado di riconoscere e classificare diverse malattie del grano oltre alla septoriosi.

Figura 5.3 – Drone 4HSE Nebula [44]

Appendice A

Operazioni di Data Augmentation

Di seguito sono definite le principali operazioni di data augmentation definite nello script **operations.py** utilizzato nella tesi.

```
from abc import abstractmethod, ABCMeta
import random
from scipy import ndimage, ndarray
from skimage.transform import rotate, resize
from skimage.util import random_noise

...

class Rotate(Operation):
    max_left_degree = None
    max_right_degree = None
    def __init__(self, probability: float, max_left_degree: int,
    max_right_degree: int) -> None:
        super().__init__(probability)
        self.max_left_degree = max_left_degree
        self.max_right_degree = max_right_degree
    def execute(self, image_array: ndarray):
        random_degree = random.uniform(-self.max_right_degree, self.
    max_left_degree)
        return rotate(image_array, random_degree)

class RandomNoise(Operation):
    def __init__(self, probability: float) -> None:
        super().__init__(probability)
    def execute(self, image_array: ndarray):
        return random_noise(image_array)

class Blur(Operation):
    width = None
    height = None
    def __init__(self, probability: float) -> None:
        super().__init__(probability)
    def execute(self, image_array: ndarray):
        return ndimage.uniform_filter(image_array, size=(11, 11, 1))

class Resize(Operation):
    def __init__(self, probability: float, width: int, height: int) -> None:
        super().__init__(probability)
        self.width = width
        self.height = height
    def execute(self, image_array: ndarray):
        return resize(image_array, (self.width, self.height))
```

```
41
42 class HorizontalFlip(Operation):
43     def __init__(self, probability: float) -> None:
44         super().__init__(probability)
45     def execute(self, image_array: ndarray):
46         return image_array[:, ::-1]
47
48 class VerticalFlip(Operation):
49     def __init__(self, probability: float) -> None:
50         super().__init__(probability)
51     def execute(self, image_array: ndarray):
52         return image_array[::-1, :]
53
54
55 class OperationPipeline:
56    operations = []
57    def blur(self, probability: float):
58         self.__add_operation(Blur(probability))
59    def rotate(self, probability: float, max_left_degree: int, max_right_degree
      : int):
60         self.__add_operation(Rotate(probability, max_left_degree,
      max_right_degree))
61    def random_noise(self, probability: float):
62         self.__add_operation(RandomNoise(probability))
63    def resize(self, probability: float, width: int, height: int):
64         self.__add_operation(Resize(probability, width, height))
65    def horizontal_flip(self, probability: float):
66         self.__add_operation(HorizontalFlip(probability))
67    def vertical_flip(self, probability: float):
68         self.__add_operation(VerticalFlip(probability))
69    def __add_operation(self, operation: Operation):
70         self.operations.append(operation)
```

Bibliografia

[1] Adama Italia. La septoria del grano. https://ita.approfondimenti.adama.com/septoria-del-grano.

[2] Claudio Cristiani Gianpiero Alvisi. Complesso della septoriosi malattia in espansione su grano. *L'Informatore Agrario*, 11, 2008.

[3] Septoriosi,ruggini e fusariosi: le malattie più dannose per i cereali autunno-vernini. https://www.syngenta.it/septoriosi-ruggini-e-fusariosi-le-malattie-piu-dannose-i-cereali-autunno-vernini, 2020.

[4] Wikipedia contributors. Agricoltura di precisione. https://it.wikipedia.org/wiki/Agricolturadiprecisione, 2020.

[5] Agricoltura 4.0: Robot ed intelligenza artificiale per le aziende agricole del futuro. https://www.feger.it/blog/agricoltura-40-robot-ed-intelligenza-artificiale-per-le-aziende-agricole-delfuturo_81.xhtml, 2020.

[6] Joseph Redmon and Ali Farhadi. Yolov3: An incremental improvement. *arXiv*, 2018.

[7] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster r-cnn: Towards real-time object detection with region proposal networks, 2016.

[8] Wikipedia contributors. Python. https://it.wikipedia.org/wiki/Python, 2020.

[9] Python Community. La crescita di python. https://www.python.it/blog, 2017.

[10] Google Brain Team. Tensorflow. https://www.tensorflow.org/, 2020.

[11] Benjamin Planche & Eliot Adres. *Hands on Computer Vision with Tensorflow.* Packt, 2019.

[12] Sebastian Raschka & Vahid Mirjalili. *Python Machine Learning Third edition.* Packt, 2019.

[13] Ravi Ranjan Singh. Tensorflow tutorial : A beginner's guide to tensorflow. TensorFlowTutorial:ABeginner\T1\textquoterightsGuidetoTensorFlow, 2020.

[14] Google Brain Team. Tensorboard. https://www.tensorflow.org/tensorboard, 2020.

[15] Ecma Internation. The json data interchange syntax. *Standard ECMA-404*, 2017.

[16] Wikipedia contributors. Xml. https://it.wikipedia.org/wiki/XML, 2020.

[17] Labelbox. https://labelbox.com, 2020.

[18] Labelbox docs. https://labelbox.com/docs/tools, 2020.

[19] Google colab per il machine learning: cos'è e come si usa. https://www.html.it/articoli/google-colab-per-il-machine-learning-cose-e-come-si-usa/, 2020.

[20] Project jupyter. https://jupyter.org/.

[21] Francois Chollet. *Deep Learning with Python*. Manning, 2018.

[22] Jacopo Kahl. I tre principali tipi di machine learning: Apprendimento supervisionato, non supervisionato e per rinforzo. https://medium.com/@jacopokahl/i-tre-principali-tipi-di-machine-learning-77ca20d0dbdd, 2020.

[23] Aaron Courville Ian Goodfellow, Yoshua Bengio. *Deep Learning*. MIT Press, 2017.

[24] Andrea Missinato. Machine learning | reti neurali demistificate. https://www.spindox.it/it/blog/ml1-reti-neurali-demistificate/, 2018.

[25] Tony Yiu. Understanding neural networks. https://towardsdatascience.com/understanding-neural-networks, 2019.

[26] Sumit Saha. A comprehensive guide to convolutional neural networks. https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53, 2018.

[27] Stanford CS class CS231n. Convolutional neural network for visual recognition. https://cs231n.github.io/convolutional-networks/, 2017.

[28] Wikipedia Contributors. Rectifer. https://en.wikipedia.org/wiki/Rectifier_(neural_networks).

[29] Teemu Kanstrén. A look at precision, recall, and f1-score. https://towardsdatascience.com/a-look-at-precision-recall-and-f1-score-36b5fd0dd3ec, 2020.

[30] Koo Ping Shung. Accuracy, precision, recall or f1? https://towardsdatascience.com/accuracy-precision-recall-or-f1-331fb37c5cb9, 2018.

[31] Jonathan Hui. map (mean average precision) for object detection. https://jonathan-hui.medium.com/map-mean-average-precision-for-object-detection-45c121a31173, 2018.

[32] Hamid Rezatofighi, Nathan Tsoi, JunYoung Gwak, Amir Sadeghian, Ian Reid, and Silvio Savarese. Generalized intersection over union. *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2019.

[33] Rokas Balsys. Yolo v3 theory explained. https://pylessons.medium.com/yolo-v3-theory-explained-33100f6d193, 2019.

[34] Ross Girshick, Jeff Donahue, Trevor Darrell, and Jitendra Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2014.

[35] Saurabh Bagalkar. Deep learning for object detection and localization using r-cnn. https://medium.com/alegion/deep-learning-for-object-detection-and-localization-using-r-cnn-e88f85ea7c16, 2019.

[36] K. E. A. van de Sande, J. R. R. Uijlings, T. Gevers, and A. W. M. Smeulders. Segmentation as selective search for object recognition. *IEEE International Conference on Computer Vision*, 2011.

[37] R. Girshick. Fast r-cnn. *2015 IEEE International Conference on Computer Vision (ICCV)*, 2015.

[38] Rohith Gandhi. R-cnn,fast r-cnn,faster r-cnn, yolo object detection algorithms. https://towardsdatascience.com/r-cnn-fast-r-cnn-faster-r-cnn-yolo-object-detection-algorithms-36d53571365e, 2018.

[39] Marcus D. Bloice, Christof Stocker, and Andreas Holzinger. Augmentor: An image augmentation library for machine learning, 2017.

[40] Image dataset generator for deep learning projects. https://github.com/tomahim/py-image-dataset-generator#create-a-custom-image-augmentation-pipeline.

[41] Labelbox Developers. Label export formats. https://labelbox.com/docs/exporting-data/export-format-detail.

[42] Yolo v3 repository. https://github.com/ultralytics/yolov3.

[43] Tensorflow models repository. https://github.com/tensorflow/models/.

[44] Evoluzione 4hse nebula, la soluzione per la diffusione di prodotti disinfettanti, disinfestanti, fitosanitari e molto altro. https://www.italdron.com/it/equipaggiamenti/evoluzione-4hse-nebula/.

Elenco delle figure