ALMA MATER STUDIORUM - UNIVERSITÀ DI BOLOGNA

SCUOLA DI INGEGNERIA

DIPARTIMENTO di
INGEGNERIA DELL'ENERGIA ELETTRICA E DELL'INFORMAZIONE
"Guglielmo Marconi"
DEI

CORSO DI LAUREA IN

INGEGNERIA ELETTRONICA E DELLE TELECOMUNICAZIONI

TESI DI LAUREA
in
CALCOLATORI ELETTRONICI

TECNICHE DI DEEP LEARNING PER IL MONITORAGGIO DEL PROCESSO NELL'INDUSTRIA DELLO PNEUMATICO

CANDIDATO

Kilian Tiziano Le Creruer

RELATORE

Chiar.mo Prof Luigi Di Stefano

CORRELATORI

Stefano Monti
Fabio Regoli

Anno Accademico
2021-2022

Sessione
I

INDICE

Introduzione…… 6

1 Computer Vision…….....8

1.1 Deep learning……8

1.2 Image classification……..8

1.3 Architetture di object detection……10

1.4 Faster RCNN……...11

1.5 Retina Net……....12

1.6 Architetture YOLO…….15

2 Struttura dati e training…… 16

2.1 Analisi delle immagini……...16

2.2 Composizione dei dataset……...……18

2.3 Preparazione del primo dataset……..18

2.4 Setup training……21

2.5 Modelli allenati……..22

2.6 Risultati……..23

3 Metriche di decisione……. 25

3.1 Struttura del problema……25

3.2 Raggruppamento dei risultati…….26

3.3 Algoritmo tradizionale……...27

3.4 Test sul secondo dataset……27

3.5 Analisi degli errori……..30

4 Modifiche alla logica e secondo *training* di Retina-Net……. 33

4.1 Annotazione del secondo dataset……33

4.2 Preparazione del dataset e training della rete……..34

4.3 Test sul secondo dataset……..35

4.4 Analisi dei *trend* su coperture non pinzate…….36

4.5 Test con controllo della posizione della copertura…….39

5 Conclusioni…… 41

LISTA DELLE TABELLE

Tabella 1.1 Risultati di alcuni benchmark di image classification…….10

Tabella 2.1 Varie parametrizzazioni dell'architettura Retina-Net……..23

Tabella 2.2 Risultati in termini di AP fra i vari modelli utilizzati……..24

Tabella 3.1 Risultati possibili in un decisore binario……..25

Tabella 3.2 Divisione del secondo dataset nelle varie macchine di produzione…….28

Tabella 3.3 Risultati test al variare del valore di soglia accettabile……29

Tabella 4.1 *Set* di parametri di *train* utilizzati…….34

Tabella 4.2 Confronto fra i due modelli sul terzo test set……35

Tabella 4.3 Dati statistici relativi alla *bounding box* della copertura……...37

LISTA DELLE FIGURE

Figura 1.1 Image classification…….9

Figura 1.2 Filtro di un convolutional layer……..10

Figura 1.3 Struttura di una rete RPN……...12

Figura 1.4 Focal loss al variare dei termini di regolarizzazione……..…13

Figura 1.5 Struttura complessiva di Retina-Net……..….14

Figura 1.6 Confronto fra vari detectors stato dell'arte……..…..14

Figura 1.7 Struttura dell'architettura Yolo-v3……..15

Figura 2.1 Primo elemento di una coppia di immagini ben posizionate……..16

Figura 2.2 Seconda immagine della coppia…….16

Figura 2.3 Schema della posizione degli elementi in una scena corretta…….17

Figura 2.4 Copertura non pinzata, scritte a malapena visibili……..18

Figura 2.5 Copertura non pinzata, scritte non visibili……..18

Figura 2.7 Copertura Upside-down, le scritte si trovano al di sotto del baricentro……..18

Figura 2.7 Copertura Upside-down, le scritte si trovano al di sotto del baricentro……..18

Figura 2.8 Struttura del dict *images*……..19

Figura 2.9 Struttura del dict *annotations*……..20

Figura 2.10 Struttura del dict *categories*……..20

Figura 2.11 Immagine annotata con COCO-Annotator……21

Figura 2.12 Risultati del modello su un'immagine upside-down…….23

Figura 2.13 Risultati del modello su un'immagine correttamente posizionata…….23

Figura 3.1 Albero decisionale a partire dagli output della rete…….27

Figura 3.2 Esempio di immagini appartenenti ai *dataset "strange"*……29

Figura 3.3 Errori dovuti al riconoscimento di macchie di gesso come scritte……..31

Figura 3.5 Errori dovuti al posizionamento erroneo della copertura……....31

Figura 4.1 Immagine contenente macchie di gesso ottenuta dal secondo dataset……..……..33

Figura 4.2 Primo falso negativo risultato dal terzo test…….……...……36

Figura 4.3 Secondo falso negativo risultato dal terzo test……36

Figura 4.4 Istogramma della mediana della coordinata y della *bounding box*…….37

Figura 4.5 Esempio di copertura non pinzata con scritte visibili……..38

Figura 4.6 Logica di decisione con test modificato per immagini non pinzate……39

Figura 4.7 L'unico errore nel test che utilizza il controllo triplo sulle immagini non pinzate.40

INTRODUZIONE

L'avanzamento tecnologico odierno, soprattutto nel mondo dello sviluppo software e hardware, ha favorito la nascita di sistemi complessi, in grado di rispondere a sfide che sarebbero sembrate impossibili solo pochi anni prima. Una di queste è sicuramente la capacità dei calcolatori di allenarsi e di imparare. Il *machine learning* è infatti una delle branche della *computer science* più affermate ai giorni nostri, in quanto permette di analizzare ed interpretare strutture dati complesse, pressoché in tempo reale. Fra le molteplici applicazioni di questa tecnologia c'è la *computer vision,* la quale, sfrutta modelli di reti neurali profonde per analizzare ed interpretare immagini. Essa a sua volta si divide in varie applicazioni, fra le quali si trova l'*object detection,* che permette di classificare oggetti all'interno di immagini e determinarne le coordinate posizionali.

Il seguente elaborato affronta un problema di controllo di processo in ambito industriale utilizzando algoritmi di *object detection.* Infatti, il progetto concordato con il professore Di Stefano si è svolto in collaborazione con l'azienda Pirelli, nell'ambito della produzione di pneumatici. Lo scopo dell'algoritmo è di verificare il preciso orientamento di elementi grafici della copertura, utilizzati dalle case automobilistiche per equipaggiare correttamente le vetture. In particolare, si devono individuare delle scritte sul battistrada della copertura e identificarne la posizione rispetto ad altri elementi fissati su di essa.

Nel caso in cui queste scritte si presentino con un orientamento non corretto, porterebbero ad un errato montaggio della copertura da parte del cliente finale, con conseguente claim.

Nella prima fase del progetto l'azienda Pirelli ha fornito un dataset di oltre 1500 immagini che sono state usate per allenare alcune architetture di *object detection,* con l'obiettivo di valutare se fossero in grado di rilevare correttamente le coperture e le scritte che vi si trovavano sopra. Dopodiché, partendo dai risultati di questi primi test, si è implementata una logica che determinasse la presenza o meno di un errore, utilizzando i valori restituiti a valle del modello scelto. Questa logica è stata testata su un secondo dataset, di oltre 6000 immagini, fornito sempre da Pirelli. A valle di questa fase, si è allenato un modello utilizzando i due dataset completi, così da ottenere una migliore generalizzazione. Infine, sfruttando la rete

appena descritta si è eseguito un ulteriore test della logica su un terzo dataset, di circa 6000 immagini, sempre fornito dall'azienda.

Nel primo capitolo si fornisce una breve panoramica sulle architetture che sono state utilizzate nel corso del progetto, con cenni alla struttura delle reti neurali, nel secondo inizia l'approccio alla parte sperimentale affrontata durante il tirocinio, con particolare riferimento alla creazione del primo dataset attraverso un *tool* di *labeling* e ai primi esperimenti di *training* dei vari *object detector* di scritte e coperture, nel terzo viene descritta la logica utilizzata per il rilevamento degli errori e vengono mostrati i risultati dei *test* effettuati con essa sul secondo *dataset* e infine, nel quarto capitolo, vengono introdotte alcune modifiche alla logica, viene riallenata l'architettura di *detection* con l'obiettivo di minimizzare gli errori rilevati nelle precedenti analisi e vengono effettuati dei nuovi test su ulteriori distribuzioni di dati.

CAPITOLO I

Computer Vision

1.1 *Deep Learning*

Nel vasto mondo del *machine learning,* ci sono branche che hanno riscosso un notevole successo in un numero enorme di applicazioni, fra cui troviamo sicuramente le reti neurali, in particolare le reti neurali profonde, su cui si basa il *deep learning*. La grande capacità di generalizzazione degli algoritmi di apprendimento profondo [1] ha fatto sì che venissero usati in praticamente qualunque *task* [2].

Le *deep neural network* sono delle reti neurali costituite da un gran numero di livelli, ognuno dei quali esegue una computazione del suo input e attiva il successivo, fino all'*output layer,* dove vengono generati i risultati. Le modalità di apprendimento di queste reti sono molteplici, ma la più comune è il *supervised learning,* nel quale si utilizzano *labeled data*[1]. Ad ogni dato in ingresso viene associato a priori un *output*, detto *groundtruth label*, dopodichè gli *input* vengono inseriti nella rete, che genera un *output* in base ai valori dei propri parametri. A partire da questa coppia di valori, si calcola una funzione errore, che ha come obiettivo mettere in luce lo scostamento fra la *groundtruth label* e l'*output* del modello, infine il gradiente della funzione errore viene propagato indietro nella rete, attraverso un algoritmo detto *backpropagation* [2], che ha l'obiettivo di aggiornare i parametri della rete di modo che forniscano una migliore approssimazione dell'output desiderato. La procedura appena descritta viene iterata molte volte e se ha successo, il modello finale sarà in grado di fornire *output* corretti su *unseen data*[2].

1.2 Image classification

Fra tutti i campi applicativi delle *deep neural network*, si trova la *computer vision,* che sfrutta algoritmi di *deep learning* per estrarre informazioni significative da immagini digitali. Fra le sue principali applicazioni si trovano una serie di *task* quali *image segmentation, localization, object detection, image classification*. Fra questi, il problema cardine è l'*image*

[1] In computer vision per *labeled data*, si intendono tutti quei dati a cui viene associata un'etichetta che contiene informazioni aggiuntive sui dati considerati, così da permettere al modello di machine learning di imparare da esso.

[2] Per *unseen data* si intendono dati provenienti dalla stessa distribuzione, che non sono stati utilizzati dal modello per la procedura di training.

classification, nel quale un modello deve imparare a riconoscere la categoria di appartenenza di un'immagine rispetto a tutte quelle possibili, a partire dalla sua struttura digitale [2].

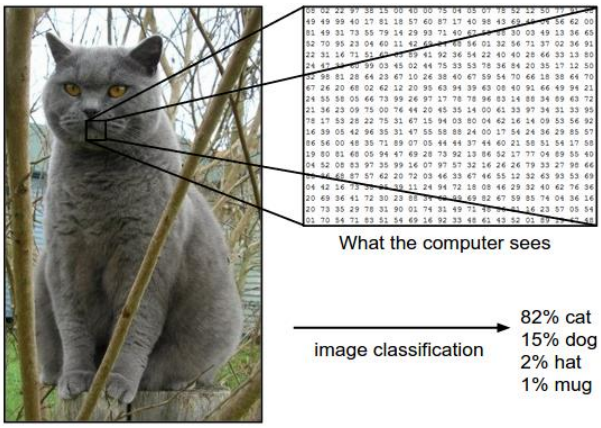


Figura 1.1 Image classification

L'*image classification* ha raggiunto risultati significativi grazie all'utilizzo di CNN, *convolutional neural networks*, che sono particolari reti neurali in cui si trovano diversi livelli con funzioni specifiche, i *fully connected layers*, i *convolutional layers* e i *pooling layers.*

I *fully connected layers* eseguono un'operazione lineare fra input e pesi[3] e generano una serie di output, alla quale viene poi applicata una *activation function.* Questo tipo di *layer* utilizza tutti gli input per la computazione di ogni valore di uscita.

$$y_{jk}(x) = f\left(\sum_{i=1}^{n_H} x_i w_{jk} + b_j\right)$$

I *convolutional layers* sono la spina dorsale di una CNN, nascono con l'obiettivo di determinare features da piccole parti di immagini. Perciò, la determinazione di un output avviene attraverso l'uso di uno o più filtri, di dimensioni molto inferiori rispetto all'input che processano, in questo modo solo una piccola parte di dati, vicini fra loro, contribuisce al risultato. I filtri vengono poi traslati lungo tutta la matrice di *input.*

[3] I pesi, o *weights* rappresentano i parametri principali di una rete neurale.

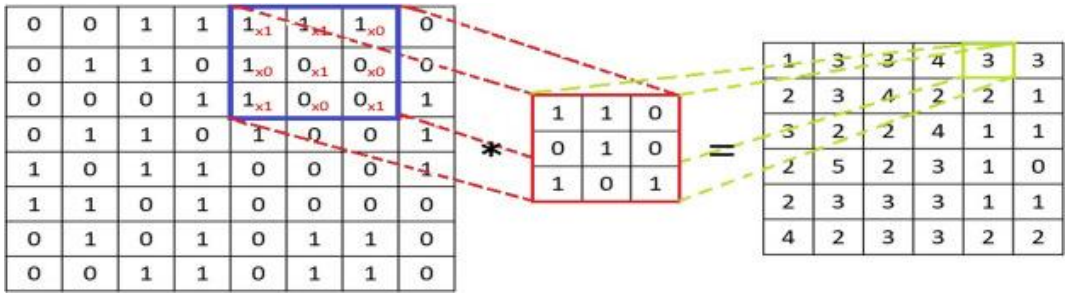


Figura 1.2 Filtro di un convolutional layer.

I *pooling layers* vengono utilizzati per effettuare un *downsampling* dell'input, riducendone le dimensioni. Esistono diverse tipologie di *pooling*, *max pooling*, *mean pooling e average pooling* [2]. Questo tipo di livelli è reso necessario per ridurre lo sforzo computazionale, che cresce esponenzialmente con l'aumentare della profondità delle reti.

Le architetture di *image classification* sono in continua evoluzione e hanno raggiunto risultati perfino superiori all'uomo in alcuni *benchmark* [3].

Model	Humans	WRNC	RNC	RN	NiN	LeNet
Acc (%)	93.91 ±1.52	96.96	96.18	95.33	89.28	80.86

Tabella 1.1 Risultati di alcuni benchmark di image classification.

1.3 Architetture di object detection

I problemi di *object detection* consistono in un'evoluzione dell'*image classificiation,* infatti in questo caso non si vuole più classificare l'immagine di *input*, ma l'obiettivo diventa quello di riconoscere molteplici caratteristiche all'interno di una singola immagine e inquadrarne la posizione attraverso una *bounding box*[4]. Ogni immagine può quindi contenere un numero qualunque di *features.*

Una delle prime soluzioni a questo problema, introdotta in [5], è RCNN, che utilizza 3 moduli distinti, il primo, detto *region proposal network*, serve a ricavare alcune migliaia di

[4] Una *bounding box* è un riquadro dell'immagine in cui è contenuta una specifica *feature.*

crop[5] dell'immagine indipendentemente dalla categoria, il secondo non è altro che una estesa CNN che crea una ricca *feature map* [6], che infine viene processata da una *support vector machine*, un algoritmo di *machine learning* molto usato per regressione e classificazione dei dati. La rete ritorna quindi una classificazione di tutte le regioni e applica dei coefficienti di regressione per la posizione delle *bounding box.*

Questo approccio risulta però estremamente lento, infatti migliaia di *proposals* devono essere processate dalla CNN, di conseguenza sono state proposte alcune alternative più efficienti, fra cui Fast-Rcnn [6], dove si esegue un *sub-sampling* dell'input prima della CNN e SPP-Net [7], dove si cerca di selezionare un numero inferiore, ma più preciso di *proposals* alla CNN, riducendo lo sforzo computazionale.

1.4 Faster-RCNN

I modelli precedenti, benché ottimizzati, sono comunque caratterizzati da problemi legati alle troppo lunghe computazioni. Faster-Rcnn [8] è il primo *object detector* a lavorare quasi in tempo reale, con *frame rate* fino a 17 fps [8].

Questo modello è diviso in due moduli, una RPN, *region proposal networks,* che seleziona i *crop* dell'immagine dove c'è una *feature* da rilevare e Fast-RCNN [6], che a partire dalle regioni proposte esegue la classificazione e la *detection,* come si vede in figura 1.3.

La prima parte dell'architettura, detta *backbone* è costituita da una profonda CNN [8] [9] che genera una grande *feature map* multiscalata. A questo punto viene applicata la RPN, che non è altro che una piccola rete convoluzionale che serve a determinare se un *crop* dell'immagine contiene una caratteristica rilevante. Questo viene fatto generando nove ancore in ogni pixel, che poi vengono classificate ed eventualmente riposizionate a valle della rete dal classificatore lineare, come è mostrato in figura 1.3.

[5] Per *crop* si intende una sezione o ritaglio dell'immagine.
[6] Una *feature map* è il risultato di una rete convoluzionale, contiene infatti caratteristiche significative di ogni parte dell'immagine.

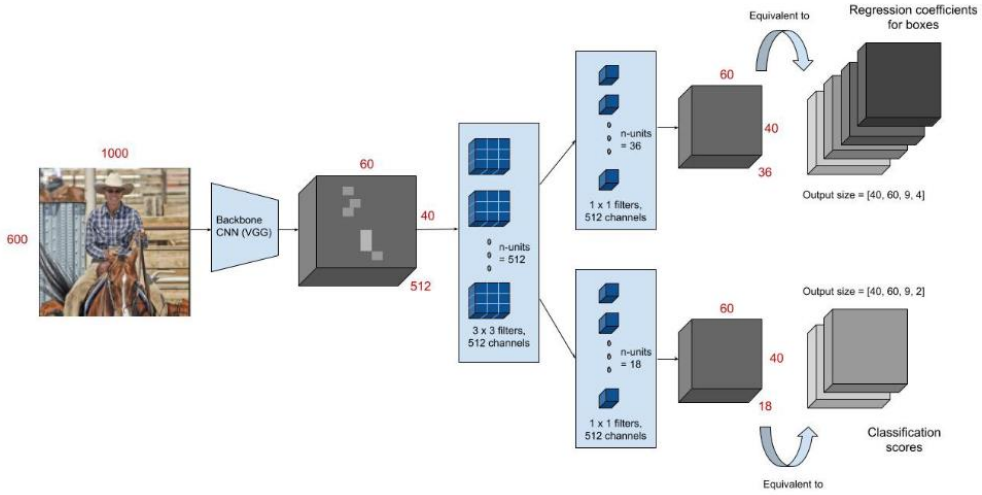


Figura 1.3 Struttura della rete RPN.

La RPN è allenata assegnando un valore binario ad ogni ancora che indica se una *feature* è presente o meno; ovvero ogni regione proposta dalla rete viene confrontata con le *ground truth* e viene associato un valore positivo solo a quella con la maggiore IoU[7], mentre si associa un valore negativo a tutte le regioni che risultino avere un IoU minore di 0.3. Inoltre, nel calcolo dell'errore si considera anche il contributo di regressione delle *bounding box.* Si utilizza infatti la *cross entropy loss.*

$$L(\{p_i\},\{t_i\}) = \frac{1}{N_{cls}}\sum_i L_{cls}(p_i, p_i^*) + \lambda\frac{1}{N_{reg}}\sum_i p_i^*\, L_{reg}(t_i, t_i^*)$$

Nonostante gli ottimi risultati sia in termini di *benchmark* che di tempistiche la struttura adottata da Faster-RCNN resta un *detector* a due stadi, che non può raggiungere inferenze veramente *real* time.

1.5 Retina Net

I primi successi dell'*object detection* [5,6,7,8] sono tutti basati su un approccio multistadio, in cui prima si ricavano le regioni da analizzare e poi si determinano le classi e le *bounding boxes.* Uno dei primi *detector* che ha raggiunto risultati paragonabili se non migliori dei multistadio è Retina-Net [11]. Il cui principio di funzionamento è strettamente correlato alla funzione errore che viene implementata per allenare la rete.

Il problema sostanziale dei detector che non utilizzano una rete di *region proposal* è che le sezioni di immagine da analizzare sono moltissime, si aggirano fra diecimila e centomila,

[7] L'IoU è una metrica molto utilizzata nell'*object detection,* che consiste nel calcolare il rapporto fra l'intersezione e l'unione fra l'area del risultato della rete è quella della ground truth.

mentre le vere *features* da rilevare sono generalmente poche all'interno di un'immagine, per cui c'è un forte sbilanciamento fra lo sfondo dell'immagine e le parti significative. Questo si traduce in due problemi sostanziali, il primo è che il *training* delle reti risulta inefficiente, in quanto il 99.99% delle regioni valutate non contribuiscono con informazioni significative, il secondo è che questo forte sbilanciamento porta facilmente i modelli a degenerare.

La soluzione offerta in [11] consiste nel creare una funzione errore che renda vicino a zero il contributo dei *crop* di sfondo dell'immagine, aumentando invece quello delle poche sezioni che contengono un oggetto. Il punto di partenza utilizzato è la *cross entropy loss function* [12].

$$CE(p, y) = -\log(pt)$$

$$p_t = \begin{cases} p & se\ y = 1 \\ 1 - p & altrove \end{cases}$$

A partire da questa definizione si costruisce la *focal loss function*, che è definita a partire da questa struttura con l'inserimento di alcuni termini di bilanciamento.

$$FL(p_t) = \alpha_t\,(1 - p_t)^{\gamma} \log(p_t)$$

In questa formulazione α_t e γ sono due termini di regolarizzazione. Infatti, garantiscono che le *proposal* appartenenti allo sfondo (non contenenti oggetti e dunque facilmente classificabili) diano pochissimo contributo all'errore, garantendo un *training* efficiente. Si veda figura 1.4.

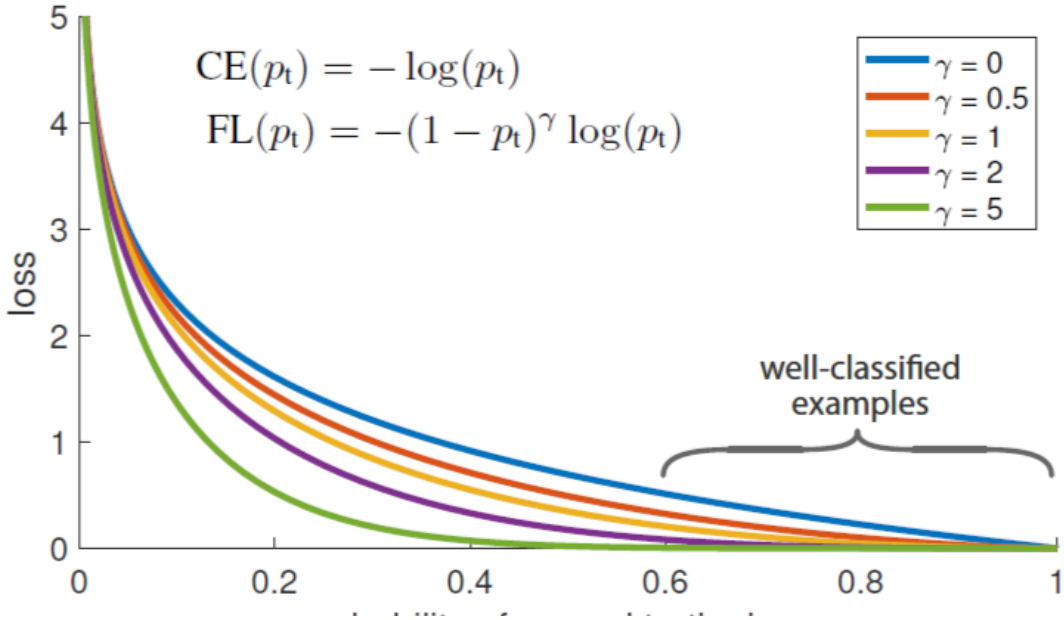


Figura 1.4 Focal loss al variare dei termini di regolarizzazione.

La struttura di Retina-Net è costituita da una prima CNN, ResNet, a cui viene applicata una FPN [13], che consiste in un'altra rete convoluzionale, ma con delle connessioni laterali che permettono di generare una *feature map* multi-scalata, seguendo quindi una struttura piramidale. Questa viene connessa a due reti ulteriori, che ritornano le classi e i coefficienti di regressione delle *bounding box.* La struttura complessiva della rete si può vedere in figura 1.5.

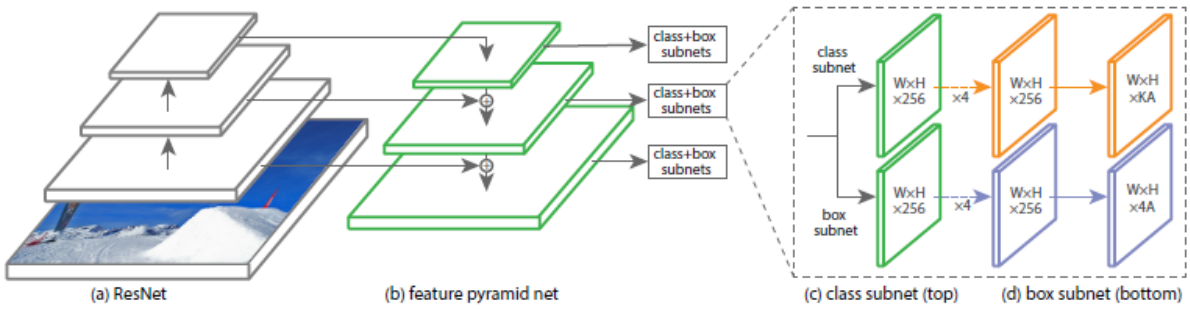


Figura 1.5 Struttura complessiva di retina-Net

Retina Net è il primo *object detector* ad aver raggiunto risultati in termini di inferenza davvero *real-time,* con tempi su GPU sotto i 200ms garantendo delle prestazioni al livello di tutti gli altri *detectors* stato dell'arte.

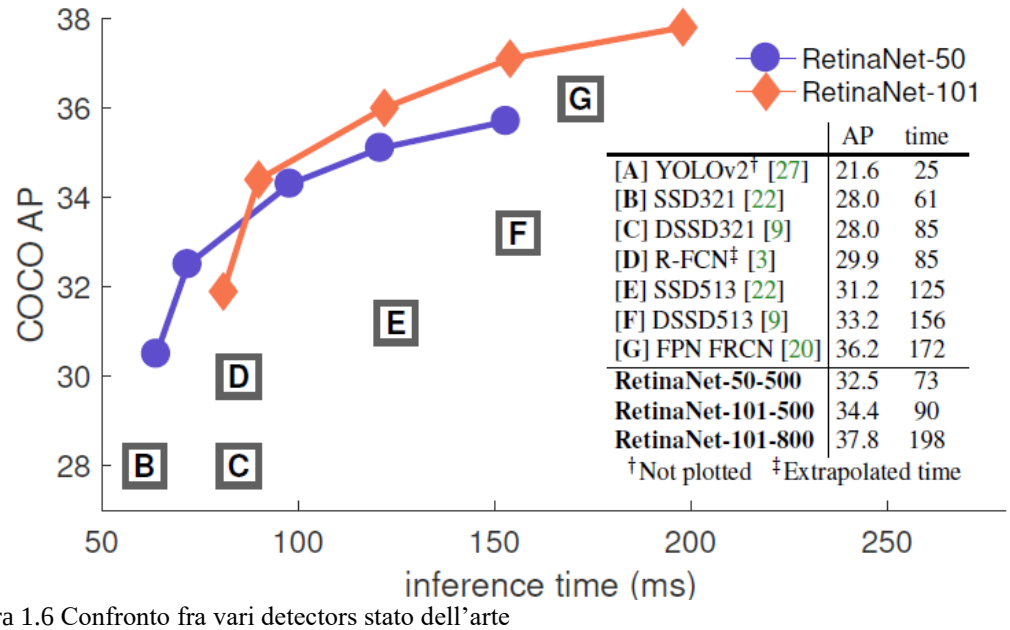


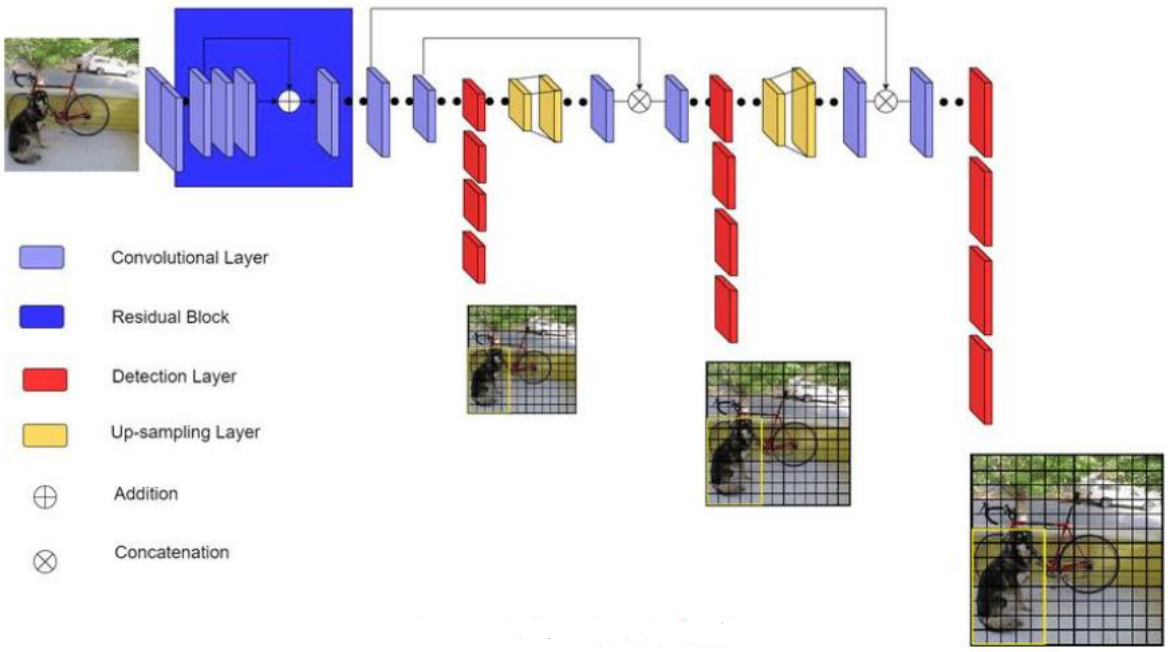
Figura 1.6 Confronto fra vari detectors stato dell'arte

1.6 Architetture Yolo

Le reti Yolo sono fra i modelli di *object detection* stato dell'arte più efficienti, sono costituite da un'architettura senza *region proposal*, che consiste nel dividere l'immagine in una griglia, i cui elementi sono responsabili del rilevamento degli oggetti che hanno il centro in quella specifica parte della griglia. Ogni cella predice quindi un fissato numero di *bounding boxes*, che consistono in 5 valori, 4 rappresentano le coordinate nell'immagine, mentre il quinto è uno *score* che indica la confidenza con cui la rete ha predetto la *box.*

Il sistema è veloce e semplice, in quanto viene allenano in una singola volta, essendo costituito da una singola CNN che predice le *bounding boxes* e lo *score,* , la quale è basata sul modello GoogLeNet [14]. Le architetture Yolo sono molteplici [15,16,17,18], si riporta in questo elaborato solo la struttura a grandi linee di Yolo v3, in quanto è l'unica utilizzata nel corso del progetto.

La struttura di Yolo v3, come mostrato in figura 1.7, è costituita da una singola CNN di 53 layer, chiamata Darknet, costruita a partire dalla rete GoogLeNet. I risultati vengono estratti direttamente dall'immagine in 3 diversi fattori di scala, dovuti al fatto che la rete sfrutta dei *layer* di *upsampling* per fornire risultati più coerenti con le diverse dimensioni delle *feature* contenute nelle immagini.



.1.7 Struttura dell'architettura Yolo-v3

CAPITOLO II

Struttura dati e training

2.1 Analisi delle immagini

Il progetto esposto in questo elaborato è stato svolto utilizzando tre *dataset* contenenti immagini di pneumatici, con l'obiettivo di creare un algoritmo in grado di riconoscere i *pattern* grafici che si trovano sulle coperture. Per ogni pneumatico vengono scattate due foto con visuale dall'alto che normalmente lo mostrano nella sua interezza, come si può vedere nelle figure 2.1 e 2.2. In generale se le coperture sono posizionate in modo corretto gli elementi grafici sono perfettamente visibili sul battistrada.

Figura 2.1 Primo elemento di una coppia di immagini ben posizionate

Figura 2.2 Seconda immagine della coppia

La posizione degli elementi grafici che si trovano sul battistrada è utilizzata dalle case automobilistiche come riferimento per equipaggiare gli pneumatici sulle vetture, per questo è fondamentale che le scritte siano stampate correttamente. Conseguentemente, si rende necessario un controllo del processo produttivo, che serve a determinare se la copertura è ben posizionata, prima di ammetterla alla fase successiva della lavorazione. Lo pneumatico risulta quindi ben fabbricato quando la copertura, nella fase in cui viene fotografata, ha le scritte nella parte superiore dell'immagine, ovvero al di sopra del proprio baricentro verticale.

Lo scopo primario dell'algoritmo è, di conseguenza, quello di determinare l'orientazione degli elementi grafici che si trovano sulla copertura. Inizialmente, si ricavano le coordinate posizionali dello pneumatico e delle scritte che si trovano su di esso, successivamente, si determina la mediana verticale della copertura e infine, si confrontano le posizioni verticali di tutte le scritte con quella del battistrada. I possibili risultati della

comparazione sono due, se le scritte si trovano tutte orientate nella parte superiore della copertura, questa è ben realizzata, altrimenti deve essere generato un allarme. Spesso si utilizzerà il termine baricentro con riferimento alla linea mediana verticale della copertura.

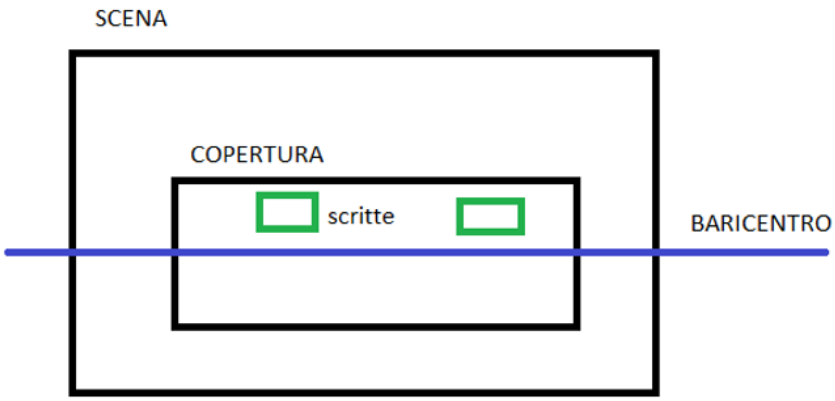


Figura 2.3 Schema della posizione corretta degli elementi in una scena.

Le immagini analizzate si distinguono principalmente in 3 categorie: le coperture con gli elementi grafici correttamente posizionati (ben fabbricate), descritte in figura 2.3, le coperture con le scritte posizionate al di sotto del baricentro dello pneumatico, spesso chiamate *upside-down* nel corso dell'elaborato, che portano l'algoritmo alla generazione di un allarme (figure 2.4 e 2.5) e le immagini dove la copertura non è stata fissata al centro dell'immagine, nelle quali le scritte sono spesso non visibili o non disposte in maniera regolare a causa della maggiore non linearità del vero baricentro della copertura, si vedano figure 2.6 e 2.7.

Le prime due categorie rappresentano i casi tipici e regolari che vengono analizzati in linea, mentre il terzo gruppo, quello delle coperture non fissate (non pinzate), è molto dinamico ed imprevedibile, in quanto non si hanno certezze sulla posizione della copertura. L'analisi di queste immagini sarà un punto chiave nel corso dell'elaborato, in quanto permetterà di ridurre i rischi.

Figura 2.4 Copertura upside-down.

Figura 2.5 Copertura upside-down, seconda immagine della coppia

Figura 2.6 Copertura non pinzata, scritte a malapena visibili.

Figura 2.7 Copertura non pinzata, scritte non visibili.

2.2 Composizione dei dataset

Il primo dei 3 dataset forniti dall'azienda contiene 1654 immagini di coperture divise a coppie, ed è stato usato per allenare delle architetture di *object detection* a individuare la posizione della copertura e delle scritte che si trovano su di essa; perciò, la valutazione delle metriche si è limitata al riconoscimento di questi *pattern.* Il secondo *dataset*, che è stato fornito da Pirelli a valle dei risultati del primo, contiene 5985 elementi, ed è stato testato sulla metrica apposita implementata nel capitolo 3 e, successivamente, annotato in maniera semiautomatica, con l'obiettivo di utilizzare un *training set* in grado di fornire una migliore generalizzazione nel riconoscimento delle *features.* Le prestazioni di questo modello sono state valutate sul terzo dataset, composto di 5393 immagini. Infine, si sono confrontati i risultati delle varie reti con quelli ottenuti dall'algoritmo di visione tradizionale attualmente implementato per affrontare il problema. Una breve introduzione alla struttura dell'algoritmo stato dell'arte viene fornita nel capitolo 3.

2.3 Preparazione del primo dataset

La prima fase del progetto è stato quello di creare un file di *groundtruth labels*, fondamentali per poter allenare un modello di *supervised learning*. Nel caso pratico dell'*object detection* significa generare un file contenente varie informazioni riguardanti la struttura del dataset, come le categorie presenti e le coordinate di tutti gli oggetti immagine per immagine. La formattazione delle *label* che si è scelta è COCO [20], che permette di registrare molteplici informazioni su di un dataset che si vuole usare per applicazioni di *computer vision.* Un dataset registrato in formato COCO è un file .json che contiene 3 elementi principali: "images" che contiene varie informazioni riguardanti le singole immagini, "annotations" che contiene le coordinate di tutte le *bounding box* immagine per immagine e "categories" che contiene il numero e i nomi delle categorie che si vogliono insegnare alla rete, figure 2.7, 2.8, 2.9.

```
"Images": imgs = [{
            "id": int,
            "width": int,
            "height": int,
            "file_name": str, }]
```

Figura 2.7 Struttura del dict images.

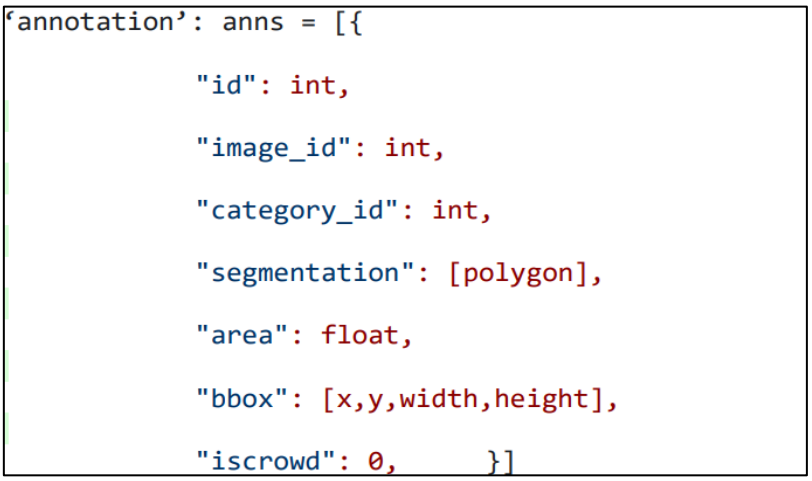


Figura 2.8 Struttura del dict annotations.

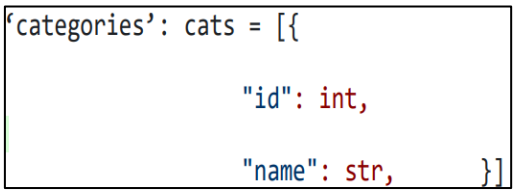


Figura 2.9 Struttura del dict categories.

La struttura del *dictionary*[8] è generalmente più complessa, può infatti contenere varie tipologie di metadati e informazioni aggiuntive, ma i dati sopra riportati sono sufficienti per lo standard COCO, che è utilizzabile nella maggior parte delle implementazioni di algoritmi di *object detection*.

Il primo dataset è stato annotato nel formato COCO utilizzando un *tool* chiamato COCO-Annotator, che permette di creare delle annotazioni manuali immagine per immagine e aggiungerle ad un file. Utilizzando poi i dati ricavati dal *tool* sono stati effettuati i primi esperimenti di *training* delle architetture precedentemente descritte.

[8] Un *dictionary* è una tipologia di formattazione dati utilizzata in moltissimi contesti per la semplicità con cui si possono leggere e modificare in maniera automatica.

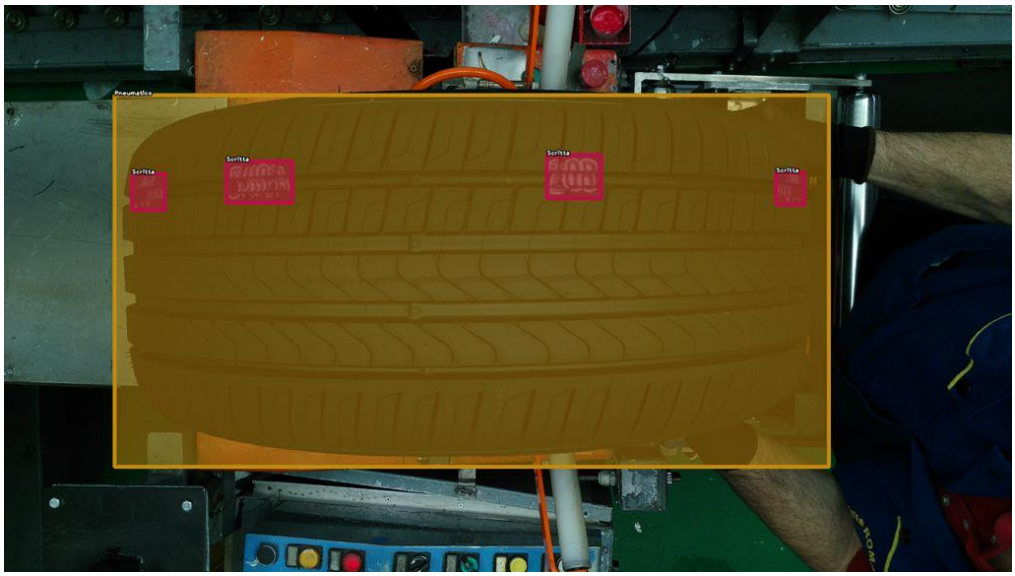


Figura 2.10 Esempio di immagine annotata con COCO-Annotator.

2.4 Setup del training

L'allenamento delle reti è stato preceduto da un'ulteriore preparazione del primo *dataset,* che è stato infatti suddiviso in 3 parti: una per il training, formata dall'80% delle immagini, una per la *validation,* composta dal 10% e infine una per il *test,* anche essa composta dal 10%. Il terzo blocco di immagini è stato poi esteso con tutte le immagini *upside-down* selezionate durante l'annotazione.

Il blocco composto dall'80% delle foto viene utilizzato durante il *training loop* per aggiornare i parametri delle reti allenate, mentre il *validation set* assume una funzione di controllo, serve infatti ad evitare che un modello incorra nell*'overfitting*[9] analizzando le prestazioni su *unsenn data* durante l'allenamento.

I modelli utilizzati sono implementati in una libreria *open source* sviluppata da *meta,* Detectron2, che si appoggia su un *framework* in *Python* molto utilizzato nel *deep learning*, Pytorch. La libreria offre un approccio ad alto livello su molti *task* di *computer vision*, oltre che garantire un controllo su tutte le fasi di sviluppo del problema. Le architetture Faster-Rcnn e Retina-Net sono state allenate e testate con detectron2, mentre gli esperimenti su Yolo-v3 sono stati svolti usando *Darknet*[10].

[9] L'*overfitting* è una tipologia di errore molto comune nel *machine learning,* che si manifesta quando il modello si adatta troppo ai dati con cui viene allenato, diventando incapace di generalizzare sugli *unseen data.*
[10] Darknet è una libreria che implementa varie architetture Yolo.

Le procedure di *training* e *test* si sono svolte nell'*environment* Google Colab, che permette l'utilizzo di GPU in *cloud,* fondamentali per rispondere alle enormi esigenze computazionali richieste per questa tipologia di problemi.

2.5 Modelli allenati

La fase che viene descritta in questa parte dell'elaborato è servita per determinare quale fosse l'architettura e quali fossero i parametri che fornivano i risultati migliori nel *task*; per questo motivo sono stati allenati tutti e tre i modelli descritti precedentemente, utilizzando diversi set di parametri, e sono stati poi valutati come *object detector,* confrontando i risultati ottenuti sul *test* set con quelli annotati a mano.

Yolo-v3 è stata allenata una sola volta, per un numero totale di 4 epoche, utilizzando un *learning rate*[11]*e* pari a 25e-5. Inoltre, sono stati modificati alcuni elementi del file di configurazione per rendere il modello un *object detector* a due classi (coperture e scritte). Il modello risultante è stato valuto manualmente, ovvero senza l'utilizzo di metriche specifiche, a causa della maggiore complessità dell'implementazione *software* scelta e degli ottimi risultati ottenuti dalle altre architetture allenate. L'implementazione scelta per allenare questa rete non offre strumenti aggiuntivi per eseguire test di metriche, per cui testare AP e mAP avrebbe significato scrivere manualmente il codice della *pipeline* di *test.*

Faster-Rcnn è stata allenata per 2000 iterazioni, con un valore di *learning rate* sempre pari a 25e-3 e un numero di immagini per *batch*[12] pari a 2, quindi sono risultate 4 *training epochs.* L'enorme *training time* e i risultati meno soddisfacenti rispetto a Retina-Net hanno portato ad accantonare l'utilizzo di questa architettura.

Retina-Net è stata allenata con diverse configurazioni e parametri, come mostrato nella tabella 2.1, è stata infatti l'architettura con i migliori risultati in ogni configurazione testata. In particolare, è risultata la rete più veloce da allenare, con i migliori risultati e l'implementazione più semplice e completa.

[11] Il *learning rate* è uno dei parametri fondamentali di un modello di *deep learning,* sostanzialmente indica quanto spazio viene percorso nella direzione opposta al gradiente durante il *backward* pass.
[12] Un *batch* è un insieme di dati che si utilizzano ad ogni iterazione del *training loop,* l'approccio utilizzato consiste nell'utilizzo di un blocco di immagini per ogni iterazione.

	Modello 1	Modello 2	Modello 3	Modello 4
Learning rate	25e-3	25e-3	25e-3	25e-3
Focal loss γ	1	0.5	2	1
Focal Loss α	0.25	0.5	0.25	0.25
Decay factor	Nessuno	Nessuno	Nessuno	0.1 dopo 4 epoche

Tabella 2.1 Varie parametrizzazioni dell'architettura Retina-Net

2.6 Risultati

I modelli allenati sono stati tutti utilizzati per il *testing* sul dataset apposito, i risultati sono stati poi valutati manualmente, e utilizzando alcune metriche specifiche, si riportano nelle figure 2.11 e 2.12 alcuni risultati ottenuti dall'architettura Retina-net nella quarta configurazione di *training.*

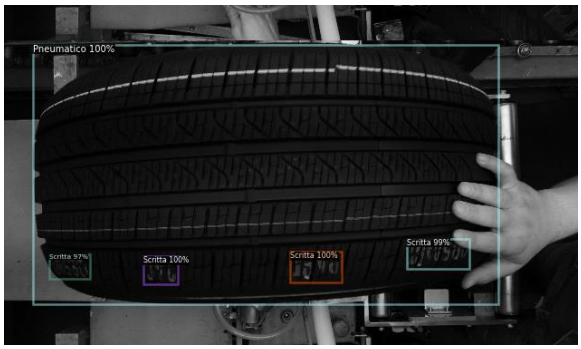


Figura 2.11 Risultati del modello su un'immagine upside-down

Figura 2.12 Risultati del modello su un'immagine correttamente posizionata

Le valutazioni a livello di metriche sono state effettuate mediante il calcolo dell'AP, e quindi dei valori di *precision* e *recall*, due metriche molto utilizzate nella data *analysis* e in particolare, in *computer vision*, ai fini della valutazione delle prestazioni degli *object detector*.

La loro applicazione nell'*object detection* è possibile implementando un algoritmo[13] che confronta la precisione fra le *bounding box* determinate dai modelli con le *groundtruth labels.*

I modelli allenati sono stati valutati secondo le metriche appena descritte, i risultati sono stati ottimi per entrambi i modelli, ma Retina-net si è dimostrato migliore sia dal punto di vista delle tempistiche di *training* e *test,* che da quello dei test veri e propri, come si può vedere nella tabella 2.2.

	Faster-Rcnn	Retina-net 1	Retina-net 4
AP Pneumatico	96.434%	99.305%	96.223%
AP scritta	72.174%	72.69%	78.958%

Tabella 2.2 Risultati in termini di AP fra i vari modelli utilizzati.

I risultati appena mostrati non forniscono informazioni riguardo la capacità della rete di rilevare errori nella filiera di produzione, che deve essere fatta riconoscendo i *pattern* descritti; perciò, nel prossimo capitolo si introdurrà la metrica apposita che è stata implementata a valle degli *output* della rete e si applicherà al secondo dataset.

[13] L'algoritmo che permette i singoli confronti fra le *bounding box* è l'IOU, che sta per *intersection over union,* che non è altro che il rapporto fra l'area di intersezione e quella di unione dei *crop* considerati.

CAPITOLO III

Metriche di decisione

3.1 Struttura del problema

La fase che viene descritta in questa parte dell'elaborato comprende sia la descrizione delle metriche utilizzate per analizzare gli elementi grafici delle coperture, compreso l'algoritmo tradizionale già attivo nella filiera, sia i risultati che questi algoritmi hanno ottenuto nei vari dataset, fornendo così un confronto preciso fra tutte le implementazioni. L'obiettivo preposto è quello di determinare, a partire dai risultati del modello utilizzato, se gli elementi grafici, identificati dal modello come "Scritte" si trovano al di sopra o al di sotto del baricentro della copertura, identificata dalla rete come "Pneumatico".

L'idea fondamentale è quella di utilizzare i risultati della rete per determinare se l'elemento del *dataset* è un negativo, in cui le scritte sono al di sopra del baricentro, oppure un positivo. La struttura generale dei dati è molto asimmetrica, infatti la percentuale di positivi è estremamente bassa rispetto alla distribuzione completa dei dati. D'altra parte, il costo associato ad ogni falso negativo è enormemente maggiore rispetto a quello associato ad un falso positivo, per cui, le metriche implementate utilizzano un approccio conservativo in cui si tende a generare un allarme in qualunque caso dubbio. Una volta determinati questi risultati è necessario confrontarli con le *ground-truth label* generate manualmente, ottenendo 4 risultati possibili, mostrati in tabella 3.1.

TN	FN	FP	TP
Negativo rilevato	Positivo mancato	Negativo mancato	Positivo rilevato

Tabella 3.1 Risultati possibili in un decisore binario

I valori ricavati vengono poi utilizzati per determinare *precision* e *recall*, che sono le due metriche finali utilizzate nella fase di test. Le due metriche forniscono indicazioni differenti circa il funzionamento di un decisore, la *precision* offre un'indicazione sull'esattezza di un modello nel riconoscere i positivi; infatti, vale 1 quando non ci sono falsi positivi, mentre la *recall* offre una misura sulla sensibilità del sistema, infatti mostra quanto il modello reagisce in maniera efficiente alla presenza di positivi.

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

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

3.2 Raggruppamento dei risultati

La *pipeline* che determina positivi e negativi a partire dai risultati su una singola immagine nasce a partire dall'analisi dei possibili risultati che il modello fornisce in uscita. Le possibili combinazioni, se consideriamo non specificati gli output dell'architettura sono molteplici, ma vengono classificate in tre gruppi, i positivi, i negativi e i *not-found.* Questi ultimi vengono assimilati a positivi nel caso in cui la metrica venga applicata su una singola immagine.

La valutazione per singola immagine avviene considerando i *not-found* come positivi, mentre nel caso in cui si valuti la metrica di coppia si ottiene un negativo quando le due immagini valutate sono entrambe negative o una negativa e una positiva, mentre si ottiene un positivo in tutti gli altri casi. In questo modo si cerca di ridurre al minimo i rischi di non rilevare un TP, cercando di ottenere una logica quanto più possibile prudente.

La prima fase consiste nel verificare la presenza di un'unica copertura fra le *bounding box,* se questa condizione non viene verificata il risultato sulla singola immagine è *not-found,* dopodiché si verifica che sia presente almeno una scritta nell'area della copertura, le *bbox* "scritte" rilevate che abbiano meno del 70% dell'area all'interno di quella della copertura vengono escluse dall'analisi. La mancata verifica di questa condizione produce un altro caso di *not-found.* Una volta verificata la presenza di almeno una scritta sul battistrada viene confrontato il punto medio della coordinata y della copertura con il punto medio della coordinata y di tutte le scritte, il risultato sarà a questo punto un positivo se anche solo una delle scritte si trovasse con il punto medio sotto la mezzeria, mentre sarà un negativo solo se tutte le scritte rilevate sulla copertura sono al di sopra del proprio baricentro.

Figura 3.1 Albero decisionale a partire dagli output della rete

La struttura appena descritta è stata parametrizzata e modificata nel tempo con l'obiettivo di migliorare le prestazioni nel maggior numero di casi possibili.

3.3 Algoritmo tradizionale

L'algoritmo già sviluppato ed utilizzato in linea, si basa su tecniche di computer vision tradizionale, le quali segmentano la scena e filtrano poi i componenti diversi dalle scritte.

Poiché la scena inquadrata è molto dinamica, in quanto la dimensione delle coperture varia in maniera significativa, ed inoltre è presente l'operatore, subentra il vantaggio nell'utilizzare algoritmi di *deep learning*, che sono più flessibili alle variazioni di contesto.

3.4 Test sul secondo dataset

La logica precedentemente descritta è stata applicata al secondo *dataset* fornito dall'azienda, la cui struttura è descritta nella tabella 3.3. Le immagini in questione sono infatti suddivise in varie sottocartelle contenenti foto provenienti da diverse macchine di produzione, inoltre, per ognuna di queste, esiste un ulteriore cartella contenente immagini particolari con macchie o errori evidenti di pinzatura, che nella maggior parte dei casi inducono in errore l'algoritmo tradizionale. Sono riportati due esempi in figura 3.2. In aggiunta a questi otto gruppi fa parte del secondo *dataset* un *burst* di sei immagini contenenti veri *upside-down.*

Il dataset completo è stato analizzato manualmente con l'obiettivo di determinare le *ground-truth* label legate alla decisone binaria. I positivi e i *not-found* sono stati copiati in una cartella specifica, che poi viene utilizzata all'interno del codice per determinare i risultati. La distinzione fra dì essi si trova nel nome del file e viene utilizzata unicamente per applicare la

metrica a coppie. Fra i positivi sono stati considerati gli *upside-down* e i casi in cui lo pneumatico è posizionato in maniera evidentemente scorretta, dove le scritte sono visibilmente anche sotto la mezzeria, mentre i casi dubbi, nei quali la posizione a priori delle scritte rispetto alla mediana non è determinabile a priori, sono stati sempre considerati negativi, in modo tale da non creare un potenziale *bias* sui falsi negativi. Infatti, ipotizzando che la rete preveda correttamente la posizione dello pneumatico e delle scritte al di sopra di esso in un'immagine, se questa fosse mappata come positiva si rischierebbe di ottenere un falso negativo senza che sia avvenuta una reale *miss detection.*

Nome	Immagini Totali	Negativi	Not found	Positivi
Matteuzzi2	1334	1322	7	5
Matteuzzi2_strange	158	158	0	0
Matteuzzi3	1526	1525	1	0
Matteuzzi3_strange	303	300	3	0
Matteuzzi4	1625	1625	0	0
Matteuzzi4_strange	283	275	4	4
Matteuzzi5	1500	1499	1	0
Matteuzzi5_strange	244	232	8	4

Tabella 3.2 Divisione del secondo dataset nelle varie macchine di produzione

I *test* effettuati nel capitolo due mostrano che i modelli allenati sono molto sensibili alle scritte sulle coperture, basti pensare al fatto che buona parte delle scritte viene rilevata dal modello con un *confidence score* di oltre il 0.9. Questo dato mette in evidenza il fatto che la scelta della minima soglia per considerare valida una predizione è un parametro con un impatto notevole sui risultati, per questo motivo si è deciso di testare la logica di decisione su singola immagine al variare dello *score treshold*, con l'obiettivo di valutare quale fosse il caso migliore, si riportano i risultati nella tabella 3.4.

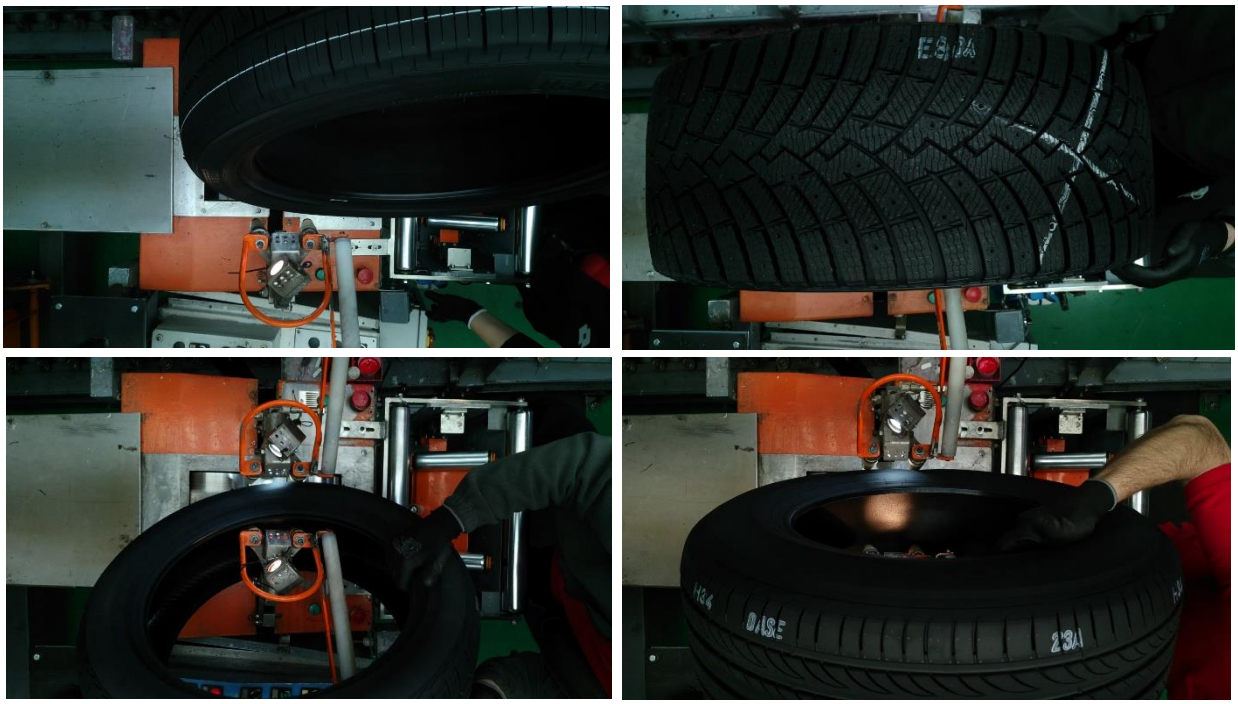


Figura 3.2 Esempio di immagini appartenenti ai *dataset "*strange*"*.

In accordo con quanto analizzato in maniera manuale, i risultati mostrano chiaramente l'ottima sensibilità del sistema, che mantiene livelli di *recall* praticamente sempre pari ad 1. L'unico caso di falso negativo è un'immagine senza scritte visibili, quindi mappata come *not-found* in cui il modello riconosce una macchia di gesso come una scritta, causando un errore di *miss detection.* Aumentando la soglia, gli errori diminuiscono in maniera drastica, come è evidente osservando i valori di *precision.* Complessivamente i risultati confermano le previsioni ottenute dall'analisi del primo test.

I valori sono calcolati su un bacino di 44 positivi provenienti dal secondo dataset, in tutte le configurazioni gli *upside down* sono stati correttamente rilevati.

Valore di soglia	*Precision*	*Recall*
0.5	0.1428	0.9736
0.6	0.2733	1.0000
0.7	0.3071	1.0000
0.8	0.3465	1.0000
0.9	0.3410	1.0000
0.95	0.2619	1.0000

I risultati appena analizzati sono abbastanza indicativi, ma bisogna tenere in considerazione il fatto che alcuni falsi positivi sono dovuti ad immagini non pinzate, che quindi hanno una posizione decentralizzata, rendendo il baricentro linearizzato utilizzato nella logica piuttosto impreciso, causando la presenza di scritte sia al di sopra che al di sotto di esso. Questi errori non sono riconducibili ad errori del modello.

La configurazione appena analizzata, con una soglia di 0.8 è stata applicata anche nel caso di coppie di immagini, fornendo risultati molto meno indicativi, poiché, soprattutto nei dataset "strange, si trovano molte coperture che contano un'unica immagine, per cui il bacino di dati è molto inferiore.

$$Precision_{coppie} = 0.113$$

$$Recall_{coppie} = 1$$

Il valore basso di *precision* dipende molto dal fatto che il numero di coppie positive è molto ridotto, restano infatti soltanto sette casi di positivi, in quanto i dataset "strange" contengono pochissime coppie complete, per cui buona parte dei positivi non vengono considerati nel calcolo.

La logica è stata infine applicata anche al *test set* del primo *dataset*, che ha fornito risultati perfetti, dovuti al fatto che i *test data* provenivano dalla stessa distribuzione di dati del training dell'architettura utilizzata.

$$Precision = 1$$

$$Recall = 1$$

3.5 Analisi degli errori

I risultati ottenuti in termini di metriche forniscono un'ottima indicazione sul funzionamento generale del sistema, ma l'ispezione manuale dei risultati ha permesso di modificare in maniera proficua la logica considerata.

La prima tipologia di errori è riconducibile ai casi nei quali vengono rilevate delle macchie di gesso come scritte che si trovano poi al di sotto del baricentro, figura 3.3. La loro causa principale è l'assenza di immagini con macchie di gesso all'interno del *train set* sul quale il modello è stato allenato; infatti, il primo e il secondo *dataset* vengono da distribuzioni di dati

differenti. Per questo motivo il funzionamento dell'architettura allenata su un *dataset* che contiene *corner-cases* del tutto nuovi non fornisce garanzie di accuratezza. Nel capitolo successivo si cercherà di sopperire a questo problema allenando un modello sull'intero secondo dataset.

La seconda tipologia di errori è dovuta alle immagini non pinzate, per le quali l'approssimazione di considerare il baricentro del battistrada come la linea mediana della *bounding box* rilevata dalla rete risulta poco precisa, in quanto la copertura risulta molto decentralizzata rispetto all'immagine. La conseguenza più comune è la presenza di scritte sia al di sopra che al di sotto della mezzeria, come viene mostrato in figura 3.4, ma anche i casi di immagini *not found* sono praticamente sempre riconducibili ad errori nella pinzatura della copertura. Questo tipo di errore non è risolvibile raffinando i parametri dell'architettura, in quanto non dipende dall'accuratezza nel rilevare scritte e pneumatico. L'approccio utilizzato nella parte successiva dell'elaborato con l'obiettivo di risolvere questo problema consiste nel riconoscere le immagini di coperture non pinzate e trattarle in maniera specifica, attraverso delle modifiche nella logica di decisione.

Figura 3.3 Errori dovuti al riconoscimento di macchie di gesso come scritte.

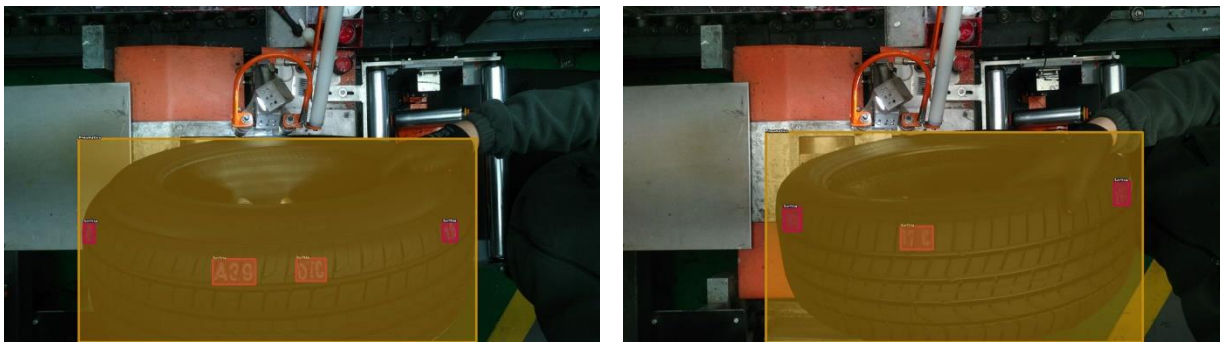


Figura 3.4 Errori dovuti al posizionamento erroneo della copertura.

Nella fase successiva dell'elaborato si analizzano possibili soluzioni alle problematiche esposte, per poi testarle su ulteriori distribuzioni di dati, in modo tale da eseguire ulteriori confronti e analisi del comportamento complessivo dell'architettura composta dal modello e dalla logica a valle di esso.

CAPITOLO IV

Modifiche alla logica e secondo *training* di Retina-Net

4.1 Annotazione del secondo dataset

I risultati ottenuti nel terzo capitolo hanno messo in luce alcuni problemi, sia causati dal modello che dalla logica a valle di esso. Per quanto riguarda il primo caso, è emerso che la rete, essendo stata allenata su una piccola distribuzione di dati, ha problemi nell'analizzare alcune tipologie di immagini provenienti dal secondo *dataset.* L'errore più tipico che si ottiene consiste nel riconoscimento di macchie di gesso come scritte, si mostra un esempio in figura 4.1. La causa principale di questo tipo di problemi è l'assenza di immagini di questo tipo all' interno del primo blocco di immagini utilizzato. La soluzione che segue cerca di risolvere il problema allenando un nuovo modello con *default weigths* su tutte le immagini fornite da Pirelli, ovvero il primo e il secondo *dataset.* Si rende quindi necessario annotare in maniera completa anche il secondo *burst* di dati forniti.

L'annotazione manuale dell'intero *dataset* sarebbe stata piuttosto lunga, perciò per ottenere un file completo di annotazioni si è scelto un approccio semiautomatico, che sfrutta gli output della rete utilizzati nel test precedente. Questi dati sono stati poi convertiti attraverso uno script Python in un file COCO leggibile dal *tool* di annotazione manuale, rendendo possibile la visualizzazione e l'eventuale modifica immagine per immagine. A questo punto le annotazioni ottenute dai risultati della rete sono state analizzate e corrette quando necessario. Questo approccio ha permesso di risparmiare tempo sfruttando gli output già precisi del modello allenato nel secondo capitolo.

Figura 4.1 Immagine contenente macchie di gesso ottenuta dal secondo dataset

4.2 Preparazione del dataset e training della rete

Il *train set* utilizzato in questa fase è stato costituito da immagini provenienti dal primo e dal secondo *dataset*. In primo luogo, entrambi sono stati privati degli *upside down*, sempre con l'obiettivo di utilizzarli in fase di test. Dopodiché sono stati divisi in due parti: *train*, composto dall'90% delle immagini rimaste e *validation*, composto dal restante 10%. Il *test set* è stato composto da un ulteriore blocco di immagini fornito da Pirelli, a cui sono stati aggiunti i positivi isolati precedentemente.

Il *training set* complessivo è risultato composto da un totale di oltre 7700 immagini provenienti dalle due distribuzioni di dati possedute ed è stato utilizzato per allenare un modello Retina-Net. I parametri scelti per il *training* sono stati selezionati in base ai risultati precedenti in questa fase.

Parametro	Valore
Learning Rate	25e-5
Numero di epoche	4 (15400 iterazioni)
Batch size	2
Decay ratio	Non utilizzato
Data augmentation	Non utilizzata
Checkpoint period	3550
Aspect ratio	*Default*

Tabella 4.1 *Set* di parametri di *train* utilizzati.

Il *checkpoint period* viene utilizzato per testare il modello allenato fino a quel momento sul *validation set*, in questo modo è possibile analizzare il comportamento del modello su *unseen data* nel corso dell'allenamento. L'obiettivo iniziale era quello di replicare la configurazione che aveva ottenuto i risultati migliori nella prima sessione, allenando il modello per dieci epoche utilizzando un *decay ratio* di 0.1 applicato dopo sei epoche, ma il *validation loss* ha raggiunto il minimo dopo quattro epoche, per cui il modello finale utilizzato è quello descritto nella tabella 4.1.

4.3 Test sul terzo dataset

Il modello appena allenato non poteva chiaramente essere testato su immagini usate per il *training*, per cui l'azienda ha fornito un ulteriore *dataset*, a cui sono stati aggiunti tutti i positivi esclusi durante la creazione del *train set.* Il numero complessivo di immagini appartenenti al *test set* è 5454, di cui fanno parte 38 positivi e 24 *not found* mappati manualmente come richiesto dalla logica di test.

I risultati sulla metrica di coppia non vengono riportati a causa dell'eccessiva perdita di informazione dovuta alla netta assenza di coppie nel gruppo di positivi estratti, inoltre si ritiene più rappresentativo il *test* se effettuato separatamente su ogni dato.

Modello	Precision	Recall
Allenato su dataset 1	0.8591	1
Allenato su dataset 1 e 2	0.9833	0.9672

Tabella 4.2 Confronto fra i due modelli sul terzo test set.

Le immagini appena descritte sono state utilizzate per testare entrambi i modelli discussi precedentemente con l'obiettivo di confrontarne i risultati riportati in tabella 4.2. Il primo modello ha ottenuto degli esiti ottimali in termini di sensibilità ai positivi, ottenendo un valore di *recall* pari a uno, mentre la presenza di dieci falsi positivi ha messo in luce il rilevamento erroneo di elementi grafici sulla copertura come scritte sul guanto dell'operatore. Infine, fra gli errori, sono risultate delle *miss* vere e proprie della copertura un due casi. Dall'altra parte, il secondo modello è risultato molto più accurato, ottenendo un solo falso positivo. Questa architettura ha però un altro problema, evidenziato dal mancato rilevamento di due *not found.*

I due errori sopracitati sono dovuti al rilevamento di scritte che in realtà sono macchie o altri elementi della scena non appartenenti al battistrada; osservando entrambe le immagini, si è notato che entrambe non sono pinzate. Conseguentemente, si evince che le coperture mal posizionate hanno un rischio associato maggiore dovuto all'assenza di scritte visibili.

Figura 4.2 Primo falso positvo risultato dal terzo test.

Figura 4.3 Secondo falso positivo risultato dal terzo test.

4.4 Analisi dei *trend* su coperture non pinzate

La fase seguente dell'elaborato analizza alcune metodologie di rilevazione delle coperture non pinzate a valle degli output della rete, con l'obiettivo di separarli dai casi standard e gestirli in maniera appropriata. Per svolgere questo particolare task si è sfruttata l'analisi statistica di alcuni dati legati alla posizione della copertura, combinata chiaramente con l'osservazione manuale di specifici blocchi di dati.

Il dato più correlato con la pinzatura dello pneumatico è sicuramente la mediana della copertura rilevata; infatti, se si ipotizza che il battistrada sia posizionato correttamente, il proprio baricentro si troverà circa alla stessa altezza del centro dell'immagine poiché la telecamera è stata centrata perfettamente rispetto al piano in cui lo pneumatico viene fissato. Una copertura non pinzata dovrebbe quindi avere un baricentro deviato rispetto al centro dell'immagine. Nel paragrafo successivo viene analizzata una parte del terzo dataset con l'obiettivo di determinare se quanto è stato ipotizzato ha una valenza in termini statistici.

Le seguenti informazioni sono state ricavate utilizzando le predizioni della rete su una parte del terzo *test set* e una libreria di analisi statistica chiamata *Pandas*, implementata in Python. La tabella 4.3 riporta alcuni dati statistici relativi alla *bounding box* della copertura, dove è mostrato chiaramente che il valor medio della sua coordinata y si discosta pochissimo dal centro. La figura 4.3 replica quanto detto mostrando la distribuzione dei dati in un istogramma, tutti i rari casi in cui la variabile interessata si trova fuori dalla zona centrale sono coperture non pinzate.

	count	mean	std	min	25%	50%	75%	max
x1	1281.0	204.897685	61.980258	22.704191	170.453490	205.798720	231.546800	365.548740
y1	1281.0	122.954528	49.734452	0.000000	88.417190	127.295800	158.786240	291.020450
x2	1281.0	1100.266922	79.803894	969.962160	1028.216000	1087.279300	1156.877200	1280.000000
y2	1281.0	568.354100	45.455920	414.244780	540.856000	557.109200	586.396850	720.000000
x_baricentro	1281.0	652.582304	44.999159	589.462311	615.460938	635.936493	690.959885	776.004425
y_baricentro	1281.0	345.654314	24.270517	208.236545	330.374191	343.066521	359.634109	503.155060

Tabella 4.3 Dati statistici relativi alla *bounding box* della copertura

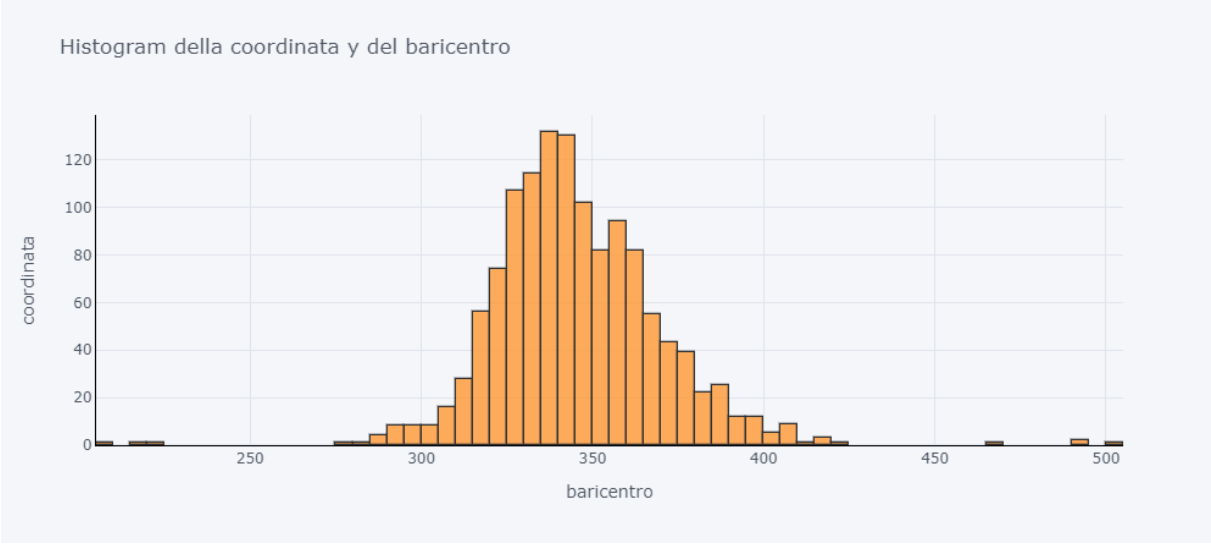


Figura 4.4 Istogramma della mediana della coordinata y della *bounding box.*

Le immagini che hanno una mediana sulla y non compresa fra 250 e 450 rappresentano tutte e unicamente coperture non pinzate. La conclusione di questa analisi è che sostanzialmente gli pneumatici mal posizionati hanno il punto medio della coordinata y della copertura notevolmente decentrato rispetto al punto medio dell'immagine sull'asse y. La precedente analisi ha portato allo sviluppo successivo di alcune modifiche nella logica decisionale, con l'obiettivo principale di gestire in maniera specifica i casi sopra discussi. In particolare, si sono proposte due alternative: la prima consiste nel considerare a priori ogni copertura non pinzata come un particolare positivo generando un allarme specifico, la seconda valuta queste immagini utilizzando una logica di decisione alternativa, nella quale una sola scritta non è sufficiente a generare un negativo.

Il vantaggio del primo sistema è che permette di tracciare in maniera accurata questo tipo di errori, ma allo stesso tempo genera dei positivi che potrebbero essere talvolta evitati. Infatti, capita spesso che se la copertura non è troppo estesa, questa sia perfettamente visibile anche quando lo penumatico non è pinzato, come si vede in figura 4.5. La seconda proposta ha lo scopo di ridurre al minimo i falsi positivi, evitando di scartare a priori questa tipologia di immagini.

Figura 4.5 Esempio di copertura non pinzata con scritte visibili.

La figura mostra chiaramente una copertura non pinzata, il cui baricentro rilevato si trova ben distante dal centro dell'immagine (sull'asse y), ma, nonostante ciò la logica lo identificherebbe correttamente come un negativo, dato che si trovano 3 scritte visibili nella parte superiore del battistrada. Il motivo dell'analisi di questa tipologia di casi è quello di ridurre al minimo i positivi, cercando di rallentare meno la *pipeline* di produzione. La soluzione proposta di seguito analizza la possibilità di trattare direttamente le coperture non pinzate utilizzando un "*hard-test"* nella logica di decisione.

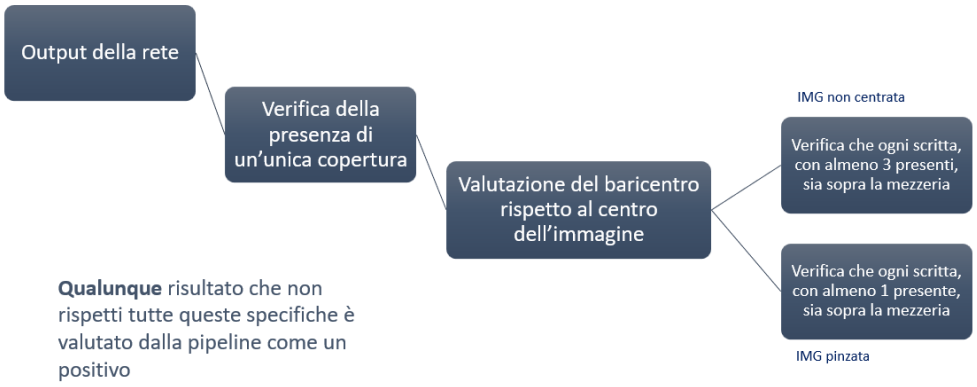


Figura 4.6 Logica di decisione con test modificato per immagini non pinzate

L'idea fondamentale è quella di non scartare a priori le immagini riconosciute come non pinzate, ma tenendone comunque conto, per cui si è proposta una logica di decisione specifica da applicare unicamente ai casi in questione. Il criterio di decisione consiste nel rendere la determinazione di un negativo più difficile, forzando la presenza di più di una scritta minima al di sopra della mezzeria. Nei test successivi si è scelto tre come numero minimo.

La rilevazione di questo specifica tendenza sulle immagini non pinzate ha permesso quindi di implementare alcune modifiche alla logica decisionale con l'obiettivo di migliorare ulteriormente i risultati del sistema nella sua interezza.

4.5 Test con controllo della posizione della copertura

I test che seguono sono applicati all'intero *test set* usato precedentemente, il modello utilizzato è quello allenato sul primo e sul secondo *dataset* mentre le logiche applicate sono entrambe quelle discusse nel sotto capitolo 4.3.

Il primo caso descritto valuta come positivi tutte le immagini con pneumatico non pinzato, a prescindere dalla presenza o assenza di scritte; perciò, risulta essere la configurazione più prudente. La percentuale di errore sul totale terzo dataset mostra chiaramente la grande efficienza del sistema ottenuto.

$$Precision = 0.7975$$

$$Recall = 1$$

$$Errori_{\%} = 0.2\%$$

L'utilizzo della seconda logica descritta risolve in maniera ottimale i problemi legati ad immagini non pinzate correttamente visibili, senza aumentare i rischi di falsi negativi. I valori finali ottenuti sul terzo dataset riportano un solo falso positivo, a dimostrazione del fatto che gli errori più comuni sono legati alla specifica categoria di immagini mal posizionate. La scelta della logica da utilizzare fra le due descritte non dipende però soltanto dall'efficienza del sistema; infatti, nell'applicazione pratica si potrebbe sfruttare il riconoscimento delle mancate pinzature permettendo in tempo reale all'operatore di riposizionare la copertura correttamente, oppure semplicemente tenere traccia del numero di errori di posizione.

Figura 4.7 L'unico errore nel test che utilizza il controllo triplo sulle immagini non pinzate

CAPITOLO V

Conclusioni

Il progetto esposto aveva come obiettivo primario quello di creare un algoritmo che risolvesse un problema di monitoraggio di processo nell'industria dello pneumatico. In particolare, si doveva essere in grado di controllare che la realizzazione della copertura fosse corretta arrivati ad una specifica fase della *pipeline.* Il controllo doveva avvenire attraverso l'analisi di elementi nelle foto che vengono scattate al battistrada; nello specifico, bisognava identificare la posizione della copertura, la mezzeria verticale della stessa e infine tutte le scritte presenti sopra la copertura. Successivamente gli elementi grafici andavano confrontati con il baricentro del battistrada.

La risoluzione del problema è stata suddivisa in due componenti specifiche, il riconoscimento degli elementi grafici e dello pneumatico immagine per immagine e il decisore che determina il risultato a partire dalle predizioni. Il primo *task* è stato gestito utilizzando architetture di *object detection,* che sono state allenate a riconoscere unicamente 2 classi, le coperture e le scritte sopra di esse, mentre la seconda parte è stata affrontata implementando una logica di decisione a valle degli *output* della rete, che determinava se la copertura era ben posizionata e ben costruita oppure generava un errore. I due elementi del problema sono stati gestiti in maniera separata, inizialmente sono state allenate vari *object detector,* che sono stati testati sulla loro capacità di identificare le classi volute, per cui senza tenere conto della logica successiva. La scelta finale del modello è dipesa sostanzialmente dai risultati in termini di AP ottenuti, ma anche dalla flessibilità e programmabilità delle implementazioni fornite dei vari modelli. Il decisore è stato invece implementato e testato solo in seguito al *training* dei primi modelli.

Le reti allenate nella prima fase hanno utilizzato come *train set* una parte del primo dataset fornito dall'azienda Pirelli, composto da oltre 1500 immagini. I risultati su questa prima distribuzione di dati si sono rivelati ottimi, specie per l'architettura Retina-Net, che è stata perciò utilizzata nel corso di tutto il progetto. In seguito alla prima implementazione della logica l'azienda ha fornito un ulteriore blocco di immagini, costituito da oltre 5500 elementi. La combinazione dei primi due dataset è stata annotata ed utilizzata per allenare un modello Retina-Net, con l'obiettivo di rendere il sistema molto più generale e flessibile rispetto a differenti distribuzioni di dati. Il modello appena descritto è stato infine utilizzato per i test finali.

La logica decisionale è stata anch'essa modificata nel tempo, cercando di risolvere alcune problematiche emerse durante il corso del progetto. La prima versione si limitava al confronto degli elementi grafici con il baricentro calcolato, successivamente si è implementato un sistema in grado di riconoscere in maniera precisa le immagini con coperture posizionate in maniera erronea, riducendo notevolmente i rischi associati.

Il sistema finale costituito da un modello Retina-Net allenato su oltre 7500 immagini e un decisore in grado di analizzare la pinzatura dello pneumatico ha ottenuto risultati eccellenti sul terzo dataset fornito dall'azienda, ben al di sopra dell'alternativa attualmente in linea, evidenziando i vantaggi nell'utilizzo di algoritmi di *deep learning* in contesti dinamici. L'implementazione effettiva in linea di produzione dell'algoritmo richiederebbe ulteriori test, su un numero ancora maggiore di immagini, per poter effettuare un confronto preciso con l'algoritmo tradizionale e poter verificare ulteriormente il funzionamento della logica complessiva.

Il problema pratico nell'utilizzo di algoritmi di *deep learning* è lo sforzo computazionale richiesto, infatti la profondità della rete, che ne garantisce il funzionamento nell'affrontare *task* complessi è una problematica in termini di tempistiche. Per questo l'utilizzo di algoritmi di questa tipologia in tempo reale è possibile solo servendosi di GPU's, che rappresentano un costo non indifferente. Le alternative all'utilizzo di *hardware* specializzato come sistemi in *cloud* sono certamente possibili, nonostante ci siano limitazioni in termini di tempo ed efficienza. La soluzione migliore dev'essere valutata in base ad una valutazione delle tempistiche di utilizzo richieste e della velocità che si vuole ottenere dall'architettura finale.

Complessivamente, si è mostrato come l'utilizzo di architetture di *deep learning* può essere una grande risorsa nello sviluppo di applicazioni di controllo del processo nella produzione dello pneumatico, in quanto queste reti sono caratterizzate da una enorme capacità di generalizzazione garantendo un'ottima flessibilità rispetto alle variazioni di contesto che possono verificarsi nella linea di produzione. Il sistema implementato nel corso del progetto è risultato infatti preciso e stabile. Il fatto che le predizioni siano molto precise offre la possibilità di riallenare il modello senza creare annotazioni manuali da zero. Questo garantisce una certa elasticità rispetto ad eventuali future variazioni del contesto della linea, come il riposizionamento della fotocamera o la presenza di un nuovo elemento nella scena fotografata.

BIBLIOGRAFIA

[1] Alom Zahangir M. *et al.*
(2018) *The History Began from AlexNet: A Comprehensive Survey on Deep Learning Approaches*.
URL: https://arxiv.org/pdf/1803.01164 .

[2] Farhana S. *et al.*
(2019) Advancements in Image Classification using Convolutional Neural Network
URL: https://arxiv.org/pdf/1905.03288 .

[3] Ho-Phuoc T.
(2018) *CIFAR10 to Compare Visual Recognition Performance between Deep Neural Networks and Humans.*
URL: https://arxiv.org/pdf/1811.07270.

[4] Girshick R. *et al.*
(2019) *Rich feature hierarchies for accurate object detection and semantic segmentation*.
URL: https://arxiv.org/pdf/1905.03288.

[5] Girshick R.
(2015) *Fast R-CNN.*
URL: https://arxiv.org/pdf/1504.08083.

[6] Kaiming H. *et al.*
(2014) *Spatial Pyramid Pooling in Deep Convolutional Networks for Visual Recognition*
URL: https://arxiv.org/pdf/1406.4729 .

[7] Shaoquin R. *et al.*
(2015) *Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks*
URL: https://arxiv.org/pdf/1506.01497 .

[8] Zeiler D. ;M. Fergus R.
(2013) *Visualizing and understaning convolutional netrworks.*
URL: https://arxiv.org/pdf/1311.2901.

[9] Simonyan K. ; Zisserman A.
(2014) *Very Deep Convolutional Networks for Large-Scale Image Recognition*
URL: https://arxiv.org/pdf/1409.1556 .

[10] Tsung-Yi L. *et al.*
(2017) *Focal loss for dense object detection.*
URL: https://arxiv.org/pdf/1708.02002 .

[11] Keren G *et al.*
(2018) *Fast Single-Class Classification and the Principle of Logit Separation.*
URL: https://arxiv.org/pdf/1705.10246.pdf.

[12] Tsung-Yi L. *et al.*
(2016) *Feature Pyramid Networks for Object Detection.*
URL: https://arxiv.org/pdf/1612.03144.

[13] Kaiming H. *et al.*
(2015) *Deep Residual Learning for Image Recognition*
URL: https://arxiv.org/pdf/1512.03385.

[14] Szegedy C *et al.*
(2014) *Going deeper with convolutions.*
URL: https://arxiv.org/pdf/1409.4842.

[15] Redmon J. ; Farhadi H.
(2016) *YOLO9000: Better, Faster, Stronger.*
URL: https://arxiv.org/pdf/1612.08242.

[16] Redmon J. *et al.*
(2015) *You Only Look Once: Unified, Real-Time Object Detection.*
URL: https://arxiv.org/pdf/1506.02640.

[17] Redmon J. ; Farhadi H.
(2018) *YOLOv3: An Incremental Improvement.*

[18] Bochkovskiy A. *et al.*
(2020) *YOLOv4: Optimal Speed and Accuracy of Object Detection.*
URL: https://arxiv.org/pdf/2004.10934.

[19] Tsung-Yi L. *et al.*
(2014) *Microsoft COCO: Common Objects in Context.*
URL: https://arxiv.org/pdf/1405.0312.