



UNIVERSITÀ DI PARMA

Dipartimento di Ingegneria e Architettura
Corso di Laurea in Ingegneria Informatica, Elettronica e delle
Telecomunicazioni

Sistema per il riconoscimento di dispositivi di protezione individuali indossati

Worn personal protective equipment recognizing system

Relatore:

Chiar.mo Prof. Jacopo Aleotti

Correlatore:

Ing. Andrea Bernardi

Tesi di Laurea di:

Luca Muroi

ANNO ACCADEMICO 2022-2023

“I limiti sono fatti per essere superati!”

Tony Stark

Ringraziamenti

Prima di intraprendere l'esposizione, desidero dedicare un momento di riflessione e riconoscenza a tutti coloro che hanno condiviso con me questa significativa fase di crescita, sia dal punto di vista personale che professionale. Esprimo il mio sincero ringraziamento al mio relatore, il Professor Aleotti, per la sua incondizionata disponibilità e tempestività nel rispondere ad ogni mia richiesta, nonché per i preziosi consigli dispensati nel corso della realizzazione del progetto e nella stesura di questo elaborato.

Un sentito ringraziamento va al tutor Bernardi, il cui prezioso supporto è stato fondamentale nel condurre le ricerche oggetto di questo elaborato svolte presso l'illustre azienda Techcab S.r.l.

Estendo il mio apprezzamento a tutto il personale dell'azienda sopracitata che si è rivelato essenziale e complementare nella realizzazione del progetto, apprezzando altresì l'ospitalità ricevuta e le competenze acquisite sul campo. A Luca del passato per aver sempre creduto nelle proprie potenzialità e per non essersi mai arreso.

A mia Madre, grazie per aver reso possibile il raggiungimento di questo obiettivo e per aver contribuito alla formazione della persona che sono diventato oggi. I nostri caratteri, forti e spesso discordanti, ci hanno portato ad affrontare sfide e divergenze, ma a modo nostro, siamo sempre stati uniti affrontando insieme ogni difficoltà.

Al mondo della Pallavolo, disciplina che ho abbracciato sin dall'età di otto anni, dedico il riconoscimento come il mio rifugio sicuro, costante compagna nell'affrontare le sfide e presenza che ha sempre arricchito ogni fase della mia

esistenza.

Ai miei colleghi e compagni di studio, Chiara e Michele, con i quali ho condiviso non solo le sessioni d'esame e le lezioni in aula ma anche innumerevoli appunti, esperienze di insuccesso, momenti di disperazione e al contempo le gioie e le soddisfazioni derivanti dal superamento di esami che, in un primo momento, sembravano insormontabili

A Veronica, per ultima ma non per importanza, grazie per essere stata il mio porto sicuro e il mio rifugio, per avermi supportato con forza in ogni sfida e sopportato in ogni momento buio. Grazie per l'immensa pazienza che hai avuto ascoltando i miei discorsi riguardanti gli esami, il codice che non compila e la matematica senza numeri. Grazie per credere in me, per farmi sentire capace e per spingermi costantemente a superare ogni mio limite. Grazie di cuore per la tua presenza e il tuo contributo in questo capitolo della mia vita. Questo traguardo è anche tuo quanto mio.

Indice

Introduzione	1
1 Stato dell'arte	3
1.1 Artificial Intelligence	3
1.1.1 Machine Learning	4
1.1.2 Deep Learning	6
1.2 Real-Time Object Detection	7
2 Strumenti utilizzati	9
2.1 Robot antropomorfo	9
2.2 Intel® RealSense D435	10
2.3 Antenna Kathrein RFID UHF Readers ARU 2400	11
2.4 ASUS VivoBook	12
2.5 Dispositivi di protezione individuali	12
2.6 Colab	15
2.7 Caffe Model	15
2.8 EfficientNetB3	16
2.9 TensorFlow e Keras	17
3 Descrizione del sistema	18
3.1 Dataset	18
3.2 Face Detection	20
3.3 Addestramento reti neurali	20
3.4 Rivelamento tag rain RFID	26

INDICE	iii
4 Risultati sperimentali	31
5 Conclusioni e Sviluppi futuri	38
Bibliografia	39

Introduzione

La presente tesi è stata condotta nell'ambito di un progetto intitolato "Cobotech", presso la Techcab S.R.L., un'azienda specializzata in Automazione Elettronica. Nel corso di un tirocinio protrattosi per sei mesi, dal marzo all'agosto del 2021, ho partecipato attivamente a tale iniziativa. Il progetto in questione si focalizza nella creazione di un sofisticato sistema di accesso, caratterizzato dall'impiego sinergico di un braccio robotico antropomorfo e di una sofisticata telecamera stereoscopica. La finalità precipua di tale sistema è determinare se un individuo è autorizzato o meno a varcare l'ingresso di un impianto industriale, un cantiere o qualsiasi contesto che prescriva determinati requisiti di accesso. La validazione dell'ingresso avviene tramite un codice QR personale, mediante il quale si determina quali dispositivi di protezione individuale siano necessari. Solo in presenza di tutti gli elementi richiesti, l'accesso sarà consentito all'individuo interessato. Durante il periodo di tirocinio, mi sono dedicato all'indagine e alla concezione di un sofisticato sistema di identificazione dei dispositivi di protezione individuali mediante l'object detection in tempo reale, affiancato da un controllo con doppio approccio mediante un lettore RFID. Il sistema in questione si articola in quattro componenti fondamentali: la prima fase consiste nella creazione di un dataset idoneo per l'addestramento dei modelli impiegati nella fase di object detection. La seconda parte si focalizza sull'identificazione, attraverso l'utilizzo di una telecamera, della presenza del volto di un individuo. La terza fase è dedicata al riconoscimento dei dispositivi di protezione individuale e alla verifica del loro corretto posizionamento, sfruttando i modelli preceden-

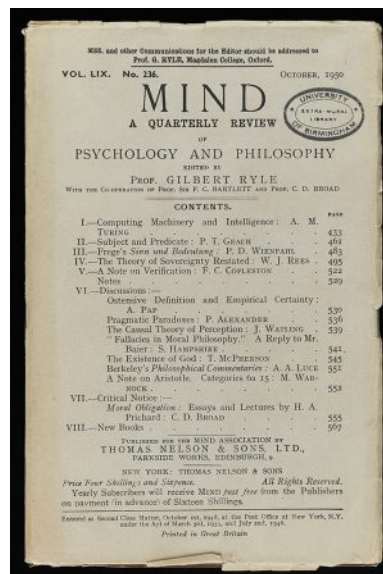
temente elaborati. La quarta parte del sistema prevede la contemporanea identificazione, durante la fase di riconoscimento della terza parte, dei tag RFID associati ad ogni dispositivo di protezione individuale. Nei successivi capitoli di questa esposizione, verranno dettagliati gli strumenti impiegati, l'evoluzione delle quattro fasi di sviluppo di questo sistema e i risultati ottenuti. Infine, saranno esposte le conclusioni tratte e le prospettive per i futuri sviluppi di questo ambizioso progetto.

Capitolo 1

Stato dell'arte

1.1 Artificial Intelligence

"Can machines think?" La domanda è estremamente attuale, sebbene risalga al 1950. È, infatti, l'incipit dell'articolo di Alan Turing, Computing machinery and intelligence, che affrontava il tema scientifico più all'avanguardia di quegli anni: l'Intelligenza Artificiale.



MIND, and other Communications for the Editor should be addressed to
Prof. G. E. Hughes, M.A., University College, Oxford.

VOL. LIX. No. 236. OCTOBER, 1950

MIND
A QUARTERLY REVIEW
OF
PSYCHOLOGY AND PHILOSOPHY
EDITED BY
PROF. GILBERT RYLE
With the Co-operation of Prof. Sir F. C. BARRETT and Prof. C. D. BRIDG

CONTENTS.

	Page
I.—Computing Machinery and Intelligence: A. M. TURING	433
II.—Subject and Predicate: P. T. GEACH	461
III.—Frege's Sense and Reference: P. D. WYENPAUL	485
IV.—The Theory of Sovereignty Restated: W. J. REES	495
V.—A Note on Verification: F. C. COLESTON	522
Notes	529
VI.—Discussions:— Ostensive Definition and Empirical Certainty: A. PAP	539
Pragmatic Paradoxes: P. ALEXANDER	539
The Causal Theory of Perception: J. WATLING	539
"Fallacies in Moral Philosophy." A Reply to Mr. BAIRD: S. HANFORD	543
The Existence of God: T. McPHERSON	545
Berkeley's Philosophical Commentaries: A. A. LUCE	551
A Note on Aristotle: Categories 9a 11: M. VAN SOEST	558
VII.—Critical Notices:— Moral Obligation: Essays and Lectures by H. A. FRICHARD: C. D. BRIDG	555
VIII.—New Books	569

PUBLISHED FOR THE MIND ASSOCIATION BY
THOMAS NELSON & SONS LTD.,
PARADE WORKS, STURBOROUGH

NEW YORK: THOMAS NELSON & SONS
Price Four Shillings and Sixpence. All Rights Reserved.
Yearly Subscribers will receive MIND post free from the Publishers
on payment (in advance) of Sixteen Shillings.
Entered as Second Class Matter, October 2nd, 1945, at the Post Office at New York, N.Y.
Under the Act of March 3rd, 1879, and July 3rd, 1911.
Printed in Great Britain

Figura 1.1: In figura l'articolo Computing machinery and intelligence

Era un periodo di grande fermento scientifico sullo studio del calcolatore e il suo utilizzo per sistemi intelligenti. Nel 1956, al Darmouth College, nel New Hampshire, si tenne un convegno al quale presero parte i maggiori esponenti dell'informatica: in quell'occasione si raccolsero i principali contributi sul tema, ponendo anche l'attenzione sugli sviluppi futuri. Durante il convegno di Darmouth ebbe un ruolo fondamentale il lavoro di Alan Turing, considerato uno dei padri dell'informatica moderna: già nel 1936 aveva posto le basi per i concetti di calcolabilità, computabilità e la macchina di Turing. Nel 1950, lo stesso Turing, scrisse l'articolo intitolato "Computing machinery and intelligence", in cui proponeva quello che sarebbe divenuto noto come test di Turing. Secondo il test, una macchina poteva essere considerata intelligente se il suo comportamento, osservato da un essere umano, fosse considerato indistinguibile da quello di una persona. L'Intelligenza Artificiale (IA) è una disciplina scientifica matura nata in seno all'informatica che ormai pervade i più svariati ambiti (scientifici e non scientifici), nonché diversi aspetti della nostra vita quotidiana. I ricercatori capirono presto che comprendere e replicare i meccanismi dell'intelligenza umana è un compito molto arduo e che richiede metodologie di progettazione molto complicate e adeguate risorse computazionali. La storia dell'IA è costellata di alti e bassi conosciuti in letteratura come "le stagioni dell'IA", oggi stiamo assistendo ad un interesse mai visto prima che sta portando allo sviluppo di nuove applicazioni, producendo molteplici effetti in contesti di vario tipo, da quello socio-economico a quello etico. Grazie al lavoro di Turing, il tema dell'Intelligenza Artificiale ricevette una forte attenzione da parte della comunità scientifica e nacquero diversi approcci: fra tutti, i principali furono la logica matematica e le reti neurali, nell'ultimo decennio la loro tecnologia è stata implementata e oggi vengono applicate nell'ambito del Deep Learning, un ramo del Machine Learning. [1]

1.1.1 Machine Learning

Il machine learning è un sottoinsieme dell'intelligenza artificiale che include

metodi come il pattern recognition, reti neurali artificiali e la statistica computazionale, ossia metodi utilizzati per migliorare le qualità degli algoritmi di riconoscimento delle regolarità all'interno dei dati mediante l'apprendimento. Il primo grande nome legato al machine learning è sicuramente quello di Alan Turing, che ipotizzò la necessità di realizzare algoritmi specifici per realizzare macchine in grado di apprendere. In quegli stessi anni, anche gli studi sull'intelligenza artificiale, sui sistemi esperti e sulle reti neurali vedevano momenti di grossa crescita alternati da periodi di abbandono, causati soprattutto dalle molte difficoltà riscontrate nelle possibilità di realizzazione dei diversi sistemi intelligenti, nella mancanza di sussidi economici e dallo scetticismo che circondava spesso chi provava a lavorarci. A partire dagli anni Ottanta, una serie di interessanti risultati ha portato alla rinascita di questo settore della ricerca: una rinascita che è stata resa possibile da nuovi investimenti nel settore. Alla fine degli anni Novanta l'apprendimento automatico trova nuova linfa vitale in una serie di innovative tecniche legate ad elementi statistici e probabilistici: si trattava di un importante passo che permise quello sviluppo che ha portato oggi l'apprendimento automatico ad essere un ramo della ricerca riconosciuto e altamente richiesto. Quando si parla di machine learning si parla di differenti meccanismi che permettono a una macchina intelligente di migliorare le proprie capacità e prestazioni nel tempo. Lo scopo del machine learning è dunque quello di analizzare le regolarità presenti all'interno di una serie di dati così da poter fare previsioni sulla base di ciò che ha analizzato e studiato. La macchina, quindi, sarà in grado di imparare a svolgere determinati compiti migliorando, tramite l'esperienza, le proprie capacità, le proprie risposte e funzioni. Alla base dell'apprendimento automatico ci sono una serie di differenti algoritmi che, partendo da nozioni primitive, sapranno prendere una specifica decisione piuttosto che un'altra o effettuare azioni apprese nel tempo. Alcuni degli ambiti in cui il machine learning è utilizzato sono il riconoscimento del linguaggio, i motori di ricerca e il face recognition. [2]

1.1.2 Deep Learning

Il deep learning, in italiano apprendimento profondo, è un sottoinsieme del machine learning, metodo in cui i computer elaborano grandi quantità di dati utilizzando reti neurali ispirate al funzionamento biologico del cervello umano. Ottenendo sempre nuove informazioni, i collegamenti esistenti possono essere modificati e ampliati, in modo che il sistema impari senza l'intervento umano e fornisca conclusioni e previsioni sempre più precise. Per l'elaborazione delle informazioni vengono utilizzate le reti neurali artificiali, che consistono in uno strato di ingresso, uno o più strati intermedi (layer) e uno strato di uscita. Le informazioni vengono inoltrate dallo strato di ingresso come vettore di input, pesate da neuroni artificiali negli strati intermedi e infine emesse come modello sullo strato di uscita. Più strati contiene una rete neurale artificiale, più complessi possono essere i compiti che l'IA deve eseguire. La classificazione delle immagini, è un compito impegnativo per i computer. Ciò che può essere immediatamente chiaro per una persona quando osserva un'immagine, non lo è per il computer che deve prima di tutto fare un'analisi delle caratteristiche delle singole immagini. Nel deep learning l'input di dati grezzi, in questo caso l'immagine, viene analizzato strato per strato. Nel primo strato di una rete neurale artificiale, ad esempio, il sistema verifica i colori di ogni singolo pixel dell'immagine. Ogni pixel viene elaborato da un proprio neurone. Nel livello successivo vengono identificati i bordi e le forme e, a seguire, vengono controllate le caratteristiche più complesse. Le informazioni raccolte sono mappate in un algoritmo flessibile. I risultati di uno strato vengono trasportati allo strato successivo e cambiano l'algoritmo, in modo tale che il computer sia in grado di decidere con una serie di operazioni se un'immagine possa essere assegnata a una categoria piuttosto che a un'altra. Inizialmente c'è una fase di addestramento in cui gli errori di assegnazione sono corretti dall'uomo per ottimizzare l'algoritmo, che in poco tempo saprà riconoscere in autonomia le immagini. Cambiando il collegamento tra i neuroni della rete e regolando la ponderazione delle variabili all'interno dell'algoritmo, alcuni modelli di ingresso portano a modelli

di uscita sempre più precisi. [3]

1.2 Real-Time Object Detection

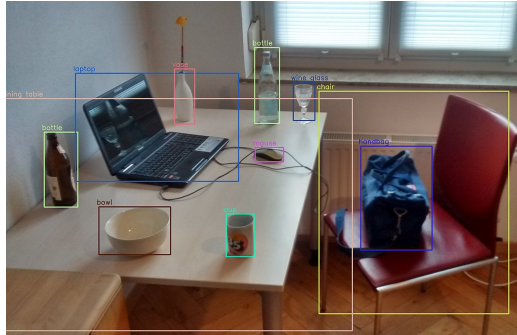


Figura 1.2: Esempio di Object Detection

L'object detection è uno dei principali problemi nell'ambito della computer vision. Negli ultimi anni, con l'avvento delle reti neurali e del deep learning, sono stati fatti notevoli progressi nei metodi per affrontare questo problema. L'obiettivo è quello di sviluppare tecniche e modelli computazionali che forniscano uno degli elementi basilari necessari per le applicazioni di computer vision: la risposta alla domanda "quali oggetti ci sono qui?". L'object detection è alla base di molte applicazioni per la computer vision, quali l'instance segmentation, l'image captioning e l'object tracking. Dal punto di vista applicativo, è possibile raggruppare l'object detection in due categorie: "general object detection" e "detection applications". Per la prima, l'obiettivo è quello di indagare i metodi per individuare diversi tipi di oggetti usando un unico framework, al fine di simulare la visione e la cognizione umana. Nel secondo caso ci si riferisce al riconoscimento, sotto specifici scenari applicativi, di oggetti di una certa classe: è il caso delle applicazioni per il pedestrian detection, il face detection o per il text detection. [4]

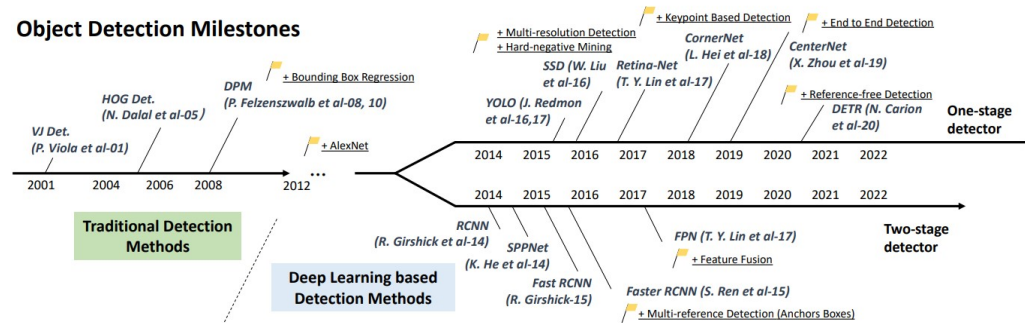


Figura 1.3: La linea del tempo dell'evoluzione dei modelli di object detection negli ultimi 20 anni.

Capitolo 2

Strumenti utilizzati

Gli strumenti utilizzati durante lo sviluppo di questo sistema sono di tipo Hardware e Software. Per la parte hardware troviamo: Robot Antropomorfo, videocamera Intel Realsense D435, Antenna Kathrein RFID UHF Readers ARU 2400, Computer ASUS Vivobook. Lato software gli strumenti utilizzati sono: Tensorflow, Keras, Pyrealsense, OpenCV Caffè Model, Colab, EfficientNetB3

2.1 Robot antropomorfo

Il prodotto "Cobotech" nasce come applicazione dell'interazione uomo-macchina per ottimizzare il controllo dei passcode all'ingresso di un ambiente di lavoro, decidendo autonomamente e automaticamente se l'utente che sta per effettuare l'ingresso è autorizzato o meno a entrare. Alcuni passcode richiedono un intervento ravvicinato da parte del sistema per verificare se la condizione imposta è presente o meno. Per risolvere questo problema "Cobotech" è provvisto di un braccio robotico antropomorfo fisso a 3 giunti (spalla, gomito, polso) (fig. 2.1) in grado di avvicinarsi alla fronte di esseri umani di altezza compresa fra 1.40 m e 2.00 m circa.



Figura 2.1: In figura è rappresentato il Robot Antropomorfo utilizzato nel progetto Cobotech

2.2 Intel® RealSense D435

La videocamera utilizzata è la Intel RealSense D435 che offre una visione stereoscopica della zona davanti a essa.



Figura 2.2: In figura è rappresentata la videocamera Intel Realsense D435 utilizzata nel progetto Cobotech.

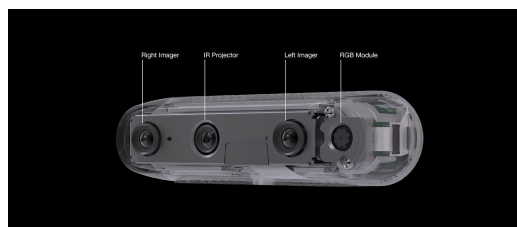


Figura 2.3: In figura vengono rappresentati nel dettaglio i sensori.

RealSense™ D435 può essere utilizzata sia in ambienti interni che in ambienti esterni con una portata ideale da 0.3m a 3m. La telecamera di profondità integrata utilizza la tecnologia stereoscopica, ha un campo visivo di 87° x 58° (FOV), una risoluzione di output di profondità fino a 1280 x 720 e un frame rate fino a 90 fps. Questo garantisce una precisione al millimetro. La telecamera RGB utilizza la tecnologia Rolling Shutter con una frequenza di 30fps e una risoluzione di 1920x1080. Questa telecamera è inserita in Cobotech su un'asta metallica all'altezza di 1.90 m in verticale, leggermente inclinata verso il basso e ruotata di 90°. Questo consente di individuare la scena nel modo più ampio possibile senza però perdere i dettagli somatici dell'utente che deve essere autorizzato all'ingresso. Grazie anche all'Intel RealSense SDK 2.0 è possibile comunicare con l'interfaccia della telecamera rendendola customizzabile e programmabile tramite il linguaggio Python. E' però necessario configurare accuratamente la videocamera.

2.3 Antenna Kathrein RFID UHF Readers ARU 2400

Kathrein ARU 2400 è un lettore rfid rain di base con antenna integrata per applicazioni interne nella logistica. In questo sistema è stata utilizzata per rilevare i tag rain presenti su ogni dispositivo di protezione individuale.



Figura 2.4: In figura vengono rappresentati i tag rain rfid

Questa lettore è stato posto dietro a Cobotech ad una altezza di 1 m da terra così da essere frontale rispetto alla posizione in cui si presenterà la persona. Questo consente di regolare il range esatto su cui deve lavorare l'antenna così da poter rilevare in modo preciso i tag presenti sui dispositivi di protezione individuali.



Figura 2.5: In figura è rappresentata l'antenna utilizzata nel progetto Cobotech

2.4 ASUS VivoBook

Per sviluppare questo sistema sono richieste adeguate specifiche e prestazione hardware poichè l'addestramento dei modelli è computazionalmente pesante. Il computer utilizzato è un Asus con processore Intel(R) Core(TM) i5-8265U CPU @ 1.60GHz 1.80 GHz, con 8 GB di RAM e scheda grafica NVIDIA GeForce MX110.

2.5 Dispositivi di protezione individuali

In questo progetto, le reti neurali sono state addestrate per riconoscere quattro tipologie specifiche di dispositivi di protezione: casco da lavoro bianco,

pettorina catarifrangente, guanti da lavoro rivestiti in poliuretano e scarpe antinfortunistiche. Qui di seguito sono riportate le foto di tali dispositivi.



Figura 2.6: Foto del casco utilizzato per la creazione del dataset



Figura 2.7: Foto della pettorina catarifrangente utilizzata per la creazione del dataset



Figura 2.8: Foto dei guanti utilizzati per la creazione del dataset



Figura 2.9: Foto delle scarpe utilizzate per la creazione del dataset

2.6 Colab

Google Colab, abbreviazione di Google Colaboratory, è un ambiente di sviluppo basato su cloud che consente di scrivere e eseguire codice Python direttamente nel browser, senza richiedere alcuna configurazione o installazione di software sul proprio computer. Questo strumento è stato progettato principalmente per il machine learning e l'analisi dati, fornendo un accesso gratuito alle risorse di calcolo GPU e TPU (Graphics Processing Unit e Tensor Processing Unit) per velocizzare le operazioni computazionali ed è particolarmente vantaggioso per progetti che richiedono elevate capacità computazionali. Colab si basa su Jupyter Notebooks, un ambiente di sviluppo interattivo che consente di combinare testo, codice eseguibile, grafici e altro ancora in un unico documento che rende la condivisione e la presentazione delle analisi molto più intuitive. Vengono fornite molte delle librerie e dei pacchetti più comuni utilizzati in ambito di machine learning e analisi dati, come TensorFlow, PyTorch, Pandas e NumPy.

2.7 Caffe Model

Caffe è un framework che fornisce modelli e esempi di lavoro per la visione artificiale, in particolare offre un modello pre-addestrato per il riconoscimento di volti in un frame 2D. Questo tipo di sistema, per funzionare, richiede la presenza del file `.prototxt` e il file `.model`. Il modello preaddestrato sviluppato da Caffe è un modello di classificazione in cui ogni classe comprende un insieme di volti di uno stesso utente. Caffe definisce una rete strato per strato nel proprio schema del modello. La rete definisce l'intero modello dal basso verso l'alto, dai dati di input al calcolo della funzione di loss. Mentre i dati e i risultati parziali dei calcoli fluiscono attraverso la rete nei passaggi forward e backward, Caffe archivia, comunica e manipola le informazioni come Blob: il Blob è l'array standard e l'interfaccia di memoria unificata per il framework. I dettagli del Blob descrivono come le informazioni vengono

archivate e comunicate all'interno tra i livelli. Il sistema Caffe fornirà in output 3 risultati:

- Coordinata iniziale $X_i Y_i$
- Coordinata finale $X_f Y_f$
- Valore di confidence

Sfruttando le due coordinate è possibile disegnare la Bounding Box del viso e il valore di confidence indica quale livello di affidabilità possiamo attribuire al risultato.

2.8 EfficientNetB3

EfficientNetB3 è un modello di rete neurale convoluzionale (CNN) sviluppato per il machine learning e la visione artificiale. Appartenente alla famiglia EfficientNet, caratteristica per la sua efficienza computazionale e prestazioni elevate con un numero relativamente basso di parametri, EfficientNetB3 è noto per il suo bilanciamento intelligente tra dimensioni e profondità della rete. L'indicazione "B3" nella sua denominazione suggerisce il livello di profondità della rete, dove varianti come B0, B1, B2, ..., B7 differiscono nella loro complessità crescente. EfficientNetB3 è stato progettato con l'obiettivo di massimizzare l'accuratezza del modello, considerando al contempo la complessità computazionale. A differenza di reti neurali più profonde, EfficientNetB3 è in grado di ottenere prestazioni elevate con un numero relativamente ridotto di parametri. Questa efficienza lo rende adatto per implementazioni su dispositivi con risorse computazionali limitate, come quelli presenti in ambienti mobili o di edge computing. La rete utilizza una combinazione di strati e blocchi, sfruttando la convoluzione "depthwise separable" per ridurre il costo computazionale. I suoi blocchi contengono livelli di convoluzioni, normalizzazioni e attivazioni, contribuendo alla sua capacità di estrarre caratteristiche complesse dalle immagini. EfficientNetB3 è ampiamente utilizzato nel trasferimento di apprendimento, in cui i modelli pre-addestrati possono essere

affinati su insiemi di dati specifici per compiti di visione artificiale come la classificazione di immagini, il rilevamento oggetti e la segmentazione semantica. In sintesi, EfficientNetB3 rappresenta un compromesso ottimale tra prestazioni e complessità, dimostrando la sua versatilità e praticità in una varietà di applicazioni di visione artificiale.

2.9 TensorFlow e Keras

TensorFlow è una libreria open-source sviluppata da Google, specializzata nell'implementazione di modelli di machine learning e deep learning. Utilizzata ampiamente in campi come l'elaborazione del linguaggio naturale e il riconoscimento di immagini, TensorFlow rappresenta i modelli attraverso un grafo computazionale, offre flessibilità nell'implementazione e supporta GPU e TPU per accelerazione hardware.

Keras è una libreria di alto livello per la creazione di reti neurali, scritta in Python e integrata ufficialmente in TensorFlow dalla versione 2.0. Con un'API chiara e intuitiva, Keras è apprezzato per la sua semplicità di utilizzo, modularità, estensibilità e adattabilità a diversi backends di calcolo. La sua integrazione con TensorFlow ha portato a un ecosistema potente e facile da usare per la creazione di modelli neurali.

Capitolo 3

Descrizione del sistema

In questo capitolo verranno descritte le quattro parti principali del sistema di riconoscimento di dispositivi di protezione individuali indossati:

- Creazione del *dataset* idoneo per l'addestramento delle reti neurali
- *Face Detection* per identificare il volto delle persone
- *Addestramento* delle reti Neurali per il riconoscimento dei dispositivi
- Rilevamento tag rain rfid tramite Kathrein ARU 2400

3.1 Dataset

Il dataset impiegato per i nostri addestramenti è stato costituito attraverso la cattura di immagini di alcuni dipendenti dell'azienda TechCab, sia in abbigliamento ordinario che indossando i dispositivi di protezione individuale oggetto del riconoscimento. Complessivamente, sono state catturate 4000 immagini, suddivise in 8 cartelle corrispondenti ai 4 dispositivi di protezione da identificare: casco, pettorina catarifrangente, guanti da lavoro e scarpe antinfortunistiche. Per ciascun dispositivo, si è reso necessario acquisire fotografie che ritraessero l'oggetto sia indossato che non indossato. Subito dopo la fase iniziale di acquisizione, è stato attuato un processo di data augmentation,

conosciuto anche come "aumento dei dati". Questo procedimento consiste essenzialmente in una serie di tecniche volte ad espandere le risorse del dataset iniziale al di là della semplice raccolta di nuovi elementi. Il suo impiego è strettamente collegato all'apprendimento automatico delle reti neurali artificiali, che migliorano la loro capacità di apprendimento con l'aumento della dimensione del dataset di training disponibile. Partendo da un set iniziale di 500 immagini per ciascun scenario, siamo giunti a un totale di 1000 immagini destinate alla fase di addestramento, 100 per la fase di validazione e 50 per la fase di test per un totale di quasi 10000 scenari. Queste tre categorie di dati rappresentano le diverse tipologie necessarie per consentire a una rete di apprendere e calibrare le sue risposte in modo efficace. In questa fase, sono state effettuate operazioni di ritaglio sulle parti rilevanti delle immagini iniziali, seguite da un adeguato ridimensionamento in base alla tipologia del dispositivo da riconoscere. Si è proceduto inoltre a una variazione casuale del 5% di luminosità ed esposizione, insieme all'applicazione di rumore alle immagini. Tale rumore si manifesta attraverso una percentuale di pixel con colorazioni contrastanti, come il bianco, il nero o tonalità di grigio, allo scopo di simulare possibili impurità presenti sulla videocamera o nell'ambiente atmosferico in cui il nostro sistema sarà impiegato. Qui di seguito riporto alcuni esempi di quanto descritto fino ad ora.



Figura 3.1: In figura vengono rappresentate 4 scenari post data augmentation

3.2 Face Detection

L'inizio del sistema implica l'utilizzo della videocamera Realsense per monitorare la presenza di un volto all'interno del frame catturato dalla stessa videocamera. Posizionata verticalmente a due metri di altezza e inclinata di 30° rispetto all'asse Y, la videocamera offre una visione più estesa della persona situata a 1.40 metri di distanza. L'implementazione del modello di Caffe richiede l'importazione del file "deploy.prototxt.txt" che definisce l'architettura del modello e del file .caffemodel contenente i pesi relativi alla fase attuale di addestramento. I frame acquisiti dalla Realsense vengono elaborati attraverso funzioni di OpenCV per generare un'immagine adatta alle esigenze del modello di Caffe. Per individuare la posizione del volto riconosciuto, è necessario fornire al modello un blob, ovvero un array contenente informazioni sull'immagine acquisita. Infine, il modello restituisce quattro coordinate (x, y) del rettangolo circostante al volto identificato e una percentuale di confidenza che indica l'affidabilità del riconoscimento.

3.3 Addestramento reti neurali

Creato il dataset specifico per ciascun dispositivo di protezione, in conformità alle specifiche restituite dal codice QR al momento della timbratura tramite Cobotech, è necessario sviluppare un modello in grado di riconoscerli. Inizialmente, si era optato per l'addestramento di un unico modello capace di riconoscere diverse casistiche. Tuttavia, questo approccio è stato abbandonato in breve tempo a causa dei risultati poco soddisfacenti e della scarsa praticità di aggiornamento, specialmente considerando la possibilità di richiedere nuovi riconoscimenti in un futuro ipotetico. Su suggerimento del mio tutor, il Professor Jacopo Aleotti, si è deciso di adottare un approccio che prevede la distinzione dei vari casi e l'addestramento di modelli individuali per il riconoscimento delle casistiche in modo binario. In particolare, tali modelli devono essere in grado di identificare il dispositivo e valutarne se è correttamente indossato o se è assente. Per sviluppare tali modelli, è stata

impiegata la rete neurale EfficientNetB3, la quale precedentemente descritta nel capitolo 2.8. Nel prosieguo di questo capitolo, verrà fornita un'illustrazione dettagliata del processo di realizzazione. Per prima cosa è stato necessario importare le diverse librerie elencate in figura 3.2.

```
from tensorflow.keras import models
from tensorflow.keras import layers
from tensorflow.keras import optimizers
from tensorflow import keras
import os
import glob
import shutil
import sys
import numpy as np
import matplotlib.pyplot as plt
from IPython.display import Image
```

Figura 3.2: Elenco librerie importate.

Successivamente scegliamo quale rete neurale importare, nel nostro caso sono stati riscontrati risultati ottimali con EfficientNetB3

```
from tensorflow.python.keras.applications.efficientnet import EfficientNetB3
```

Figura 3.3: Rete neurale scelta per l'addestramento.

Dopo aver importato le librerie e definito la rete, è stata elaborata la funzione principale per l'addestramento dei modelli. In questa funzione, i parametri passati includono le dimensioni in pixel delle immagini e il nome che vogliamo assegnare al modello finale. Il primo passo implica la definizione dei parametri passati e parallelamente la specifica degli indirizzi delle cartelle che contengono i nostri dataset 3.1.

```
def allenamento(Width, Height, name, epochs):  
    #define input height and width  
    width = Width #300  
    height = Height #300  
    input_shape = (height, width, 3)  
    nome= name  
    conv_base = EfficientNetB3(weights='imagenet', include_top=False, input_shape=input_shape)  
  
    train_dir = './efficientnet_keras_transfer_learning-master/guanti/train/'  
    valid_dir = './efficientnet_keras_transfer_learning-master/guanti/valid/'  
    test_dir = './efficientnet_keras_transfer_learning-master/guanti/test/'  
    batch_size = 4
```

Figura 3.4: Codice di caricamento del dataset nella funzione principale

Ora andremo a configurare un generatore di immagini per l'addestramento e la validazione del nostro modello di rete neurale. Viene utilizzata una classe fornita dalla libreria Keras, *ImageDataGenerator(rescale=1./255)*, per generare batch di dati per l'addestramento di modelli di reti neurali per il riconoscimento delle immagini, il parametro contenuto al suo interno specifica una trasformazione di riscalamento delle immagini che originariamente sono costituite da pixel con valori compresi tra 0 e 255 (0 per il nero e 255 per il bianco), la trasformazione divide tutti i valori dei pixel per 255, portandoli su una scala compresa tra 0 e 1. Questo è un passo importante poiché le reti neurali tendono ad avere prestazioni migliori quando le variabili sono normalizzate in un intervallo più piccolo. Quindi, *ImageDataGenerator(rescale=1./255)* crea un generatore di immagini che normalizza i valori dei pixel dividendo ogni valore per 255, rendendo le immagini pronte per essere utilizzate nell'addestramento di una rete neurale convoluzionale.

```
from tensorflow.keras.preprocessing.image import ImageDataGenerator

train_datagen = ImageDataGenerator(
    rescale=1./255)

# Note that the validation data should not be augmented!
test_datagen = ImageDataGenerator(rescale=1./255)

train_generator = train_datagen.flow_from_directory(
    # This is the target directory
    train_dir,
    # All images will be resized to target height and width.
    target_size=(height, width),
    batch_size=batch_size,
    # Since we use categorical_crossentropy loss, we need categorical labels
    class_mode='categorical')

validation_generator = test_datagen.flow_from_directory(
    valid_dir,
    target_size=(height, width),
    batch_size=batch_size,
    class_mode='categorical')
train_generator.class_indices
```

Figura 3.5: Codice di preparazione ed elaborazione del dataset

- *train_generator* *validation_generator* Questi sono generatori di batch di dati che fluiscono direttamente dalle directory di addestramento e validazione.
- *flow_from_directory* è un metodo che semplifica notevolmente la generazione dei dati per l'addestramento dei modelli. I parametri specificati includono: il percorso della directory (*train_dir* e *valid_dir*), le dimensioni target delle immagini (*target_size*), la dimensione del batch (*batch_size*), e il tipo di classe (*class_mode*) che è impostato su 'categorical' poiché si sta utilizzando una loss di tipo *categorical_crossentropy*.

A questo punto procederemo a definire la struttura del nostro modello che vediamo raffigurata nell'immagine 3.6. Utilizzando Keras 2.9 viene definito un modello di tipo sequenziale composto da diversi strati, tra cui convoluzionali, di max pooling e dropout completamente connessi.

L'ultimo strato `model.add(layers.Dense(num_classes, activation='softmax'))` utilizza la funzione di attivazione *softmax* poiché si tratta di un problema di classificazione multi-classe. Sostanzialmente, il codice in questione sta assemblando un nuovo modello che incorpora quello di EfficientNetB3 pre-addestrato come parte del suo input, aggiungendo strati personalizzati in cima per adattarsi al problema specifico in questione.

```
NUM_TRAIN = sum([len(files) for r, d, files in os.walk(train_dir)])
NUM_TEST = sum([len(files) for r, d, files in os.walk(valid_dir)])
dropout_rate = 0.2

num_classes = len(os.listdir(train_dir))
print('building network for ' + str(num_classes) + ' classes')

#struttura
model = models.Sequential()
model.add(layers.Conv2D(256, kernel_size=(3, 3),
                        activation='relu',
                        input_shape=input_shape))
model.add(layers.MaxPooling2D(pool_size=(2, 2)))
model.add(layers.Conv2D(256, (3, 3), activation='relu'))
model.add(layers.MaxPooling2D(pool_size=(2, 2)))
model.add(layers.Dropout(0.25))
model.add(layers.Flatten())
model.add(layers.Dense(128, activation='relu'))
model.add(layers.Dropout(0.5))
model.add(layers.Dense(64, activation='relu'))
model.add(layers.Dropout(0.5))
model.add(layers.Dense(num_classes, activation='softmax'))

model.summary()

print('This is the number of trainable layers '
      'before freezing the conv base:', len(model.trainable_weights))

conv_base.trainable = False

print('This is the number of trainable layers '
      'after freezing the conv base:', len(model.trainable_weights))
```

Figura 3.6: Codice di specifica della struttura del modello

- `model.summary()` questo comando stampa un riepilogo del modello, che include le dimensioni di output di ciascun layer e il numero totale di parametri addestrabili.
- `conv_base.trainable = False` imposta il modello EfficientNetB3 3.3 pre-addestrato come non addestrabile. In altre parole, i pesi di questo modello non verranno aggiornati durante l'addestramento del modello complessivo. Questo è un passo comune nel trasferimento di apprendimento per mantenere le informazioni apprese dal modello pre-addestrato.
- attraverso la funzione `len(model.trainable_weights)` viene evidenziato e stampato a video la quantità di parametri che il modello ha cercato di adattare, questo è utile per verificare che il processo sia avvenuto con successo.

Una volta definito il modello, questo può essere addestrato come qualsiasi altro modello di classificazione standard utilizzando `model.compile()` e `model.fit()`.

```
#compilare il modello (nuova struttura)
model.compile(loss=keras.losses.categorical_crossentropy,
              optimizer=keras.optimizers.Adadelta(),
              metrics=['acc'])

# allenare il modello
history = model.fit(
    train_generator,
    steps_per_epoch= NUM_TRAIN //batch_size,
    epochs=epochs,
    validation_data=validation_generator,
    validation_steps= NUM_TEST //batch_size,
    verbose=1,
    use_multiprocessing=False,
    workers=1)
```

Figura 3.7: Compilazione e avvio dell'allenamento del modello

3.4 Rivelamento tag rain RFID

Come precedentemente illustrato nell'introduzione di questo documento, oltre alla modalità di rilevamento basata sull'object detection, il sistema si avvale dell'utilizzo di un lettore RFID, in particolare il modello Kathrein ARU 2400 [2.3]. Tale dispositivo consente di leggere i tag RFID UHF presenti su ciascun dispositivo, ognuno dei quali è dotato di codici univoci. Questi codici vengono registrati in corrispondenza di ciascun individuo, facilitando la verifica dell'avvenuta lettura e garantendo l'identificazione accurata. In seguito, all'interno di questo paragrafo, sarà presentata un'analisi dettagliata dell'implementazione del codice che sfrutta il lettore sopra citato insieme alla comunicazione con un database SQL per gestire in modo efficace la rilevazione e l'autenticazione dei tag. Prima di procedere con l'esposizione del codice, è opportuno delineare la struttura del database in questione. Quest'ultimo è composto da tre tabelle che riguardano rispettivamente il ruolo e il codice che lo identifica, i dispositivi necessari per ciascun ruolo specifico e i codici RFID univoci associati a ciascun tag presente sui dispositivi.

	Tipo_Ruolo	Helmet	Mask	Glove	Shoes	Reflective	Riserva1	Riserva2	Riserva3	Riserva4	Riserva5	Riserva6	Riserva7	Riserva8	Riserva9	Riserva10
1	Elettricista	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0
2	Full	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0
3	Programmatore	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0
4	prova	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0
5	Prova1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0

Figura 3.8: Struttura tabella dei ruoli

	Tipo_DPI	Code_DPI	Stato	Code_USER
1	Shoes	0000058200010TLA	1	130499
2	Shoes	0000158200010TLA	1	130499
3	Helmet	0000458200010TLA	1	130499
4	Gloves	0000648200010TLA	1	130499
5	Gloves	0000748200010TLA	1	130499
6	Mask	0000948200010TLA	1	130499
7	Reflective	0000962300010TLA	1	130499

Figura 3.9: Struttura tabella dei codici univoci

Avendo delineato la configurazione del database, procederò ora con l'illustrazione del codice dedicato alla gestione del lettore. Per prima cosa è stato necessario importare le diverse librerie elencate in figura.

```
import socket
import sys
import binascii
import pyodbc
import os
```

Figura 3.10: Elenco librerie importate.

Il codice è formato da una classe *Antenna()* contenente sei funzioni che tratterò nello specifico.

```
class Antenna():
    def __init__(self):...

    # Resetta tutte le variabili
    def clear():...

    #ritorna una lista prendendo i tag liberi
    def get(self, DPI, stato):...

    #aggiorna i valori nella tabella
    def set(self, DPI):...

    #estrae il ruolo associato al badge e setta i booleani dei dpi richiesti
    def setDPIforROLE(self):...

    #funzione che va a leggere i tag e consente o nega l'accesso
    def detection(self, cod_operaio):...
```

Figura 3.11: Struttura del codice

- *__init__(self)* questa funzione inizializza tutte le variabili e le liste locali necessarie per il proseguo del codice. In aggiunta, gestisce la connessione al server del database attraverso una socket e attiva il lettore RFID eseguendo le operazioni iniziali di configurazione.

```
#liste di appoggio già entrati
self.helmet_list_dentro = list()
self.mask_list_dentro = list()
self.reflective_list_dentro = list()
self.gloves_list_dentro = list()
self.shoes_list_dentro = list()

#variabili d'appoggio dpi letti
self.bytes_helmet = ""
self.bytes_mask = ""
self.bytes_reflective = ""
self.bytes_gloves_1 = ""
self.bytes_gloves_2 = ""
self.bytes_shoes_1 = ""
self.bytes_shoes_2 = ""
```

Figura 3.12: Codice di alcune delle variabili e liste locali

```
#database
self.database = ('Driver={SQL server};'
                 'Server=LAPTOP-NB064SD0\SQLEXPRESS;'
                 'Database=dbtechcab_muroni;'
                 'Trusted_Connection=yes;')

#parametri e connessione antenna
self.IPV4 = "192.168.1.68"
self.port = 4007

try:
    self.s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)# creazione socket client
    print(f"socket: {self.s}")

    self.host = socket.getaddrinfo(self.IPV4, self.port)
    print(self.host)

    self.s.connect(self.host[0][4]) #connessione al server
    print(f"Connessione al Server: {self.IPV4} effettuata.")
    print(f"Socket: {self.s}")

except socket.error as errore:
    print(f"Qualcosa è andato storto, riprovo... \n{errore}")
    connection(self.indirizzo, self.port)
```

Figura 3.13: Codice di collegamento al database SQL

```
#comandi da inserire prima e dopo il messaggio
self.Start = "5A50"
self.Stop = "A5"

#comandi completi
self.ping = self.Start + "03000000" + "5A5FA5" + self.Stop
self.ASyncGetEPCs = self.Start + "02000000" + "1101" + self.Stop

#settaggio potenza di lettura
self.NumAntenna = "01"
self.PotenzaAntenna = "57"
self.SetPortPower = self.Start + "04000000" + "0600" + self.NumAntenna + self.PotenzaAntenna + self.Stop
self.s.send(bytes.fromhex(self.SetPortPower))
```

Figura 3.14: Codice per il collegamento e configurazione iniziale del lettore

- *clear()* questa funzione ha lo scopo di ripristinare tutte le variabili locali alla loro configurazione iniziale, un'operazione eseguita al termine della scansione di un soggetto in preparazione al successivo.
- *get()* questa funzione stabilisce una connessione al database e successivamente esegue una query per recuperare la lista dei tag univoci associati ai dispositivi considerando il flag *stato* specificato per indicare se i DPI sono già entrati o meno.

```
def get(self, DPI, stato):
    DPI_list = []
    conn = pyodbc.connect(self.database)
    cursor = conn.cursor()
    cursor.execute(f"select Code_DPI from DPI where Tipo_DPI = '{DPI}' and Stato = '{stato}'")
    for row in cursor:
        DPI_list.append(row[0].strip())
```

Figura 3.15: Codice per ricevere la lista dei DPI già entrati o meno

- *set(self, DPI)* questa funzione aggiorna i valori nella tabella 3.9 del database relativa agli abbinamenti dei codici univoci dei dispositivi di protezione e quelli degli utenti. Viene stabilita una connessione al database e successivamente esegue una query SQL dove comunica i dati letti per aggiornare questi due valori appena citati in tabella. Infine, i cambiamenti vengono confermati con un'operazione di *commit()* sulla connessione al database.

```
def set(self, DPI):
    conn = pyodbc.connect(self.database)
    cursor = conn.cursor()
    cursor.execute(f"UPDATE DPI SET Stato = 1, Code_USER = '{self.cod_operai}' where Tipo_DPI = '{DPI}'")
    conn.commit()
```

Figura 3.16: Codice per aggiornare la tabella 3.9

- *setDPIforROLE(self)* questa funzione stabilisce una connessione al database, dove successivamente esegue una query per recuperare le informazioni relative ai dispositivi di protezione individuali associati al ruolo dell'utente. I booleani corrispondenti ai DPI richiesti vengono impostati a True e vengono inoltre ottenute le liste associate a ciascun DPI sia per quelli esterni che interni. Le liste vengono ottenute utilizzando la funzione *Antenna.get*. In breve, questa funzione si occupa di adattare dinamicamente le impostazioni dei DPI in base al ruolo associato al badge dell'utente nel contesto del sistema di monitoraggio.

```
def setDPIforROLE(self):
    conn = pyodbc.connect(self.database)
    cursor = conn.cursor()
    cursor.execute(f"SELECT DPI_Ruolo.Helmet, DPI_Ruolo.Mask, DPI_Ruolo.Glove, DPI_Ruolo")
    for row in cursor:

        if row[0]:
            self.helmet = True
            self.helmet_list = Antenna.get(self, "Helmet", 0)
            self.helmet_list_dentro = Antenna.get(self, "Helmet", 1)

        if row[1]:
            self.mask = True
            self.mask_list = Antenna.get(self, "Mask", 0)
            self.mask_list_dentro = Antenna.get(self, "Mask", 1)
```

Figura 3.17: Parte del codice della funzione che si occupa di impostare dinamicamente i flag in base al ruolo

- *detection(self, cod_operai)* questa è la funzione principale che gestisce il processo di rilevamento dei tag dai DPI attraverso il lettore, è strutturata in modo dettagliato per gestire in modo efficace il rilevamento, lo stato e l'aggiornamneto delle informazione nel database.

Capitolo 4

Risultati sperimentali

In questo capitolo saranno presentati i risultati derivati dallo sviluppo delle reti neurali descritte precedentemente. Innanzitutto, presento i grafici ottenuti durante l'addestramento dei modelli, concentrandomi specificamente su quelli relativi al riconoscimento del casco, della pettorina catarifrangente e delle scarpe antinfortunistiche. Questi grafici rivestono una particolare importanza nell'identificazione che il modello non abbia subito l'overfitting, un comportamento indesiderato nel machine learning che si manifesta quando il modello fornisce previsioni accurate sui dati di addestramento ma non su casistiche future su cui dovrà lavorare.

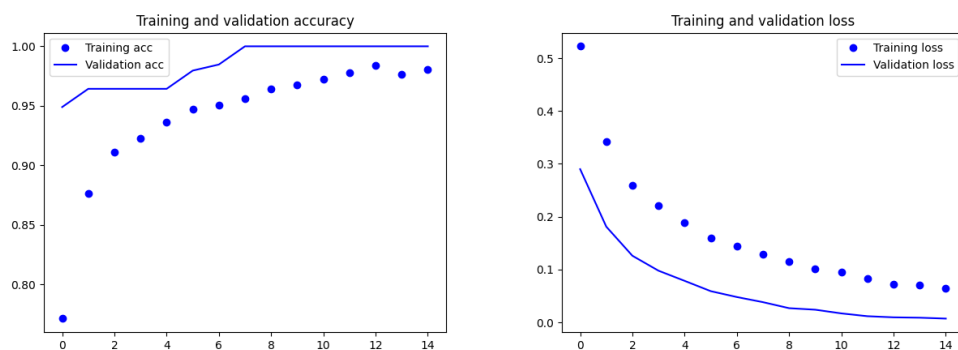


Figura 4.1: Grafici del modello per il casco

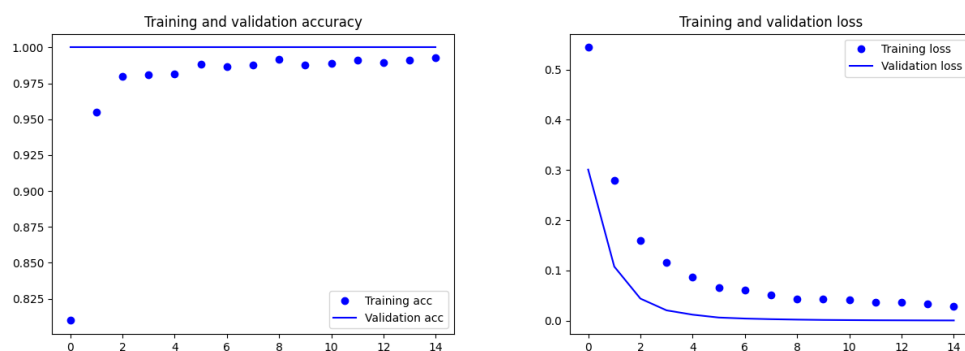


Figura 4.2: Grafici del modello per la pettorina

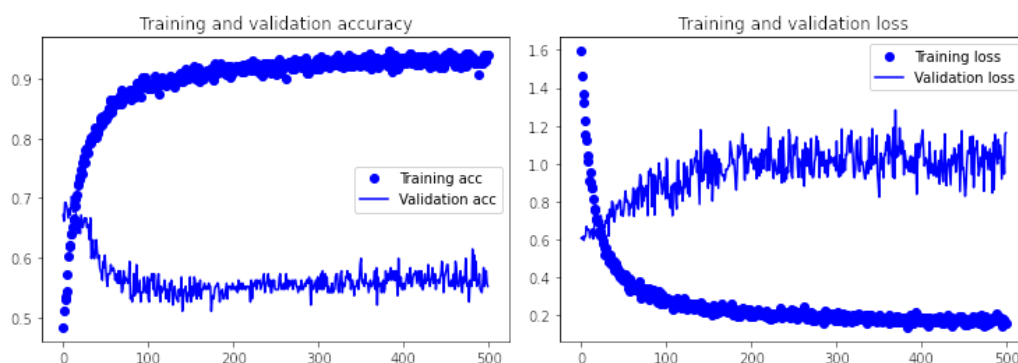


Figura 4.3: Grafici del modello per le scarpe

L'efficacia dei modelli è stata valutata in base alla precisione delle identificazioni, considerando la percentuale di risultati corretti durante la fase di test e la capacità di lavorare in tempo reale durante la scansione. Nella fase iniziale, definita come statica, i modelli sono stati testati su un dataset specifico per ogni dispositivo composto da 1000-1400 immagini. È stata impostata una soglia di affidabilità pari a 0.8, significa che durante ciascuna scansione se la percentuale di riconoscimento è inferiore il dispositivo non viene riconosciuto. Al contrario, se la percentuale è compresa tra 0.8 e 1, il dispositivo viene riconosciuto. Nel caso in cui la scansione indichi l'assenza

del dispositivo qualsiasi livello di affidabilità è accettato. Per i modelli dedicati al riconoscimento del casco, della pettorina e delle scarpe da lavoro, dopo alcune regolazioni nella fase di allenamento, è stata raggiunta un'affidabilità media compresa tra 0.86 e 0.99. Il modello che inizialmente sembrava avere prestazioni inferiori è quello della pettorina catarifrangente. Tuttavia, un'analisi più approfondita dei risultati ha rivelato che il valore medio di affidabilità per il riconoscimento risulta penalizzato per via dei casi in cui il dispositivo non è presente. Come già anticipato, questa soglia non è rilevante nel contesto del nostro progetto, poiché nel momento in cui viene identificata questa casistica, qualsiasi valore viene accettato per garantire la massima sicurezza. Qui di seguito verranno riportati i risultati dei test di questi tre modelli.

effettivo	rilevato	percentuale			accuratezza media	numero di immagini	risultati esatti	risultati sbagliati
casco	casco	0,986411572	Casco_105.jpg	VERO	0,92551903	1000	998	2
casco	casco	0,988638222	Casco_106.jpg	VERO				
casco	casco	0,982682228	Casco_112.jpg	VERO				
casco	casco	0,994360745	Casco_113.jpg	VERO				
casco	casco	0,995787442	Casco_114 - Copia.jpg	VERO				
casco	casco	0,995787442	Casco_114.jpg	VERO				
casco	casco	0,996370554	Casco_115 - Copia.jpg	VERO				
casco	casco	0,996370554	Casco_115.jpg	VERO				

Figura 4.4: Risultati dei test di rilevamento del casco

effettivo	rilevato	percentuale			accuratezza media	numero di immagini	risultati esatti	risultati sbagliati
normali	normali	0,688832879	NoPettorina_1.jpg	VERO	0,86702884	1100	1067	34
normali	normali	0,95012778	NoPettorina_10.jpg	VERO				
normali	normali	0,746309459	NoPettorina_100.jpg	VERO				
normali	normali	0,754918933	NoPettorina_100.jpg.rf.e9153b3494c6410bt	VERO				
normali	normali	0,949204922	NoPettorina_101.jpg	VERO				
normali	normali	0,72294867	NoPettorina_102.jpg	VERO				
normali	normali	0,671476126	NoPettorina_103.jpg	VERO				

Figura 4.5: Risultati dei test di rilevamento della pettorina

Scarpe effettive indossate nella foto	Scarpe rilevate nella foto	Percentuale	Risultato	accuratezza media	numero di immagini	risultati esatti	risultati sbagliati
normali	normali	0,999984622	VERO	0,996081819	1408	1402	6
normali	normali	0,999994874	VERO				
normali	normali	1	VERO				
normali	normali	1	VERO				
normali	normali	1	VERO				
normali	normali	1	VERO				
normali	normali	1	VERO				

Figura 4.6: Risultati dei test di rilevamento delle scarpe

I risultati ottenuti durante la fase di test hanno mostrato elevate percentuali di affidabilità. Di conseguenza, nel contesto del progetto finale, in cui tali modelli sono impiegati, la soglia di riconoscimento è stata aumentata a 0.9. Per quanto riguarda il modello progettato per il rilevamento dei guanti la situazione cambia poiché durante le fasi dei test statici sono state raggiunte buone percentuali di rilevamento, tuttavia durante la fase dinamica in cui è richiesta la rilevazione in tempo reale tali risultati non sono stati sufficienti per il riconoscimento di tali dispositivi. Al momento dello studio di questa ricerca non era possibile modificare la struttura del robot quindi non è stato possibile optare per l'aggiunta di una videocamera più vicina che potesse inquadrare le mani con maggiore risoluzione e dettaglio pertanto, lo sviluppo di questo modello è stato rimandato per una versione successiva. Per quanto concerne i tre modelli menzionati in precedenza, i test dinamici hanno evidenziato eccellenti risultati in termini di precisione e tempestività nel riconoscere la presenza o l'assenza dei dispositivi. Di conseguenza, essi sono stati integrati con successo nel software finale del progetto. Nel prosieguo di questo capitolo, a scopo illustrativo, saranno presentati esempi di riconoscimento partendo da quelli ottenuti durante lo studio e l'addestramento dei modelli fino ad arrivare ai riconoscimenti dei modelli finali precedentemente menzionati.



Figura 4.7: Prima fase in cui il modello aveva il compito di riconoscere ogni dispositivo, abbandonata per gli scarsi risultati ottenuti.



Figura 4.8: Risultati dei primi test delle reti neurali adoperando modelli allenati al riconoscimento di un singolo dispositivo o alla sua assenza



Figura 4.9: Risultati dei test sui modelli ultimati che sono stati implementati nel codice finale.

Capitolo 5

Conclusioni e Sviluppi futuri

L'elaborazione di questo progetto ha richiesto un impegno considerevole in termini di tempo e studio tuttavia sono certo nel fatto che gli obiettivi posti da Techcab S.R.L. siano stati pienamente raggiunti. Il sistema sviluppato può essere impiegato come sistema di controllo dei requisiti per l'accesso a diverse aree o sezioni specifiche. Naturalmente, ci sono spazi per miglioramenti, come nell'ottimizzazione della fase di acquisizione del dataset e delle immagini. Ad esempio, è possibile considerare l'eliminazione dello sfondo sfruttando appieno la tecnologia della Realsense. Inoltre, è possibile raffinare la procedura di addestramento dei modelli per ottenere risultati ancora più accurati. Un aspetto che ha contribuito significativamente è la suddivisione del compito di rilevare ciascun dispositivo in modelli distinti. Questo approccio ha comportato miglioramenti sia nella fase di addestramento che sulle prestazioni complessive del software finale. Tale ottimizzazione consente l'utilizzo del software su computer con prestazioni medie, rendendolo versatile e accessibile.

Bibliografia

- [1] Redazione osservatori digital innovation. Storia dell'intelligenza artificiale: da turing ai giorni nostri, 2023.
- [2] intelligenzaartificiale.it. Machine learning.
- [3] Deep learning: quando il cervello umano diventa un modello di riferimento, 2020.
- [4] Zhenwei Shi Member IEEE Yuhong Guo Zhengxia Zou, Keyan Chen and IEEE Jieping Ye, Fellow. Object detection in 20 years: A survey, 2023.
- [5] Paolo Poli François Chollet. *Deep Learning con Python*. Apogeo, 2020.
- [6] Alberto Fernandez Villan. *Mastering OpenCV 4 with Python*. Packt Publishing Ltd., 2019.
- [7] Keras.
- [8] Tensorflow.
- [9] Colab.