POLITECNICO DI TORINO

Laurea Magistrale in Ingegneria Informatica

Tesi Magistrale

Tecniche di Pattern Recognition per il riconoscimento e la classificazione di interruttori elettrici

Uno studio sperimentale in ambito enterprise per la messa in sicurezza degli stabilimenti

Relatore:
Santa Di Cataldo

Candidato:
Vincenzo Lacovara

Dicembre 2019

Tecniche di Pattern Recognition per il riconoscimento e la classificazione di interruttori elettrici

Uno studio sperimentale in ambito enterprise per la messa in sicurezza degli stabilimenti

Vincenzo Lacovara

Relatore

Santa Di Cataldo

Dipartimento di Automatica e Informatica

La sicurezza sul posto di lavoro è una delle più importanti conquiste dei lavoratori del dopoguerra e, oltre ad essere un diritto del lavoratore, è un interesse per la società nel suo insieme, che così riduce i costi sociali per infortuni e malattie professionali.
Questo studio propone delle tecniche di pattern recognition per il riconoscimento dello stato di interruttori elettrici con lo scopo di riconoscere se un interruttore è stato messo in sicurezza. Esso nasce dall'esigenza di un'importante azienda italiana leader nel settore energetico di fornire un supporto automatico alla messa in sicurezza dei propri impianti elettrici durante le operazioni di manutenzione.
A tale scopo sono stati progettati e sviluppati dei modelli di Deep Learning in grado, data l'immagine, di riconoscere lo stato dell'interruttore elettrico e di verificarne la corretta messa in sicurezza. Sono state utilizzate reti neurali convoluzionali, le quali sono particolarmente adatte nell'ambito del riconoscimento di immagine. Per integrare il riconoscimento dello stato dell'interruttore nel processo di messa in sicurezza dell'impianto del cliente sono stati sviluppati dei microservizi, i quali ricevono l'immagine da classificare e interrogano le reti neurali per ottenere la predizione.
Si è deciso di utilizzare reti pre-addestrate in modo da far fronte alle ridotte dimensioni del dataset a disposizione. In particolare, sono state sperimentate diverse reti pre-addestrate, come AlexNet, VGGnet16 e Resnet18 e sono stati confrontati i risultati ottenuti. Per evitare il fenomeno dell'overfitting, sono state applicate diverse tecniche di data augmentation.
Le reti neurali e i microservizi sono stati implementati con successo. I risultati hanno mostrato che la rete più adatta per questo specifico task e che ha fornito le prestazioni migliori è stata la rete pre-addestrata Resnet18 con un'accuratezza media del 99.5%.
Grazie agli ottimi risultati ottenuti, il progetto, nato come sperimentazione, è stato approvato dal cliente ed è ora operativo in una delle sue principali centrali elettriche.

Indice

1 Introduzione **12**

2 Background **15**
2.1 Deep learning 16
2.1.1 Feedforwad neural network 18
2.1.2 Reti neurali convoluzionali 22
2.1.3 Tecniche di regolarizzazione 27
2.1.4 Transfer Learning 29

3 Stato dell'Arte **33**

4 Metodologia **37**
4.1 Scelta tipologie di interruttori e tagging 40
4.2 Creazione del dataset 42
4.3 Training e valutazione dei modelli 43
4.4 Integrazione con l'applicazione esistente 43
4.5 Test e rilascio in produzione 44

5 Sviluppo del Lavoro **46**
5.1 Descrizione delle quattro tipologie 46
5.1.1 Tipologia window-panel 46
5.1.2 Tipologia knob 49
5.1.3 Tipologia switch 50
5.1.4 Tipologia lever 52
5.2 QR code Reader e crop automatico 54
5.3 Reti neurali 58
5.3.1 Tipologia window-panel 60
5.3.2 Tipologia knob 63
5.3.3 Tipologia switch 63
5.3.4 Tipologia lever 64
5.4 Microservizi 66

6 Risultati sperimentali **71**
6.1 Risultati Tipologia window-panel 74
6.2 Risultati Tipologia knob 82
6.3 Risultati Tipologia switch 86
6.4 Risultati Tipologia lever 90

7 Conclusioni **97**

Elenco delle figure

2.1 Esempio di modello di Deep Learning [4]. Il primo layer è detto "visibile" perchè contiene le variabili che possono essere osservate direttamente (i pixel nel caso delle immagini). A seguire ci sono una serie di hidden layer che estraggono progressivamente le features dall'immagine. Questi layer sono detti "hidden" perchè il loro valore non è fornito dai dati. Dai pixel iniziali, il primo hidden layer è in grado di identificare facilmente i bordi mentre il secondo hidden layer riesce, a partire dai bordi, ad estrapolare gli angoli e i contorni. Il terzo layer è in grado di individuare intere parti degli oggetti presenti nell'immagine, a partire dagli angoli e dai bordi. Il layer finale viene infine utilizzato per riconoscere gli oggetti presenti nell'immagine. . . 17
2.2 Perceptron . . . 18
2.3 La funzione sigmoide è così definita: $F(x) = \frac{1}{1+e^{-t}}$ . . . 19
2.4 Rete neurale Feedforward . . . 19
2.5 Training di una rete neurale . . . 20
2.6 Gradient descent . . . 20
2.7 Estrazione di una feature map . . . 22
2.8 Connettività sparsa[4]: Nella figura sono evidenziate l'unità di input x_3 e l'unità di output s che sono interessate da x_3. (In alto) Quando s è generato con una convoluzione con un kernel di dimensione 3, solo tre output sono interessati da x_3. (In basso) Quando s è generato da una moltiplicazione matriciale, la connettività non è più sparsa e quindi tutti gli output sono interessati da x_3. . . . 23
2.9 Composizione di una rete neurale convoluzionale . . . 24
2.10 Operazione di convoluzione con un filtro 3x3. Il filtro viene fatto scorrere sull'immagine e viene calcolato il prodotto scalare. L'output è chiamato feature map ed è più piccolo dell'immagine principale. . . 24
2.11 Esempio di input 7x7 e di filtro convoluzionale 3x3 con stride 2: l'activation map risultante sarà 3x3. Infatti $\frac{7-3+0}{2} + 1 = 3$. . . . 25
2.12 A sinistra la funzione di attivazione ReLu. A destra la Leaky ReLu dove il parametro a indica la pendenza della funzione. Un tipico valore di a è 0.01 . . . 26
2.13 Max pooling e average pooling . . . 27
2.14 Dropout . . . 28
2.15 Skip connection . . . 31

3.1 Confronto tra diversi modelli di CNN. [18] . . . 34

4.1 Metodologia utilizzata . . . 37

4.2 Fasi del processo di manutenzione AS-IS 38
4.3 Fasi del processo di manutenzione TO-BE 38
4.4 Soluzione tecnologica. **(1)** L'operatore invia l'immagine dell'interruttore ad un microservizio che effettua delle operazioni di preprocessing sull'immagine. **(2)** L'immagine elaborata viene inviata ad un altro microservizio che interroga i modelli di rete neurali per ottenere una predizione sullo stato dell'interruttore. **(3)** Il microservizio fornisce la predizione della rete neurale all'applicazione dell'operatore che la visualizza sullo schermo. **(4)** L'operatore sottomette l'operazione di sezionamento con le evidenze fotografiche della messa in sicurezza dell'interruttore e la predizione dell'intelligenza artificale. Il backend del sistema elabora e salva la richiesta dell'operatore. **(5)** Il capo turno, tramite un browser, accede alla web application e verifica che l'operazione di sezionamento sia avvenuta correttamente. **(6)** Se tutto è andato a buon fine il capo turno autorizza la richiesta. **(7)** Le immagini sottomesse dall'operatore vengono salvate in modo da poter capire cosa è successo in caso di errore e per usi futuri. **(8)** Le immagini salvate possono essere utilizzate per riaddestrare le reti neurali in modo da raffozzarne le performance. 39
4.5 Esempio di Tipologia *window-panel* 40
4.6 Esempio di Tipologia *knob* 40
4.7 Esempio di Tipologia *switch* 41
4.8 Esempio di Tipologia *lever* 41

5.1 finestra della Tipologia window-panel - On 46
5.2 finestra della Tipologia window-panel - Off 47
5.3 Pannello della Tipologia window-panel - Inserito 47
5.4 Pannello della Tipologia window-panel - Estratto 48
5.5 Pannello della Tipologia window-panel - Messo a terra 48
5.6 Tipologia knob - On 49
5.7 Tipologia knob - Off 50
5.8 Tipologia knob - Estratto 50
5.9 Tipologia switch - On 51
5.10 Tipologia switch - Off 51
5.11 Tipologia switch - Estratto 52
5.12 Tipologia lever - On 52
5.13 Tipologia lever - Off 53
5.14 Tipologia lever - Estratto 53
5.15 Esempio di crop dell' immagine. L'id di questo interruttore è 11-MME-006B-1. Il QR code in alto a sinistra codificherà il testo 11-MME-006B-1T, mentre quello in basso a destra codificherà il testo 11-MME-006B-1B. In questo esempio il microservizio rileverebbe tre qr code, ma solo due di questi codificano l'identificativo dell'interruttore e quindi è in gradio di ritagliare l'immagine correttamente. 55
5.16 Immagine ruotata di 90°. Il microservizio riconosce che l'immagine è ruotata di 90°perchè il QR code Top si trova alla destra di quello Bottom. 55

5.17 Immagine ruotata di 270°. Il microservizio riconosce che l'immagine è ruotata di 270°perchè il QR code Top e più in basso di quello Bottom. 55
5.18 Procedura di crop dell'immagine 56
5.19 Cross Entropy Loss[21] 59
5.20 Object detection per individuare e croppare l'area di interesse per il riconoscimento dello stato del pannello Tipologia window-panel . . . 61
5.21 Object detection per individuare e croppare l'area di interesse per il riconoscimento dello stato della finestra di Tipologia 1 window-panel . 61
5.22 Crop del pannello dell'interruttore di Tipologia lever 64
5.23 Flusso logico per determinare lo stato dell'interruttore di Tipologia lever 64
5.24 Flusso logico dell'API composta dai moduli *switch-recognition-engine* e *switch-recognition-classifier* 69

6.1 Esempio di curva recall - precision. [26] 72
6.2 Formula per il calcolo della intersection over union. [27] 73
6.4 Confusion Matrix delle CNN della finestra di Tipologia window-panel 75
6.6 Test accuracy durante il training delle CNN della finestra di Tipologia window-panel 76
6.8 Loss durante il training delle CNN della finestra di Tipologia window-panel 76
6.10 Confusion Matrix delle CNN del pannello dell'interruttore di Tipologia window-panel 79
6.12 Test accuracy durante il training delle CNN del pannello dell'interruttore di Tipologia window-panel 80
6.14 Loss durante il training delle CNN del pannello dell'interruttore di Tipologia window-panel 80
6.16 Confusion Matrix delle CNN di Tipologia knob 83
6.18 Test accuracy durante il training delle CNN di Tipologia knob 84
6.20 Loss durante il training delle CNN di Tipologia knob 84
6.22 Confusion Matrix delle CNN di Tipologia switch 87
6.24 Test accuracy durante il training delle CNN di Tipologia switch . . . 88
6.26 Loss durante il training delle CNN di Tipologia switch 88
6.28 Confusion Matrix delle CNN di Tipologia lever 91
6.30 Test accuracy durante il training delle CNN di Tipologia lever 92
6.32 Loss durante il training delle CNN di Tipologia lever 92
6.34 Confusion Matrix delle CNN di Tipologia lever estratto/non estratto 94
6.36 Test accuracy durante il training delle CNN di Tipologia lever estratto/non estratto 95
6.38 Loss durante il training delle CNN di Tipologia lever estratto/non estratto 95

Elenco delle tabelle

4.1 Numero totale di immagini del dataset per ogni tipologia di interruttore 43

5.1 Numero totale di immagini del dataset per ogni stato della finestra dell' interruttore di Tipologia window-panel 62
5.2 Numero totale di immagini del dataset per ogni stato del pannello dell' interruttore di Tipologia window-panel 62
5.3 Numero totale di immagini del dataset per ogni stato dell' interruttore di Tipologia knob 63
5.4 Numero totale di immagini del dataset per ogni stato dell' interruttore di Tipologia switch 63
5.5 Numero totale di immagini del dataset per ogni stato dell' interruttore di Tipologia lever 65
5.6 Numero totale di immagini del dataset per ogni stato del pannello dell' interruttore di Tipologia lever 65
5.7 Campi della request per la chiamata al microservizio del modulo *switch-recognition-engine* 66
5.8 Campi della response del microservizio del modulo *switch-recognition-engine* in caso esito positivo 67
5.9 Campi della response del microservizio del modulo *switch-recognition-engine* in caso esito negativo 67
5.10 Possibili messaggi di errore nella response del microservizio del modulo *switch-recognition-engine* 68

6.1 Training set e Test set della finestra dell'interruttore di Tipologia window-panel 74
6.2 Risultati finestra di Tipologia window-panel 74
6.3 Training set e Test set del pannello dell'interruttore di Tipologia window-panel 78
6.4 Risultati del pannello dell'interruttore di Tipologia window-panel . . 78
6.5 Training set e Test set dell'interruttore di Tipologia knob 82
6.6 Risultati Tipologia knob 82
6.7 Training set e Test set dell'interruttore di Tipologia switch 86
6.8 Risultati Tipologia switch 86
6.9 Training set e Test set dell'interruttore di Tipologia lever 90
6.10 Risultati Tipologia lever On/Off 90
6.11 Training set e Test set dell'interruttore di Tipologia lever estratto/non estratto 93
6.12 Risultati Tipologia lever estratto/non estratto 93

Capitolo 1

Introduzione

La sicurezza sul posto di lavoro è una delle più importanti conquiste dei lavoratori del dopoguerra. Le aziende devono garantirla tramite opportuni corsi di formazione che insegnino ai lavoratori come comportarsi per salvaguardare la propria incolumità, ma soprattutto investendo in procedure e tecnologie che limitino sempre più i rischi legati all'ambito lavorativo.

La salute sul posto di lavoro, oltre ad essere un diritto del lavoratore, è un interesse sia per l'impresa, che in questo modo garantisce un ambiente sempre più confortevole e di conseguenza più produttivo, sia per la società nel suo insieme, che così riduce i costi sociali per infortuni e malattie professionali.[1]
Il Bureau of Labor Statistics (BLS), un'unità del Dipartimento del Lavoro degli Stati Uniti d'America, ha pubblicato dei dati relativi al 2017 in cui si sono verificati 2,9 infortuni sul lavoro ogni 100 dipendenti a tempo pieno, per un totale di circa 3,3 milioni a livello nazionale. Collettivamente, gli infortuni sul lavoro costano all'economia degli Stati Uniti dai 52 ai 60 miliardi all'anno, vale a dire almeno 1 miliardo di dollari a settimana.[2]

Il mondo del lavoro sta cambiando rapidamente. In particolare stanno aumentando sempre di più le applicazioni dell'intelligenza artificiale in vari ambiti del mondo lavorativo. L'Intelligenza artificiale (IA), ovvero la capacità di un computer di elaborare ragionamenti tipici della mente umana, può essere uno strumento utile per prevenire gli incidenti sul lavoro. Un recente articolo del MIT Technology Review ha analizzato l'applicazione dell'IA nei cantieri, evidenziando il caso di Suffolk, una società di costruzioni con sede a Boston, nel Massachusetts. Da oltre un anno l'azienda lavora su un sistema di sicurezza sul lavoro basato sull'intelligenza artificiale con l'obiettivo di prevedere quando si verificheranno incidenti, in modo da aumentare la capacità di prevenirli.
Il sistema utilizza un algoritmo che combina il monitoraggio in tempo reale sia con le informazioni derivanti dai casi di incidenti passati, sia con quelle derivanti dall'analisi delle immagini esistenti del cantiere. Lo scopo è quello di trovare situazioni in cui è probabile che si verifichino incidenti. Ad esempio, è in grado di segnalare la presenza sul cantiere di un lavoratore senza guanti o senza casco protettivo.[3]

Il progetto esposto in questa tesi nasce dall'esigenza di un'importante azienda italiana leader nel settore energetico di fornire un supporto automatico alla messa in

sicurezza dei propri impianti elettrici durante le operazioni di manutenzione. Tale processo prevede la messa in sicurezza degli interruttori elettrici che si trovano nell'area che deve essere sottoposta a manutenzione, in modo da non far circolare corrente elettrica che potrebbe essere potenzialmente letale per coloro che devono eseguire l'operazione.

Attualmente le operazioni di verifica della messa in sicurezza dell'impianto si basano esclusivamente sul controllo da parte degli esseri umani che, per natura, commettono errori. Tuttavia, un errore in questo settore rischia di essere fatale per i lavoratori. Pertanto l'azienda ha deciso di inserire all'interno del proprio processo di verifica di messa in sicurezza dell'impianto una componente tecnologica e automatizzata, in modo da affiancare l'operato umano e aumentare il livello di sicurezza dell'intero processo.

A tale scopo è stato deciso di progettare e sviluppare dei modelli di Machine Learning in grado, data l'immagine, di riconoscere lo stato dell'interruttore elettrico e di verificarne la corretta messa in sicurezza. Sono state utilizzate tecniche di Deep Learning, in particolare reti convoluzionali, le quali sono particolarmente adatte nell'ambito del riconoscimento di immagine.

Il Deep Learning è strettamente legato all'intelligenza artificale e al machine learning. L'intelligenza artificiale è l'insieme di teorie e algoritmi che permettono ai computer di imitare l'intelligenza umana, mentre il machine learning è un sottoinsieme dell'intelligenza artificiale che, tramite metodi statistici, permette ai computer di migliorare nelle proprie decisioni tramite i dati e l'esperienza.
Il Deep Learning è una sottocategoria del machine learning in grado di raggiungere ottime prestazioni e grande flessibilità poichè impara a rappresentare il mondo come una gerarchia nidificata di concetti, con ogni concetto definito in relazione a concetti più semplici e rappresentazioni più astratte calcolate in termini di quelle meno astratte.[4]

I capitoli successivi approfondiranno maggiormente il lavoro svolto e i risultati ottenuti: nel Capitolo 2 verrà approfondito il concetto di Deep Learning e di come le reti convoluzionali siano particolarmente adatte per il riconoscimento di immagine; nel Capitolo 3 verranno esposte le tecniche e le modalità con le quali negli ultimi anni è stato applicato il Deep Learning nell'ambito della sicurezza sul lavoro; nel Capitolo 4 verrà approfondito il processo di messa in sicurezza dell'impianto e verranno esposte le metodologie adottate per raggiungere lo scopo del progetto; nel Capitolo 5 verrà esposto il lavoro svolto, in particolare le tecniche utilizzate per realizzare le reti neurali convoluzionali; nel Capitolo 6 verranno analizzati e valutati i modelli realizzati e infine, nel Capitolo 7, verranno riportati i risultati ottenuti esponendo gli aspetti più rilevanti del progetto svolto.

Capitolo 2

Background

Il machine learning è l'insieme di metodi in grado di rilevare automaticamente modelli dai dati e quindi utilizzare tali modelli per prevedere dati futuri o eseguire altri tipi di processi decisionali in condizioni di incertezza.[5]

Esistono due principali tecniche di machine learning: supervised learning e unsupervised learning. La prima ha come obiettivo quello di generare un modello statistico a partire da un insieme di dati detto training set che è composto da coppie $\{x_\mathrm{i}, y_\mathrm{i}\}$ con i che va da 1 a N, dove N è il numero di campioni. Tale modello deve essere in grado di predire la risposta y_i per future osservazioni di cui non si conosce la risposta. In generale X_i è un vettore di dimensione D che rappresentà un'entità (ad esempio l'altezza e il peso di una persona). Le componenti di tale vettore sono chiamate features o attributi. X_i può essere un oggetto strutturato molto complesso, come ad esempio una email, una molecola o un'immagine. Il secondo tipo di tecnica di machine learning è il cosiddetto unsupervised learning, il cui obiettivo è invece quello di trovare dei pattern interessanti nei dati. Un esempio è la tecnica del clustering che ha come obiettivo quello di raggruppare i dati in gruppi omogenei secondo delle caratteristiche di rilievo.[5]

Poichè il progetto applicativo esposto in questo elaborato si basa sul riconoscimento di immagine, verrà approfondito il supervised learning, in particolare la classificazione tramite reti neurali convoluzionali. In generale, l'obiettivo della classificazione è quello di stimare una funzione f:

$$y = f(x)$$

dove $y \in \{1...C\}$ e C è il numero di classi.
La funzione f viene stimata a partire da un training set contenente dati etichettati e viene successivamente utilizzata per predire input di cui non si conosce l'etichetta:

$$\hat{y} = \hat{f}(x)$$

(il simbolo del cappello viene utilizzato per indicare una stima).

Per la classificazione di immagini le reti neurali convoluzionali rappresentano la soluzione migliore. I paragrafi successivi illustreranno cosa sono le reti convoluzionali e il perchè del loro grande successo nell'ambito del riconoscimento di immagine.

2.1 Deep learning

Una delle sfide più importanti della ricerca sull'intelligenza artificiale è stata quella di riuscire a emulare l'efficienza e la robustezza del cervello umano nel rappresentare le informazioni derivanti dal mondo che ci circonda. L'essere umano ogni giorno riceve una quantità di dati sensoriali molto elevata ed è in grado di estrapolarne gli aspetti più importanti e riutilizzarli in maniera coincisa per i propri scopi. Inizialmente l'intelligenza artificiale era riuscita a risolvere problemi intellettualmente difficili per gli esseri umani, ma relativamente semplici per i computer: problemi che possono essere descritti da un elenco di regole matematiche formali. Tuttavia la vera sfida dell'intelligenza artificiale è stata quella di risolvere problemi che l'essere umano è in grado di risolvere facilmente ed intuitivamente osservando e analizzando i dati sensoriali che lo circondano (come ad esempio riconoscere un volto o capire le parole di un discorso), ma che sono difficili da descrivere in maniera strutturata e formale per i computer.

La soluzione a questo problema è quella di permettere ai computer di imparare dall'esperienza e di rappresentare il mondo in termini di gerarchia di concetti: inizialmente riconoscono i componenti locali dei dati e in seguito li uniscono in una panoramica globale dell'intero dato.

Il termine "deep" indica la presenza di più strati (layer), ognuno dei quali estrae delle informazioni (features) dall'input in modo da formare una gerarchia di rappresentazioni, dalle features di basso livello a quelle di medio livello, fino all'identità dell'oggetto. Ad esempio, nell'ambito del riconoscimento di immagine, l'input grezzo potrebbe essere una matrice di pixel che rappresenta un'immagine. Il primo layer potrebbe astrarre i pixel e codificarli in bordi; il secondo layer potrebbe comporre e codificare le disposizioni dei bordi; il terzo può codificare naso e occhi; il quarto è in grado di riconoscere che l'immagine contiene un volto (Figura 2.1).

I primi predecessori del deep learning moderno sono stati i modelli lineari, i quali furono progettati per ottenere un output y a partire da un set di n valori $\{x_1...x_{\mathrm{n}}\}$. Questi modelli hanno lo scopo di imparare un insieme di pesi $\{w_1...w_{\mathrm{n}}\}$ e calcolare il loro output:

$$f(x, w) = x_1w_1 + x_2w_2 + ... + x_nw_n$$

I modelli lineari hanno alcune limitazioni, in particolare non sono in grado di risolvere problemi non lineari, i quali rappresentano la grande maggioranza dei problemi reali.

Il concetto di deep learning non è recente. Questo termine fu introdotto nella comunità del machine learning da Rina Dechter nel 1986[6][7] e da allora studi e ricerche hanno contribuito a renderlo il potente strumento di cui oggi disponiamo. Il concetto di rete neurale invece è ancora più antecedente: negli anni 50, il perceptron fu il primo modello in grado di apprendere i pesi definendo delle categorie a partire da esempi di input di ciascuna categoria. Dall'unione di più perceptron si origina il cosiddetto multilayer perceptron (MLP), anche detto Feedforwad neural network.
La Sezione 2.1.1 è dedicata al MLP, che può essere definito come l'hello world del deep learning, pertanto è necessario comprenderlo a pieno per poter capire gli aspet-

ti più complessi del deep learning.

Poichè i primi esperimenti furono condotti molti anni fa, ci si potrebbe chiedere come mai il deep learning solo recentemente sia stato riconosciuto come una tecnologia di fondamentale importanza per le applicazioni pratiche. La risposta è che al giorno d'oggi possiamo fornire a questi algoritmi le risorse di cui necessitano per funzionare al meglio. Per raggiungere ottime performance, i modelli di deep learning devono essere sufficientemente complessi (in termini di profondità e di numero di connessioni) e necessitano di una grande mole di dati che devono essere processati dai computer. La moderna potenza di calcolo e di memoria di cui disponiamo, l'avvento delle GPU, una connessione di rete sempre più veloce, sono stati elementi di fondamentale importanza per il successo applicativo di tali algoritmi.

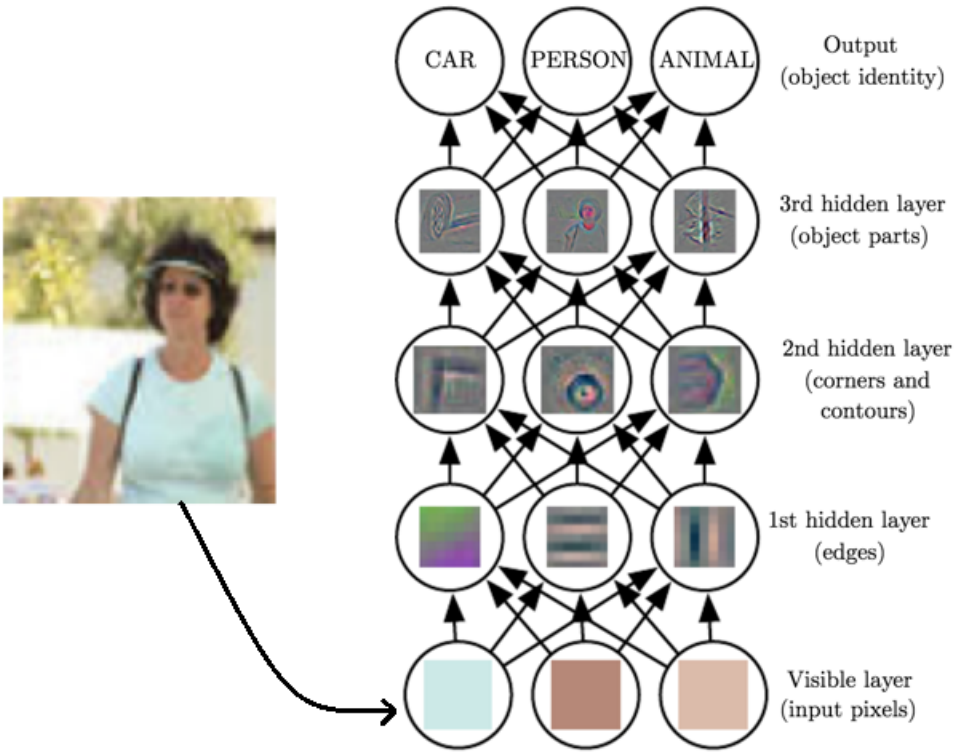


Figura 2.1: Esempio di modello di Deep Learning [4]. Il primo layer è detto "visibile" perchè contiene le variabili che possono essere osservate direttamente (i pixel nel caso delle immagini). A seguire ci sono una serie di hidden layer che estraggono progressivamente le features dall'immagine. Questi layer sono detti "hidden" perchè il loro valore non è fornito dai dati. Dai pixel iniziali, il primo hidden layer è in grado di identificare facilmente i bordi mentre il secondo hidden layer riesce, a partire dai bordi, ad estrapolare gli angoli e i contorni. Il terzo layer è in grado di individuare intere parti degli oggetti presenti nell'immagine, a partire dagli angoli e dai bordi. Il layer finale viene infine utilizzato per riconoscere gli oggetti presenti nell'immagine.

2.1.1 Feedforwad neural network

Come anticipato nel paragrafo introduttivo sul Deep Learning, per poter comprendere cos'è una rete neurale è necessario introdurre il perceptron che può essere considerato una rete neurale con un singolo layer.

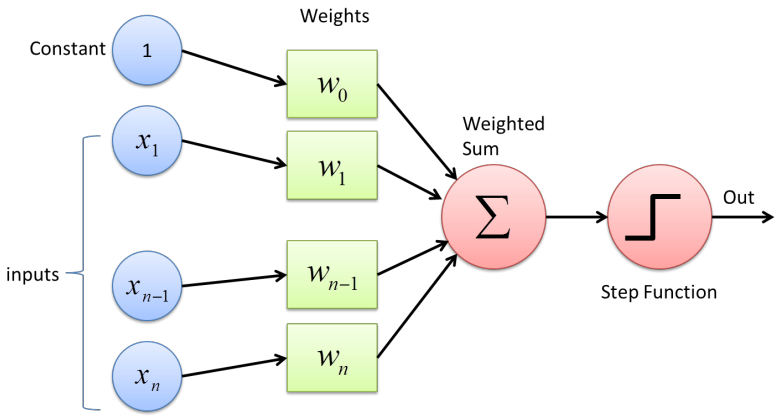


Figura 2.2: Perceptron

Il perceptron è un classificatore lineare binario, riceve n input e produce un output. I parametri di questo modello sono i pesi w_i che indicano l'importanza di uno specifico input rispetto all'output e una funzione di attivazione che è usata per mappare l'input tra i valori richiesti (ad esempio tra 0 e 1).

$$output = \begin{cases} 0, & \text{se } \sum_j w_j x_j \leq soglia \\ 1, & \text{se } \sum_j w_j x_j > soglia \end{cases} \tag{2.1}$$

Una rete neurale è un sistema computazionale che si ispira alle connessioni neurali biologiche del cervello. Sono composte da nodi computazionali detti neuroni che sono connessi tra di loro in modo analogo a come le sinapsi collegano i neuroni all'interno del cervello. Una rete neurale riceve un segnale e lo processa tramite la rete di neuroni e connessioni.
Proprio come nel caso del perceptron, l'importanza di ciascun input è fornita da un parametro w_i detto peso, ma, a differenza del perceptron, l'output di ciascun neurone è calcolato tramite una funzione non lineare della somma pesata dei suoi input. Tipicamente i neuroni sono aggregati in più layer, ognuno dei quali opera trasformazioni diverse dell'input.
Una delle funzioni non lineari più utilizzate è la funzione sigmoide, mostrata nella Figura 2.3

Una rete neurale si dice "feedforward" quando le connessioni tra i neuroni non formano cicli e le informazioni si muovono in avanti solo in una direzione, dai nodi di ingresso, attraverso i nodi intermedi, fino ai nodi di uscita.

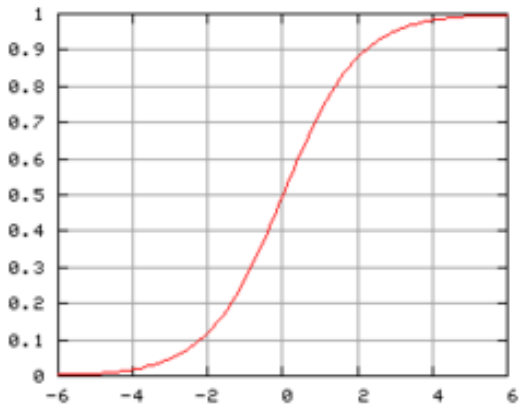


Figura 2.3: La funzione sigmoide è così definita: $F(x) = \frac{1}{1+e^{-t}}$

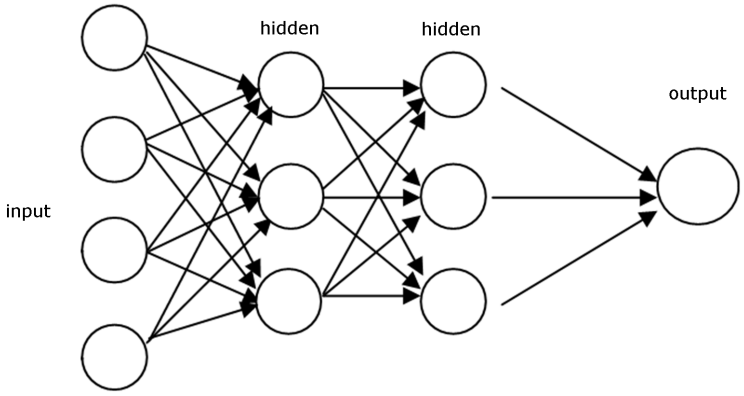


Figura 2.4: Rete neurale Feedforward

L'obiettivo di una rete neurale è quella di definire una funzione $y = f(x; \vec{\boldsymbol{w}})$ e imparare il valore del parametro $\vec{\boldsymbol{w}}$ che approssimi al meglio la funzione f al valore vero. Il parametro $\vec{\boldsymbol{w}}$ è il vettore dei pesi delle varie connessioni. Ogni campione x_i del training set (di cui si conosce la classe vera y_i) viene dato in input alla rete, la quale fornisce come output la classe predetta $\hat{y}_i$. Questo processo è detto forward-propagation. Una volta ottenuto l'output, viene calcolato l'errore $e = \hat{y} - y$ commesso dalla rete. È opportuno scegliere la giusta funzione di costo per valutare l'errore compiuto. L'errore viene utilizzato per aggiornare i pesi delle varie connessioni in modo da ridurre l'errore alla prossima iterazione. Questo processo è invece detto back-propagation. Per ogni campione del training set vengono compiute più iterazioni, dette epoche, fino a quando l'errore non è sotto una certa soglia ritenuta accettabile o si è raggiunto un numero di epoche sufficiente.

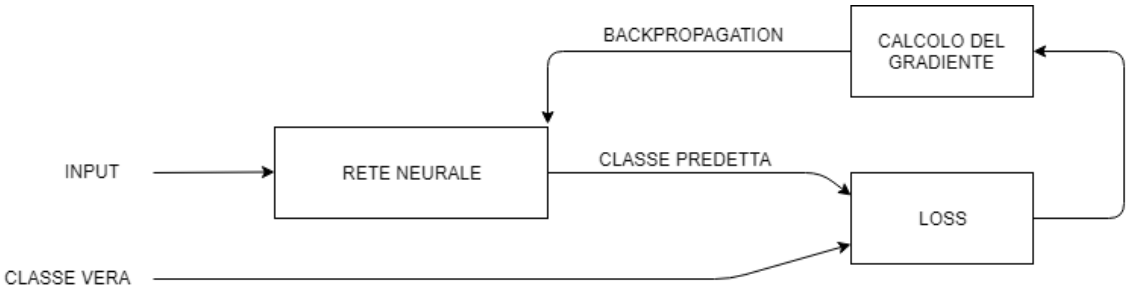


Figura 2.5: Training di una rete neurale

Lo scopo della fase di training è quella di minimizzare l'errore compiuto dopo ogni iterazione aggiornando i pesi della rete. Per valutare l'errore è necessario definire una funzione di costo, anche detta funzione di loss, e usare delle tecniche per minimizzarla. Esistono diverse funzioni di costo, tra le più utilzzate vi è la cross-entropy loss che sfrutta il principio della maximum likelihood, ovvero i parametri del modello vengono stimati in modo da massimizzare la probabilità che un certo campione appartenga ad una determinata classe.
Per trovare il minimo di una funzione si utilizza un algoritmo di ottimizzazione iterativo del primo ordine chiamato gradient descent, in particolare si calcola il gradiente della funzione e si procede in modo proporzionale verso la direzione negativa del gradiente della funzione nel punto corrente (Figura 2.6).

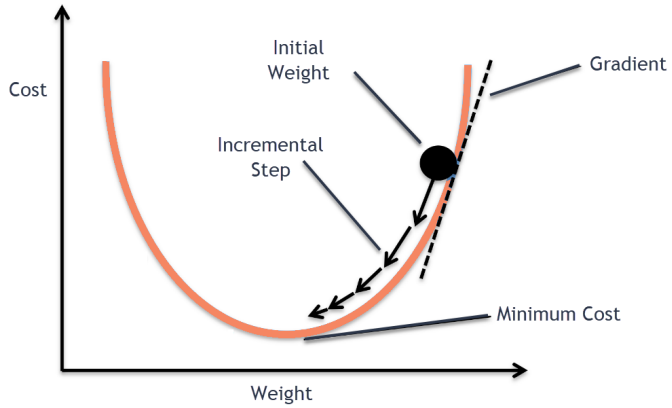
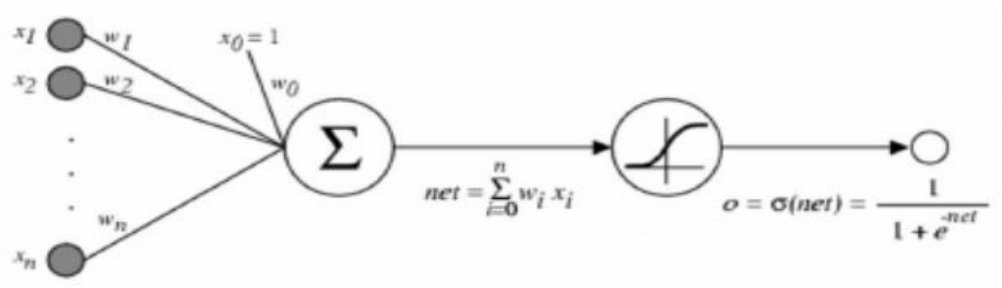


Figura 2.6: Gradient descent

Una volta calcolato il gradiente si utilizza l'algoritmo del back-propagation per aggiornare i pesi della rete. Di seguito si riportano gli step matematici che vengono eseguiti durante il training di una rete neurale.

$$net = \sum_{n=0}^{n} w_i x_i$$

$$\sigma = \sigma(net) = \frac{1}{1 + e^{-net}}$$



L'obiettivo è trovare il vettore dei pesi $\vec{\boldsymbol{w}}$ che massimizzi la probabilità che un campione appartenga ad una determinata classe:

$$\vec{\boldsymbol{w}} = \arg\max_{\vec{\boldsymbol{w}}} \lg \prod_i P(Y_i|X_i;\vec{\boldsymbol{w}}) = \arg\min_{\vec{\boldsymbol{w}}} \sum_i \frac{1}{2}\left(Y_i - \hat{f}(X_i;\vec{\boldsymbol{w}})\right)^2 = \arg\min_{\vec{\boldsymbol{w}}} E(\vec{\boldsymbol{w}})$$

Dunque la funzione di loss da minimizzare è:

$$E(\vec{\boldsymbol{w}}) = \frac{1}{2}\sum_i \left(\underbrace{Y_i}_{\text{output vero}} - \underbrace{\hat{f}(X_i;\vec{\boldsymbol{w}})}_{\text{output stimato}} \right)^2$$

Viene calcolato il gradiente di $E(\vec{\boldsymbol{w}})$

$$\nabla E(\vec{\boldsymbol{w}}) = \left[\frac{\partial E}{\partial w_0}, \frac{\partial E}{\partial w_1}, ..., \frac{\partial E}{\partial w_n}\right]$$

Una volta calcolato il gradiente, si aggiornano i pesi con i seguenti delta:

$$\Delta\vec{\boldsymbol{w}} = -\eta\nabla E(\vec{\boldsymbol{w}}) \rightarrow \Delta w_i = -\eta\frac{\partial E}{\partial w_i}$$

dove η è un parametro detto learning rate che controlla il valore di aggiornamento dei pesi rispetto al valore del gradiente.

Ci sono diversi modi in cui è possibile applicare il gradient descent:

1. Modalità batch
   Per calcolare il gradiente viene considerato l'intero training set. Questa modalità è costosa dal punto di vista computazionale.

2. Modalità incrementale
   Il gradiente viene calcolato considerando un singolo campione del training set per volta. Il rischio in questo caso è di raggiungere un minimo locale e non riuscire ad eseguire uno step sufficientemente grande per abbandonare il minimo locale e convergere verso il minimo assoluto.

3. Modalità mini-batch
   Questa modalità rappresenta un compromesso tra le due precedenti. Il gradiente viene calcolato su blocchi del training set composti da più campioni.

2.1.2 Reti neurali convoluzionali

Le reti neurali neurali convoluzionali (CNN) hanno svolto un ruolo importante nella storia del deep learning. Sono un esempio chiave di come studi e ricerche sull'apprendimento automatico abbiano avuto successo nelle applicazioni pratiche. Infatti sono stati i primi modelli di deep learning che abbiano avuto ottime prestazioni, tali da rendere il deep learning non solo un campo di ricerca, ma anche un'importante risorsa per applicazioni commerciali.[4]

Le CNN sono reti neurali con una struttura di connettività specializzata. Il nome "rete neurale convoluzionale" indica che la rete utilizza un'operazione chiamata convoluzione che viene utilizzata al posto della moltiplicazione matriciale in almeno uno dei suoi layer. La formula della convoluzione è la seguente:

$$s(t) = (x \circledast w)(t)$$

Nella terminologia delle reti convoluzionali, il primo argomento (x nella formula precedente) è detto input e il secondo argomento (w) è detto kernel. L'output è invece chiamato feature map. L'input è solitamente un array multidimensionale di dati e il kernel è un array multidimensionale di parametri che vengono adattati dall'algoritmo di apprendimento.[4]

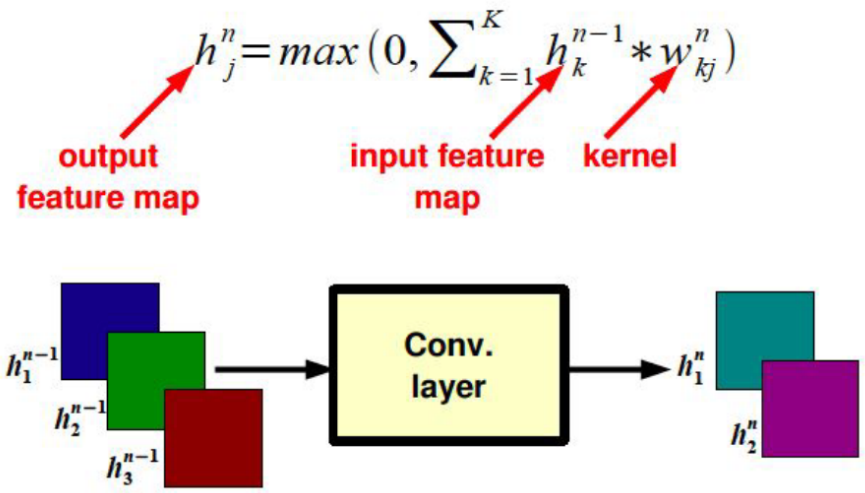


Figura 2.7: Estrazione di una feature map

Il grande successo delle reti convoluzionali può essere sintetizzato in due concetti principali: sparsa interazione e condivisione dei parametri.
I layer delle reti neurali tradizionali usano la moltiplicazione di matrici per una matrice di parametri con un parametro separato che descrive l'interazione tra ciascuna unità di input e ciascuna unità di output. Ciò significa che ogni output di un livello interagisce con ogni input del livello successivo. Le reti neurali convoluzionali, invece, hanno interazioni sparse poichè il kernel è più piccolo dell'input. Ad esempio, quando viene processata un'immagine, questa può avere in genere migliaia di milioni di pixel, ma è possibile rilevare piccole features significative come i contorni di un oggetto con kernel che occupano solo decine o centinaia di pixel. Dunque è necessario memorizzare un numero inferiore di parametri, riducendo così i requisiti di memoria del modello e migliorando la sua efficienza. Inoltre sono richieste meno

operazioni per calcolare l'output.
Per quanto riguarda la condivisione dei parametri, nelle reti neurali tradizionali ogni elemento della matrice dei pesi è usato esattamente una volta per calcolare l'output del layer. In una rete convoluzionale ogni membro del kernel è usato in ogni posizione dell'input. Dunque, invece di imparare un set di parametri separati per ogni posizione, ne viene imparato uno solo. Ciò non influisce sul tempo di esecuzione della forward propagation, ma riduce ulteriormente i requisiti di memoria del modello.[4][8].

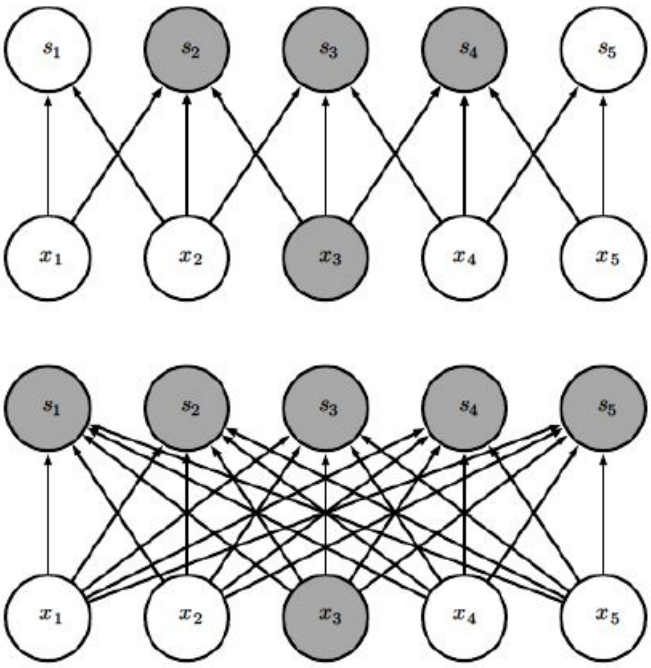


Figura 2.8: Connettività sparsa[4]: Nella figura sono evidenziate l'unità di input x_3 e l'unità di output s che sono interessate da x_3. (In alto) Quando s è generato con una convoluzione con un kernel di dimensione 3, solo tre output sono interessati da x_3. (In basso) Quando s è generato da una moltiplicazione matriciale, la connettività non è più sparsa e quindi tutti gli output sono interessati da x_3.

Tipicamente una rete convoluzionale è composta da tre operazioni principali (Figura 2.9). La prima operazione è quella di convoluzione (Figura 2.10), usata per produrre un set di attivazioni lineari. La seconda operazione è la cosiddetta "detector stage" in cui l'attivazione lineare passa attraverso una funzione non lineare. La terza operazione è quella di pooling che ha lo scopo di ridurre dimensionalmente l'output di un layer. Gli ultimi layer sono uno o più fully connected layer, usati per classificare gli oggetti[4].

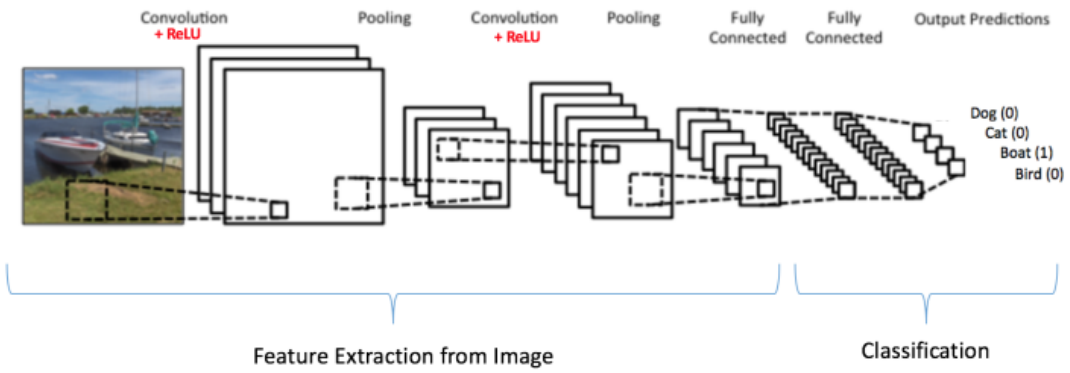


Figura 2.9: Composizione di una rete neurale convoluzionale

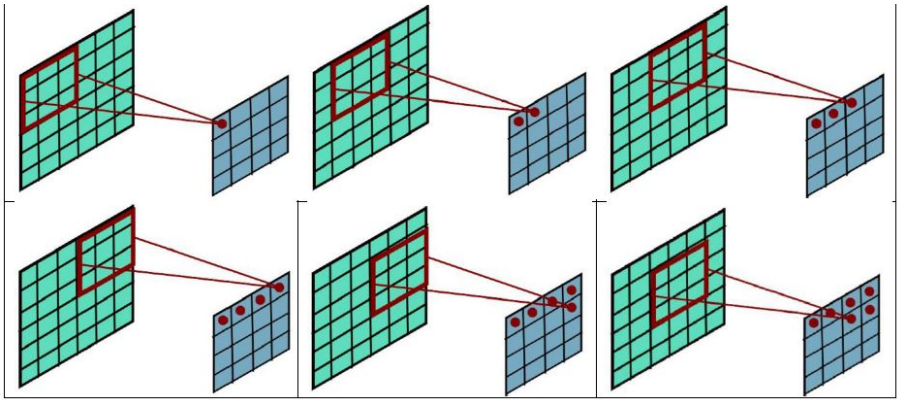


Figura 2.10: Operazione di convoluzione con un filtro 3x3. Il filtro viene fatto scorrere sull'immagine e viene calcolato il prodotto scalare. L'output è chiamato feature map ed è più piccolo dell'immagine principale.

Layer convoluzionale

Ci sono quattro parametri che controllano la dimensione dell'output di un layer convoluzionale:

- Numero di filtri K
  Definisce la profondità dell'activation map risultante. Solitamente è una potenza di 2.

- Dimensione del filtro F
  Tipicamente si applica un filtro 3x3, ma, specialmente nel primo layer, potrebbe essere più grande (ad esempio 5x5 o 7x7).

- Stride S
  Definisce come muovere il filtro lungo l'input: può essere mosso un pixel alla volta o più di uno. Più lo stride aumenta e più la dimensione dell'activation map diminuisce.

- Zero padding P
  A volte conviene riempire di zero parti dell'input per preservare la dimensione

spaziale dell'input stesso o per rendere possibile la convoluzione con un certo valore di stride.

La dimensione dell'activation map può essere calcolata nel modo seguente:

$$D = \frac{N - F + 2P}{S} + 1$$

dove D, N e F sono rispettivamente la dimensione dell'activation map, dell'input e del filtro convoluzionale. S è lo stride e P il padding.
La profondità dell'activation map è uguale al numero di filtri convoluzionali.

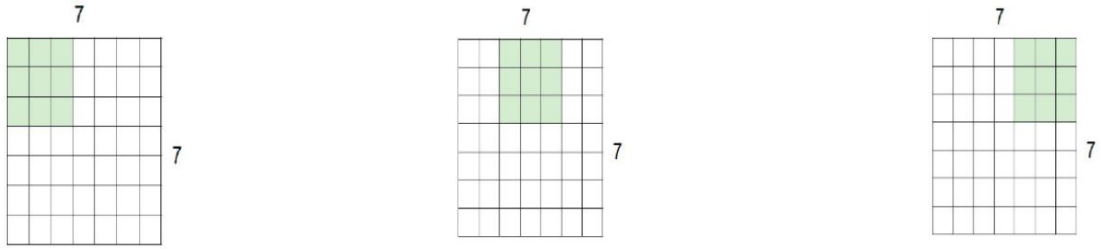


Figura 2.11: Esempio di input 7x7 e di filtro convoluzionale 3x3 con stride 2: l'activation map risultante sarà 3x3. Infatti $\frac{7-3+0}{2} + 1 = 3$.

Funzioni di attivazione non lineari

I neuroni calcolano la somma pesata dei loro input e, in base a tale valore, la funzione di attivazione decide se attivare il neurone o meno.

Nella Sezione precedente è stata introdotta la funzione sigmoide (Figura 2.3). Tra i valori di X che vanno da -2 a +2, i valori di Y sono molto ripidi. Ciò significa che qualsiasi piccola modifica dei valori di X in quella regione provocherà cambiamenti significativi nei valori di Y e quindi tale funzione ha la tendenza a portare Y a valori verso le estremità della curva facendo chiare distinzioni sulla previsione. Il problema è che verso entrambe le estremità i valori di Y tendono a rispondere molto meno ai cambiamenti di X. Si può dunque presentare il cosiddetto problema del gradiente evanescente (gradient vanishing problem), ovvero il valore del gradiente è così basso da non riuscire a portare a cambiamenti significativi della rete, la quale si rifiuta di apprendere ulteriormente o è drasticamente lenta.

La funzione di attivazione non lineare maggiormente usata nelle reti convoluzionali è la ReLU (Rectified Linear Unit):

$$A(x) = max(0, x)$$

La ReLU restituisce 0 se riceve un input negativo e restituisce l'input stesso se questo è positivo. Il motivo per cui questa funzione è molto utilizzata è che se ad esempio considerassimo una rete neurale molto grande con molti neuroni, usando la funzione sigmoide i neuroni si attiverebbero in modo analogo. In altre parole l'attivazione è densa e questo è costoso dal punto di vista computazionale e di memoria. Usando

la ReLu, un numero inferiore di neuroni si attiverebbe e la rete risulterebbe molto più leggera.

Tuttavia anche la ReLu presenta alcuni problemi. Poichè per valori negativi di X tale funzione vale 0, il gradiente può andare verso valori prossimi allo zero e dunque i pesi non verrebbero aggiornati durante la back-propagation. Ciò significa che quei neuroni smetteranno di rispondere alle variazioni dell'errore commesso dalla rete e dunque una buona parte della rete rimarrebbe passiva. Per far fronte a questo problema, noto come problema della ReLu morente, è possibile rendere la funzione leggermente inclinata per valori negativi di x. Questa variante è detta leaky ReLu. L'idea principale è quella di non permettere al gradiente di azzerarsi, ma di rigenerarsi durante il training.

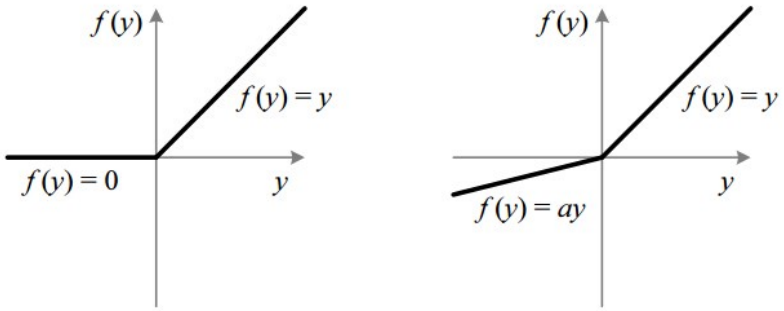


Figura 2.12: A sinistra la funzione di attivazione ReLu. A destra la Leaky ReLu dove il parametro a indica la pendenza della funzione. Un tipico valore di a è 0.01

Pooling layer

Il pooling layer ha lo scopo di ridurre la dimensione dell'output di un layer sostituendo l'output della rete in una certa posizione con un valore riassuntivo degli output vicini. Il pooling aiuta a rendere la rappresentazione approssimativamente invariante rispetto a piccole traslazioni dell'input. Intuitivamente, l'esatta posizione di una feature è meno importante della sua posizione relativa rispetto alle altre feature.
Poichè il pooling riassume la risposta di un intero vicinato, è possibile usare meno unità di pooling rispetto a quelle di detection. Ci sono diverse modalità con cui applicare il pooling, quelle più utilizzate sono il max pooling e l'average pooling. La prima seleziona il valore massimo su un insieme di parti non sovrapposte dell'input e ha lo scopo di salvare solo le informazioni significative. La seconda invece calcola la media dei valori in modo da non perdere troppa informazione (può essere utile soprattutto per pooling di grandi dimensioni).

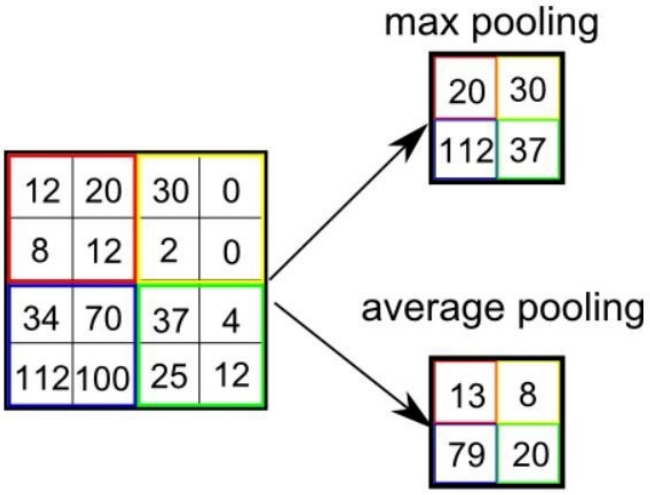


Figura 2.13: Max pooling e average pooling

2.1.3 Tecniche di regolarizzazione

Le tecniche di regolarizzazione vengono utilizzate per cercare di evitare che si verifichi l'overfitting, situazione in cui la rete neurale si è adattata così tanto ai campioni del training set che non è più in grado di generalizzare e fare predizioni su nuovi dati.

I dati di training oltre all'informazione sul dato stesso, contengono anche errori o dati accidentali proprio a causa dei particolari casi di training scelti. Quando il modello viene addestrato, non è in grado di riconoscere quali regolarità sono vere e quali causate da errori. Dunque se la capacità della rete è troppo grande, essa imparerà non solo le features, ma anche gli errori e non sarà più in grado di generalizzare su nuovi campioni da classificare.

Esistono diverse tecniche di regolarizzazione per contrastare il fenomeno dell'overfitting.

Weight Decay

Una delle soluzioni più efficaci è il weight decay, tecnica che penalizza i pesi con valore elevato aggiungendo un termine di regolarizzazione alla funzione di costo della rete (L2 penalty). L'obiettivo è quello di mantenere bassi i valori dei pesi in quanto pesi con valori troppo elevati portano la rete a imparare anche gli errori. Applicando il weight decay la nuova funzione di loss diventa:

$$L(\vec{\boldsymbol{w}}) = L(\vec{\boldsymbol{w}}) + \frac{\lambda}{2}\vec{\boldsymbol{w}}^2$$

Applicando il gradient descent, si ottiene:

$$w_i \leftarrow w_i - \eta\frac{\partial L}{\partial w_i} - \eta\lambda w_i$$

Dropout

Il dropout è una tecnica che ha come obiettivo quello di evitare il co-adattattamento dei neuroni all'interno della rete. Infatti è possibile che un neurone cambi il proprio comportamento in modo da correggere gli errori commessi da altre unità. Ciò può portare a complessi adattamenti e all'overfitting. Applicando il dropout, durante la fase di training ogni neurone ha una probabilità p (tipicamente del 50%) di essere "spento" e quindi temporaneamente ignorato. Di conseguenza, l'addestramento sarà più difficile e richiederà più tempo per convergere, ma la rete imparerà a classificare utilizzando diversi set di features ad ogni iterazione, evitando così il co-adattamento. Durante la fase di testing, il dropout non viene utilizzato e tutte le features sono disponibili per la classificazione.

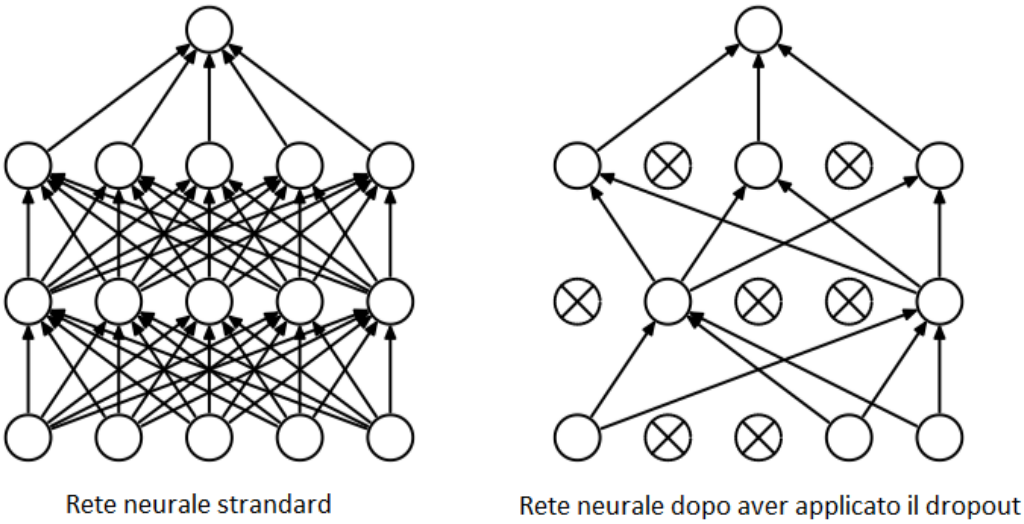


Figura 2.14: Dropout

Batch Normalization

In generale, qualsiasi algoritmo di machine learning lavora meglio su dati normalizzati (media zero e, solitamente, varianza unitaria). Nel caso della CNN, dopo ogni layer convoluzionale e dopo ogni fully connected layer, a causa delle funzioni di attivazioni non lineari, i dati non sono più normalizzati.

La soluzione è inserire un layer di batch normalization che normalizza l'output di un layer precedente sottraendo la media del batch e dividendo per la deviazione standard del batch.

$$\hat{x_k} = \frac{x_k - \mathbb{E}[x_k]}{\sqrt{Var[x_k]}}$$

La batch normalization è largamente utilizzata nelle recenti reti convoluzionali dopo ogni layer convoluzionale e dopo ogni fully connected layer (tranne dopo quello finale di classificazione) in quanto tende a limitare il problema del gradiente evanescente permettendo dunque la presenza di più layer. Inoltre velocizza di molto la convergenza della rete e ha un effetto di regolarizzazione.

Data Augmentation

Uno degli accorgimenti più importanti nel machine learning e in particolare nel deep learning è quello di sfruttare correttamente i propri dati. La data augmentation ha lo scopo di trasformare i dati di input con qualche variazione, in modo da aumentare la dimensione del dataset e di variarne leggermente il contenuto, migliorando la capacità della rete di generalizzare ed evitando così l'overfitting. Naturalmente le trasformazioni devono essere significative rispetto ai dati: nel caso delle immagini, queste potrebbero essere allungate, ruotate, capovolte, ritagliate o se ne potrebbero cambiare i colori, senza però che queste trasformazioni cambino l'etichetta di classe.

Alcune tecniche popolari di data augmentation sono:

- Horizontal flip
  Ogni campione del training set viene capovolto orizzontalmente
- Random crops/scales
  Il dato viene tagliato o scalato in maniera casuale e la rete deve essere ancora in grado di classificarlo correttamente. Ad esempio, se in un'immagine è presente un gatto e viene ritagliata solo la testa, la rete deve essere ancora in grado di riconoscere il gatto.
- Color jitter
  Questo metodo consiste nel cambiare i colori all'immagine, ad esempio il contrasto o la luminosità.

2.1.4 Transfer Learning

Per addestrare le reti neurali ed ottenere delle buone performance è necessario avere un dataset molto numeroso, ma spesso, nelle applicazioni pratiche, questo non avviene. L'idea del transfer learning è quella di partire da una rete pre-addestrata su dataset di grandi dimensioni (ad esempio ImageNet) e adattarla in base alle esigenze applicative.

Ci sono due modi principali per applicare il transfer learning:

- Feature extractor
  Se il dataset a disposizione è piccolo o i pesi della rete pre-addestrata sono sufficientemente tarati per risolvere il task applicativo, è possibile non modificare la rete pre-addestrata, ma aggiungere solo un fully connected layer finale per adattare il modello al proprio problema specifico.
- Finetuning
  Se si ha a disposizione un dataset di buone dimensioni, è possibile specializzare i pesi della rete in base al problema specifico, migliorando le prestazioni.

Di seguito vengono riportati alcune delle più importanti reti convoluzionali progettate che possono essere utilizzate come reti pre-addestrate per risolvere specifici problemi applicativi. Molte di queste reti sono nate nell'ambito della competizione annuale chiamata ImageNet Large-Scale Visual Recognition Challenge (ILSVRC)

dove diverse reti neurali vengono testate sul dataset ImageNet per valutarne le prestazioni. ImageNet è un dataset contenente oltre 15 milioni di immagini ad alta risoluzione etichettate, appartenenti a circa 22000 categorie.

AlexNet

Inventata da Krizhevsky nel 2012, AlexNet è il pionere del Deep Learning. È una delle reti più importanti, che diede un enorme contributo nel miglioramento dell'accuratezza nella competizione ImageNet.

AlexNet è composta da cinque layer convoluzionali, tre layer di pooling e tre fully connected layer. Fu la prima ad utilizzare la ReLu come funzione di attivazione. Inoltre, sfrutta fortemente la data augmentation, usa dropout 0.5, ha un batch size di 128 e usa L2 weight decay.[9]

VGGnet

VGGnet fu inventata nel 2014 da Simonyan e Zisserman. Si basa su AlexNet a cui vengono aggiunti altri layer convoluzionali. La grande novità di VGGnet rispetto ad AlexNet è che vengono utilizzati filtri 3x3 nei primi layer invece di filtri 11 e 5 rispettivamente nel primo e secondo layer convoluzionale di AlexNet. Infatti, dato un certo receptive field (l'area effettiva dell'immagine di input da cui dipende l'output), conviene usare più kernel di piccole dimensioni rispetto ad usarne uno solo di maggiori dimensioni perchè più layer non lineari aumentano la profondità della rete permettendole di impare più feature complesse e a minor costo (il numero di parametri è inferiore).[10]

Resnet

Quando le reti neurali sono molto profonde, la batch normalization non è sufficiente per evitare il problema del vanishing gradient. Infatti più la rete è profonda e più l'accuratezza aumenta, ma dopo un certo numero di layer inizia a decrescere perchè ogni layer ha una funzione di attivazione non lineare e quindi viene aggiunto del "rumore" alla pendenza del gradiente. Pertano durante la back-propagation il layer viene aggiornato con il rumore e la rete non può essere addestrata propriamente. Le residual neural network sono state progettate proprio per risolvere tale problema. Infatti esse utilizzano le cosiddette skip connection (Figura 2.15) in cui all'output viene sommato l'input per rendere il gradiente più "pulito" nella back-propagation. In questo modo l'errore diminuisce più velocemente ed è possibile ottenere reti più profonde.[11]

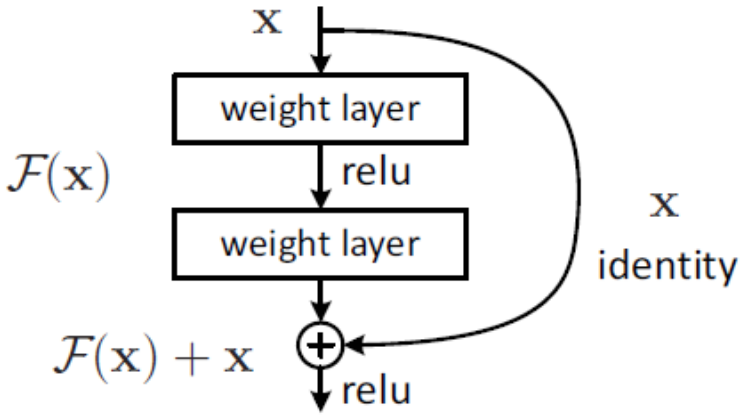


Figura 2.15: Skip connection

Le ResNet rappresentano una delle più importanti invenzioni degli ultimi anni nell'ambito del deep learning. In base al numero di layer si distinguono in ResNet-18, Resnet-34 e così via.

Capitolo 3

Stato dell'Arte

In letteratura sono presenti diversi esempi di Image Recognition tramite reti neurali convoluzionali in ambito industriale. Tuttavia sono pochi i lavori strettamente legati all'industria elettrica ed in particolare agli interruttori elettrici.

Un esempio di studio sul riconoscimento di interruttori elettrici è riportato in un paper intitolato *State Recognition of Electric Control Cabinet Switches Based on CNNs* [12] che propone un algoritmo di riconoscimento dello stato degli interruttori elettrici del quadro elettrico nelle sottostazioni elettriche. Tale algoritmo combina la tecnica di elaborazione delle immagini digitali con il riconoscimento dello stato tramite CNN. Infatti, secondo gli autori, il numero di quadri elettrici nelle sottostazioni è in aumento ed è essenziale ottenere lo stato degli interruttori in tempo reale per determinare la presenza di eventuali sprechi di energia e per controllare che la griglia elettrica funzioni in sicurezza. Essi sottolineano come l'ispezione manuale richiederebbe tempo e manodopera ed aumenterebbe il rischio di commettere errori. Pertanto, è molto importante utilizzare il sistema software per riconoscere automaticamente lo stato degli interruttori.
La CNN sviluppata riprende lo schema di AlexNet.
Questo lavoro ha dimostrato che processare l'immagine prima di sottoporla alla rete neurale convoluzionale porta diversi vantaggi in termini di accuratezza.

Altri studi hanno utilizzato tecniche diverse dalle CNN per il riconoscimento dello stato degli interruttori.
Il paper *Application of image recognition technology in electrical equipment on-line monitoring*[13] propone un algoritmo di classificazione basato su metodi tradizionali di proiezione e un algoritmo di corrispondenza del modello basato sull'analisi delle caratteristiche della forma dell'immagine.
Un altro tipo di approccio è quello utilizzato dagli autori di *Monitoring and identification system of substation switch division based on SURF*[14], i quali utilizzano un algoritmo basato su SURF che usa il GPRS per trasmettere alla sede di controllo le immagini degli interruttori monitorati e le informazioni dei guasti che si sono verificati.

Tra le varie tecniche presenti in letteratura, le CNN hanno dimostrato di avere prestazioni migliori in diversi problemi di visual recognition, sia per l'accuratezza elevata, sia per la velocità di predizione, come sottolineato anche dal paper *Switch*

state recognition method of intelligent substation based on full convolution neural network[15].

Oltre all'ambito dell'industria elettrica, ci sono diversi studi che testimoniano l'utilizzo delle CNN per la classificazione di immagine, ottenendo ottimi risultati.
Il paper *Convolutional Neural Networks for Food Image Recognition: An Experimental Study* [16] ha applicato le CNN al riconoscimento del cibo, con lo scopo di fornire un'analisi nutrizionale più completa e salutare possibile al consumatore finale. Il paper si è soffermato sulla scelta di architettura di CNN da adoperare per il riconoscimento di immagine. Hanno mostrato come le varie architetture hanno caratteristiche diverse in termini di complessità e dimensioni e non è noto a priori quale di queste sia maggiormente performante nel proprio ambito di studio.
Gli autori dell'articolo hanno evidenziato l'importanza della dimensione del dataset. Per ottenere un livello di accuratezza adeguato, bisogna avere un minimo di circa 300 immagini per categoria. Nel caso il dataset fosse troppo piccolo, è conveniente partire da reti preaddestrate su alcuni dataset contenenti milioni di immagini di varie categoria, come ad esempio ImageNet. Infatti hanno dimostrato che partire da pesi preaddestrati migliora le performance.
Lo studio sottolinea anche l'importanza delle tecniche di data augmentation per ottenere prestazioni migliori.

Un altro studio interessante in ambito industriale è quello riportato nell'articolo *Convolutional neural network for industrial egg classification* [17], dove è stata sviluppata una CNN per la classificazione delle uova in modo da valutarne la qualità. Questo lavoro ha dimostrato come le CNN siano efficaci non solo per la classificazione di immagini su dataset di grandi dimensioni, ma anche per quelli di piccole dimensioni. In questo caso la rete di partenza su cui sono stati eseguiti gli esperimenti è stata AlexNet.

Gli autori dell'articolo *Automatic Pearl Classification Machine Based on a Multistream Convolutional Neural Network* [18] hanno invece sviluppato una CNN per la classificazione di perle. Anche in questo caso la rete utilizzata è stata AlexNet, ma sono stati eseguiti anche esperimenti con altre reti preaddestrate, come VGG19, GoogleNet e ResNet34. La Figura 3.1 riporta i risultati da loro ottenuti.

Method	Batch Size	Training Time (h)	Testing Time (s)	Accuracy (%)
AlexNet	20	5.64	**15.25**	87.48
VGG19	5	18.30	74.35	**89.48**
GoogleNet	20	**2.87**	23.69	89.29
ResNet34	20	3.78	30.69	83.57

Figura 3.1: Confronto tra diversi modelli di CNN. [18]

Nel confronto sono stati valutati alcuni parametri, tra cui il training time e l'accuratezza. VGG19 è la più time-consuming a causa di un più piccolo batch suze e un numero maggiore di parametri.

Tale studio, insieme a quello riportato nel paper *Identification of Rice Diseases using Deep Convolutional Neural Networks* [19], ha anche dimostrato come per il task del riconoscimento di immagine, le CNN siano molto più performanti di altre tecniche, come ad esempio SVM, basati sull'estrazione manuale delle features.

Come sottolineato dal paper *State Recognition of Electric Control Cabinet Switches Based on CNNs* [12], la fase di preprocessing dell'immagine, da effettuare prima della predizione della rete neurale, è di fondamentale importanza per raggiungere alti livelli di performance della rete. Dunque, in fase di progetto, si è deciso di elaborare l'immagine originale in modo da tagliare la porzione di interesse contenente l'interruttore elettrico. Le modalità di tale operazione saranno approfondite nella Sezione 5.2 .
Dagli studi sopra riportati è anche emerso che le CNN rappresentano il metodo migliore per l'image recognition e di come sia importante la scelta dell'architettura di CNN in base al proprio specifico task. Pertanto, nella Sezione 5.3, verranno condotti esperimenti con diverse tipologie di rete in modo da confrontarne le prestazioni e scegliere la più performante. Tutti questi esperimenti veranno effettuati con reti pre-addestrate su ImageNet, in quanto, come evidenziato da *Convolutional Neural Networks for Food Image Recognition: An Experimental Study* [16], per dataset di piccole dimensioni le prestazioni migliorano.
Durante la fase di training, verranno applicate tecniche di data augmentation perchè la maggior parte dei casi in letteratura ha evidenziato come queste siano molto efficaci nell'evitare overfitting e migliorare l'accuratezza della rete.

Capitolo 4

Metodologia

In questo Capitolo verrà esposta la metodologia utilizzata per la realizzazione del progetto, dalla definizione del problema, fino al test sul campo e al rilascio in produzione. Ogni aspetto della metodologia mostrata in Figura 4.1 verrà analizzato più nel dettaglio nelle Sezioni successive.

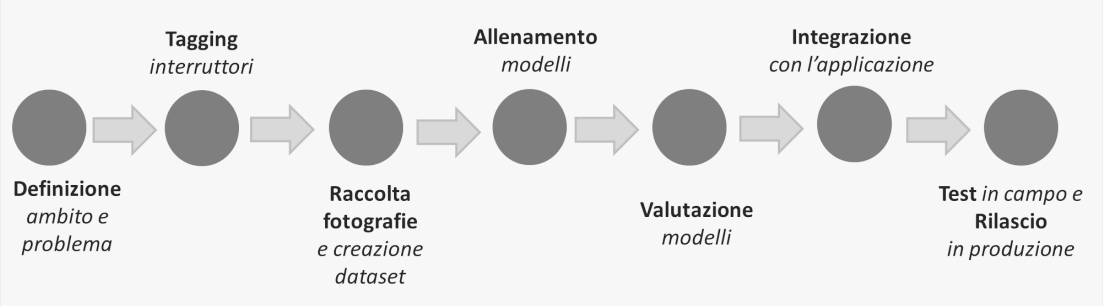


Figura 4.1: Metodologia utilizzata

Lo scopo del progetto è stato quello di introdurre un livello di sicurezza aggiuntivo nel processo di attivazione dei lavori di manutenzione, nelle centrali elettriche del cliente. Prima di poter progettare la soluzione applicativa, è stato necessario comprendere il processo AS-IS del cliente.
Le fasi del processo (Figura 4.2) sono le seguenti:

1. **Ordine di lavoro**
   Viene aperto un ordine di lavoro per le attività di manutenzione a seguito di una segnalazione.

2. **Creazione permesso di lavoro**
   Il capo turno crea il permesso di lavoro che resta bloccato in attesa del sezionamento.

3. **Sezionamento impianto**
   L'operatore seziona l'impianto in modo da metterlo in sicurezza per le attività di manutenzione.

4. **Rilascio permesso di lavoro**
   Completato il sezionamento, il capo turno rilascia il permesso di lavoro e autorizza la manutenzione.

5. **Intervento squadra**
   La squadra esterna di manutenzione accede alla sezione di impianto messa in sicurezza ed esegue l'attività.

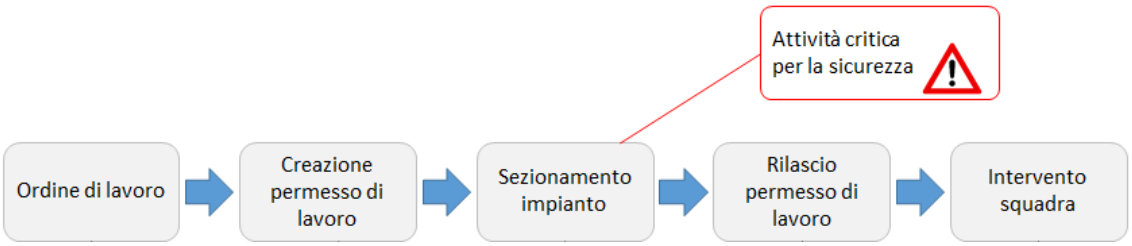


Figura 4.2: Fasi del processo di manutenzione AS-IS

I due attori principali del processo sono: il capo turno che crea la check-list con i task di sezionamento tramite l'applicazione web e schedula la check-list nel turno richiesto; l'operatore che, prima prende in carico la check-list tramite un'app installata sui dispositivi della centrale, poi esegue l'attività di sezionamento elettrico e infine inserisce nella check-list il riscontro sul nuovo stato. A questo punto raccoglie le evidenze fotografiche e le invia al capo turno il quale ha il compito di verificare il corretto sezionamento dell'impianto e, in caso positivo, di autorizzare le operazioni di manutenzione.

Tuttavia può capitare che il capo turno sia impegnato in altre attività di cui è responsabile e che presti poca attenzione alla verifica del sezionamento. In questo caso il rischio che possano essere commessi degli errori è elevato. Per questo motivo è nata l'esigenza da parte del cliente di aggiungere all'interno di questo processo un ulteriore livello di verifica automatizzato, tramite intelligenza artificiale. In particolare l'operatore prima di inviare le evidenze fotografiche del proprio operato al capo turno, interroga, tramite dei microservizi, i modelli di rete neurale che forniscono la risposta sullo stato dell'interruttore. Le fotografie e la predizione delle reti neurali vengono inviate al capo turno che riceve un allarme se predizione della rete e quella dell'operatore differiscono.
Il processo TO-BE è descritto in Figura 4.3

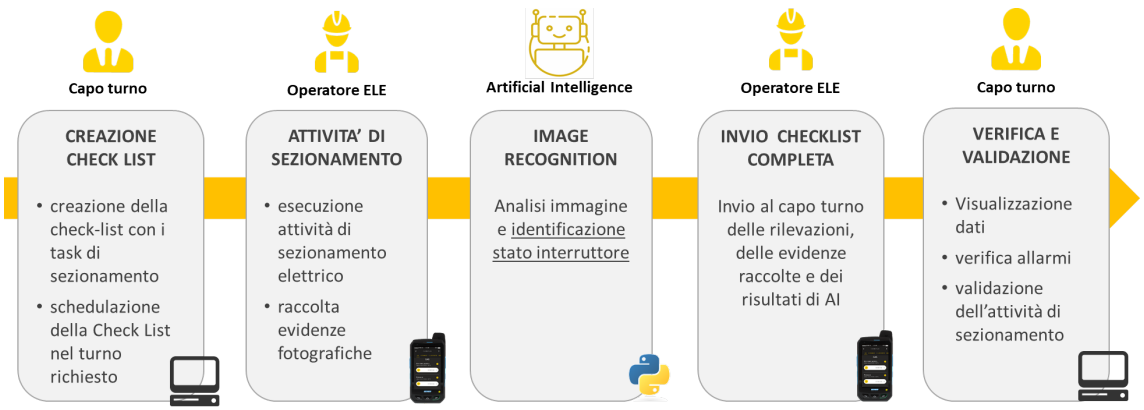


Figura 4.3: Fasi del processo di manutenzione TO-BE

I microservizi rappresentano una tecnologia di ultima generazione nella progettazione, nello sviluppo e nella fornitura di servizi software. Costituiscono un approccio all'architettura di software e sistemi che si basa sul concetto consolidato di modularizzazione. Ogni modulo, ovvero ogni microservizio, è implementato e gestito come un sistema piccolo ma indipendente, offrendo accesso alla sua logica interna e ai dati attraverso un'interfaccia di rete ben definita. Ciò aumenta l'agilità del software perché ogni microservizio diventa un'unità indipendente di sviluppo, implementazione, controllo delle versioni e ridimensionamento.[20]

La Figura 4.4 rappresenta graficamente la soluzione tecnologica adottata.

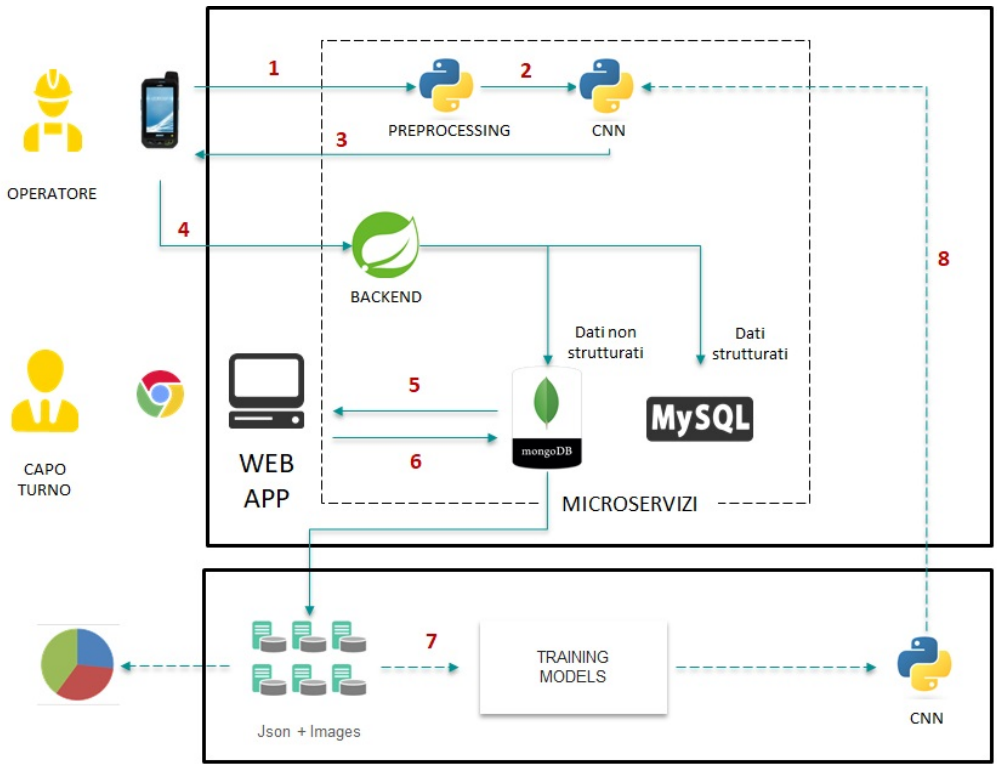


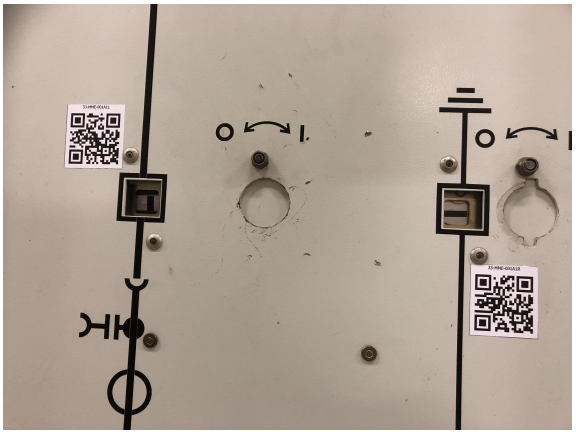
Figura 4.4: Soluzione tecnologica.
(1) L'operatore invia l'immagine dell'interruttore ad un microservizio che effettua delle operazioni di preprocessing sull'immagine. **(2)** L'immagine elaborata viene inviata ad un altro microservizio che interroga i modelli di rete neurali per ottenere una predizione sullo stato dell'interruttore. **(3)** Il microservizio fornisce la predizione della rete neurale all'applicazione dell'operatore che la visualizza sullo schermo. **(4)** L'operatore sottomette l'operazione di sezionamento con le evidenze fotografiche della messa in sicurezza dell'interruttore e la predizione dell'intelligenza artificale. Il backend del sistema elabora e salva la richiesta dell'operatore. **(5)** Il capo turno, tramite un browser, accede alla web application e verifica che l'operazione di sezionamento sia avvenuta correttamente. **(6)** Se tutto è andato a buon fine il capo turno autorizza la richiesta. **(7)** Le immagini sottomesse dall'operatore vengono salvate in modo da poter capire cosa è successo in caso di errore e per usi futuri. **(8)** Le immagini salvate possono essere utilizzate per riaddestrare le reti neurali in modo da raffozzarne le performance.

4.1 Scelta tipologie di interruttori e tagging

All'interno delle centrali elettriche del cliente ci sono numerose tipologie di interruttori. Per raggiungere delle prestazioni ottimali ho deciso di creare diversi modelli dei rete neurale, uno per ogni tipologia. Infatti le varie tipologie differiscono molto tra di loro e pertanto creare un unico modello sarebbe stato poco efficiente.

Ho scelto, insieme al cliente, quattro tipologie di interruttori, ognuna con le proprie caratteristiche, che sono state analizzate separatamente. Sono state scelte queste quattro tipologie perchè rappresentano l'80% degli interruttori presenti negli stabilimenti del cliente, in modo da poter inserire la componente di intelligenza artificiale in quasi tutte le centrali presenti su suolo nazionale. Di seguito si riporta un esempio per ogni tipologia di interruttore.

(a) finestra



(b) pannello

Figura 4.5: Esempio di Tipologia *window-panel*

Figura 4.6: Esempio di Tipologia *knob*

Figura 4.7: Esempio di Tipologia *switch*

Figura 4.8: Esempio di Tipologia *lever*

Ogni interruttore ha un proprio codice identificativo all'interno dello stabilimento. Come è possibile notare dalla Figura 4.8, gli interruttori possono essere anche molto vicini tra di loro e questo rende molto difficile per l'operatore inquadrare solo l'interruttore di interesse durante la raccolta delle evidenze fotografiche. Se l'operatore inquadrasse più interruttori, la rete neurale non saprebbe quale classificare. Pertanto ho deciso, in fase di progettazione, di applicare alcuni QR code nell'angolo in alto a destra e in basso a sinistra di ogni interruttore. Tali QR code codificano al loro interno l'id dell'interruttore. In questo modo, se nella fotografia dell'operatore sono presenti più QR code, quelli che non codificano l'id dell'interruttore di interesse vengono ignorati. Naturalmente, questo implica che l'operatore debba specificare l'id dell'interruttore di cui sta raccogliendo l'evidenza fotografica.
Una volta fotografato l'interruttore, i QR code vengono utilizzati per delimitare l'area di interesse dell'immagine e ritagliarla, in modo da fornire alla rete neurale esclusivamente la porzione dell'immagine contenente l'interruttore, aumentandone le performance. Il funzionamento di questa procedura verrà spiegato più nel dettaglio nella Sezione 5.2 .

4.2 Creazione del dataset

Diversi studi riportati nel Capitolo precedente, sottolineano che la dimensione del dataset è un aspetto cruciale per la realizzazione di una rete neurale convoluzionale sufficientemente performante. Tuttavia in molti casi raccogliere decine di migliaia di immagini è molto oneroso. L'approccio seguito è stato quello del paper *Automatic Pearl Classification Machine Based on a Multistream Convolutional Neural Network* [18], ovvero partire da reti preaddestrate su dataset conteneti milioni di immagini di varie categorie, come ImageNet, e utilizzare le nostre immagini per addestrare gli ultimi layer della rete.

Mi sono recato in una delle fabbriche del cliente e ho fotografato le varie tipologie di interruttori. In totale sono state raccolte più di settemila immagini, cercando di variare l'angolazione e le condizioni di luce già in questa fase. Inoltre le immagini sono state raccolte con dispositivi diversi, in modo da variare anche la risoluzione dell'immagine. I dispositivi utilizzati sono stati:

- Un dispositivo ECOM con certificazione Atex.
  Questo è il dispositivo utilizzato dagli operatori nelle centrali del cliente. Ha una fotocamera da 12 MP e una risoluzione delle immagini di 3264x2448.
- Uno smartphone xiaomi Redmi Note 5
  Ha una fotocamera da 12 MP e una risoluzione delle immagini di 4032x3024
- Un iphone 8
  Ha una fotocamera da 12 MP e una risoluzione delle immagini di 3264x2448

Durante l'acquisizione delle immagini, le condizioni di illuminazione erano buone. La luce artificiale presente era prodotta da plafoniere a led.

Dopo aver raccolto le immagini, è stata necessaria una fase di preprocessing. Le foto sono state classificate manualmente in modo da suddividerle nei vari stati delle varie tipologie. Ho implementato una procedura software per croppare automaticamente le immagini a partire dai QR code. Tuttavia, è possibile che tale procedura non rilevi i QR code nell'immagine. Infatti può capitare che l'operatore non inquadri pienamente i QR code o che l'immagine sia sfocata. Nei casi in cui la procedura non è in grado di ritagliare le immagini è necessario cropparle manualmente.

La Tabella 4.1 riporta, per ogni dispositivo usato, il numero di immagini raccolte per ogni tipologia.

Tipologia	Ecom	Xiaomi	Iphone 8	Totale
Tipologia window-panel finestra	1736	560	505	2801
Tipologia window-panel pannello	1219	567	315	2101
Tipologia knob	226	77	129	432
Tipologia switch	1115	769	580	2464
Tipologia lever	918	292	329	1539
Tipologia lever pannello	1121	415	541	2077

Tabella 4.1: Numero totale di immagini del dataset per ogni tipologia di interruttore

4.3 Training e valutazione dei modelli

Le reti neurali convoluzionali progettate per il riconoscimento degli interruttori sono state sviluppate su un noto framework di Deep Learning chiamato Pytorch, sviluppato principalmente dal Facebook AI Research (FAIR) group. Pytorch ha guadagnato una enorme popolarità fra gli sviluppatori grazie alla combinazione di semplicità ed efficienza.
Le reti sono state addestrate utilizzando la GPU NVIDIA Tesla K80.

Come anticipato nella Sezione precedente, poichè il dataset a disposizione non è sufficientemente grande per poter raggiungere livelli di performance adeguate, sono state utilizzate delle reti preaddestrate, aggiornandone i pesi tramite il nostro training set. Sono state scelte alcune tra le più famose reti neurali preaddestrate, come AlexNet, VGGnet16 e Resnet18 e sono stati confrontati i risultati ottenuti con le varie reti.

4.4 Integrazione con l'applicazione esistente

Una volta sviluppati i modelli di Deep Learning, è stato necessario integrarli nel processo di sezionamento dell'impianto. Per far ciò sono stati sviluppati dei microservizi in Python che, dopo aver ricevuto le immagini dall'operatore, interrogano i modelli di rete neurali e restituiscono la predizione.
I microservizi sono stati creati utilizzando Flask, un microframework web per la realizzazione di applicazioni web in Python.

I microservizi implementati saranno approfonditi nella Sezione 5.4 .

4.5 Test e rilascio in produzione

L'ultima fase del progetto è stata quella di testing, sia del corretto funzionamento dei modelli di Deep Learning sia del corretto funzionamento dei microservizi. Sono stati effettuati i test in sede insieme al cliente che ha potuto provare direttamente le nuove funzionalità dell'applicazione. Il Capitolo 6 illustrerà nel dettaglio i risultati ottenuti.

Capitolo 5

Sviluppo del Lavoro

5.1 Descrizione delle quattro tipologie

In questa Sezione verranno descritte nel dettaglio le quattro tipologie di interruttori.

5.1.1 Tipologia window-panel

L'interruttore di Tipologia window-panel è formato da due porzioni: una finestra collocata in alto sul lato sinistro contenente al suo interno tre fori posti uno sotto l'altro; una pannello in basso contenente due piastrelle di forma quadrata, al cui interno è presente una linea che può essere verticale (ad indicare passaggio di corrente) o orizzontale (ad indicare interruzione di corrente). Lo stato dell'interruttore di Tipologia window-panel è dato dalla combinazione dello stato della finestra e di quello del pannello.

I possibili stati della finestra dell'interruttore di Tipologia window-panel sono:

On

Nello stato On, il foro centrale della finestra ha al suo interno una linea verticale bianca.

Figura 5.1: finestra della Tipologia window-panel - On

Off

Nello stato Off, il foro centrale della finestra ha al suo interno un cerchietto nero.

Figura 5.2: finestra della Tipologia window-panel - Off

I possibili stati del pannello dell'interruttore di Tipologia window-panel sono:

Inserito

Nello stato Inserito, la piastrella di sinistra ha l'interno in posizione verticale e la piastrella di destra ha l'interno in posizione orizzontale.

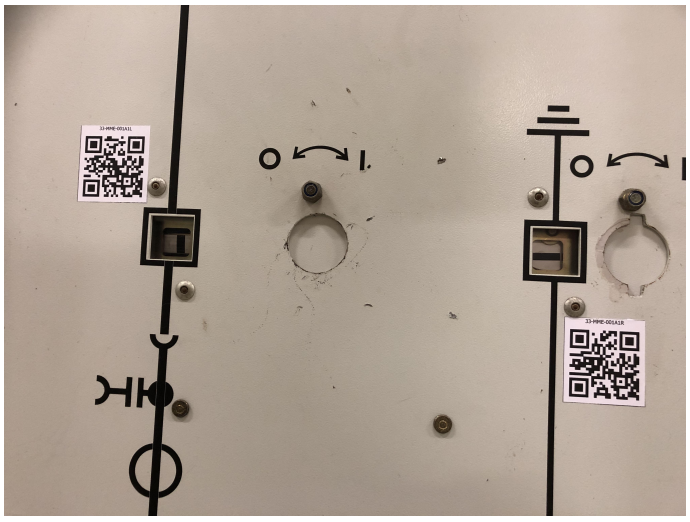


Figura 5.3: Pannello della Tipologia window-panel - Inserito

Estratto

Nello stato Estratto, la piastrella di sinistra ha l'interno in posizione orizzontale e la piastrella di destra ha l'interno in posizione orizzontale.

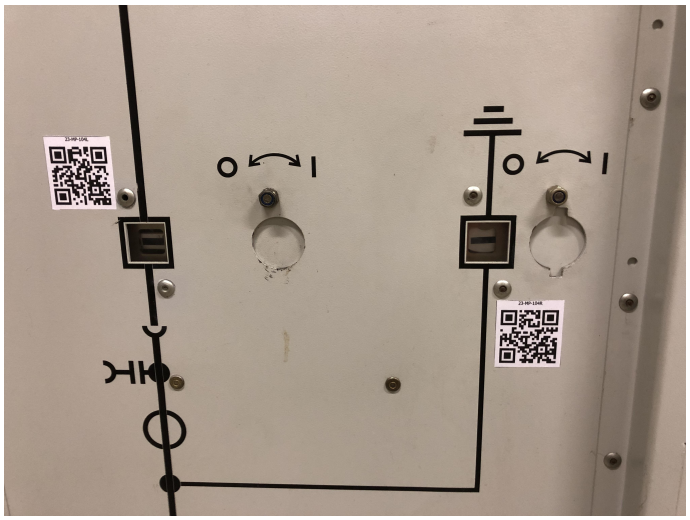


Figura 5.4: Pannello della Tipologia window-panel - Estratto

Messo a terra

Nello stato Messo a terra, la piastrella di sinistra ha l'interno in posizione orizzontale e la piastrella di destra ha l'interno in posizione verticale.

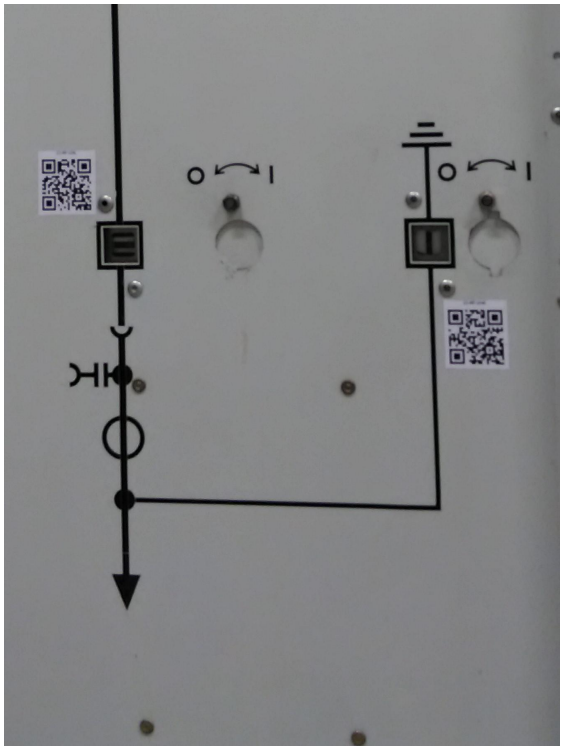


Figura 5.5: Pannello della Tipologia window-panel - Messo a terra

In base agli stati di finestra e pannello, i possibili stati per la Tipologia window-panel sono:

- Inserito - On
- Inserito - Off
- Estratto - On
- Estratto - Off
- Messo a terra - On
- Messo a terra - Off

5.1.2 Tipologia knob

L'interruttore di Tipologia knob è formato da: un blocco quadrato di colore grigio che costituisce la base dello stesso, con fascia verticale rossa sulla destra; una maniglia di colore grigio, con la parte centrale di colore bianco.

L'interruttore di Tipologia knob può assumere uno dei seguenti stati:

On

Nello stato On, la maniglia di colore grigio è in posizione orizzontale

Figura 5.6: Tipologia knob - On

Off

Nello stato Off, la maniglia di colore grigio è in posizione verticale.

Figura 5.7: Tipologia knob - Off

Estratto

Nello stato Estratto, la maniglia di colore grigio è in posizione verticale e alla parte centrale di colore bianco, che è sollevata, vi è agganciata una tenaglia di sicurezza. Quando l'interruttore è nello stato Estratto, significa che è stato messo in sicurezza.

Figura 5.8: Tipologia knob - Estratto

5.1.3 Tipologia switch

L'interruttore di Tipologia switch è formato da un grande pannello di colore bianco sul quale sono inseriti: quattro led di forma circolare nella parte alta, due dei quali di colore bianco e gli altri due di colore verde e rosso; una barra rettangolare nella parte medio-bassa di colore nero, con al centro una levetta bianca che può essere alzata o abbassata.

L'interruttore di Tipologia switch può assumere uno dei seguenti stati:

On

Nello stato On, la levetta posta all'interno della barra nera rettangolare nella parte medio-bassa del pannello è posizionata verso l'alto.

Figura 5.9: Tipologia switch - On

Off

Nello stato Off, la levetta posta all'interno della barra nera rettangolare nella parte medio-bassa del pannello è posizionata verso il basso.

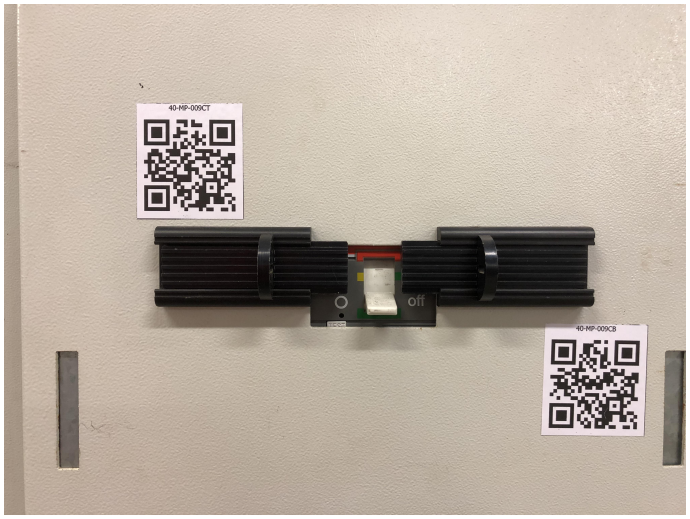


Figura 5.10: Tipologia switch - Off

Estratto

Il riconoscimento di questo stato è dato dall'estrazione fisica del pannello dalla sua ubicazione e dalla conseguente identificazione di un marker (QR Code) posto all'interno della porzione non visibile con l'interruttore inserito. Quando l'interruttore è nello stato Estratto, significa che è stato messo in sicurezza.

Figura 5.11: Tipologia switch - Estratto

5.1.4 Tipologia lever

L'interruttore di Tipologia lever è formato da due porzioni fondamentali: una leva di metallo di colore nero sul lato destro che può trovarsi in posizione orizzontale o in posizione verticale e un pannello di colore nero sul lato sinistro su cui sono presenti 3 led di colore bianco, verde e rosso oltre ad un display di controllo. Tale pannello, nel caso in cui la leva sia in posizione verticale, può essere estratto.

L'interruttore di Tipologia lever può assumere uno dei seguenti stati:

On

Nello stato On, la leva di destra è in posizione orizzontale e il pannello di sinistra non è estratto

Figura 5.12: Tipologia lever - On

Off

Nello stato Off, la leva di destra è in posizione verticale e il pannello di sinistra non è estratto

Figura 5.13: Tipologia lever - Off

Estratto

Nello stato Estratto, la leva di destra è in posizione verticale e il pannello di sinistra è estratto. Eventualmente può essere applicata una tenaglia alla leva. Quando l'interruttore è nello stato Estratto, significa che è stato messo in sicurezza.

Figura 5.14: Tipologia lever - Estratto

5.2 QR code Reader e crop automatico

Ogni interruttore ha un codice identificativo. Quando l'operatore scatta la foto dell'interruttore che verrà analizzata dalla rete neurale , c'è il rischio che nella foto vengano inquadrati più interruttori. In tal caso la rete neurale non saprebbe quale degli interruttori classificare. Pertanto ho deciso di applicare dei marker nell'angolo in alto a sinistra e in basso a destra dell'interruttore. Ognuno di questi marker codifica all'interno il codice dell'interruttore.

Sono stati valutati i vantaggi e gli svantaggi di diverse tipologie di marker

- Aztec Code: è un codice a barre bidimensionale così chiamato per la somiglianza dello schema centrale del codice ad una piramide azteca. Tale soluzione è stata scartata poichè le librerie Python presenti in rete riguardano principalmente la generazione degli Aztec Code e non la detection. Inoltre la documentazione presente è minima.

- Aruco Code: è un marker quadrato composto da un ampio bordo nero e una matrice binaria interna che ne determina l'identificatore. Anche questa tipologia di marker non è stata scelta perchè è possibile codificare al suo interno pochissima informazione, in particolare solo numeri.

- **QR Code**: Un QR Code è un codice a barre bidimensionale composto da moduli neri disposti all'interno di uno schema bianco di forma quadrata. **Abbiamo deciso di utilizzare i QR code** per la loro capacità di codificare una buona mole di informazione, sia numerica che testuale. Inoltre in rete sono presenti numerose librerie in Python per la detection, ben documentate.

- Bar Code: sono dei marker che rappresentano i dati tramite barre verticali, variandone la larghezza e la distanza tra di esse. Questo tipo di marker rappresenta una buona scelta, ma abbiamo deciso di non adottarli in quanto abbiamo appurato empiricamente che le librerie Python sono meno performanti rispetto a quelle dei QR code.

Un primo vantaggio derivante dall'applicazione dei QR code è quello di poter tagliare l'immagine in modo da poter ottenere esclusivamente la porzione di interesse. L'operatore quando invia la foto al microservizio che interroga la rete neurale per ottenere la classificazione, invia anche il codice identificativo dell'interruttore. In questo modo il microservizio è in grado di selezionare l'interruttore corretto all'interno dell'immagine nel caso ne fossero presenti più di uno. Per poter effettuare il crop dell'immagine correttamente, è necessario specificare all'interno della codifica del QR code se si tratta di quello in alto a sinistra o di quello in basso a destra. Per risolvere tale problema è stato deciso di aggiungere come suffisso la lettera T (Top) per quello in alto a sinistra e B (Bottom) a quello in basso a destra.

L'applicazione dei QR code è molto utile anche per risolvere un altro problema che potrebbe ridurre notevolmente le performance delle reti neurali. Se per qualsiasi motivo il dispositivo dell'operatore inviasse l'immagine ruotata di 90°o 270°, le reti neurali di alcune Tipologie, come knob e lever, fornirebbero una classificazione errata dell'interruttore. Infatti gli stati dell'interruttore di queste tipologie si basano

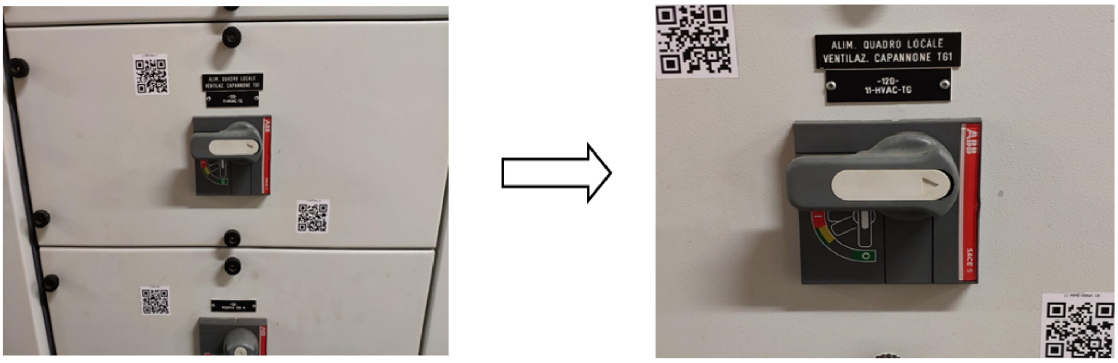


Figura 5.15: Esempio di crop dell' immagine.
L'id di questo interruttore è 11-MME-006B-1. Il QR code in alto a sinistra codificherà il testo 11-MME-006B-1T, mentre quello in basso a destra codificherà il testo 11-MME-006B-1B.
In questo esempio il microservizio rileverebbe tre qr code, ma solo due di questi codificano l'identificativo dell'interruttore e quindi è in gradio di ritagliare l'immagine correttamente.

sull'orientamento di una leva sull'asse verticale o orizzontale.
In base a come sono stati applicati i QR code quello Top è sempre alla sinistra di quello Bottom ed è posizionato più in alto. Se questi vincoli non sono rispettati, il microservizio è in grado di riconoscere che l'immagine è ruotata ed è in grado di capire di quanti gradi ruotarla per riportarla nello stato conforme.

Figura 5.16: Immagine ruotata di 90°. Il microservizio riconosce che l'immagine è ruotata di 90°perchè il QR code Top si trova alla destra di quello Bottom.

Figura 5.17: Immagine ruotata di 270°. Il microservizio riconosce che l'immagine è ruotata di 270°perchè il QR code Top e più in basso di quello Bottom.

Può capitare a volte che la foto fatta dall'operatore sia sfocata. In tal caso c'è il rischio che uno o più QR code non vengano rilevati dal microservizio. Se nessuno dei due viene rilevato, viene inviato un messaggio di errore all'operatore (si rimanda alla Sezione 5.4). Se viene rilevato un solo QR code è possibile comunque ritagliare correttamente l'immagine grazie ad alcune tecniche statistiche che sono state implementate. Per ogni Tipologia sono state analizzate tutte le foto del dataset ed è stata calcolata la posizione relativa dei QR code all'interno dell'immagine. Grazie a questi dati statistici è possibile ritagliare l'immagine a partire da un solo QR code, poichè si conosce la distanza media relativa tra i QR code all'interno dell'immagine. Nel caso di rilevamento di un solo QR code non è tuttavia possibile ruotare l'immagine.
La figura 5.18 illustra il funzionamento della procedura di crop dell'immagine.

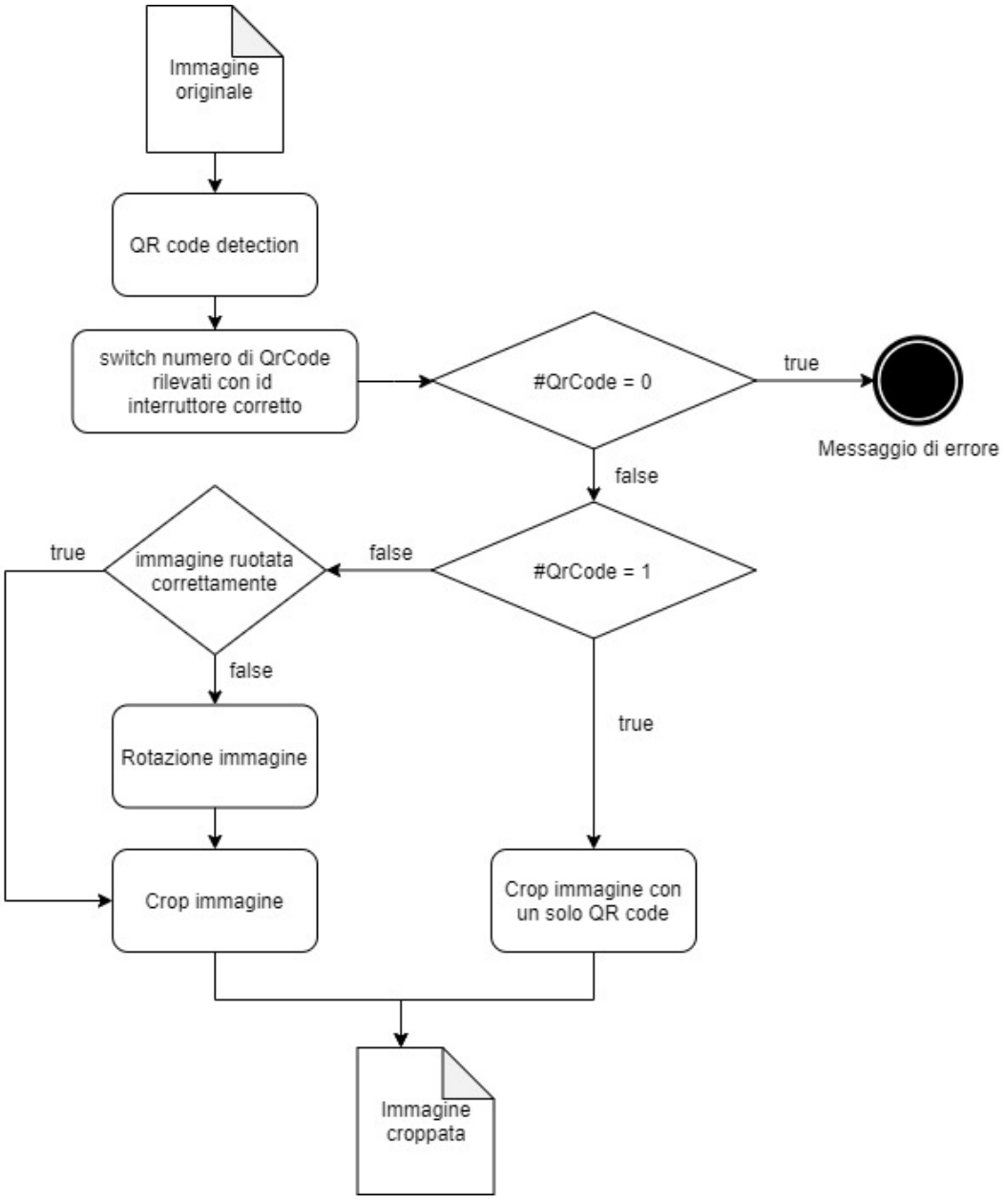


Figura 5.18: Procedura di crop dell'immagine

La libreria utilizzata per il cropping dell'immagine è *Pyzbar* che, data un'immagine, tramite il metodo *decode*(), è in grado di individuare i QR code presenti. Tale metodo restituisce una lista di oggetti, ognuno dei quali rappresenta un QR code rilevato. Ogni oggetto contiene al suo interno il testo codificato nel QR code e le coordinate dei quattro vertici del QR code. Una volta ottenuti i QR code Bottom e Top e le loro posizioni in pixel, è possibile ritagliare l'immagine tramite il metodo *crop*() della libreria PIL che accetta come parametri le coordinate del vertice in alto a sinistra e quello in basso a destra dell'area da ritagliare.

5.3 Reti neurali

Come anticipato nei capitoli precedenti, è stata sviluppata una rete neurale convoluzionale per ogni tipologia di interruttore. Per alcune tipologie, come Tipologia window-panel e Tipologia lever, sono state sviluppate due reti neurali, come approfondito nelle Sezioni 5.3.1 e 5.3.4. Sono stati condotti esperimenti con diverse reti pre-addestrate in modo da poterne confrontare le prestazioni e scegliere quella che maggiormente si adattava alle nostre esigenze. Le reti preaddestrate utilizzate sono state Resnet18, VGG16 e AlexNet per tutte le tipologie.
Ho deciso di impostare 10 epoche per il training delle varie reti perchè si è visto empiricamente che sono sufficienti per farle convergere.

Le tecniche di data augmentation utilizzate sono:

- HorizontalFlipping
  L'immagine viene capovolta orizzontalmente
- RandomRotation
  Sono state applicate all'immagine piccole rotazioni in senso orario e antiorario di massimo 6°in modo da simulare eventuali rotazioni dello schermo del dispositivo durante lo scatto della foto.
- ColorJitter
  Viene modificata la luminosità e il contrasto dell'immagine

Durante la progettazione delle reti neurali, è stato necessario scegliere alcuni parametri molto importanti di seguito descritti.

Funzione di Loss

Come riportato nel Capitolo 2, la funzione di Loss è una funzione di costo e serve per valutare quanto bene l'algoritmo modella il dataset, ovvero permette di calcolare l'errore compiuto dalla rete nella fase di forward in modo da poter aggiornare i pesi della rete di conseguenza.
Esistono diverse funzioni di loss. Per le nostre reti neurali abbiamo deciso di utilizzare la *CrossEntropy Loss* che è molto utilizzata nei problemi di classificazione. Questa funzione di loss è rappresentata dalla seguente formula:

$$-(y \log p + (1-y) \log (1-p))$$

Da questa formula si nota che quando la classe vera è 1, il termine $(1-y) \log (1-p)$ si annulla, mentre quando la classe vera è 0, $y \log p$ è nullo. Ciò significa che il suo valore è semplicemente rappresentato dalla moltiplicazione del logaritmo della probabilità calcolata per la classe predetta.
Abbiamo deciso di utilizzare questa funzione di loss perchè penalizza fortemente le alte confidenze nelle predizioni errate [21]. Infatti, nel nostro caso di studio, affermare con una confidenza elevata una predizione errata è molto più pericoloso rispetto ad affermare con bassa confidenza una predizione corretta.

La Figura 5.19 mostra l'andamento di tale funzione.

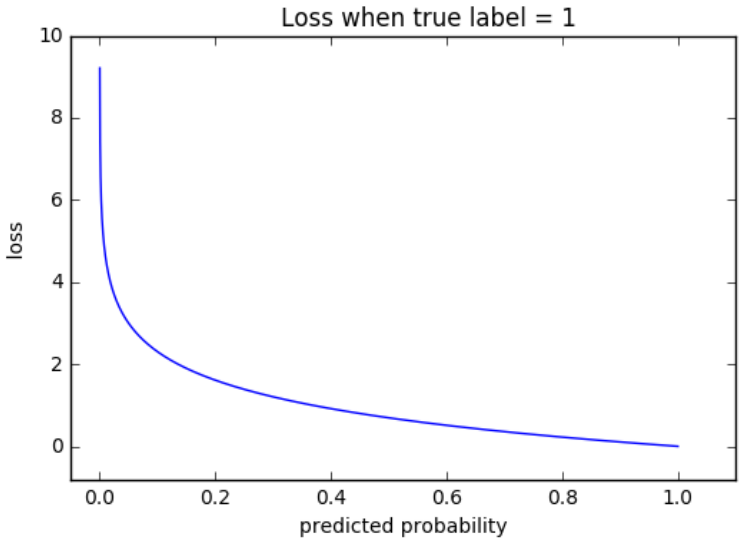


Figura 5.19: Cross Entropy Loss[21]

Optimizer

Il ruolo dell'optimizer è quello di aggiornare i parametri del modello in risposta all'output della funzione di loss. L'ottimizzatore scelto per le reti neurali sviluppate è l'ottimizzatore *Adam* che sta per *adaptive moment estimation* [22].

Adam è un algoritmo di ottimizzazione che può essere usato al posto del classico stochastic gradient descent (descritto nel Capitolo 2), per aggiornare i pesi della rete. Adam si differenzia dallo stochastic gradient descent perchè, mentre quest'ultimo mantiene un singolo learning rate per tutti gli aggiornamenti dei pesi e durante tutto il training, adam adatta il learning rate in base al momento del primo ordine (la media), utilizzando anche il momento del secondo ordine del gradiente (la varianza non centrata) [23].

Ho scelto adam perchè è un algoritmo molto popolare nel campo del Deep Learning e perchè permette di raggiungere ottime prestazioni rapidamente. Nel paper in cui fu presentato, intitolato *Adam: A Method for Stochastic Optimization* [24], è stato dimostrato emipiricamente che Adam è in grado di risolvere con ottimi risultati problemi pratici del Deep Learning [23].

Learning rate

Il learning rate è un parametro che viene utilizzato per controllare il valore di aggiornamento dei pesi rispetto al valore del gradiente. Ho scelto un valore piccolo (0.001) per questo parametro. Infatti modificare i valori dei pesi aggiungendo o sottraendo un valore troppo elevato potrebbe causare il non raggiungimento del valore ottimale per un certo peso, non permettendo alla rete di convergere. Allo stesso tempo tale valore non deve essere troppo piccolo perchè ci sarebbe il rischio di rimanere intrappolati in un minimo locale, non riuscendo più a convergere.

5.3.1 Tipologia window-panel

L'interruttore di Tipologia window-panel è il più complesso dei quattro analizzati. Infatti si compone di due parti, la finestra e il pannello, e lo stato finale è dato dalla combinazione dello stato delle due parti. Inoltre, sia lo stato della finestra che quello del pannello si basano su elementi molto specifici all'interno dell'immagine. La finestra si basa su un cerchietto o una linea verticale all'interno del foro centrale dei tre che compongono la finestra stessa. Il pannello si basa a sua volta sulla combinazione dello stato di due piastrelle, una a sinistra e una a destra. Entrambe possono avere all'interno una linea orizzontale o una verticale.
L'approccio utilizzato si diversifica rispetto a quello utilizzato per le altre tipologie. Infatti sono state addestrate due reti neurali: una per la finestra e una per il pannello.

Le prime reti neurali sviluppate prendevano l'immagine completa e provavano a stabilire lo stato del pannello e quello della finestra. I risultati non sono stati ottimali proprio a causa della specificità degli elementi che caratterizzano la distinzione tra i vari stati all'interno dell'immagine. Dunque ho deciso di cambiare strategia.
Ho introdotto un modulo di object detection in grado di rilevare il foro all'interno della finestra e la piastrella all'interno del pannello. In questo modo si può ritagliare una porzione molto specifica dell'immagine da far analizzare in un secondo step alla rete neurale, aumentandone le performance.

Come abbiamo visto, l'image classification ha lo scopo di assegnare un'etichetta di classe all'immagine di input. L'object detection invece ha lo scopo di classificare vari oggetti e stabilirne la posizione all'interno dell'immagine.
Per l'object detection è stato utilizzato Pytorch ed in particolare l'architettura Faster-RCNN, presentata per la prima volta in un paper del 2015 [25]. Questa è una tra le più famose architetture di object detection. Essa deriva dall'architettura RCNN che è composta dalle seguenti parti:

1. Un algoritmo di *region proposal* per generare dei box attorno a possibili oggetti nell'immagine
2. Uno step di estrazione delle features dei possibili oggetti individuati, tramite CNN
3. Un layer di classificazione per predire a quale classe gli oggetti individuati appartengono.
4. Un layer per rendere più precise le coordinate dei box che individuano gli oggetti nell'immagine

La Faster-RCNN rappresenta è un'evoluzione rispetto alla RCNN. Essa risulta più performante in quanto usa un'altra rete CNN (l'RPN) per generare le region proposal. Ciò non solo riduce notevolmente il tempo di generazione di queste regioni, ma permette anche allo stage di generazione delle region proposal di condividere layer con il successivo stage di feature detection.

L'operazione di object detection e di crop dell'area di interesse sono rappresentati in Figura 5.20 per il pannello e Figura 5.21 per la finestra.

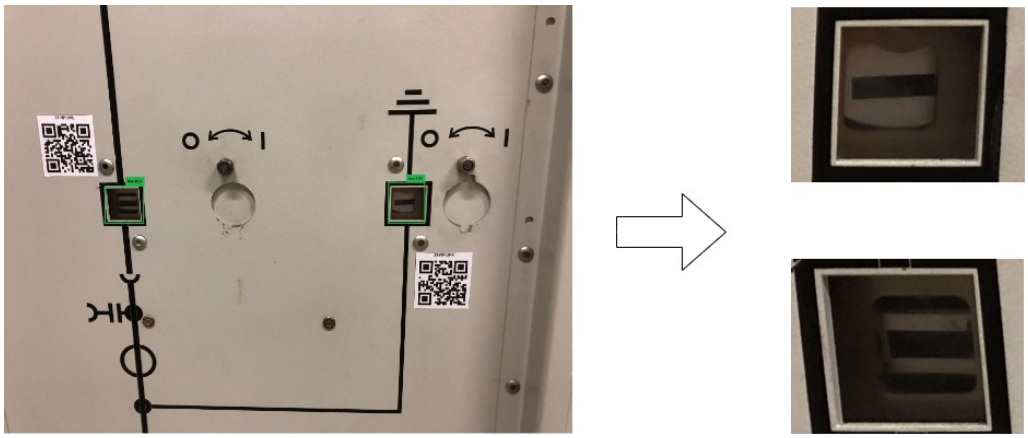


Figura 5.20: Object detection per individuare e croppare l'area di interesse per il riconoscimento dello stato del pannello Tipologia window-panel

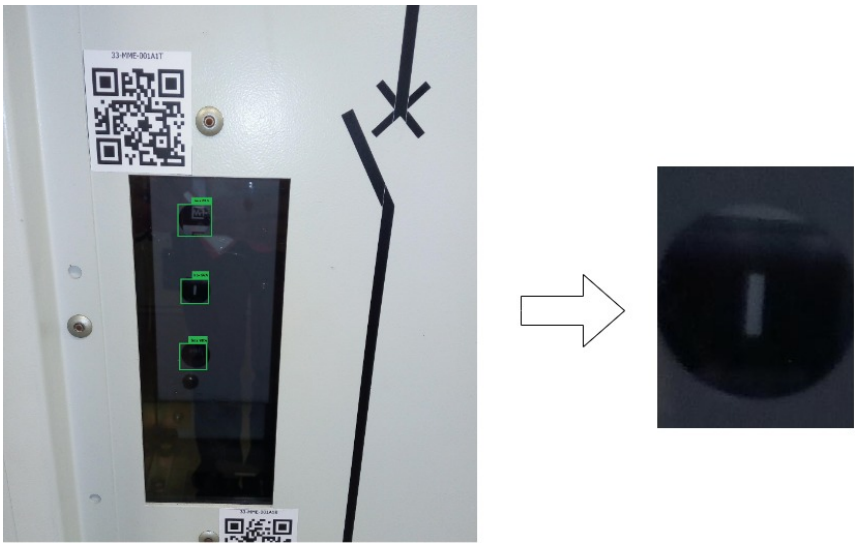


Figura 5.21: Object detection per individuare e croppare l'area di interesse per il riconoscimento dello stato della finestra di Tipologia 1 window-panel

La rete neurale utilizzata per classificare il pannello di Tipologia window-panel riceve l'immagine di ciascuna delle due piastrelle ed è in grado di stabilire se all'interno è presente una linea orizzontale o verticale. In base alla combinazione della predizione delle due piastrelle è possibile stabilire lo stato del pannello, secondo le regole esposte nella Sezione 5.1.1. Affinchè tale operazione abbia successo, è necessario distinguere quale delle due piastrelle è quella di sinistra e quale quella di destra. Questa distinzione può essere fatta a partire dalle coordinate delle piastrelle individuate dal modulo di object detection.
Nell'esempio di Figura 5.20, la piastrella di sinistra e quella di destra presentano entrambi una linea orizzontale, dunque lo stato finale predetto per il pannello sarà Estratto.

La rete neurale per classificare la finestra di Tipologia window-panel riceve l'immagine del foro che può contenere il cerchietto (stato Off) o la linea verticale (stato On) ed è in grado di distinguere questi due stati. Anche in questo caso per individuare il

foro centrale, vengono calcolate le posizioni relative dei fori individuati dal modulo di object detection.

Il numero di immagini utilizzate per il training della rete della finestra di Tipologia 1 window-panel sono indicati nella Tabella 5.1, mentre quelle per il pannello sono in Tabella 5.2

Stato	**Numero di immagini**
On	1394
Off	1407

Tabella 5.1: Numero totale di immagini del dataset per ogni stato della finestra dell' interruttore di Tipologia window-panel

Stato	**Numero di immagini**
Inserito	254
Estratto	307
Messo a terra	1540

Tabella 5.2: Numero totale di immagini del dataset per ogni stato del pannello dell' interruttore di Tipologia window-panel

5.3.2 Tipologia knob

Gli stati possibili di questo interruttore sono On, Off e Estratto (Sezione 5.1.2).

Il numero di immagini di questa tipologia che compongono il dataset sono indicati nella Tabella 5.3

Stato	Numero di immagini
On	170
Off	163
Estratto	99

Tabella 5.3: Numero totale di immagini del dataset per ogni stato dell' interruttore di Tipologia knob

5.3.3 Tipologia switch

Gli stati possibili di Tipologia switch sono On,Off e Estratto. Quest'ultimo stato è individuato dall'estrazione fisica dell'interruttore, come riportato nella Sezione 5.1.3. Pertanto viene riconosciuto esclusivamente con la detection del QR code applicato lateralamente, visibile solo se il cassetto è estratto. L'identificativo dell'interruttore codificato nel QR code ha come suffisso lettera la I, in modo da poter essere distinto dai due QR code Bottom e Top usati per croppare l'immagine.
Per distinguere tra gli stati On e Off è stata addestrata una rete neurale convoluzionale utilizzando i parametri discussi precedentemente.
I risultati ottenuti saranno approfonditi nella Sezione 6.3.

Il numero di immagini di questa tipologia che compongono il dataset sono indicati nella Tabella 5.4

Stato	Numero di immagini
On	1108
Off	1052
Estratto	304

Tabella 5.4: Numero totale di immagini del dataset per ogni stato dell' interruttore di Tipologia switch

5.3.4 Tipologia lever

Come esposto nella Sezione 5.1.4 gli stati della Tipologia lever sono On, Off e Estratto. Poichè lo stato Estratto è individuato dall'estrazione del pannello a sinistra della leva, sono state sviluppate due reti neurali: una rete che individua se la leva è in posizione verticale (stato Off) o orizzontale (stato On); una seconda rete che, data l'immagine del pannello , controlla se è estratto o meno. In particolare viene effettuato un nuovo crop dell'immagine in modo da selezionare esclusivamente la porzione contenente il pannelo come mostrato in Figura 5.22 .

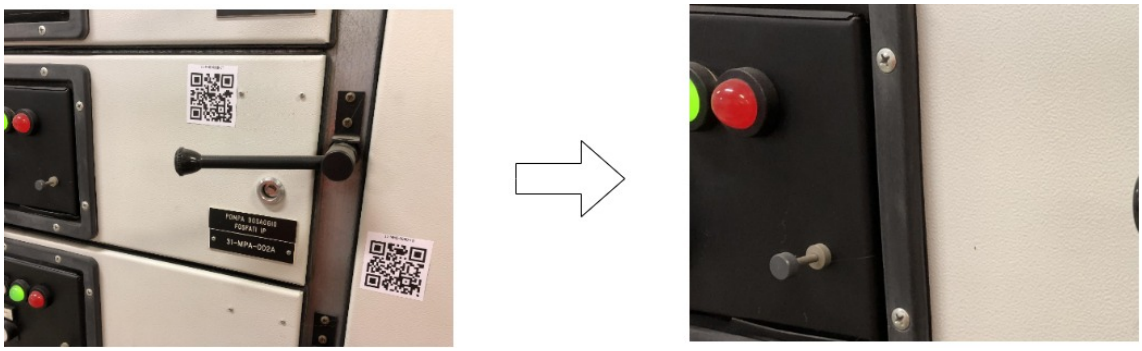


Figura 5.22: Crop del pannello dell'interruttore di Tipologia lever

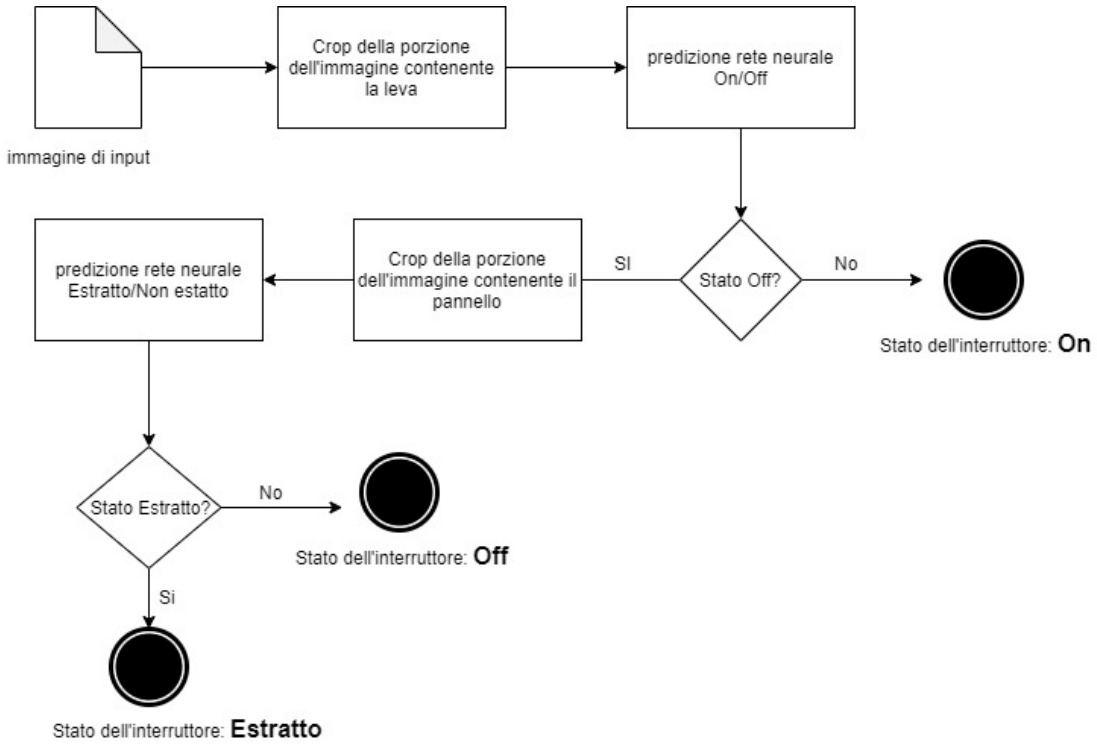


Figura 5.23: Flusso logico per determinare lo stato dell'interruttore di Tipologia lever

L'interruttore può essere nello stato Estratto solo se la leva è in posizione verticale. Dunque la seconda rete neurale che distingue tra pannello estratto e pannello non estratto viene interrogata esclusivamente se lo stato della leva rilevato dalla prima rete neurale è Off. Il flusso logico di tali operazioni è rappresentato in Figura 5.23 .

Per facilitare la distinzione tra stato Estratto e stato Non Estratto, è stato applicato un marker nella parte interna del pannello in modo da renderlo visibile quando questo è estratto e non visibile quando non lo è. Il marker è composto da 4 ellissi nere su sfondo bianco (Figura 5.14).

Il numero di immagini di questa tipologia che compongono il dataset sono indicati nella Tabella 5.5 per la rete neurale On/Off, e in Tabella 5.6 per la rete Estratto/-Non Estratto.

Stato	**Numero di immagini**
On	753
Off	786

Tabella 5.5: Numero totale di immagini del dataset per ogni stato dell' interruttore di Tipologia lever

Stato	**Numero di immagini**
Estratto	1083
Non estratto	994

Tabella 5.6: Numero totale di immagini del dataset per ogni stato del pannello dell' interruttore di Tipologia lever

5.4 Microservizi

Sono stati implementati due moduli di microservizi Python distinti, chiamati *switch-recognition-engine* e *switch-recognition-classifier*.

switch-recognition-engine

switch-recognition-engine espone un solo microservizio che effettua operazioni di pre-processing dell'immagine. In particolare effettua le operazioni di detection dei QR code e di crop dell'immagine descritte nella Sezione 5.2. Gestisce inoltre tutte le situazioni anomale, come ad esempio parametri della richiesta errati o errori dovuti all'immagine sfocata. Una volta ottenuta l'immagine croppata, questa viene inviata al modulo switch-recognition-classifier per ottenere la predizione della rete neurale. Una volta ottenute, il servizio salva su un database non relazionale, chiamato MongoDB, l'immagine originale e quella croppata codificate in base64 in modo da alimentare il dataset a disposizione.
Per chiamare questo servizio è necessario effettuare una chiamata HTTP POST contenente alcune informazioni nel body sotto forma di form-data. I campi richiesti nella request sono descritti nella Tabella 5.7 .

Nome Campo	Tipologia	Descrizione	Note
image	FileStorage	Rappresenta l'immagine dell'interruttore	Campo obbligatorio. Nel caso di Tipologia window-panel questa è la foto del pannello
additionalImage	FileStorage	Rappresenta l'immagine della finestra dell'interruttore di Tipologia window-panel	Campo obbligatorio solo per Tipologia window-panel
id	String	Rappresenta l'identificativo dell'interruttore	Campo obbligatorio
type	Int	Rappresenta la tipologia dell'interruttore	Campo obbligatorio

Tabella 5.7: Campi della request per la chiamata al microservizio del modulo *switch-recognition-engine*

Il microservizio esposto in questo modulo fornisce una risposta in formato JSON. Se le operazioni sono andate a buon fine, i campi della risposta sono quelli descritti nella Tabella 5.8. Se invece si è verificato un errore, i campi della risposta sono quelli descritti nella Tabella 5.9.

Nome Campo	Tipologia	Descrizione
success	Boolean	Indica se la chiamata è andata a buon fine
id	String	Rappresenta l'identificativo dell'interruttore
status	String	Rappresenta lo stato dell'interruttore predetto dalla rete neurale
message	String	Messaggio informativo della response

Tabella 5.8: Campi della response del microservizio del modulo *switch-recognition-engine* in caso esito positivo

Un esempio del body della risposta con esito positivo in formato JSON è il seguente:

```
{
    "success": "true"
    "data": {
        "id": "11-HSV-308"
        "status": "on"
        "message": "OK"
    }
}
```

Nome Campo	Tipologia	Descrizione
success	Boolean	Indica se la chiamata è andata a buon fine
message	String	Messaggio informativo della response
code	Int	Codice di errore

Tabella 5.9: Campi della response del microservizio del modulo *switch-recognition-engine* in caso esito negativo

Un esempio del body della risposta con esito negativo in formato JSON è il seguente:

```
{
    "success": "false"
    "data": {
        "message": "Nessun QR code trovato"
        "code": 1
    }
}
```

I possibili messaggi di errore sono riportati nella Tabella 5.10

Codice di errore	Messaggio	Descrizione
1	Nessun QR code trovato	Indica che non è stato possibile rilevare QR code nell'immagine
2	Parametri errati	Indica che uno o più parametri della request sono errati o mancanti
3	ID non coincide con quelli sul qr code	Indica che l'id specificato nella request non corrisponde con quello dei QR code rilevati
4	Errore interno	Indica che si è verificato un errore interno al server

Tabella 5.10: Possibili messaggi di errore nella response del microservizio del modulo *switch-recognition-engine*

switch-recognition-classifier

Questo modulo espone quattro microservizi tramite metodo POST, uno per ogni tipologia. Il modulo di *switch-recognition-engine* una volta ricevuta la chiamata, in base alla tipologia dell'interruttore chiama il servizio corrispondente del modulo *switch-recognition-classifier*, fornendo le immagini da classificare. Ogni microservizio contiene una propria logica interna che dipende dalla tipologia dell'interruttore. Poichè le reti neurali sono distinte per tipologia, ogni servizio richiama la rete neurale corrispondente e, una volta ottenuta la predizione, la restituisce al chiamante, ovvero al modulo di engine, il quale a sua volta restituisce questa informazione all'operatore.

Il flusso logico dell'intero blocco di microservizi è rappresentato in Figura 5.24 .

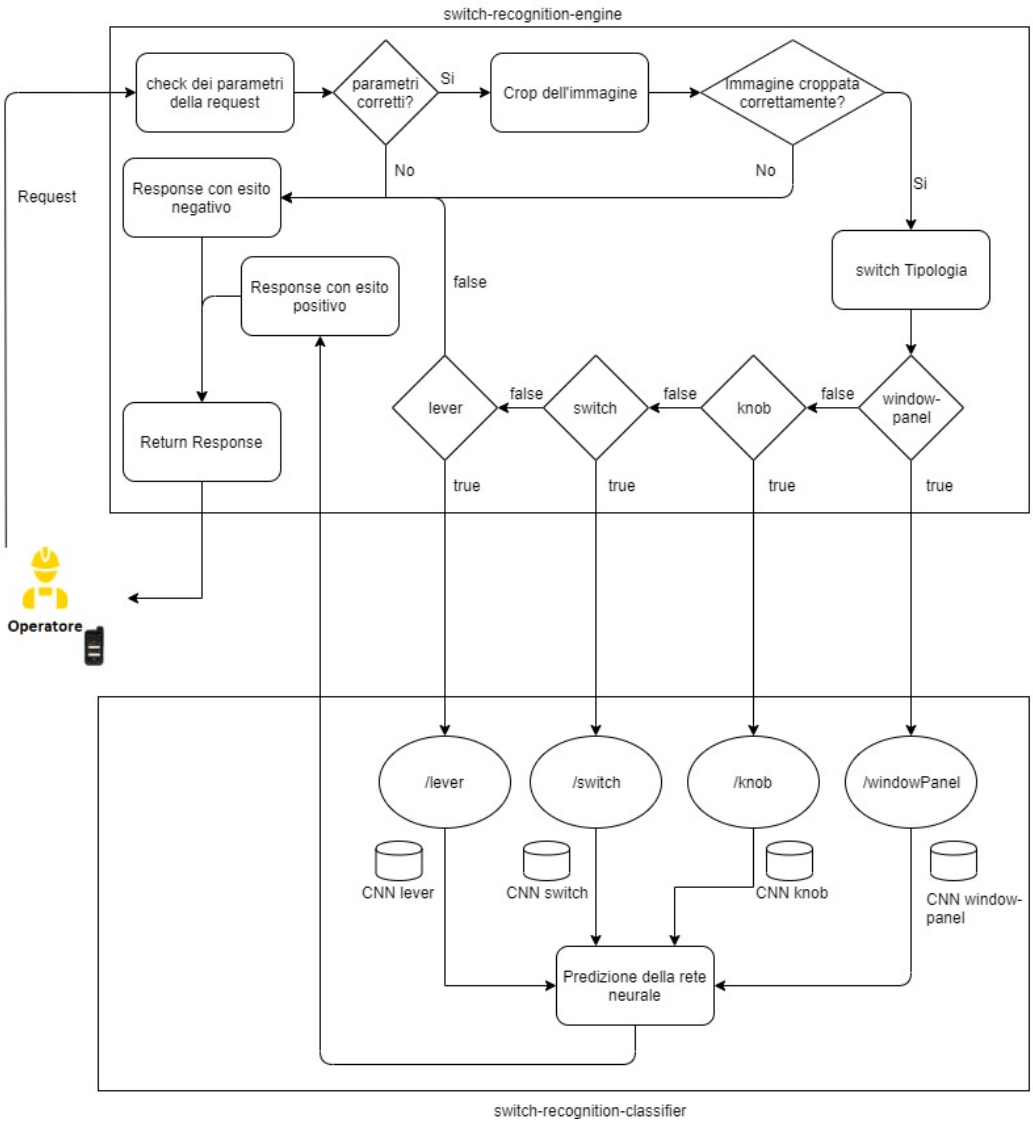


Figura 5.24: Flusso logico dell'API composta dai moduli *switch-recognition-engine* e *switch-recognition-classifier*

Capitolo 6

Risultati sperimentali

Per valutare le reti neurali sono state prese in considerazioni diverse metriche. Oltre all'accuratezza, è stato valutato il tempo necessario per il training della rete neurale e il tempo impiegato dalla rete neurale per predire un nuovo campione. Infatti, poichè il microservizio che fornisce le predizioni deve rispondere il più velocemente possibile per garantire una buona user experience all'operatore, il tempo di predizione della CNN deve essere il più basso possibile. Ho dunque condotto dei test per misurare il tempo impiegato dalle varie reti neurali per fornire la predizione su CPU.

Dopo aver addestrato la rete, in Pytorch è possibile estrarre un file in formato .pth che rappresenta il modello. Questo file può essere utilizzato per caricare in memoria la rete e utilizzarla per effettuare le predizioni. Durante il deploy del modulo python *switch-recognition-classifier*, vengono scaricati i vari modelli da un repository online dove sono stati precedentemente caricati. Dunque il tempo necessario per il deploy dipende soprattutto dalla dimensione di questi file. Pertanto ho deciso di considerare anche la dimensione del file del modello come parametro di valutazione. Inoltre, più un file è di grandi dimensioni, più significa che il modello è complesso e di conseguenza il tempo di predizione della rete dovrebbe aumentare.

Non sempre l'accuratezza è la metrica migliore per valutare un modello. In particolare quando una classificazione errata ha un'importanza diversa in base a quale classe è stata erroneamente classificata, è preferibile utilizzare altre metriche come la recall, precision e F-measure. Nel caso oggetto di studio, indicare che un interruttore è stato messo in sicurezza quando invece non lo è, rappresenta un errore molto più grave rispetto ad indicare che l'interruttore non è stato messo in sicurezza quando invece lo è stato.
La precision può essere vista come la probabilità che le predizioni per la classe C siano corrette ed è definita come il rapporto tra i *true positive*, ovvero il numero di oggetti appartenenti ad una classe etichettati correttamente, e il numero totale di elementi etichettati come appartenenti alla classe:

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

La recall può essere vista come la probabilità di classificare correttamente un campione appartenente alla classe C ed è definita come il rapporto tra il numero di veri positivi e il numero totale di elementi che appartengono effettivamente alla classe:

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

La F-measure è la media armonica di recall e precision ed è così definita:

$$\text{F-measure} = 2 * \frac{precision * recall}{precision + recall}$$

Per valutare un modello di object detection si utilizza una metrica chiamata *mean average precision* (mAP). Un modello di object detection, non solo deve determinare quali e quanti oggetti di una classe sono presenti nell'immagine, ma anche la loro posizione. La metrica mAP si compone a sua volta di due metriche: *Average Precision* (AP) e *Intersection over Union* (IoU). La prima viene utilizzata per valutare il task della classificazione, mentre la seconda per valutare la corretta localizzazione degli elementi all'interno dell'immagine.
L'AP viene calcolata a partire da una curva generata dalle misure di recall e precision. Viene calcolata la media dei valori di precision corrispondenti a 11 valori di recall equidistanziati. Un esempio di curva Recall-Precision è mostrata in Figura 6.1.

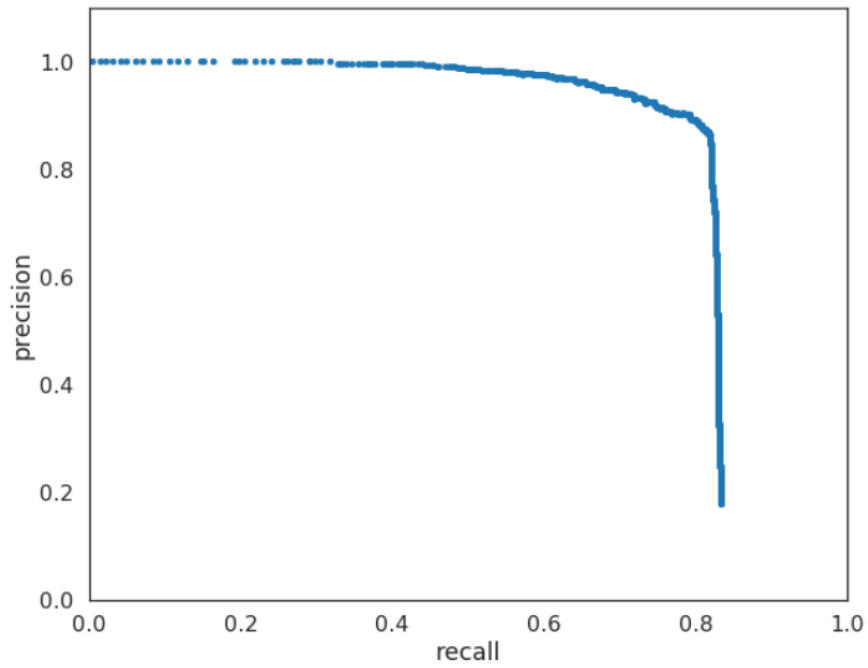


Figura 6.1: Esempio di curva recall - precision. [26]

$$AP = \frac{1}{11} \sum_{Recall_i} Precision(Recall_i)$$

dove $Recall_i = [0, 0.1, 0.2, ..., 1.0]$

L' intersection over union indica se la localizzazione degli oggetti effettuata dal modello di object detection corrisponde alla loro vera localizzazione. Essa è definita come il rapporto tra l'intersezione e l'unione delle aree di localizzazione predette e quelle vere.

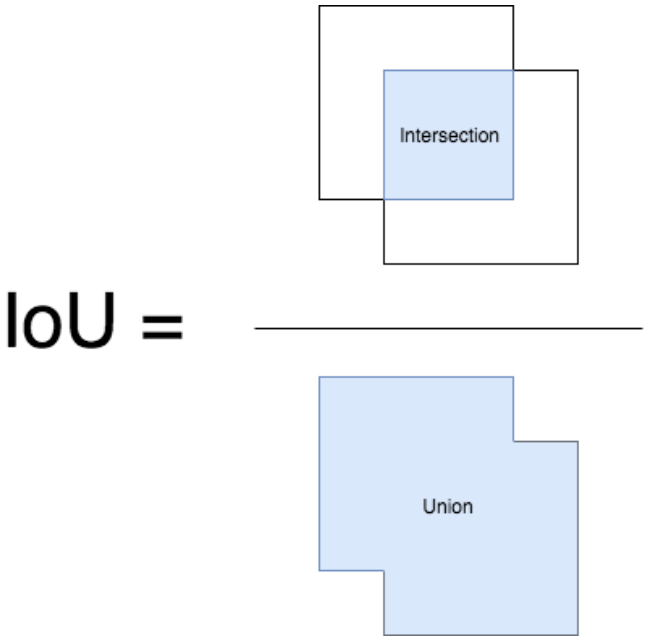


Figura 6.2: Formula per il calcolo della intersection over union. [27]

Il modello di object detection individua la posizione degli oggetti nell'immagine con un certo valore di confidenza. È possibile stabilire una soglia tramite la quale selezionare solo le localizzazioni con valori di confidenza al di sopra della soglia. Il valore finale di mAP è dato dalla media AP su tutte le classi degli oggetti la cui confidenza di localizzazione è sopra la soglia scelta.

Le Sezioni che seguono riportano i risultati ottenuti per le varie tipologie di interruttori. Oltre ad utilizzare diverse reti pre-addestrate per capire quale di queste ottenesse i risultati migliori, è stata addestrata anche una rete neurale composta da due layer convoluzionali, due layer di max pooling e due fully connected layer. Questa CNN nel corso della trattazione verrà chiamata *rete neurale custom*.

6.1 Risultati Tipologia window-panel

Finestra

Per il modello di object detection sviluppato per riconoscere i fori all'interno dell'immagine della finestra dell'interruttore di Tipologia window-panel, è stata impostata la soglia di confidenza dell'IoU a 0.5, dunque solo le localizzazioni degli oggetti con una confidenza al di sopra di 0.5 sono stati considerati true positive. Per il task di classificazione è stata impostata una soglia di confidenza di 0.6.
Il modello ha ottenuto un punteggio di 0.9 di mAP ed è quindi risultato essere affidabile nel riconoscimento e nella localizzazione dei fori della finestra all'interno dell'immagine.

La rete neurale convoluzione usata per riconoscere lo stato On e lo stato Off della finestra dell'interruttore di Tipologia window-panel, è stata addestrata applicando i parametri descritti nella Sezione 5.3.1.
Il dataset è stato diviso in training set e test set rispettivamente con il 70% e il 30% dei campioni. La Tabella 6.1 riporta il numero di immagini per ciascuno dei due dataset.

Stato	Numero di immagini
On	976
Off	985

(a) Training set

Stato	Numero di immagini
On	418
Off	422

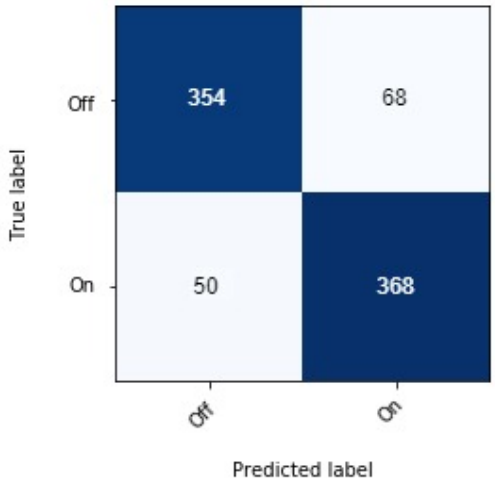
(b) Test set

Tabella 6.1: Training set e Test set della finestra dell'interruttore di Tipologia window-panel

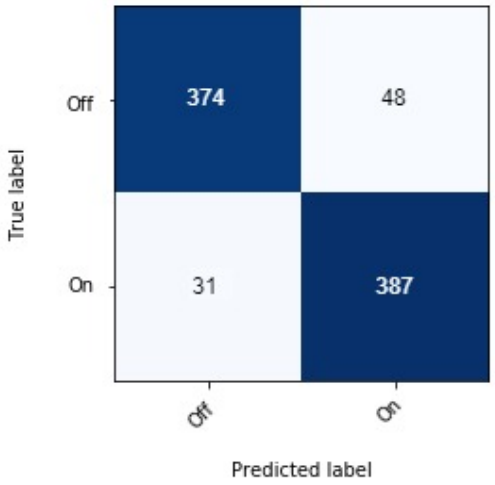
Metrica	Rete Custom	AlexNet	VGGnet16	Resnet18
Accuratezza	85.9%	90.5%	94.6%	98.7%
Recall	87.6%	92.3%	95.9%	99%
Precision	83.9%	88.6%	93.4%	98.3%
F-measure	85.7%	90.4%	94.6%	98.7%
Tempo di training (min)	23	22	28	24
Tempo di predizione medio (ms)	132	183	792	179
Peso della rete (MB)	20	218	512	43

Tabella 6.2: Risultati finestra di Tipologia window-panel

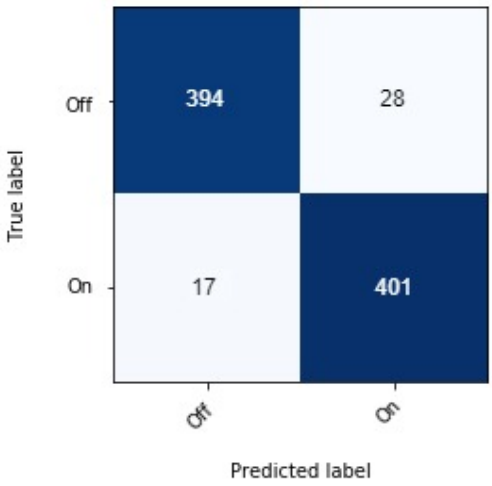
Le figure seguenti riportano le matrici di confusione ottenute per le varie reti neurali utilizzate, i grafici dell'accuratezza e l'andamento della loss durante il training. La Tabella 6.2 riporta i risultati ottenuti.



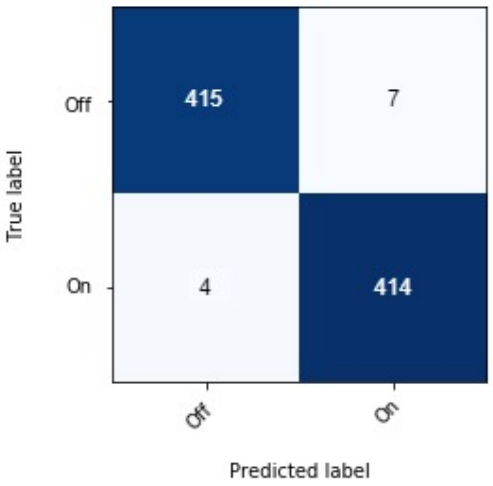
(a) CNN Custom



(b) AlexNet



(c) VGGNet16

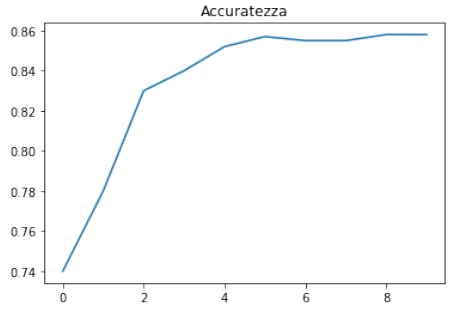


(d) Resnet18

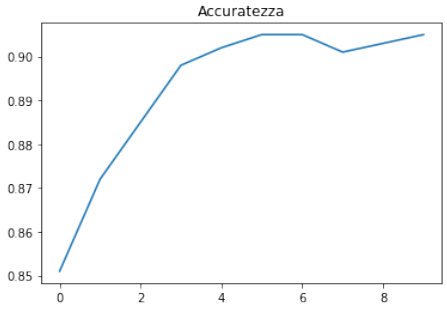
Figura 6.4: Confusion Matrix delle CNN della finestra di Tipologia window-panel

Le performance del modello di rete neurale addestrato per il riconoscimento dello stato On e Off della finestra dell'interruttore di Tipologia window-panel sono leggermente inferiori rispetto a quelle delle altre tipologie. Infatti non sempre l'immagine analizzata dalla rete neurale dopo il crop del modulo di object detection è nitida. A volte le condizioni di luminosità e soprattutto il riflesso dovuto dal vetro della finestra rendono difficoltosa la corretta predizione da parte della rete neurale.

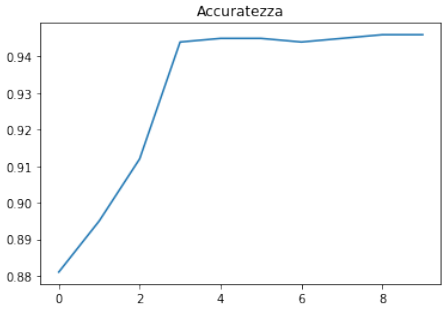
Nonostante ciò, la Resnet18 ha ottenuto degli ottimi risultati, con un'accuratezza del 98.7%. Le altre reti pre-addestrate presentano dei buoni risultati, seppure infe-



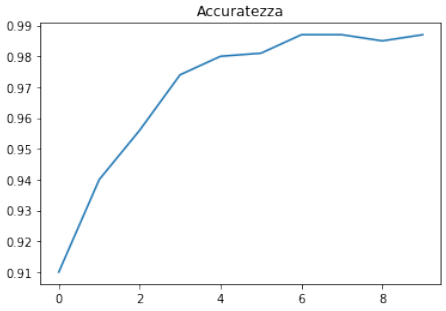
(a) CNN Custom



(b) AlexNet

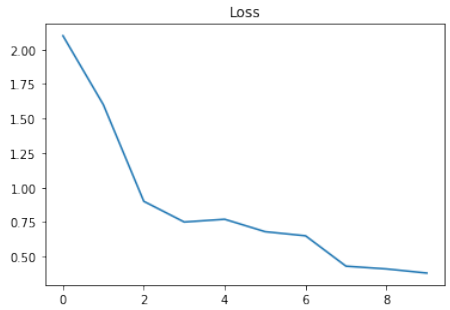


(c) VGGNet16

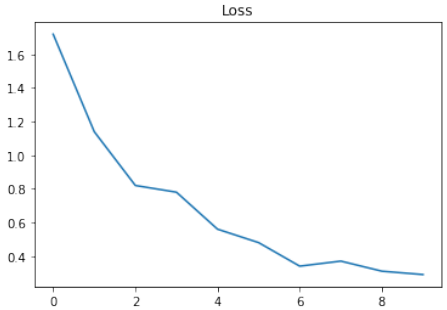


(d) Resnet18

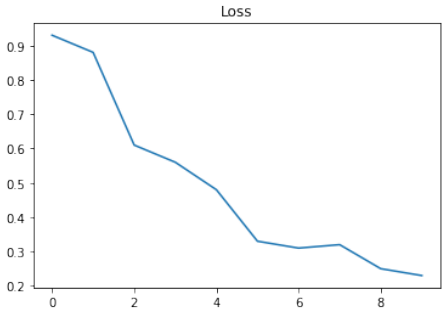
Figura 6.6: Test accuracy durante il training delle CNN della finestra di Tipologia window-panel



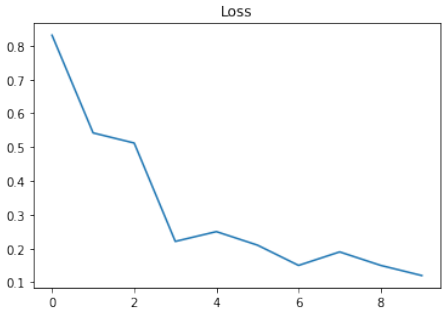
(a) CNN Custom



(b) AlexNet



(c) VGGNet16



(d) Resnet18

Figura 6.8: Loss durante il training delle CNN della finestra di Tipologia window-panel

riori ai requisiti di progetto (che prevedono un'accuratezza minima del 95%).

Come atteso, la rete neurale custom non è risultata essere sufficientemente performante, sia per la struttura poco complessa con cui è stata costruita, sia per le ridotte dimensione del dataset a disposizione.

Il tempo di training e il tempo di predizione medio hanno valori simili tra le CNN custom, AlexNet e Resnet18, mentre VGGnet16 presenta dei valori molto più alti.

Precision e recall sono state calcolate rispetto alla classe Off.

Pannello

Anche per il modello di object detection sviluppato per riconoscere le piastrelle all'interno dell'immagine del pannello dell'interruttore di Tipologia window-panel è stata impostata la soglia di confidenza dell'IoU a 0.5. Per il task di classificazione è stata impostata una soglia di confidenza di 0.6.
Il modello ha ottenuto un punteggio pari a 1 di mAP ed è quindi risultato essere affidabile nel riconoscimento e nella localizzazione delle piastrelle del pannello all'interno dell'immagine.

La rete neurale convoluzionale del pannello dell'interruttore di Tipologia window-panel è stata addestrata utilizzando i parametri descritti nella Sezione 5.3.1. Essa è in grado di distinguere se all'interno delle due piastrelle del pannello sono presenti una linea verticale o una orizzontale. Poichè ogni immagine del pannello di Tipologia window-panel ha al suo interno due piastrelle, il dataset a disposizione è superiore rispetto a quello rappresentato nella Tabella 5.2. Nella trattazione che segue lo stato della piastrella contenente la linea verticale verrà chiamato On, mentre quello della piastrella contenente la linea orizzontale verrà chiamato Off.
Il dataset è stato diviso in training set e test set rispettivamente con il 70% e il 30% dei campioni. La Tabella 6.3 riporta il numero di immagini per ciascuno dei due dataset.

Stato	Numero di immagini
On	1256
Off	1471

(a) Training set

Stato	Numero di immagini
On	538
Off	630

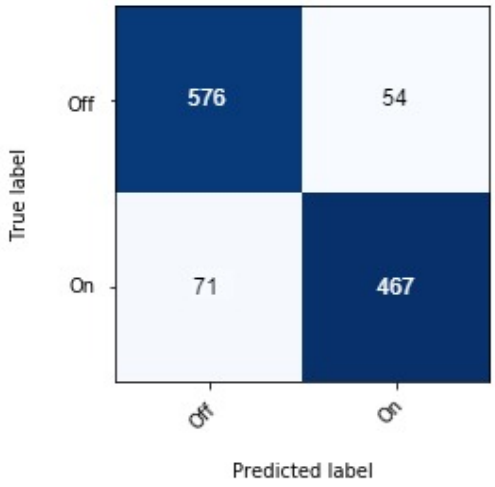
(b) Test set

Tabella 6.3: Training set e Test set del pannello dell'interruttore di Tipologia window-panel

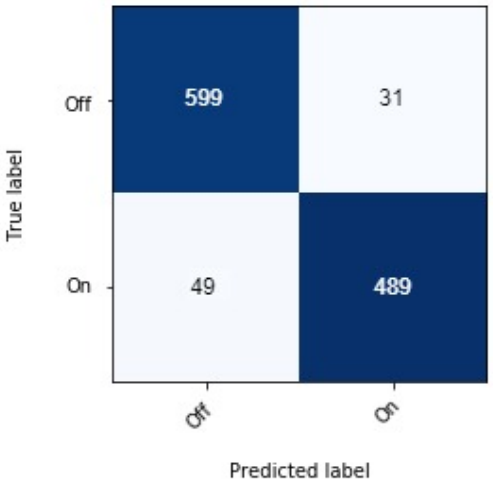
Metrica	Rete Custom	AlexNet	VGGnet16	Resnet18
Accuratezza	89.3%	93.1%	96%	99.3%
Recall	89%	92.4%	95.6%	99%
Precision	91.4%	95%	97.1%	99.7%
F-measure	90.2%	93.7%	96.4%	99.4%
Tempo di training (min)	27	28	35	31
Tempo di predizione medio (ms)	112	151	698	191
Peso della rete (MB)	20	218	512	43

Tabella 6.4: Risultati del pannello dell'interruttore di Tipologia window-panel

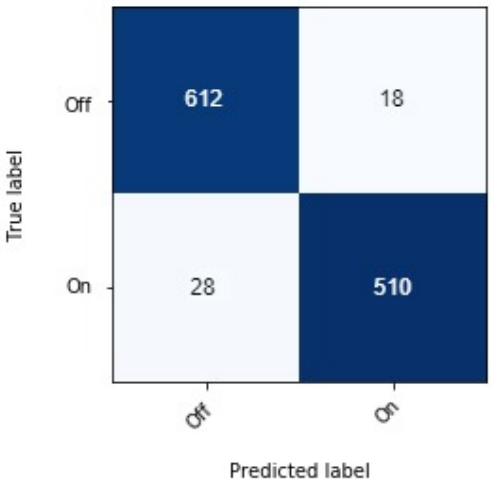
Le figure seguenti riportano le matrici di confusione ottenute per le varie reti neurali utilizzate, i grafici dell'accuratezza e l'andamento della loss durante il training. La Tabella 6.4 riporta i risultati ottenuti.



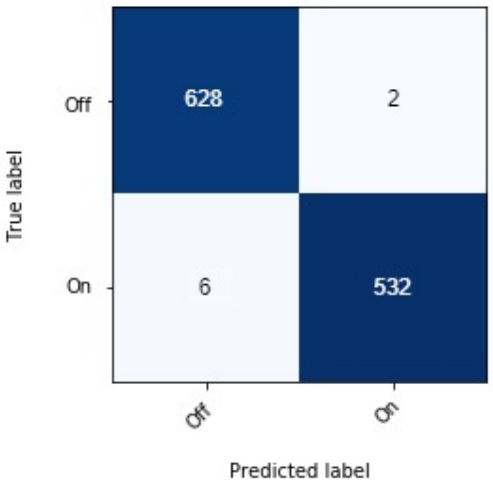
(a) CNN Custom



(b) AlexNet



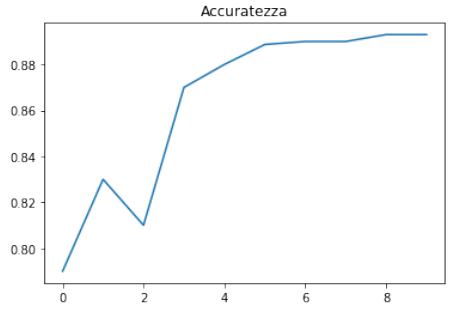
(c) VGGNet16



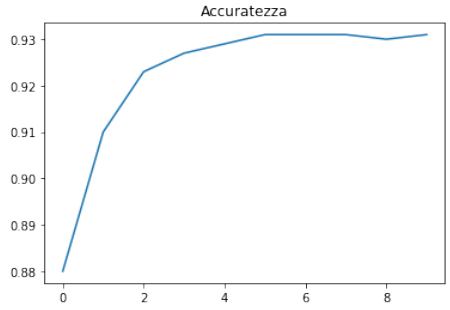
(d) Resnet18

Figura 6.10: Confusion Matrix delle CNN del pannello dell'interruttore di Tipologia window-panel

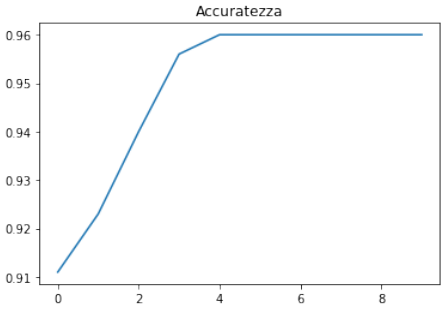
Dalle matrici di confusione in Figura 6.10 si può notare che la maggior parte dei campioni classificati erratamente sono quelli classificati come Off quando invece la classe vera era On. Ciò è dovuto al fatto che all'interno della piastrella di sinistra sono presenti due bordi neri orizzontali in alto e in basso (Figura 5.3). Dunque, quando lo stato della piastrella di sinistra è On (linea verticale), la rete neurale può essere tratta in inganno dalle due linee orizzontali classificandola erroneamente come Off.



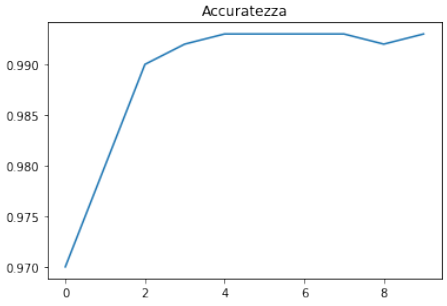
(a) CNN Custom



(b) AlexNet

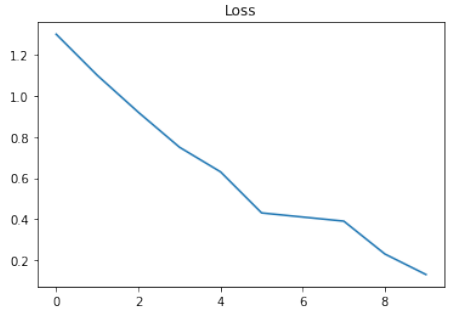


(c) VGGNet16

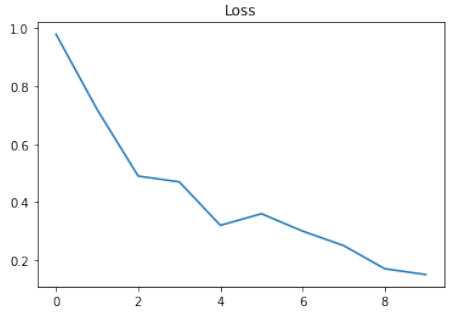


(d) Resnet18

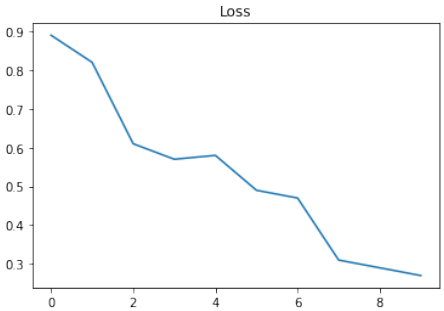
Figura 6.12: Test accuracy durante il training delle CNN del pannello dell'interruttore di Tipologia window-panel



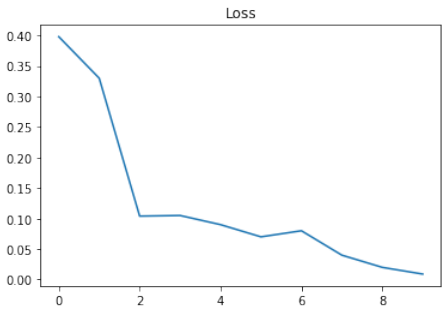
(a) CNN Custom



(b) AlexNet



(c) VGGNet16



(d) Resnet18

Figura 6.14: Loss durante il training delle CNN del pannello dell'interruttore di Tipologia window-panel

I risultati della Tabella 6.4 mostrano come le varie reti abbiano raggiunto dei buoni risultati. Anche in questo caso Resnet18 ha dimostrato di essere la rete più performante, ottenendo degli ottimi valori in tutte le metriche. Da notare inoltre che Resnet18 ha una dimensione in MB della rete di gran lunga inferiore rispetto alle altre reti pre-addestrate. Questo permette di ridurre notevolmente il tempo necessario per effettuare il deploy dei microservizi.

Il tempo di training per la rete del pannello di Tipologia window-panel è più elevato rispetto alle altre tipologie soprattutto a causa della dimensione del training set che è più numeroso rispetto alle altre.

I grafici in Figura 6.12 mostrano come le varie reti raggiungano il valore di accuratezza massimo dopo poche epoche, mentre quelli in Figura 6.14 mostrano come le varie reti convergano rapidamente.

I valori di recall e precision sono stati calcolati rispetto alla classe Off.

6.2 Risultati Tipologia knob

La rete neurale convoluzionale dell'interruttore di Tipologia knob è stata addestrata utilizzando i parametri descritti nella Sezione 5.3.2.
Il dataset di Tipologia knob è stato diviso in training set e test set rispettivamente con il 70% e il 30% dei campioni. La Tabella 6.5 riporta il numero di immagini per ciascuno dei due dataset.

Stato	Numero di immagini
On	118
Off	114
Estratto	69

(a) Training set

Stato	Numero di immagini
On	52
Off	49
Estratto	30

(b) Test set

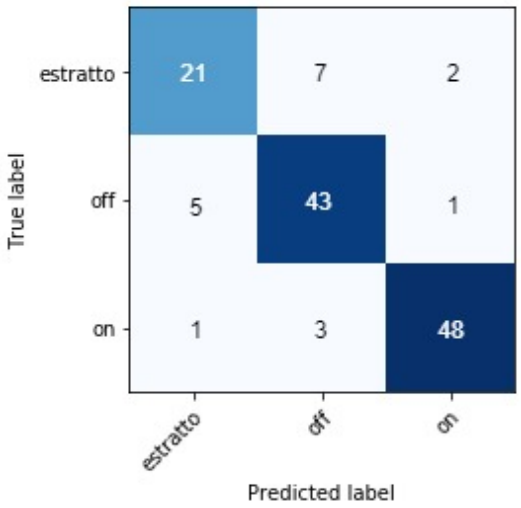
Tabella 6.5: Training set e Test set dell'interruttore di Tipologia knob

Le figure seguenti riportano le matrici di confusione ottenute per le varie reti neurali utilizzate, i grafici dell'accuratezza e l'andamento della loss durante il training. La Tabella 6.6 riporta i risultati ottenuti.

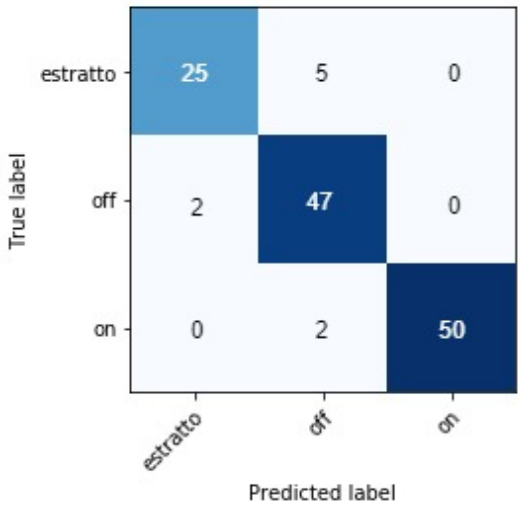
Metrica	Rete Custom	AlexNet	VGGnet16	Resnet18
Accuratezza	85.5%	93%	95%	100%
Recall	70%	83.3%	90%	100%
Precision	77.8%	92.6%	93.1%	100%
F-measure	73.7%	87.7%	91.5%	100%
Tempo di training (min)	4	6	7	6
Tempo di predizione medio (ms)	156	161	741	187
Peso della rete (MB)	20	218	512	43

Tabella 6.6: Risultati Tipologia knob

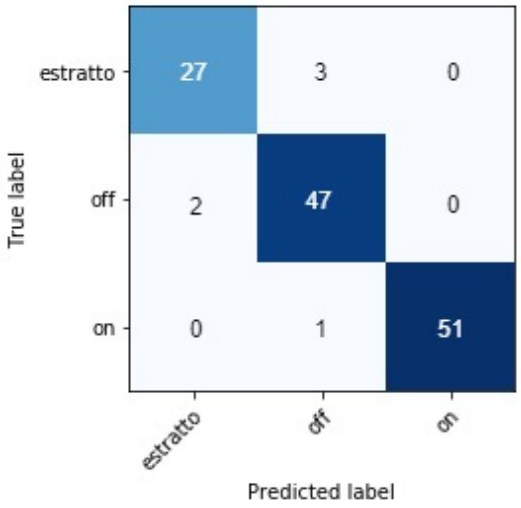
Il dataset a disposizione per questa tipologia è di dimensioni ridotte. La rete neurale custom, formata da solo due layer convoluzionali non garantisce delle performance sufficienti. Lo stato Off e Estratto sono molto simili e difficilmente distinguibili, poichè le uniche due differenze sono che, nel caso Estratto, la porzione bianca dell'interruttore è rialzata e vi è applicata una tenaglia. Dalla matrice di confusione della Figura 6.16(a), si può notare come la rete custom abbia difficoltà nel distinguere questi due stati. Anche le reti pre-addestrate AlexNet e VggNet16 presentano



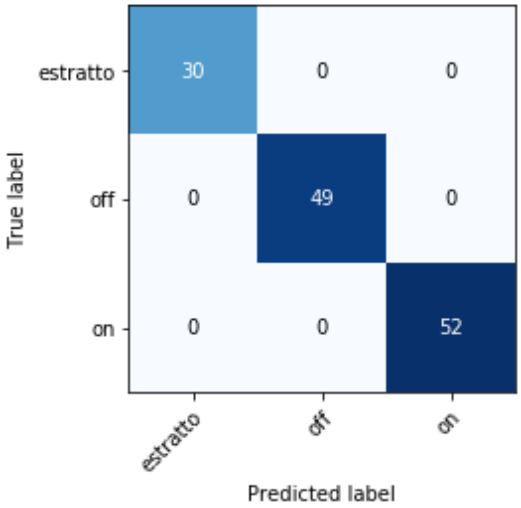
(a) CNN Custom



(b) AlexNet



(c) VGGNet16



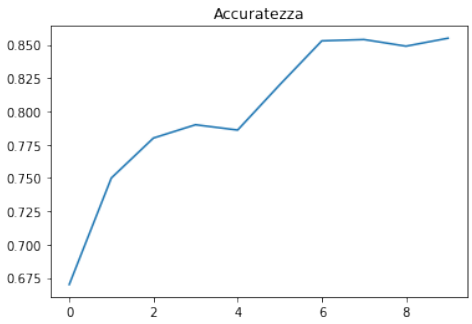
(d) Resnet18

Figura 6.16: Confusion Matrix delle CNN di Tipologia knob

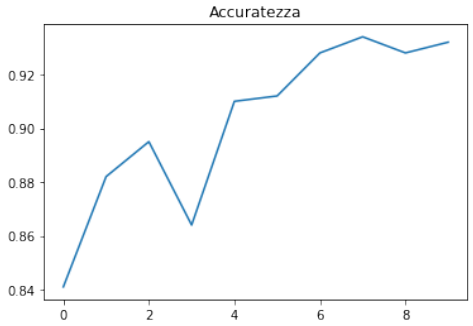
dei casi in cui questi due stati non vengono riconosciuti correttamente.

Dal punto di vista dell'accuratezza, la Resnet18 è indubbiamente quella più performante. Ma non è solo questo parametro a rendere questa rete preferibile rispetto alle altre. I valori di recall per le altre reti non sono sufficienti. Questo dato rivela infatti che dei campioni classificati da AlexNet come Estratto, solo l'83% lo sono davvero.
Inoltre le dimensioni della rete Resnet18 sono di gran lunga inferiori rispetto ad AlexNet e VggNet.

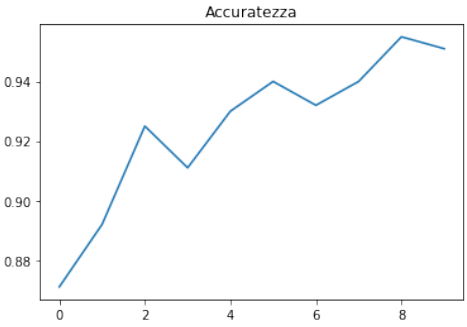
Nel caso di Tipologia knob, il tempo di training è simile tra le varie reti preaddestrate ed è leggermente inferiore quello della rete custom.



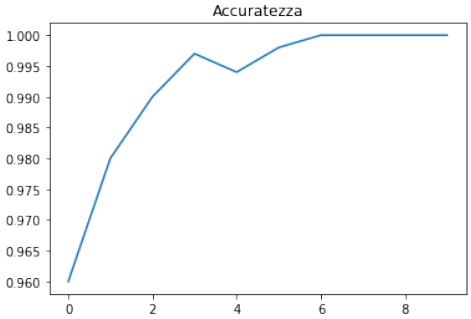
(a) CNN Custom



(b) AlexNet

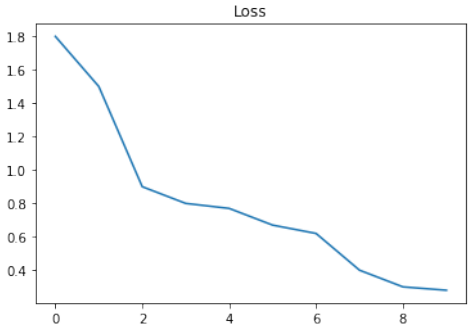


(c) VGGNet16

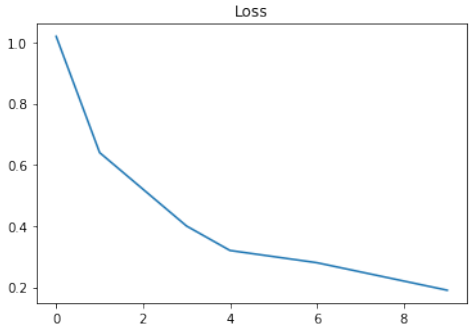


(d) Resnet18

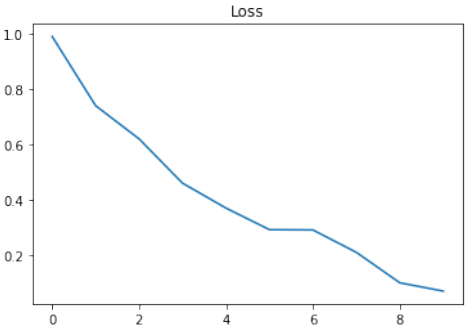
Figura 6.18: Test accuracy durante il training delle CNN di Tipologia knob



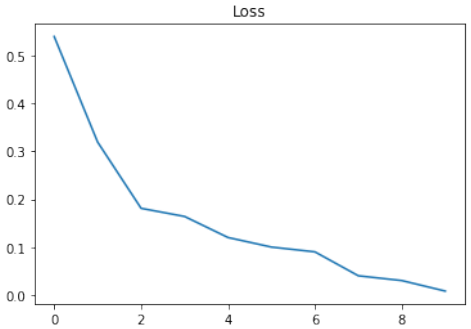
(a) CNN Custom



(b) AlexNet



(c) VGGNet16



(d) Resnet18

Figura 6.20: Loss durante il training delle CNN di Tipologia knob

Le figure 6.18 e 6.20 mostrano i grafici relativi all'andamento dell'accuracy e della loss durante il training. Da notare come nel caso di Resnet18 l'accuratezza parta da valori molto alti. Ciò significa che sono sufficienti poche epoche per ottenere prestazioni significative.

Resnet18 e Alexnet sono risultate essere le reti neurali più veloci, mentre VGGnet16 la più lenta. Questi dati sono in accordo con il dato sul peso in MB della rete neurale riportato in Tabella 6.6.

I valori di recall e precision sono stati calcolati rispetto alla classe Estratto.

6.3 Risultati Tipologia switch

La rete neurale convoluzionale dell'interruttore di Tipologia switch è stata addestrata utilizzando i parametri descritti nella Sezione 5.3.3.
Il dataset di Tipologia switch è stato diviso in training set e test set rispettivamente con il 70% e il 30% dei campioni. La Tabella 6.7 riporta il numero di immagini per ciascuno dei due dataset.

Stato	Numero di immagini
On	775
Off	736

(a) Training set

Stato	Numero di immagini
On	333
Off	316

(b) Test set

Tabella 6.7: Training set e Test set dell'interruttore di Tipologia switch

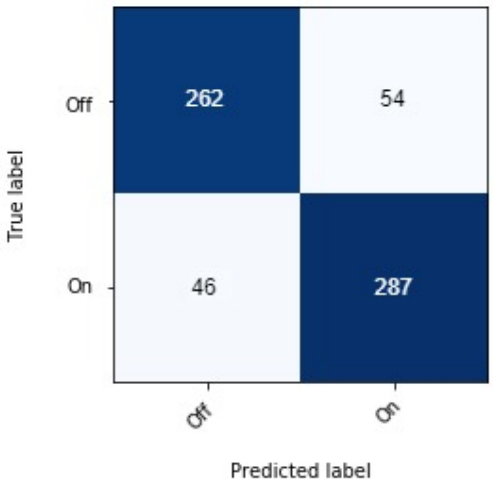
Le figure seguenti riportano le matrici di confusione ottenute per le varie reti neurali utilizzate, i grafici dell'accuratezza e l'andamento della loss durante il training. La Tabella 6.8 riporta i risultati ottenuti.

Metrica	Rete Custom	AlexNet	VGGnet16	Resnet18
Accuratezza	84.6%	92.7%	94%	99.2%
Recall	83%	94%	92.9%	98.7%
Precision	85%	91.4%	95.2%	99.7%
F-measure	84%	92.7%	94%	99.2%
Tempo di training (min)	7	7	7	11
Tempo di predizione medio (ms)	111	115	757	234
Peso della rete (MB)	20	218	512	43

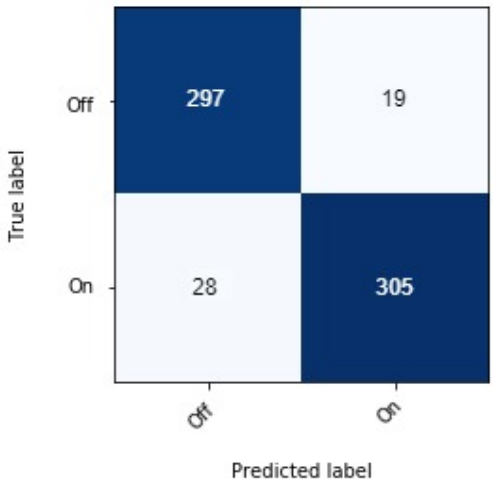
Tabella 6.8: Risultati Tipologia switch

Il riconoscimento dello stato Off e dello stato On si basa sulla posizione della levetta di colore bianco al centro dell'interruttore, come descritto nella Sezione 5.1.3. Questo rende la distinzione tra i due stati difficile da riconoscere.

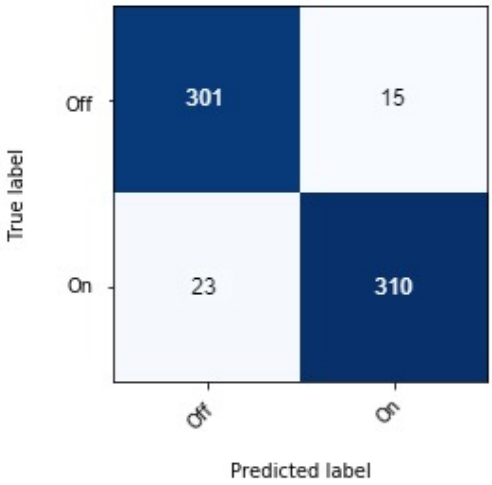
Le prime reti neurali sviluppate per questa tipologia avevano come training set le immagini degli interruttori croppate includendo i due QR code. Dai primi test ho notato che la rete non era in grado di generalizzare nel caso di QR code con identificativi diversi. Infatti le prime reti sono state addestrate con immagini aventi lo



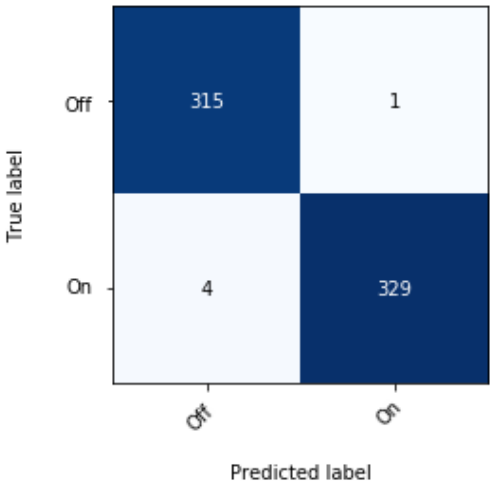
(a) CNN Custom



(b) AlexNet



(c) VGGNet16

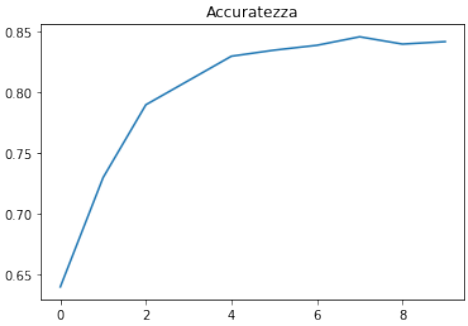


(d) Resnet18

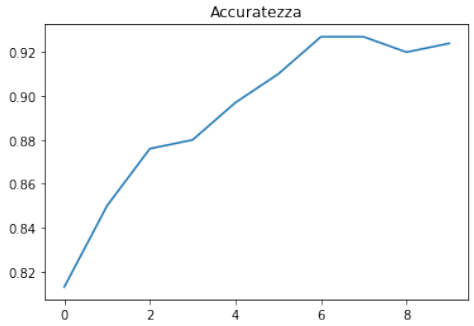
Figura 6.22: Confusion Matrix delle CNN di Tipologia switch

stesso id nei QR code. La rete ha appreso che lo stato Off e lo stato On dipendessero anche da quel particolare identificativo nel QR code. La rete riusciva a classificare correttamente l'interruttore con lo stesso identificativo di quello presente nelle immagini del training set, ma non era in grado di distinguere correttamente i due stati nelle immagini di interruttori con diverso identificativo.
Per risolvere questo problema, ho deciso di rimuovere i QR code dalle immagini durante la fase di crop. In questo modo le reti neurali imparano a distinguere i due stati solo ed esclusivamente in base alla posizione della levetta.
Questo fenomeno è stato riscontrato solo per questa tipologia di interruttore.

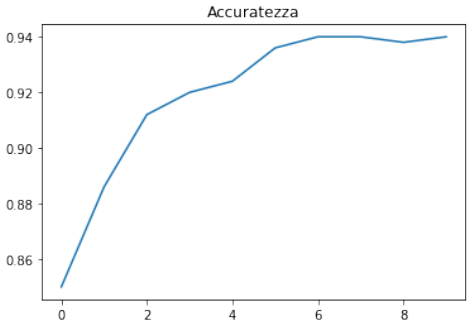
Anche per Tipologia switch, la Resnet18 è risultata essere la rete neurale più performante, con un'accuratezza di quasi il 100% e ottimi valori di recall e precision. A differenza di Tipologia knob, il tempo di training è superiore rispetto alle altre reti



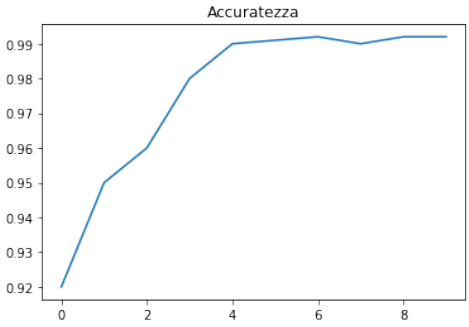
(a) CNN Custom



(b) AlexNet

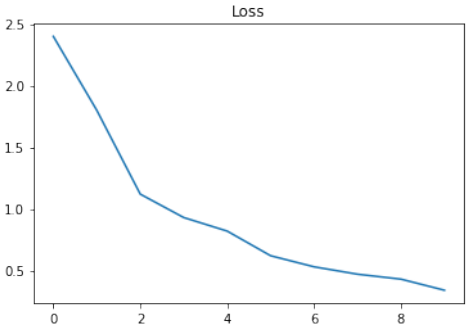


(c) VGGNet16

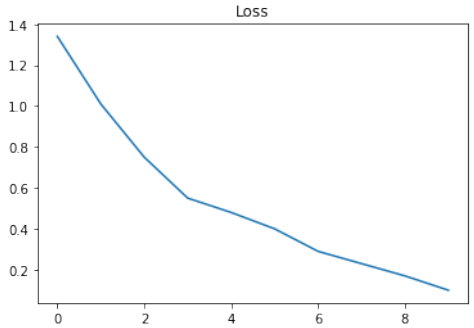


(d) Resnet18

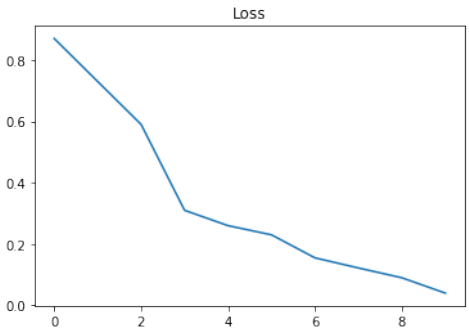
Figura 6.24: Test accuracy durante il training delle CNN di Tipologia switch



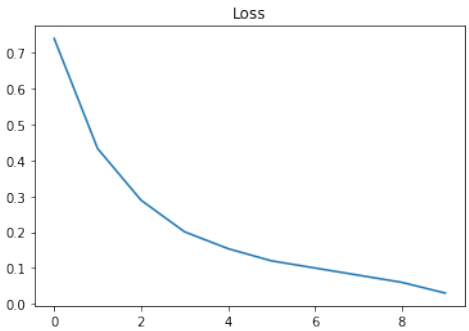
(a) CNN Custom



(b) AlexNet



(c) VGGNet16



(d) Resnet18

Figura 6.26: Loss durante il training delle CNN di Tipologia switch

pre-addestrate.

La rete custom anche in questo caso non è stata sufficiente per raggiungere delle prestazioni adeguate, sia in termini di accuratezza che in termini di recall e precision.

Lo stato Estratto per questa tipologia si verifica quando l'interruttore viene estratto fisicamente dalla sua ubicazione. Questo stato è riconosciuto dalla detection di un QR code, come indicato nella Sezione 5.3.3.

I valori di recall e precision sono stati calcolati rispetto alla classe Off.

6.4 Risultati Tipologia lever

Tipologia lever On/off

La rete neurale convoluzionale dell'interruttore di Tipologia lever per distinguere tra lo stato On e lo stato Off è stata addestrata utilizzando i parametri descritti nella Sezione 5.3.4.
Il dataset di Tipologia lever On/Off è stato diviso in training set e test set rispettivamente con il 70% e il 30% dei campioni. La Tabella 6.9 riporta il numero di immagini per ciascuno dei due dataset.

Stato	Numero di immagini
On	527
Off	550

(a) Training set

Stato	Numero di immagini
On	226
Off	236

(b) Test set

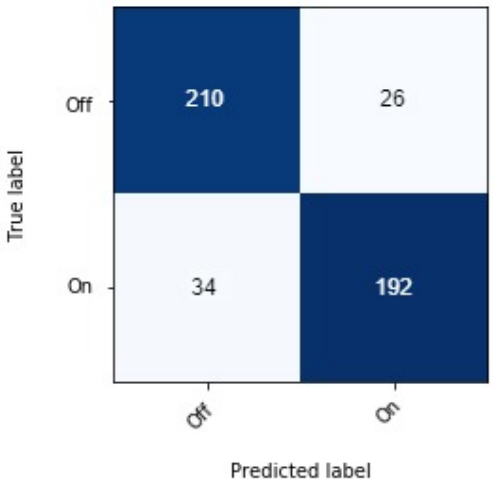
Tabella 6.9: Training set e Test set dell'interruttore di Tipologia lever

Le figure seguenti riportano le matrici di confusione ottenute per le varie reti neurali utilizzate, i grafici dell'accuratezza e l'andamento della loss durante il training. La Tabella 6.10 riporta i risultati ottenuti.

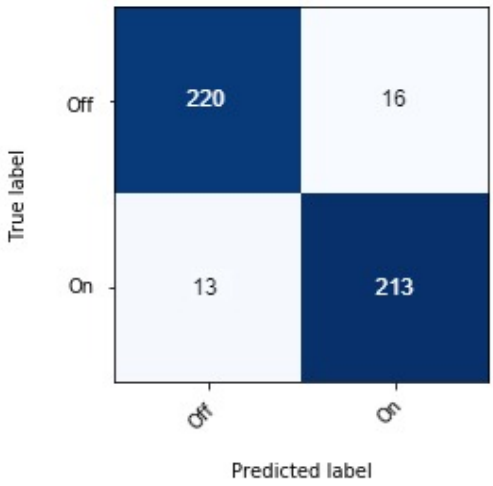
Metrica	Rete Custom	AlexNet	VGGnet16	Resnet18
Accuratezza	87%	93.7%	95.9%	100%
Recall	89%	93.2%	95.7%	100%
Precision	86%	94.4%	96.1%	100%
F-measure	87.5%	93.8%	96%	100%
Tempo di training (min)	11	11	18	14
Tempo di predizione medio (ms)	136	571	731	247
Peso della rete (MB)	20	218	512	43

Tabella 6.10: Risultati Tipologia lever On/Off

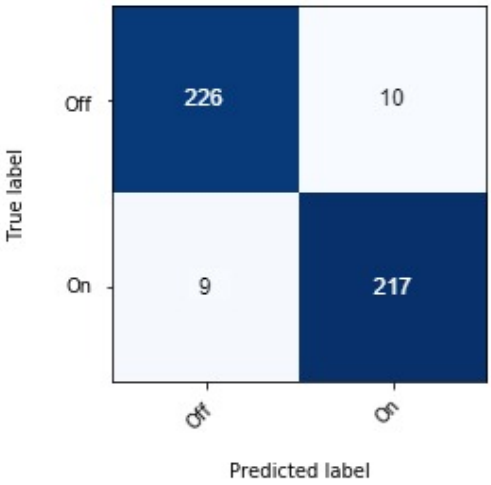
Il training set a disposizione per Tipologia lever On/Off è più numeroso rispetto alle tipologie precedenti e di conseguenza il trainig time per le varie reti è superiore. La Tabella 6.10 evidenzia come VGGnet16 abbia un tempo di training nettamente superiore rispetto alle altre.
Le performance sono elevate per tutte le reti pre-addestrate, ma anche in questo caso la migliore è stata Resnet18 con un'accuratezza, recall, precision e F-measure



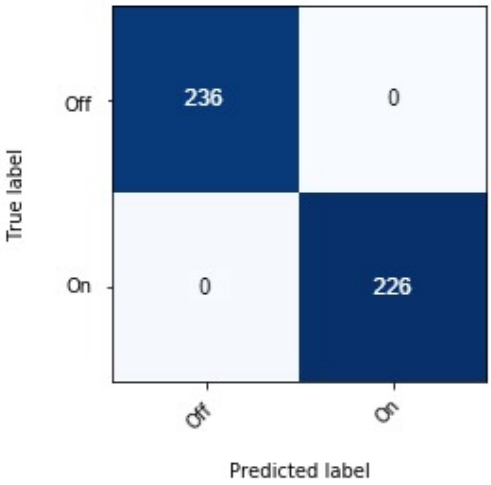
(a) CNN Custom



(b) AlexNet



(c) VGGNet16

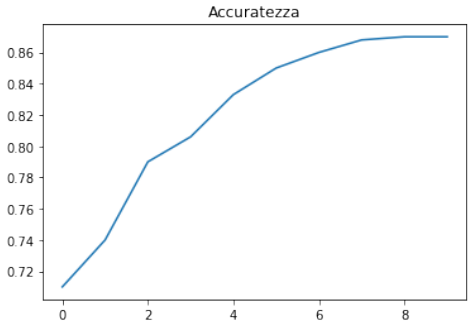


(d) Resnet18

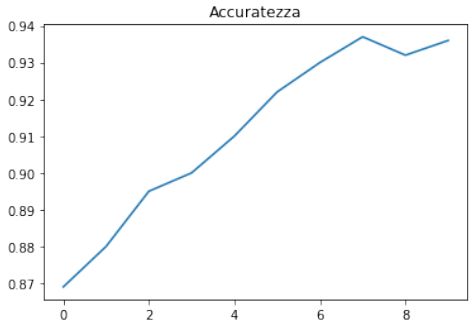
Figura 6.28: Confusion Matrix delle CNN di Tipologia lever

del 100%. Resnet18 presenta anche il minor tempo di predizione medio rispetto alle altre reti pre-addestrate.

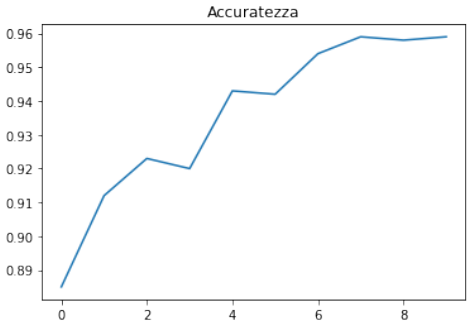
I valori di recall e precision sono stati calcolati rispetto alla classe Off.



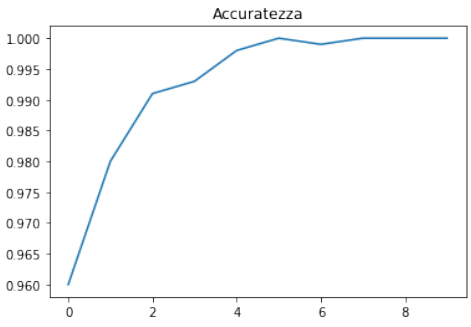
(a) CNN Custom



(b) AlexNet

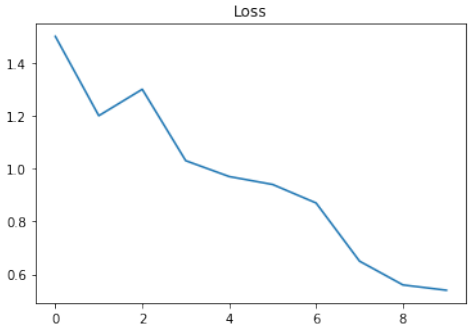


(c) VGGNet16

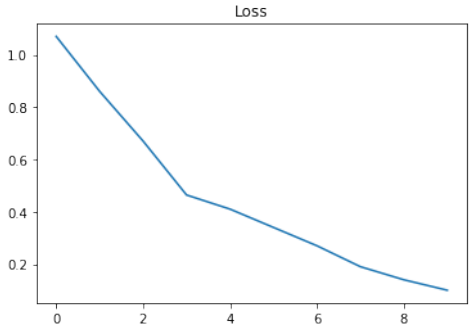


(d) Resnet18

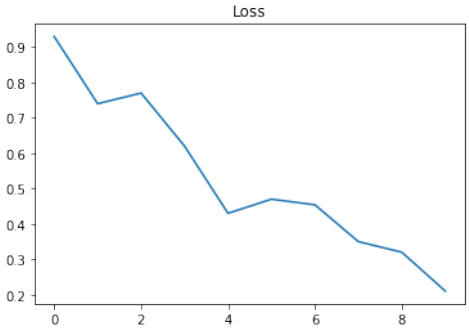
Figura 6.30: Test accuracy durante il training delle CNN di Tipologia lever



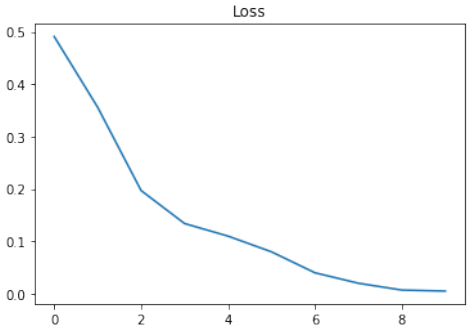
(a) CNN Custom



(b) AlexNet



(c) VGGNet16



(d) Resnet18

Figura 6.32: Loss durante il training delle CNN di Tipologia lever

Tipologia lever Estratto/Non estratto

La rete neurale convoluzione usata per riconoscere lo stato del pannello dell'interruttore di Tipologia lever è stata addestrata utilizzando i parametri descritti nella Sezione 5.3.4.
Il dataset di Tipologia lever Estratto/Non estratto è stato diviso in training set e test set rispettivamente con il 70% e il 30% dei campioni. La Tabella 6.11 riporta il numero di immagini per ciascuno dei due dataset.

Stato	Numero di immagini
Estratto	758
Non Estratto	695

(a) Training set

Stato	Numero di immagini
Estratto	325
Non Estratto	299

(b) Test set

Tabella 6.11: Training set e Test set dell'interruttore di Tipologia lever estratto/non estratto

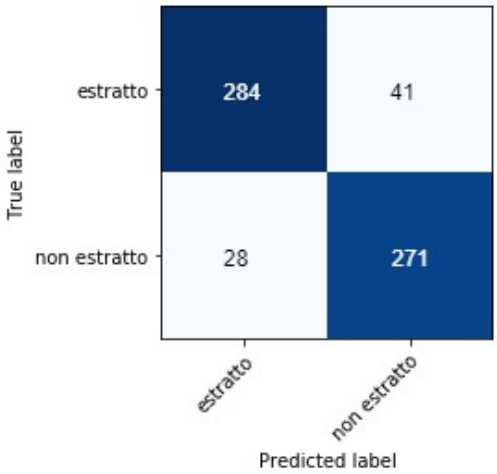
Le figure seguenti riportano le matrici di confusione ottenute per le varie reti neurali utilizzate, i grafici dell'accuratezza e l'andamento della loss durante il training. La Tabella 6.12 riporta i risultati ottenuti.

Metrica	Rete Custom	AlexNet	VGGnet16	Resnet18
Accuratezza	88.9%	95.8%	95.3%	100%
Recall	87.4%	95.4%	95%	100%
Precision	91%	96.6%	96%	100%
F-measure	89.2%	96%	95.5%	100%
Tempo di training (min)	19	19	24	23
Tempo di predizione medio (ms)	129	241	781	197
Peso della rete (MB)	20	218	512	43

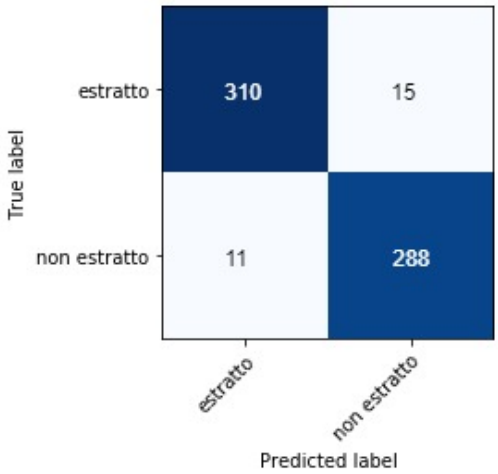
Tabella 6.12: Risultati Tipologia lever estratto/non estratto

I risultati ottenuti confermano ancora una volta come la Resnet18 sia la rete preaddestrata più efficiente. Inoltre, dalle figure 6.34 e 6.38 si può notare che la Resnet18 converge più velocemente tra quelle proposte e raggiunge valori di accuratezza vicini al 100% già dalle prime epoche.

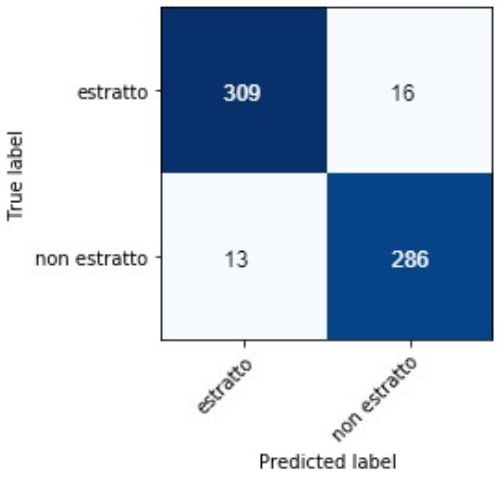
I valori di recall e precision sono stati calcolati rispetto alla classe Estratto.



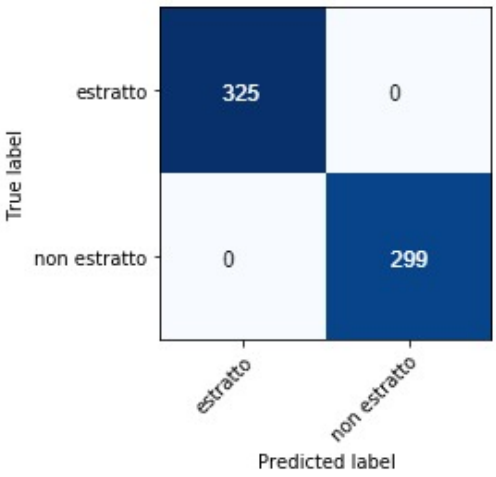
(a) CNN Custom



(b) AlexNet

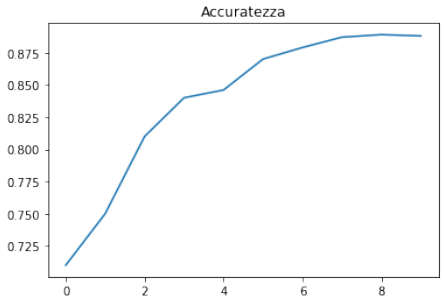


(c) VGGNet16

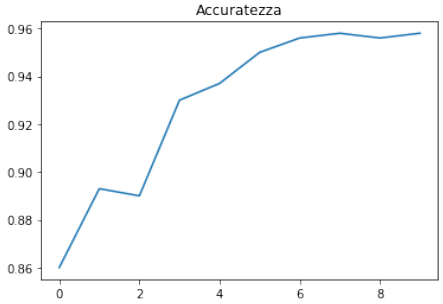


(d) Resnet18

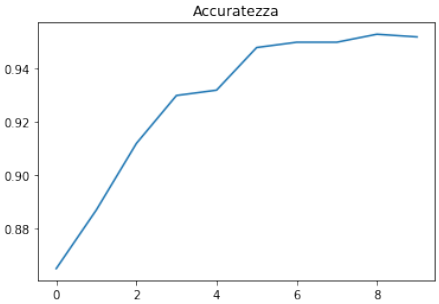
Figura 6.34: Confusion Matrix delle CNN di Tipologia lever estratto/non estratto



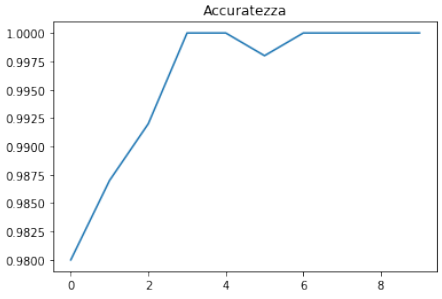
(a) CNN Custom



(b) AlexNet

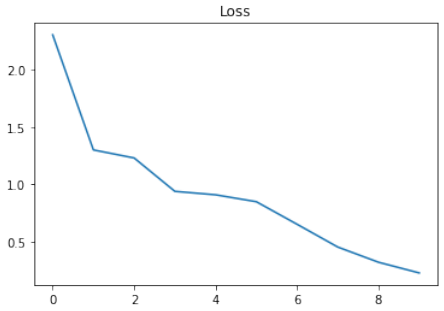


(c) VGGNet16

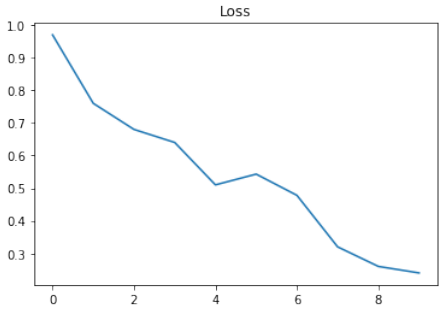


(d) Resnet18

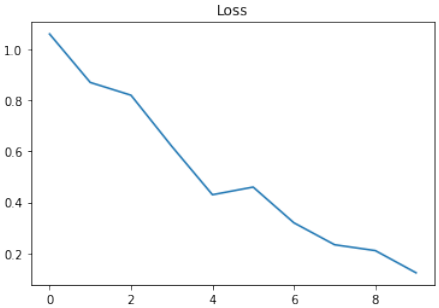
Figura 6.36: Test accuracy durante il training delle CNN di Tipologia lever estratto/non estratto



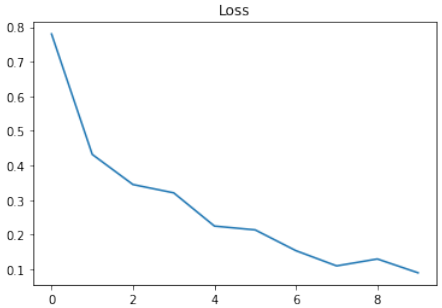
(a) CNN Custom



(b) AlexNet



(c) VGGNet16



(d) Resnet18

Figura 6.38: Loss durante il training delle CNN di Tipologia lever estratto/non estratto

Capitolo 7

Conclusioni

In questo studio sono state mostrate quali tecniche di Pattern Recognition possono essere utilizzate in ambito industriale per il riconoscimento dello stato di interruttori elettrici. Sono state analizzate le possibili soluzioni per aggiungere un processo automatizzato di intelligenza artificiale per aumentare la sicurezza dei lavoratori durante le operazioni di manutenzione. Ho deciso di utilizzare delle reti neurali convoluzionali come tecnica di Pattern recognition per il riconoscimento dello stato degli interruttori elettrici, motivato dai numerosi esempi di successo presenti in letteratura.

Sono state progettate varie reti neurali convoluzionali, separate per ognuna delle quattro tipologie di interruttori scelte. Sono state utilizzate delle reti pre-addestrate in modo da far fronte alle ridotte dimensioni del dataset a disposizione. Sono state sperimentate diverse reti pre-addestrate, come AlexNet, VGGnet16 e Resnet18 e una rete convoluzionale avente solo due layer convoluzionali.
I risultati hanno mostrato che la rete più adatta per questo specifico task e che ha fornito le prestazioni migliori è stata la rete pre-addestrata Resnet18. I risultati ottenuti con questa rete sono stati ottimi e il cliente ha deciso di approvare la sperimentazione e di portare il progetto in produzione, in modo da rendere il nuovo processo di sezionamento disponibile in una delle loro centrali elettriche.
Attualmente il nuovo processo di sezionamento che utilizza anche la componente di intelligenza artificiale è operativo da circa tre mesi, con ottimi risultati.

Il buon lavoro svolto e i risultati ottenuti mi spingono a continuare a sperimentare nuove soluzioni per risolvere problemi sempre più complessi. Possibili lavori futuri sono la progettazione di nuove reti neurali per le tipologie di interruttori rimanenti, in modo da coprire ogni casistica presente nelle fabbriche del cliente. La sfida più grande sarà sicuramente quella di provare a progettare un'unica rete neurale per classificare tutti gli stati di tutte le tipologie di interruttori, riuscendo comunque a mantenere dei livelli di performance elevati.

Bibliografia

[1] IAL Srl. *L'importanza della sicurezza sul lavoro*. URL: https://www.ialnazionale.com/formazione-per-la-sicurezza/43-l-importanza-della-sicurezza-sul-lavoro.

[2] Bureau of Labor Statistics. *Census of Fatal Occupational Injuries (CFOI) - Current and Revised Data*. 2019. URL: https://www.bls.gov/iif/oshcfoi1.htm.

[3] Will Knight. *Artificial intelligence sees construction site accidents before they happen*. 2019. URL: https://www.technologyreview.com/s/613738/artificial-intelligence-sees-construction-site-accidents-before-they-happen/.

[4] Ian Goodfellow, Yoshua Bengio e Aaron Courville. *Deep Learning*. http://www.deeplearningbook.org. MIT Press, 2016.

[5] Kevin P.Murphy. *Machine-Learning-A-Probabilistic-Perspective*. MIT Press, 2012.

[6] Rina Dechter. "Learning While Searching in Constraint-Satisfaction-Problems." In: *AAAI* (gen. 1986), pp. 178–185.

[7] J. Schmidhuber. "Deep Learning". In: *Scholarpedia* 10.11 (2015). revision #184887, p. 32832. DOI: 10.4249/scholarpedia.32832.

[8] Itamar Arel, Derek C.Rose e Thomas P-Karnowski. "Deep Machine Learning—A New Frontier in Artificial Intelligence Research". In: (2010).

[9] Manoj krishna et al. "Image classification using Deep learning". In: (2018).

[10] KOUSTUBH. *Understanding various architectures of Convolutional Networks*. 2018. URL: https://cv-tricks.com/cnn/understand-resnet-alexnet-vgg-inception/.

[11] K. He et al. "Deep residual learning for image recognition". In: (2015).

[12] X. Mou et al. "State Recognition of Electric Control Cabinet Switches Based on CNNs". In: *2018 IEEE 3rd International Conference on Image, Vision and Computing (ICIVC)*. Giu. 2018, pp. 183–187. DOI: 10.1109/ICIVC.2018.8492853.

[13] ZHANG Hao, WANG Wei e XU Lijie. "Application of image recognition technology in electrical equipment on-line monitoring". In: *Power System Protection and Control* (2010), pp. 88–91.

[14] XU Lijie ZHANG Hao WANG Wei. "Monitoring and identification system of subatation switch division based on SURF". In: *Instrument technique and sensor* (2017), pp. 56–58.

[15] CHEN Tuo. "Switch state recognition method of intelligent substation based on full convolution neural network". In: *35th Chinese Control Conference* (2016).

[16] Yi Sen Ng et al. "Convolutional Neural Networks for Food Image Recognition: An Experimental Study". In: *Proceedings of the 5th International Workshop on Multimedia Assisted Dietary Management*. MADiMa '19. Nice, France: ACM, 2019, pp. 33–41. ISBN: 978-1-4503-6916-9. DOI: 10.1145/3347448.3357168. URL: http://doi.acm.org/10.1145/3347448.3357168.

[17] R. Shimizu et al. "Convolutional neural network for industrial egg classification". In: *2017 International SoC Design Conference (ISOCC)*. Nov. 2017, pp. 67–68. DOI: 10.1109/ISOCC.2017.8368830.

[18] Q. Xuan et al. "Automatic Pearl Classification Machine Based on a Multistream Convolutional Neural Network". In: *IEEE Transactions on Industrial Electronics* 65.8 (ago. 2018), pp. 6538–6547. DOI: 10.1109/TIE.2017.2784394.

[19] Yang Lu et al. "Identification of Rice Diseases using Deep Convolutional Neural Networks". In: *Neurocomputing* 267 (lug. 2017). DOI: 10.1016/j.neucom.2017.06.023.

[20] P. Jamshidi et al. "Microservices: The Journey So Far and Challenges Ahead". In: *IEEE Software* 35.3 (mag. 2018), pp. 24–35. DOI: 10.1109/MS.2018.2141039.

[21] Algorithmia. *Introduction to Loss Functions*. 2018. URL: https://algorithmia.com/blog/introduction-to-loss-functions.

[22] Algorithmia. *Introduction to Optimizers*. 2018. URL: https://algorithmia.com/blog/introduction-to-optimizers.

[23] Jason Brownlee. *Gentle Introduction to the Adam Optimization Algorithm for Deep Learning*. 2017. URL: https://machinelearningmastery.com/adam-optimization-algorithm-for-deep-learning/.

[24] Diederik P. Kingma e Jimmy Ba. *Adam: A Method for Stochastic Optimization*. 2014. arXiv: 1412.6980 [cs.LG].

[25] Shaoqing Ren et al. *Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks*. 2015. arXiv: 1506.01497 [cs.CV].

[26] Timothy C Arlen. *Understanding the mAP Evaluation Metric for Object Detection*. 2018. URL: https://medium.com/@timothycarlen/understanding-the-map-evaluation-metric-for-object-detection-a07fe6962cf3.

[27] Tarang Shah. *Measuring Object Detection models — mAP — What is Mean Average Precision?* 2018. URL: https://towardsdatascience.com/what-is-map-understanding-the-statistic-of-choice-for-comparing-object-detection-models-1ea4f67a9dbd.

Ringraziamenti

Vorrei ringraziare in primis la professoressa Santa Di Cataldo per il suo costante supporto durante la stesura della Tesi. I suoi consigli sono stati importanti e mi hanno guidato durante tutto il percorso intrapreso per realizzare questo elaborato.

Ringrazio l'azienda Blue Reply che mi ha accolto nella sua grande famiglia circa due anni fa e che mi ha permesso di iniziare un percorso che mi ha insegnato tanto e mi ha permesso di mettere piede nel mondo lavorativo fin da subito. Ringrazio i manager e i colleghi della Business Unit del *Cognitive*, che mi hanno seguito e supportato durante ogni fase del progetto esposto in questa Tesi. Ci tengo inoltre a ringraziare i manager e i colleghi della Business Unit *Digital Commerce* dove ho lavorato part-time in questi due anni, per avermi formato lavorativamente e per la grande flessibilità che mi è stata concessa, fondamentale affinchè potessi concludere gli studi nei tempi prestabiliti.

In questi anni ho conosciuto tante persone fantastiche. Ringrazio tutti gli amici universitari che mi hanno accompagnato in questo percorso perchè ci siamo sempre supportati e, a volte, sopportati a vicenda. Grazie a Simo, Davide, Fuffolo, Gabri, Ilaria, Susanna, Nicola, Luca, Fede e Antonio.

Ringrazio il CUS Torino Volley che mi ha permesso di scaricare tutto lo stress universitario nello sport che amo di più, ma soprattutto mi ha fatto trovare degli amici stupendi che sono stati compagni di mille avventure in questi cinque anni. Ringrazio il mio capitano e grande amico Andrea, compagno di mille battaglie pallavolistiche e non solo, che è stato al mio fianco per tutti questi anni. Grazie a Gimmi per i suoi consigli sempre saggi e per il suo modo di fare sempre così euforico che tanto amo. Un grazie a Gennaro e Roberta per la loro amicizia. Gennaro mi hai insegnato cos'è un pixel e che "delegare" non è solo un verbo, ma un vero e proprio stile di vita. Grazie al mio pazzesco figlioccio Lorenzo che mi ha insegnato cosa significa il karma. Con te è stato amore a prima vista.
Ringrazio tutti i miei splendidi compagni di squadra. Dai che quest'anno si va ai playoff!! (Ma anche a Bibione).

Quando sei lontano da casa, c'è il rischio che amicizie di lunga data possano pian piano svanire. Per fortuna non è questo il mio caso. Un grazie a Nigro, Fortugno, Gramazio, Giorgia, Piera e Giulia, i miei amici compagni di classe del liceo perchè con voi ho condiviso tanti momenti della mia adolescenza, belli ma anche brutti, che mi hanno fatto crescere.
Non potevano mancare i ringraziamenti per alcune delle persone più importanti della mia vita presente e passata. Grazie a Cassio perchè da quando quel giorno la

Mingione ci ha reso compagni di banco è nata un'amicizia speciale che nè il tempo e nè la distanza potrà mai spezzare. Un grazie al Nota che, nonostante tutti i soldi persi a FIFA a Campomarino, è sempre stato un grande amico e consigliere. Grazie a Campanello e a quel succo di frutta alla pera che in prima media ha dato il via a una delle mie più belle amicizie. Grazie a Simone, lui c'è sempre, non importa dove e quando ma lui è sempre lì. Con voi ho praticamente passato ogni momento della mia adolescenza, con voi sono cresciuto. Seppur lontani vi ho sempre sentito al mio fianco. Torino sarebbe stato un posto migliore con voi qui.
Un grazie ovviamente anche alle mie amiche foggiane Roberta, Ale, Domi, Vale e Annamaria e alle mie cognatine Gaia e Robertina.

Ringrazio soprattutto loro, la mia splendida famiglia per il costante supporto che mi hanno dato non solo in questi anni, ma da sempre. Ringrazio i miei genitori per l'educazione ricevuta, per essersi sempre fidati di me e per avermi sempre supportato in ogni decisione presa. Nonostante la distanza, siete sempre stati presenti per me, nei momenti più felici, ma soprattutto nei momenti più difficili e di sconforto. Un grazie a te mamma, per la tua dolcezza e sensibilità, per l'amore che mi dimostri ogni giorno. Grazie a te papà, perchè per me sei sempre stato un modello da seguire e perchè quando gli altri mi dicono "sembra di sentir parlare tuo padre" io non posso far altro che esserne orgoglioso. Un grazie alla mia splendida sorellona, coinquilina per qualche anno che mi ha accompagnato nei primi anni universitari. Amo il nostro rapporto e la nostra complicità, e, nonostante spesso ci ritroviamo in parti diverse del mondo, sei per me sempre un punto di riferimento.

Un grazie alla mia fantastica fidanzata. Grazie Rosy per essermi sempre stata vicina in questi anni. Sono sicuro al 100% che senza di te non sarei mai riuscito a realizzare questi obiettivi. Sei stata un punto di riferimento fisso, mi hai sostenuto in ogni momento e hai sempre creduto in me. Quello che sono oggi lo devo a noi, al rapporto che ci lega ormai da sei meravigliosi anni.

Infine, vorrei ringraziare anche me stesso per la forza di volontà e la passione con cui in questi anni ho affrontato il mio percorso universitario. Ho deciso di accettare sfide sempre più ambiziose per mettermi in gioco e dimostrare a me stesso che con sacrificio e determinazione ogni obiettivo è raggiungibile.

Grazie