

UNIVERSITÀ DEGLI STUDI DI UDINE

Dipartimento di Scienze Matematiche, Informatiche e Fisiche

Corso di Laurea Triennale in Informatica

Tesi di Laurea

AUTOENCODER PER
L'ANOMALY DETECTION
APPLICATI IN
AMBITO AUTOMOTIVE

Relatore:

Prof. GIUSEPPE SERRA

Laureando:

TRISTANO MUNINI

Correlatore:

Ing. DANIELE FORNASIER

ANNO ACCADEMICO 2018-2019

Abstract

In questa tesi si affronta il problema di rilevare la presenza di colla sul fondo di carcasse metalliche per motori elettrici. Il *dataset* fornito, composto da qualche migliaio di immagini, comprende due classi, ma una di queste è rappresentata da un numero veramente esigui di esemplari. Si è deciso di utilizzare tecniche i *Machine Learning* per il riconoscimento di anomalie. In particolare è stato sviluppato un modello basato su reti neurali del tipo *autoencoder*, poiché permettono di essere allenati utilizzando solamente una classe. Applicando algoritmi di manipolazione e trasformazione delle immagini si è migliorato il *dataset*. La soluzione proposta si dimostra all'altezza dei risultati attesi, perché ha permesso di identificare quasi tutte le colle, senza però generare troppi falsi positivi. In questo studio non è stato possibile determinare se la soluzione proposta sia applicabile in modo affidabile in ambienti industriali. A tal fine sarebbe necessario ampliare il *dataset* ed effettuare delle verifiche sul campo. Il lavoro viene comunque considerato positivo perché ha permesso di esplorare le capacità combinate di algoritmi di manipolazione immagini e degli *autoencoder*.

Indice

Abstract	iii
1 Introduzione	1
1.1 Gli Obiettivi da Raggiungere	5
2 Il Problema	7
2.1 Le carcasse per motori elettrici	7
2.2 Il processo di produzione	9
3 Il Dataset	11
3.1 Problematiche principali	11
3.1.1 Dataset piccolo e sbilanciato	11
3.1.2 Differenze tra immagini	15
3.2 Pre-Processing	17
3.2.1 Passaggio da RGB a GrayScale	18
3.2.2 Masking	19
3.2.3 Image cropping and resizing	21
3.2.4 Histogram equalization	22
3.2.5 Gaussian blur	24
3.2.6 Sobel operator	25
3.2.7 Canny edge Detector	28
3.2.8 Hough circle transform	29
3.3 Applicazione degli algoritmi descritti	32
3.4 Data Augmentation	39
3.4.1 Generazione scarti sintetici	39
4 Gli Autoencoder	41
4.1 La struttura di un autoencoder	41
5 Risultati Ottenuti	49
5.1 Obiettivo	49

5.2	Metriche di valutazione	50
5.3	Il modello	50
5.4	Post-Processing	55
5.5	I Risultati	57
6	Conclusioni	59
6.1	Commento Dei Risultati	59
	Bibliografia	61

Capitolo 1

Introduzione

In campo industriale è sempre più frequente la scelta di integrare processi già esistenti con fotocamere per l'acquisizione di immagini. Le fotocamere, che hanno dimensioni sempre più ridotte ma permettono comunque di scattare fotografie ad ottima risoluzione, possono essere inserite facilmente nella maggior parte dei processi industriali, permettendo di monitorare la produzione senza comprometterla. Inoltre, le immagini raccolte possono andare a formare un *dataset* che, utilizzato per addestrare algoritmi di *machine learning*, permette non solo di monitorare il processo di produzione, ma anche di controllarlo.

In questa tesi si affronta il problema di rilevare la presenza di colla sul fondo di carcasse per motori elettrici, per mezzo di fotografie digitali. I pezzi da analizzare hanno una forma cilindrica cava con una delle due estremità sigillata. All'intero della cavità verrà alloggiato un motore elettrico per tergilampi. Per poter fissare il motorino è richiesto che un anello di colla sia versato sulla parete verticale interna del pezzo. Poiché la colla è liquida è possibile che goccioli fino a raggiungere il fondo del pezzo, questo comporta il malfunzionamento del motore. Da ciò nasce la necessità di rilevare in modo automatizzato quali pezzi presentano colla sul fondo, così da poterli scartare. Nello specifico si vuole che il sistema riconosca il maggior numero di carcasse che presentano colla sul fondo, queste verranno poi rimosse automaticamente dal processo di produzione.

Il lavoro svolto in questa tesi nasce per cercare una risposta ad un problema reale. La soluzione che si propone è stata sviluppata durante un tirocinio, della durata di sei mesi, svolto presso beanTech¹. Il principale strumento utilizzato è stato il linguaggio di programmazione *python*, arricchito da librerie come: *opencv*² e *pillow*³ per la manipolazione e trasformazione delle immagini; *num-*

¹Collegamento al sito ufficiale URL: <https://www.beantech.it/>

²URL: <https://opencv.org/>

³URL: <https://pillow.readthedocs.io/en/stable/>

*py*⁴ e *scikit-learn*⁵ per la gestione di dati numerici; *pytorch*⁶ per tutto ciò che riguarda le reti neurali e convolutive; *matplotlib*⁷ e *pandas*⁸ per la visualizzazione dei dati. Per l’allenamento dei modelli si è potuto sfruttare una NVIDIA® DGX Station⁹ con quattro NVIDIA® TESLA® V100 da 32 GB/GPU l’una. Durante il tirocinio si è sviluppata anche una buona conoscenza dello strumento Docker, che permette di creare spazi virtuali in cui sviluppare, testare e distribuire le proprie soluzioni.

Le prime soluzioni proposte si sono rivelate non soddisfacenti, ma poiché fanno parte del percorso che ha portato allo sviluppo della soluzione descritta in questo documento, si è deciso di dedicare un breve commento alla tecnica che più si discosta dalla soluzione finale.

ResNet e One Class SVM Come si vedrà meglio poi, il *dataset* è fortemente sbilanciato. Si dispone di pochissimi esemplari con colla sul fondo, per cui si è considerata soltanto la classe di carcasse idonee. La maggior parte dei modelli di *machine learning* per la classificazione viene allenato con almeno due classi. In questo modo il modello, tramite il confronto, può trovare le caratteristiche che distinguono gli elementi di una classe dall’altra e quindi imparare a riconoscerli. Quando si dispone di una sola classe ciò non è possibile, bisogna quindi applicare metodi alternativi come la *One Class Support Vector Machine* (OCSVM). Variazione dell’algoritmo di classificazione *Support Vector Machine*, è stato pensato per poter essere allenato avendo dati di un’unica classe. Il modo più intuitivo per descrivere il funzionamento di una OCSVM è immaginare che il suo scopo sia creare la più piccola sfera contenente tutti i punti del *dataset*. Così facendo tutti gli elementi con caratteristiche differenti da quelle degli elementi del *dataset* saranno posizionati a grande distanza dal centro della sfera, probabilmente cadranno all’esterno di essa. La classificazione viene effettuata verificando se l’elemento cade entro i confini della sfera, in caso affermativo l’elemento appartiene alla classe utilizzata durante l’allenamento.

Poiché l’informazione a nostra disposizione è contenuta in immagini, si è deciso di usare una rete ResNet18 [4] a cui è stato rimosso l’ultimo strato, in questo modo ritornerà vettori di 512 elementi. Solitamente una rete convolutiva sintetizza i dati contenuti in un’immagine e ritorna il valore della classe più probabile, ma rimuovendo l’ultimo strato si vanno a rimuovere quei neuroni

⁴URL: <https://numpy.org/>

⁵URL: <https://scikit-learn.org/>

⁶URL: <https://pytorch.org/>

⁷URL: <https://matplotlib.org/>

⁸URL: <https://pandas.pydata.org/>

⁹URL: <https://www.nvidia.com/en-us/data-center/dgx-station/>

che trasformano il vettore di 512 elementi nel valore della classe più probabile. Così facendo è possibile, per ogni immagine, ottenere un vettore di dimensioni ridotte. L'insieme di questi vettori andrà a costituire il *dataset* con cui allenare la OCSVM.

La soluzione appena descritta si è rivelata inadatta ai nostri scopi, fornendo un'accuratezza poco superiore al 60%, molto inferiore agli obiettivi che si vogliono raggiungere.

La soluzione, che verrà descritta con maggiore dettaglio nelle prossime pagine, sfrutta un particolare tipo di rete neurale che prende il nome di *autoencoder*. Il funzionamento di un *autoencoder* può essere suddiviso in due momenti: nel primo i dati in ingresso vengono mappati in uno spazio con dimensionalità inferiore a quella di partenza, si può dire che il dato venga compresso; nel secondo il dato in forma compressa viene riportato nel suo spazio originale, venendo quindi decompresso. Un buon *autoencoder* è in grado di fornire in *output* un dato estremamente simile a quello che è stato ricevuto in ingresso, non bisogna pensare però che il suo scopo sia simulare la funzione identità; infatti un compito del genere non sarebbe di alcuna utilità. L'origine degli *autoencoder* non è ben definita però ad oggi sono utilizzati per vari scopi.

ampliare in modo non tecnico

Le robe tecniche spostarle nella parte AE

Dimensionality Reduction Sappiamo che gli *autoencoder* possono essere sfruttati per trovare, fissata una dimensionalità, spazi latenti efficaci. Questa capacità ricorda tecniche di *dimensionality reduction* come la *Principal Component Analysis*(PCA). Non a caso infatti molti studi, come ad esempio quello intitolato *PCA vs Autoencoders for Dimensionality Reduction* [19] mettono in luce come gli *autoencoder* siano in grado di equiparare, se non superare, gli approcci classici. Il principale vantaggio degli *autoencoder* è la possibilità di creare una funzione di compressione specifica per il problema che si vuole risolvere. Mentre gli approcci come la PCA operano seguendo uno stesso algoritmo per ogni applicazione.

Denoising Il *denoising* è una tecnica con cui si rimuove, a partire da un dato non puro, tutta l'informazione considerata rumore. In figura 1.1 si può vedere come dopo aver introdotto del rumore alle immagini più a sinistra, ottenendo quelle nella seconda colonna, un *autoencoder* sia in grado di ripristinarle. L'ultima colonna mostra le immagini generate dalla rete. Questo tipo di applicazione può essere sfruttato per effettuare *denoising* ad immagini di radiografie in campo medico.

Anomaly Detection *Anomaly detection* significa letteralmente rilevamento delle anomalie. Questa tecnica permette, dopo aver definito quali sono i valori

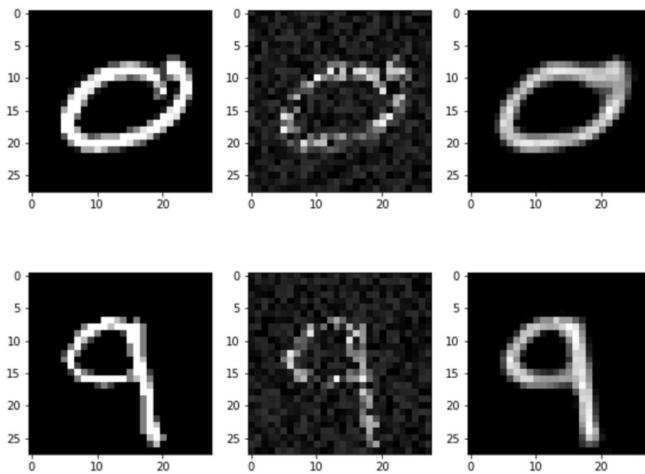


Figura 1.1: Esempio di applicazione di un *Denoising Autoencoder* URL: http://users.cecs.anu.edu.au/~Tom.Gedeon/conf/ABCs2018/paper/ABCs2018_paper_20.pdf

attesi, di riconoscere quando i dati ricevuti si allontanano da quelli che ci si aspettava. Poiché un *autoencoder* può essere facilmente allenato con la classe di valori canonici, imparerà una funzione di compressione adatta a comprimere i valori più frequenti. Questo significa che la rete fornirà un *output* molto simile a ciò che ha ricevuto in ingresso, se e solo se l'*input* ha caratteristiche simili agli elementi del *dataset*. Il dato in ingresso, se molto distante dai valori abituali, sarà compresso in modo che, probabilmente, non ne mantenga le caratteristiche principali. Ciò porta, dopo la decompressione, ad ottenere un *output* distante dal dato originale, permettendo di classificare quest'ultimo come anomalia.

Il problema descritto in questa tesi si presta per essere affrontato come un problema di *anomaly detection*.

1.1 Gli Obiettivi da Raggiungere

togliere titolo e sistemare nell'introduzione

Di seguito sono riportati alcuni dati numerici riguardo i processi appena descritti.

La colla viene depositata su circa 5000 pezzi al giorno, la probabilità che gocce di colla cadano sul fondo delle carcasse è estremamente bassa. Purtroppo non esistono dati numerici esatti ma si stima che il macchinario abbia un tasso d'errore di una carcassa al mese o poco più. Questi dati possono essere trasformati in probabilità approssimative osservando che:

$$\begin{array}{ll} \text{colla depositata al mese:} & 5000 * 31 = 155000 \\ \text{colla depositata male al mese:} & 2 \end{array}$$

Quindi la probabilità che il macchinario sbagli e pari allo 0.00001%.

Il sistema di intelligenza artificiale deve riconoscere i pezzi non conformi ma soprattutto, tenendo conto della probabilità di cui sopra, deve generare un numero bassissimo di falsi positivi. Ricordano che per falsi positivi (detti anche FP o *False Positive*) si intendono tutte le carcasse che il sistema considera non conformi ma che in realtà non presentano difetti. Una AI troppo rigida, che, quando indecisa, propende per scartare il pezzo, inciderebbe negativamente sulla produzione. Si rischierebbe infatti di creare un enorme danno economico andando a scartare molte più carcasse del necessario.

Il nostro obiettivo è creare un sistema che generi un numero di falsi positivi che sia inferiore al 2%, cercando di riconoscere più carcasse non conformi possibili.

Capitolo 2

Il Problema

All'inizio di questo capitolo verranno illustrati forma e fine ultimo dei pezzi meccanici analizzati. Seguirà una descrizione sommaria del processo industriale e delle macchine che manipolano e depositano la colla all'interno dei pezzi, nonché del sistema di acquisizione immagini. Si conclude con una formalizzazione del problema da risolvere e le metriche con cui valutare la soluzione proposta.

2.1 Le carcasse per motori elettrici

Osservando le fotografie in figura 2.1, possono essere definite le caratteristiche principali delle carcasse:

- il pezzo ha una struttura cilindrica cava;
- il fondo presenta tre gradini;
- sono presenti due balze (nella foto laterale se ne vede soltanto una), posizionate una di fronte all'altra, prima del primo gradino;
- i supporti alla bocca della carcassa presentano due fori.

Il pezzo è stato pensato per avvolgere e proteggere motori elettrici. Nello specifico i pezzi in foto, sui quali è stato svolto lo studio, sono carcasse per motori elettrici per tergilampade. I due fori aiuteranno a fissare con delle viti il pezzo su dei supporti in plastica. All'interno della cavità verrà depositata della colla liquida, successivamente verrà alloggiato il motore. La colla verrà fatta solidificare tramite calore, in modo che il motore, una volta posizionato, non possa vibrare all'interno della carcassa, evitando che eventuali urti possano danneggiarlo. Dalle foto in figura 2.1 si può notare che la colla è distribuita

in modo da formare un anello ad una altezza di circa 4cm dal fondo della carcassa.

La colla è stata depositata correttamente se:

1. l'anello non presenta né sbavature né discontinuità;
2. sul fondo non c'è presenza di colla.

In questo documento ci concentreremo esclusivamente sul secondo punto, infatti la presenza di colla sul fondo della carcassa causa il malfunzionamento del motorino dopo un limitato tempo d'utilizzo, molto inferiore al tempo di vita atteso. Per questo motivo è fondamentale che la colla venga depositata correttamente.



Figura 2.1: Visioni laterale e superiore di una carcassa

2.2 Il processo di produzione

Chiarire le modalità con cui le carcasse vengono manipolate, la colla viene depositata e le foto vengono acquisite risulta fondamentale. Senza queste informazioni mancherebbe la base sulla quale costruire ipotesi e considerazioni riguardo le immagini del *dataset*. Analizzando le condizioni in cui le foto vengono scattate, si definiscono i vincoli ed i confini entro i quali le soluzioni proposte possono considerarsi verosimili ed applicabili al mondo reale.

Le carcasse, già presenti in grandi quantità in magazzino, raggiungono il macchinario e vengono caricate, con la concavità rivolta verso l'alto, su un disco rotante. Ad ogni ciclo macchina la colla, tramite due ugelli, viene depositata simultaneamente su due carcasse distinte. Al contempo due sonde dotate di luce scendono nelle due carcasse su cui la colla era stata depositata il ciclo precedente, fino ad una distanza di circa 3cm dal fondo. Le carcasse ritenute conformi procedono lungo un rullo trasportatore, mentre quelle che non sono idonee vengono scartate.

Vanno precisati vari aspetti. Le carcasse, nonostante siano tutte dello stesso tipo, possono differire per quanto riguarda colore, graffi superficiali, sporco, incrostazioni oppure macchie. Inoltre non vengono orientate tutte allo stesso modo rispetto all'asse verticale: questo ha delle ripercussioni dirette sulle foto raccolte, infatti le due balze non si presenteranno in posizioni fisse.

Il sistema assicura che le foto vengano scattate sempre alla stessa profondità e che la sonda sia centrata rispetto al pezzo (considerando come centro il centro della cavità cilindrica). La distanza fissa è condizione sufficiente per garantire la messa a fuoco di ognuno dei tre gradini sul fondo. Purtroppo non sono stati specificati dei vincoli riguardo l'illuminazione.

Si fa notare che il processo appena descritto viene eseguito da almeno due macchinari distinti, ovviamente questo aggiunge un ulteriore grado di sfida: non si può supporre che i macchinari siano sempre calibrati esattamente allo stesso modo.

In conclusione le foto che saranno da analizzare vengono raccolte da un totale di quattro fotocamere distinte, delle quali si assicura

- con un grado di precisione soddisfacente la distanza dal fondo;
- con un grado di precisione accettabile la centratura delle immagini.

Capitolo 3

Il Dataset

Innanzitutto si specifica che con la parola "scarto" si indica l'immagine di una carcassa che presenta colla sul fondo, pezzo che quindi dovrà essere scartato. Invece con la parola "conforme" si indica l'immagine di una carcassa nella quale la colla è stata depositata correttamente, quindi con fondo pulito.

In questo capitolo verranno descritti il *dataset*, le problematiche più importanti delle immagini e gli algoritmi principali con cui sono state manipolate. Il capitolo si chiude con la tecnica del *data augmentation*.

3.1 Problematiche principali

3.1.1 Dataset piccolo e sbilanciato

La prima difficoltà insorge ancora prima di ispezionare le immagini del *dataset*. Infatti questo non solo comprende solamente 1749 immagini, ma è anche fortemente sbilanciato:

- 1719 immagini sono conformi;
- 30 immagini sono scarti.

A questo punto è corretto chiedersi se le quasi duemila immagini del *dataset* siano sufficienti ai nostri scopi. Nel campo del *Machine Learning*, ed ancora di più in quello del *Deep Learning*, non è raro che il numero di elementi in un *dataset* sia dell'ordine delle decine di migliaia se non di quello delle centinaia di migliaia. Basti pensare ai *dataset* più famosi ed usati:

- MNIST [1] è un *dataset* molto famoso contenente 70000 cifre disegnate a mano appartenenti a 10 classi in totale. È allo stesso tempo sia il punto di partenza dei principianti, perché di facile manipolazione, sia il campo

di prova degli esperti, sul quale vengono allenati nuovi modelli prima di testarli su compiti più complessi.

- CIFAR-10 [2] contiene 60000 immagini a colori suddivise in 10 classi da 6000 immagini l'una. Le classi comprendono alcuni tipi di veicoli ed animali. Date le dimensioni ridotte delle immagini, appena 32x32 pixel, il *dataset* occupa meno di 200MB, questo permette di allenare reti neurali in tempi ragionevoli.
- ImageNet [3] con i suoi 14 milioni di immagini è il più grande *dataset* che sia mai stato creato. I più grandi esperti nel campo del *Machine Learning* si sfidano annualmente su questo *dataset*. Contiene decine di migliaia di classi e le sue immagini sono in gran parte fotografie scattate da volontari, che quindi hanno condizioni di luminosità e colori molti diversi tra loro. Possono variare anche la dimensione e l'orientamento, orizzontale o verticale, delle immagini.

Ciascuno di questi *dataset* è stato etichettato¹ a mano. Ciò significa che ad ogni immagine è stata assegnata, a mano, una classe di appartenenza. Prendendo in esempio MNIST, se un'immagine raffigura la cifra 7 allora sarà etichettata con il *label seven* ed apparirà alla classe di immagini in cui compare la cifra 7. Allo stesso modo le immagini di ImageNet hanno etichette come *dog*, *cat*, *bird*, *car*, *bike*, ecc...

Sembrerebbe che possedere 2000 immagini appena renda impossibile applicare algoritmi di *Machine Learning*. In realtà osservando i conformi, in figura 3.1 sono stati riportati alcuni esemplari significativi, ci si accorge che i pezzi sono molto simili. Questo non ci sorprende, dato che i pezzi vengono generati meccanicamente, è perfettamente lecito supporre che nessuna carcassa abbia delle caratteristiche completamente differenti dalle altre. Le principali diversità sono dovute ad imperfezioni superficiali come graffi o macchie. Si può quindi dire che la distribuzione dei pezzi conformi è descritta bene anche da un numero limitato di esemplari. Purtroppo il sistema di acquisizione immagini introduce ulteriori fattori di diversità come la posizione delle balze e la luminosità, ma questi fattori potranno essere gestiti senza troppe difficoltà. Concludiamo che il numero di conformi a nostra disposizione è sufficiente ai nostri scopi.

Non possiamo dire lo stesso per gli scarti. Se già la statistica ci lascia sospettare che 30 esemplari non possono ritenersi significativi, allora questo sospetto diventa certezza quando si analizzano le caratteristiche della colla nelle immagini scarto. Come si vede in figura 3.2 la colla può presentarsi in

¹in inglese *labeled* da *label*, etichetta

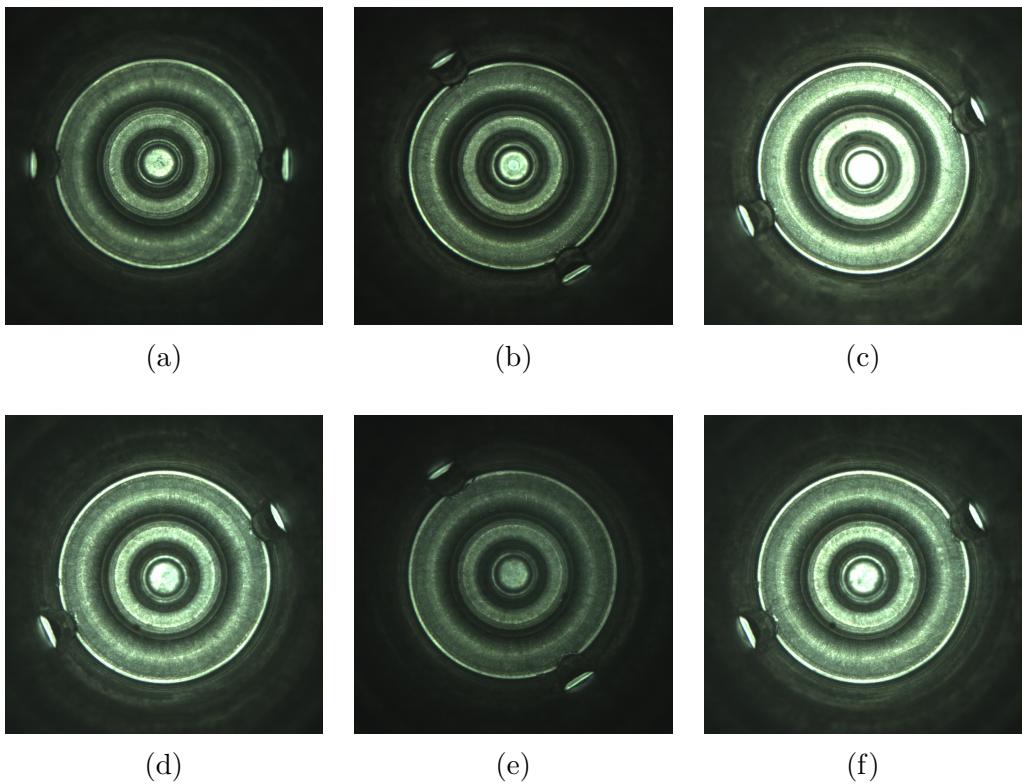


Figura 3.1: Alcune carcasse conformi

forma di gocce più o meno circolari oppure come sbaffi di grossezza e lunghezza variabili. Anche la quantità di superficie coperta dalla colla può variare notevolmente, passando da aree ridotte e localizzate ad aree estese e di conformazioni singolari. Infine notiamo che la posizione del rimasuglio di colla all'interno della carcassa non è in relazione con la posizione delle balze e che la presenza dei gradini sul fondo non la obbliga in alcun modo a scivolare fino al centro.

Per analizzare meglio le modalità con cui potrebbe essere generato uno scarto si supponga che il macchinario abbia commesso un errore: dall'ugello è uscita un certa quantità di colla in esubero. A seconda della posizione dell'ugello rispetto alla carcassa si può immaginare che la colla raggiunga il fondo in vari modi, proviamo ora ad illustrarne due:

- nel primo caso si immagina che il braccio abbia già depositato l'anello di colla e che si stia allontanando dalla carcassa. La colla in esubero cadrebbe sotto forma di goccia fino a raggiungere il fondo del pezzo. Questo potrebbe esse il caso per la figura 3.2d;

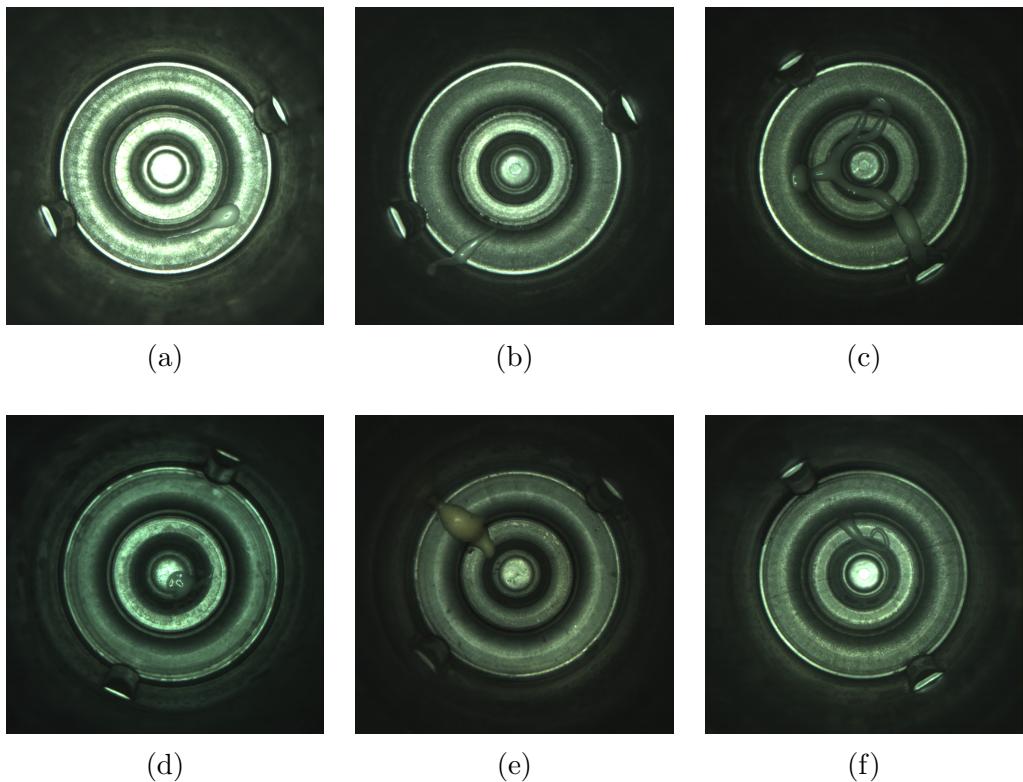


Figura 3.2: Alcune carcasse scarto

- nel secondo caso si ipotizza che la colla in eccesso faccia parte dell'anello appena depositato e che, a causa delle vibrazioni o di altri fattori simili, colì raggiungendo il fondo della carcassa. Questo potrebbe essere il caso per la figura 3.2b.

Data la grande varietà di modi in cui la colla può presentarsi, concludiamo che gli esemplari forniti per la classe scarto la descrivono soltanto in modo parziale. Ciò rappresenta la maggiore difficoltà del *dataset*. Alla base di ogni applicazione di tecniche di *machine learning* c'è l'ipotesi che il *training set*² contenga informazione sufficiente a descrivere l'intero spazio campionario. Senza questa assunzione non è possibile pretendere che il modello sia performante su *input* non ancora osservati. Si conclude che il numero di scarti fornito non è sufficiente per effettuare un allenamento classico di un qualche classificatore binario.

Ora è doveroso spendere alcune parole per commentare la colorazione delle immagini rispetto ai veri colori delle carcasse e della colla. In figura 2.1 a pagina

²Sezione del *dataset* su cui viene effettuato l'allenamento della rete.

8 abbiamo visto che la superficie del pezzo è di colore grigio ma nella foto risulta di colore verdastro. Allo stesso modo anche la colla, in realtà di colore bianco sporco, nella fotografia assume tonalità verdognole. Per certi compiti possedere immagini in falsi colori può risultare problematico ma fortunatamente non è questo il caso: l'importante è che venga mantenuta l'informazione che ci permette di distinguere la colla dalla superficie della carcassa. Come vedremo poi, le immagini verranno trasformate in scala di grigi. Quindi, nonostante sia preferibile avere immagini a colori reali, i falsi colori non sono da considerarsi problematici.

3.1.2 Differenze tra immagini

Ora che abbiamo una visione d'insieme sul dataset possiamo concentrare la nostra attenzione sulle proprietà principali delle immagini. Innanzitutto ogni immagine ha una risoluzione di 896x896 *pixel*, dimensione che ci permette di esplorare varie possibilità. Ad esempio, si può pensare di ridurre l'immagine ad una dimensione tale da occupare meno spazio in memoria, quindi in RAM durante l'allenamento della rete, ed allo stesso tempo di mantenere un livello di dettaglio sufficiente ai nostri scopi. Oppure si può pensare di suddividere l'immagine in quadranti da analizzare singolarmente, così da mantenere la qualità dell'immagine originale ma senza dover creare una rete che accetti immagini troppo grandi. Infatti, una rete che prende in input immagini di grandi dimensioni, solitamente avrà un numero di parametri maggiore di una che accetta immagini piccole. Questo porta non solo ad occupare più spazio in memoria, ma significa anche che la rete impiegherà più tempo in fase d'allenamento, perché dovranno essere trovati un maggior numero di parametri. Per avere un termine di paragone basti pensare che le immagini di MNIST sono 64x64 *pixel* mentre quelle di ImageNet, che ricordiamo hanno dimensioni variabili, vengono solitamente scalate a 224x224*px*. Quest'ultima è la dimensione accettata da due delle reti che storicamente hanno ottenuto risultati allo stato dell'arte su ImageNet. Prima, nel 2014, la rete VGG [5] e, l'anno dopo, l'architettura ResNet [4].

Osservando nuovamente figura 3.1 ci si accorge che le immagini hanno varie proprietà. Verranno ora elencate a partire da quella da considerarsi meno problematica fino ad arrivare a quella più problematica.

- Ogni immagine presenta tre circonferenze concentriche, con centro il centro del pezzo. Ciascuna circonferenza è definita da una transizione da una zona più scura ad una più chiara. Sappiamo che le zone più scure corrispondono alle pareti verticali del pezzo mentre le zone chiare ai tre

gradini sul fondo. Questa proprietà non è problematica, anzi può essere sfruttata a nostro vantaggio.

- Dato che le immagini vengono raccolte ad una distanza costante dal fondo, la dimensioni delle circonferenze sono fissate e si mantengono coerenti tra le immagini. Anche questa proprietà può essere usata a nostro vantaggio.
- Le due balze sulla parete verticale sono ben visibili e possono presentarsi, sempre una di fronte all'altra, in ogni posizione lungo una circonferenza di raggio pari al raggio della cavità cilindrica. Possono essere considerate un problema in quanto rappresentano informazione superflua e variabile. Ricordiamo che la posizione delle balze non ha alcuna correlazione con la presenza della colla, tanto meno con la sua posizione.
- Le superfici dei pezzi si assomigliano: tutte presentano un effetto chiamato “sale e pepe” con granuli di grandezze e luminosità variabili. Bisogna prestare particolare attenzione alle macchie scure presenti sul fondo di alcune carcasse. La posizione delle macchie non è fissa, perdipiù anche la loro dimensione è variabile. Queste caratteristiche superficiali non saranno da sottovalutare in fase di elaborazione delle immagini.
- Ad un primo sguardo potrebbe sembrare che le immagini siano tutte centrate allo stesso modo, invece in molte il centro dell'immagine non corrisponde con il centro del pezzo. Nonostante la distanza massima tra centro del pezzo e centro dell'immagine sia tale per cui il fondo della carcassa sia sempre visibile interamente, sarebbe stato preferibile fossero centrate tutte correttamente.
- La variazione di luminosità tra le varie foto è una problematica. Infatti, alcune immagini hanno una luminosità così alta da far risultare alcune superfici bianche. Altre immagini, invece, sono molto più scure, tanto che anche le zone che normalmente rifletterebbero sono illuminate appena.

Ora, facendo riferimento alla figura 3.2, possiamo elencare le proprietà esclusive degli scarti. In questo caso sono tutte non problematiche, anzi sono sfruttabili poiché rappresentano informazione con cui si può distinguere uno scarto da un conforme:

- La colla ha alcune caratteristiche particolari: ha un colore bianco-verde solitamente più chiaro della superficie della carcassa e presenta sempre delle zone con dei riflessi.

- La colla è localizzata. Significa che, se presente, non appare come tante gocce sparse, ma come un corpo unico più o meno allungato.
- Tracciando un diametro a piacere ci si accorge che gli scarti sono sempre asimmetrici; invece i conformi, a meno di piccole differenze superficiali, sono sempre simmetrici.

3.2 Pre-Processing

Questa sezione è divisa in due parti: nella prima verranno illustrate alcune tra le principali tecniche di elaborazione digitale delle immagini, esponendo i dettagli matematici ed esplorando le loro applicazioni; nella seconda si spiegherà quali di queste tecniche, in che ordine e per quali motivi sono state utilizzate.

Come prima cosa è bene ricordare che con *elaborazione digitale delle immagini* si intende il modificare immagini digitali per mezzo di algoritmi eseguibili da un calcolatore. Mentre con *pre-processing* si indica la manipolazione delle immagini digitali per renderle utilizzabili da altri algoritmi.

Gli algoritmi utilizzati hanno precise formulazioni matematiche, perché ogni immagine viene rappresentata come una matrice bidimensionale se in scala di grigi, oppure tridimensionale se a colori. Gli elementi di una matrice bidimensionale appartengono all'intervallo $[0, 255]$, nel quale 0 corrisponde al colore nero mentre 255 corrisponde al bianco. Disponendo una sopra l'altra tre matrici come quelle appena descritte si ottiene un'immagine a colori: ogni matrice rappresenta uno dei canali principali (*Red*, *Green*, *Blue* da cui il famoso acronimo RGB) dell'immagine. Sia I un'immagine a tre canali (RGB), il colore del *pixel* in posizione (i, j) è dato dalla tripletta $(I[i, j, 0], I[i, j, 1], I[i, j, 2])$, in cui: $(0, 0, 0)$ indica il colore nero, $(255, 255, 255)$ indica il colore bianco, $(255, 0, 0)$ indica il colore rosso, $(0, 255, 0)$ indica il colore verde, e così via. Quindi un'immagine avrà un numero finito di elementi, detti *pixel*, la cui quantità si può ottenere moltiplicando il numero di colonne della matrice per il numero di righe.

Uno dei vantaggi del rappresentare le immagini come matrici è quello di poter applicare operazioni classiche come somma, sottrazione, prodotto e divisione. Ma la nostra attenzione si concentrerà soprattutto sulle convoluzioni.

Convoluzioni Nell'ambito dell'elaborazione digitale delle immagini con convoluzione si intende l'operazione che permette di effettuare, per ogni *pixel* dell'immagine, una somma pesata tra il *pixel* e gli elementi a lui vicini. I pesi sono definiti in una matrice, detta *kernel* o filtro, di dimensioni non superiori

a quelle dell'immagine di partenza. Solitamente i *kernel* hanno dimensione 3x3 o 5x5. Una convoluzione è composta da semplici passi, come illustrato in [6]. Prima il filtro viene centrato su un *pixel*. A questo punto è come se il *kernel* coprisse un'area quadrata dell'immagine: moltiplichiamo i valori dei *pixel* con i rispettivi valori del filtro. I prodotti così ottenuti dovranno essere sommati assieme, il risultato della somma sarà il nuovo valore del *pixel* su cui il *kernel* era stato centrato. Effettuiamo questa operazione per ogni elemento dell'immagine. Solitamente le convoluzioni si effettuano da sinistra a destra e dall'alto verso il basso, ma si fa presente che l'ordine d'esecuzione non deve modificare il risultato. È infatti importante aggiornare i *pixel* solo dopo aver ottenuto i nuovi valori di tutta l'immagine.

Feature Extraction Prima di passare alla descrizione degli algoritmi utilizzati si vuole specificare che cosa significa *Feature Extraction*, in italiano Estrazione di Caratteristiche. Citando parte della definizione data in [7] :

Feature extraction is the name for methods that select and/or combine variables into features, effectively reducing the amount of data that must be processed, while still accurately and completely describing the original data set.

Quindi, in generale, indica un procedimento con cui, da un insieme di dati complesso, si estrapola un gruppo di informazioni ridotte, ma con la stessa capacità espressiva. Nell'ambito dell'elaborazione digitale delle immagini le tecniche di estrazione delle caratteristiche possono essere raccolte in varie categorie. Ciascuna permette di ottenere nuove immagini a partire da immagini digitali, oppure coordinate di zone d'interesse dell'immagine in ingresso. Noi esploreremo tecniche come: il rilevamento dei contorni o *edge detection*, la sogliatura o *thresholding* e le trasformate di Hough.

3.2.1 Passaggio da RGB a GrayScale

La conversione di un'immagine da RGB in scala di grigi è un'operazione estremamente facile, ma rimane comunque alla base di molti algoritmi di elaborazione delle immagini. Infatti, per molti compiti l'informazione sul colore non è necessaria, basti pensare al rilevamento di forme od il riconoscimento di testo. Il modo più semplice per combinare i tre canali RGB in un unico canale è quello di effettuare la media dei valori *pixel* per *pixel*. Siano R , G e B i tre canali di un'immagine a colori I e sia Y l'immagine in bianco e nero che si vuole ottenere:

$$Y = (R + G + B)/3 \quad (3.1)$$

Nell'equazione 3.2.1 ogni canale partecipa allo stesso modo, quindi Y è semplicemente la media aritmetica dei tre canali di partenza. Sappiamo, però, che l'occhio umano è più sensibile ai colori verdi, quindi per certe applicazioni potrebbe essere preferibile dare più importanza al secondo canale:

$$Y = 0.299 * R + 0.587 * G + 0.114 * B \quad (3.2)$$

I pesi utilizzati nell'equazione 3.2.1 seguono lo standard CCIR 601 [8]. Un esempio di applicazione dell'equazione 3.2.1 è illustrato in figura 3.3.



Figura 3.3: A sinistra l'immagine originale. A destra la versione in scala di grigi

3.2.2 Masking

La tecnica del *masking* permette di nascondere parti di immagine a cui non siamo interessati. Per fare questo abbiamo bisogno di una maschera binaria e dell'immagine che si vuole mascherare. Per immagine binaria si intende un'immagine in cui ogni *pixel* appartiene all'insieme $\{0, 1\}$, cioè è nero oppure bianco. Solitamente la maschera viene generata a mano con semplici *editor*. Si fa notare che quest'ultima deve avere le stesse dimensioni dell'immagine di partenza. L'operazione consiste nell'effettuare un AND logico, *pixel* per *pixel*, tra l'immagine e la maschera. Così facendo tutti i *pixel* dell'immagine di partenza corrispondenti a zone di valore 0 della maschera verranno impostati a 0, diventando quindi neri. Il resto dell'immagine non viene alterato.

Nell'esempio in figura 3.4 si è deciso di rimuovere l'informazione relativa allo sfondo.

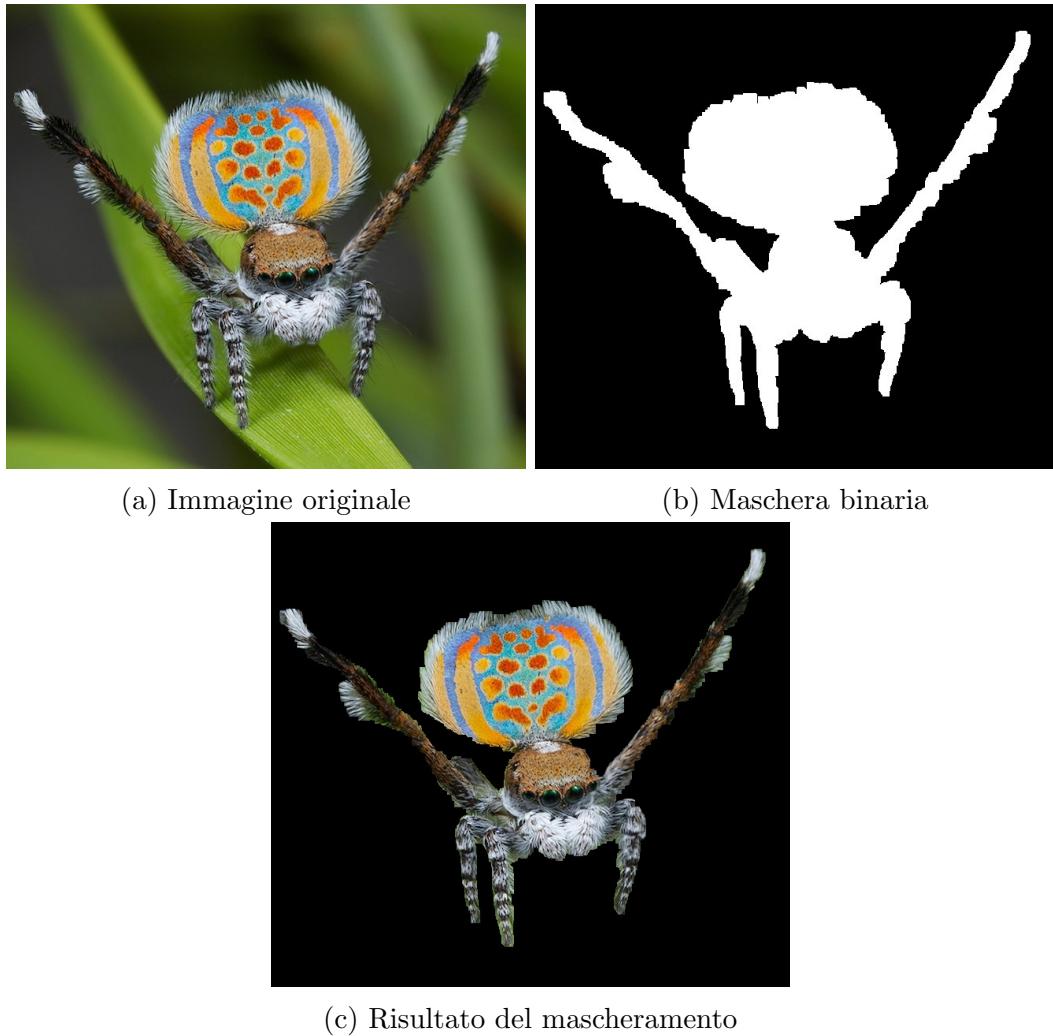


Figura 3.4: Esempio di mascheramento di un'immagine a colori

3.2.3 Image cropping and resizing

Quando si effettua un *image cropping* si ritaglia una porzione dell'immagine, che viene chiamata ROI (*Region Of Interest*), sulla quale si vuole concentrare l'attenzione. Dato che ogni immagine è rappresentata con una matrice, una ROI non sarà nient'altro che una matrice di dimensioni minori in cui sono stati copiati i valori dell'area interessata. Una matrice di questo tipo viene anche chiamata vista o *view*.

Con l'*image resizing* si aumentano (o diminuiscono) le dimensioni di un'immagine. Nel primo caso, poiché si vuole aumentare il numero di *pixel* dell'immagine finale, bisognerà utilizzare tecniche di *upsampling* ed interpolare i dati a disposizione per generarne di nuovi che siano verosimili.

Uno fra gli algoritmi più semplici è *Nearest-Neighbor Interpolation*: il *pixel* che deve essere aggiunto ottiene il valore del *pixel* a lui più vicino. Un criterio di scelta dovrà essere definito nel caso in cui ci siano più *pixel* alla stessa distanza, ma con valori differenti. Un criterio possibile è quello di assegnare al nuovo *pixel* il valore del *pixel* alla sua sinistra ed in alto.

Una tecnica leggermente più complessa, ma che fornisce risultati soddisfacenti nella maggior parte delle occasioni, è l'interpolazione bilineare. Con questa tecnica si effettuano due interpolazioni lineari in cascata, una orizzontale ed una verticale. Con l'interpolazione bilineare i nuovi *pixel* si ottengono come media dei valori noti, pesata rispetto allo loro distanza dal *pixel* che si vuole colorare. In questo modo si ottengono immagini con transizioni di colore più dolci. Maggiori informazioni ed immagini esemplificative possono essere trovate, per quanto riguarda l'interpolazione lineare, in [9] e per l'interpolazione bilineare, in [10].

Nel caso in cui si voglia ridurre le dimensioni dell'immagine si dovranno usare tecniche di *downsampling* ed *anti-aliasing*. Il *downsampling* permette di selezionare un numero limitato di *pixel* che poi verranno usati per colorare la matrice di dimensione ridotta. Applicare soltanto questa tecnica può portare alla creazione di artefatti sintetici nell'immagine risultato. Significa che l'immagine ridotta potrebbe contenere gruppi di *pixel* di colori sbagliati. Sfruttando tecniche come l'*anti-aliasing* si può evitare, o quantomeno limitare, la creazione di tali artefatti. Un *low-pass filter* è un tipo di filtro che smorza tutti i valori al di sopra di una certa soglia, mentre lascia passare tutti i valori al di sotto. Nel campo dell'elaborazione digitale delle immagini vengono utilizzati filtri di *blur* applicati tramite convoluzione. Il termine *blur* o *smooth* indica applicare un effetto sfocato, che tende a rendere più dolci le transizioni da un colore all'altro. Andando quindi anche a ridurre valori troppo alti (o troppo bassi) avvicinandoli a valori più probabili.

3.2.4 Histogram equalization

Citando Wikipedia [11]:

Histogram equalization is a method in image processing of contrast adjustment using the image's histogram.

Si ricorda che il contrasto è definito come la differenza in intensità luminosa e colore che permette di distinguere gli oggetti in un'immagine. In figura 3.5 sono state riportate un'immagine a basso contrasto e l'immagine risultato dopo l'applicazione della *histogram equalization*. Nella figura si possono osservare anche gli istogrammi relativi alle immagini: sull'asse delle x abbiamo ogni possibile valore di un *pixel*, quindi nell'intervallo da 0 a 255; sull'asse delle y è riportato il numero di occorrenze di quel colore nell'immagine. Notare che l'immagine in ingresso deve essere in scala di grigi.

Vediamo ora come la costruzione dell'istogramma e la sua manipolazione possono essere formulate matematicamente. Siano X l'immagine di partenza ed Y l'immagine risultato. Facendo riferimento a quanto scritto nel documento [12] l'istogramma può essere definito come:

$$p_n = \frac{\text{numero di pixel di colore } n}{\text{numero totale di pixel}} \quad \text{con } n \in [0, 255]$$

Poiché p_n descrive la probabilità che un *pixel* scelto a caso dall'immagine abbia valore n , possiamo considerare X una variabile casuale discreta in $[0, 255]$. Quindi X avrà funzione di ripartizione, tracciata in blu nel grafico in figura 3.5c, pari ad:

$$F_X(x) = \sum_{n=0}^x p_n$$

Vorremmo che la differenza di colore tra i pixel fosse più netta, così da aumentare il contrasto dell'immagine. Per raggiungere i nostri scopi possiamo ridistribuire equamente i valori di X nell'intervallo dei valori possibili:

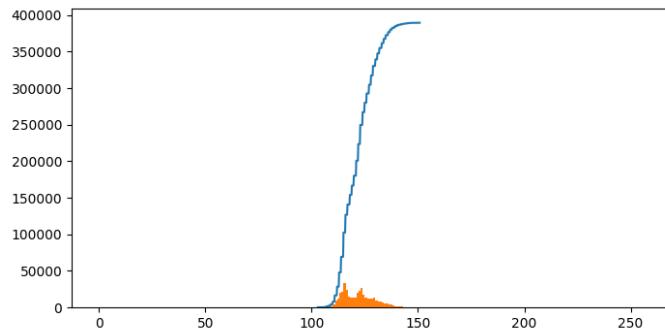
$$T(x) = \text{floor}(255 * F_X(x))$$

dove *floor()* arrotonda verso il basso all'intero più vicino. In questo modo possiamo ottenere la variabile casuale trasformata $Y = T(X)$, ossia l'immagine equalizzata. Sostanzialmente abbiamo ricolorato ogni pixel dell'immagine di partenza con colori più distanti tra loro, come si può vedere nell'istogramma in figura 3.5d.

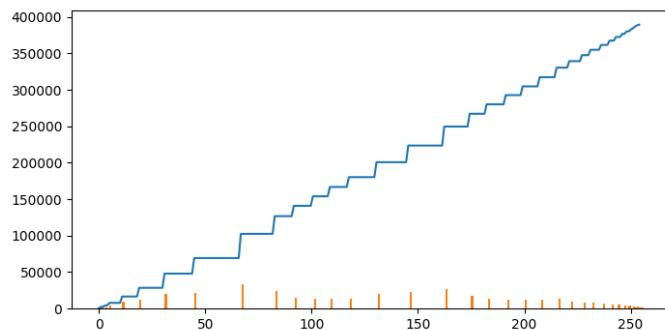


(a) Immagine originale

(b) Immagine equalizzata



(c) Istogramma dell'immagine originale



(d) Istogramma dell'immagine equalizzata

Figura 3.5: Esempio di applicazione di *histogram equalization*

3.2.5 Gaussian blur

Conosciuto anche come *Gaussian smoothing*, il *Gaussian blur* permette di sfocare un'immagine sfruttando una convoluzione con *kernel* gaussiano. L'immagine così ottenuta risulta meno nitida, con meno dettagli e, quindi, con meno rumore. L'obiettivo principale di questa tecnica è mantenere soltanto l'informazione caratterizzante, rimuovendo quella non necessaria od anomala. Si ricorda che la funzione di Gauss ad un parametro è

$$G(x) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{x^2}{2\sigma^2}\right)$$

Si dimostra che la funzione di Gauss a due parametri equivale al prodotto di due funzioni come quella appena definita, in formule

$$\begin{aligned} G(x, y) &= G(x)G(y) \\ &= \frac{1}{2\pi\sigma^2} \exp\left(-\frac{x^2 + y^2}{2\sigma^2}\right) \end{aligned}$$

in cui x ed y sono le distanze dagli assi di riferimento, mentre σ è la deviazione standard.

La matrice sottostante rappresenta un'approssimazione di un filtro gaussiano quadrato di lato 7 con $\sigma = 2$.

$$\frac{1}{100} * \begin{bmatrix} 0.5 & 0.9 & 1.3 & 1.5 & 1.3 & 0.9 & 0.5 \\ 0.9 & 1.7 & 2.4 & 2.8 & 2.4 & 1.7 & 0.9 \\ 1.3 & 2.4 & 3.6 & 4.0 & 3.6 & 2.4 & 1.3 \\ 1.5 & 2.8 & 4.0 & 4.6 & 4.0 & 2.8 & 1.5 \\ 1.3 & 2.4 & 3.6 & 4.0 & 3.6 & 2.4 & 1.3 \\ 0.9 & 1.7 & 2.4 & 2.8 & 2.4 & 1.7 & 0.9 \\ 0.5 & 0.9 & 1.3 & 1.5 & 1.3 & 0.9 & 0.5 \end{bmatrix}$$

Ricordiamo che durante una convoluzione si effettua la somma dei prodotti elemento per elemento, ossia una media pesata in cui i pesi sono definiti nel kernel. Dato che i valori al centro del filtro sono più grandi di quelli ai bordi, daremo maggior peso ai *pixel* nell'intorno dell'elemento su cui il kernel viene centrato. All'aumentare di σ cresce il raggio alla base della campana di Gauss. Questo significa che, con σ grandi, verrà data importanza anche ai *pixel* distanti, ciò farà risultare l'immagine in uscita molto più sfocata.

In figura 3.6 è riportata un'immagine prima e dopo l'applicazione del filtro riportato in 3.2.5. Si fa notare come i dettagli più piccoli sono stati rimossi, mentre l'oggetto dell'immagine rimane distinguibile.

aggiungere
ref per studio
futuro



Figura 3.6: A sinistra l’immagine originale. A destra il risultato del *blur*

3.2.6 Sobel operator

Prima di descrivere cosa sia il *Sobel operator*, detto anche filtro Sobel, bisogna dare una definizione di gradiente. Nel calcolo vettoriale, il gradiente è la generalizzazione della derivata. La derivata di una funzione ad una variabile associa ad ogni punto uno scalare, mentre il gradiente di una funzione a più variabili f associa ad ogni punto un vettore multidimensionale. Quest’ultimo è composto dall’insieme delle derivate parziali di f nel punto considerato. Il gradiente rappresenta la pendenza della tangente al grafico della funzione in un punto. La sua direzione indica il più grande incremento della funzione mentre il modulo³ è il tasso d’incremento.

Possiamo pensare un’immagine (in scala di grigi) come una funzione a due variabili che associa ad ogni punto (x, y) un valore in $[0, 255]$. Il gradiente dell’immagine sarà composto da vettori direzionati in modo da uscire dalle zone scure ed entrare nelle zone più chiare (mantenendo la convenzione per cui a zero è associato il colore nero). La magnitudine sarà tanto più grande quanto più grande il contrasto, quindi la differenza di colore e luminosità.

Ora possiamo procedere con la descrizione del filtro di Sobel, facendo riferimento a quanto scritto in [14]. Il *Sobel filter* viene utilizzato per creare immagini in cui si enfatizzano gli *edge*. Per *edge* si intendono tutte quelle zone dell’immagine che corrispondono a bordi, margini o spigoli degli oggetti rappresentati nell’immagine. Ciascuna di queste zone deve necessariamente essere associata almeno ad un cambio di colore oppure ad un cambio di luminosità,

³Detto *magnitude* in inglese.

altrimenti l'oggetto in questione non sarebbe distinguibile dallo sfondo. Quindi, osservando il gradiente dell'immagine, ad ogni *edge* corrisponderà una serie di vettori con modulo più grande rispetto ai moduli dei vettori circostanti. Lo scopo del filtro Sobel è generare un'approssimazione del gradiente dell'immagine sfruttando due convoluzioni distinte con uno specifico kernel. Nonostante il risultato sia abbastanza grossolano, la sua efficacia e rapidità lo rendono uno dei principali strumenti per la *edge detection*, tecnica che esploreremo fra poco.

L'applicazione del filtro Sobel avviene come mostrato nelle equazioni 3.3 e 3.4, dove $*$ denota una convoluzione ed I è un'immagine in scala di grigi.

$$G_x = I * \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix} \quad (3.3)$$

$$G_y = I * \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix} \quad (3.4)$$

L'equazione 3.3 fornisce un'approssimazione del gradiente rispetto all'asse x , mentre l'equazione 3.4 rispetto ad y . L'equazione 3.5, invece, combina le precedenti fornendo informazione riguardo al modulo del gradiente dell'immagine. Con l'equazione 3.6 si ottiene la direzione del gradiente.

$$G = \sqrt{G_x^2 + G_y^2} \quad (3.5)$$

$$\Theta = \text{atan}\left(\frac{G_y}{G_x}\right) \quad (3.6)$$

Ora verranno fatte delle considerazioni sul kernel mostrato nella formula 3.3. Poiché il filtro nell'equazione 3.4 è una semplice trasposta del precedente, tali considerazioni sono valide anche per questo filtro, facendo riferimento all'asse delle y . Il kernel nell'equazione 3.3 assegnerà valori in assoluto più grandi a *pixel* in una posizione di transizione di colore, rispetto a *pixel* posizionati in una zona con colorazione uniforme. Prendiamo un *pixel* p di colore bianco posizionato in una zona in cui tutti i *pixel* sono di colore bianco, si avrà che il *kernel*, pesando i *pixel* a destra allo stesso modo di quelli a sinistra ma con segno opposto, attribuisce un valore pari a 0 a p . Se p fosse stato in una zona di transizione dal bianco al nero, avrebbe ottenuto un valore negativo molto grande: tutti i valori nella prima colonna del filtro sarebbero stati moltiplicati per 255 mentre tutti quelli dell'ultima colonna sarebbero stati annullati, essendo moltiplicati per 0. Uno stesso ragionamento può essere effettuato considerando zone completamente nere oppure di transizione dal nero al bianco.

Nell'immagine sottostante viene mostrata l'applicazione dell'operatore Sobel prima rispetto ad x , poi rispetto ad y ed infine il risultato della combinazione delle precedenti.

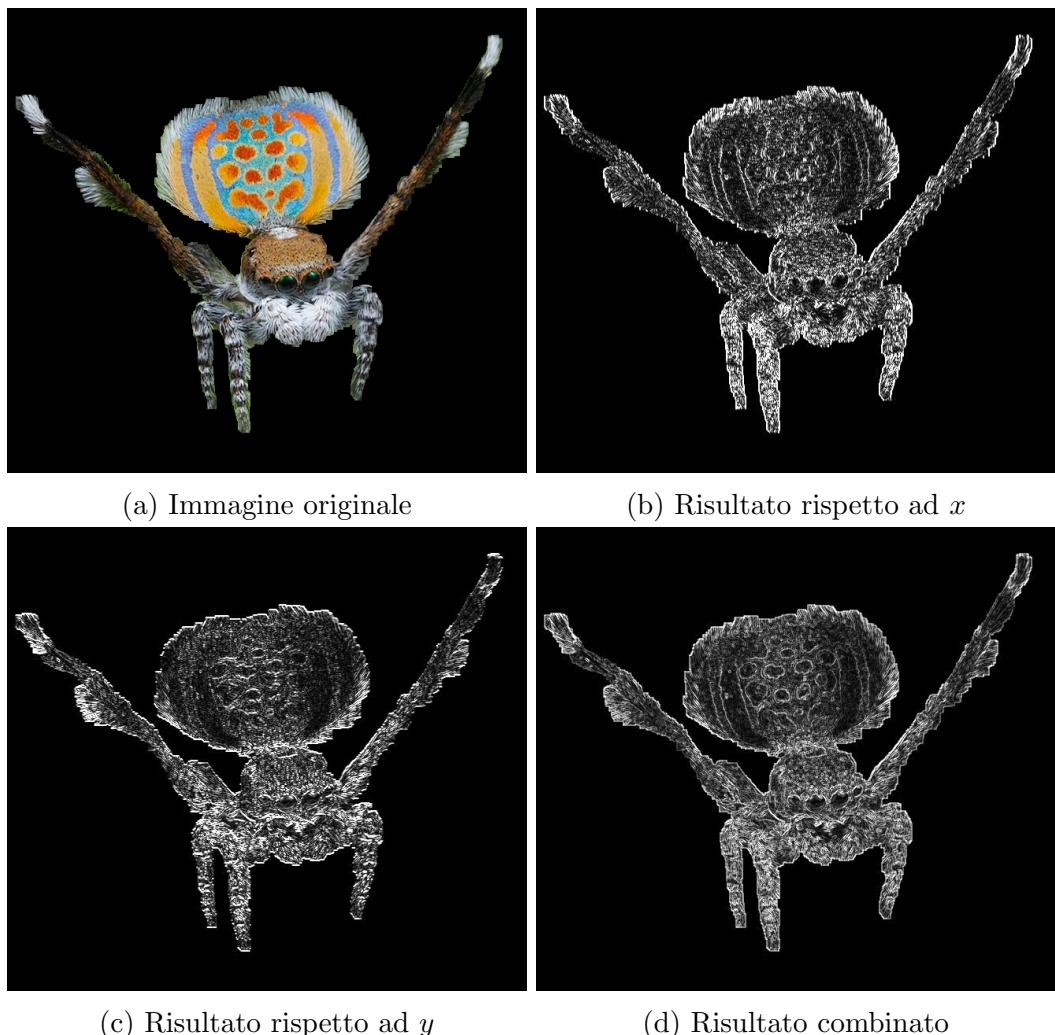


Figura 3.7: Esempio di applicazione del filtro Sobel

3.2.7 Canny edge Detector

Composto da vari passaggi ed algoritmi, il *Canny Edge Detector* permette di estrarre i principali *edge* di un’immagine ottenendo risultati soddisfacenti ed adattabili alle necessità dell’utilizzatore. Questo rilevatore viene sfruttato largamente come tecnica di estrazione delle caratteristiche perché permette di rimuovere molta informazione dall’immagine, mantenendo solo quella strettamente necessaria. Nell’ambito del *edge detection* è bene che tutti gli *edge* presenti nell’immagine vengano identificati correttamente, ma è altrettanto importante che non vengano generati falsi positivi, cioè che delle aree nel risultato presentino contorni che nell’immagine di partenza non esistevano. Per questo motivo *Canny edge detector* effettua molte operazioni di rimozione di falsi positivi ed allo stesso tempo tende ad evidenziare molto bene quelli che sembrano essere *edge* autentici. Ora verranno elencati in ordine i vari passaggi del rilevatore:

1. Viene applicato un filtro gaussiano per effettuare uno *smooth* all’immagine. Questo passaggio è fondamentale perché vengono rimossi valori anomali che, risultando come picchi positivi o negativi, potrebbero contribuire a generare falsi positivi. Inoltre può essere usato per rimuovere dettagli superflui.
2. Utilizzando il filtro Sobel si ottiene il modulo dei gradienti dell’immagine, ossia tutti gli *edge* candidati, che dovranno poi essere manipolati e selezionati.
3. Con una doppia sogliatura si ottengono due effetti: come prima cosa vengono rimossi tutti quegli *edge* con magnitudine troppo bassa, perché considerati rumore. Successivamente gli *edge* sopravvissuti vengono classificati come forti e deboli, questa informazione verrà sfruttata nel prossimo passaggio. Effettuare una sogliatura significa semplicemente osservare ogni valore di un’immagine (nel nostro caso ogni *pixel* indica il modulo del gradiente) e mapparlo ad un valore dato se soddisfa determinate condizioni. In questo passaggio sono presenti due soglie. La prima ci permette di annullare tutti gli *edge* con valori troppo piccoli. La seconda viene usata per etichettare i *pixel* superiori alla soglia come *edge* forti. Tutti i *pixel* intermedi vengono considerati *edge* deboli.
4. A questo punto si procede con il rimuovere tutti gli *edge* deboli che non facciano parte di nessun *edge* forte. Significa che, per ogni *pixel* etichettato come *edge* debole, vengono osservati gli otto *pixel* circostanti. Se nessuno di questi è un *edge* forte allora il *pixel* centrale viene annullato.

Va fatto notare che i valori delle due soglie devono essere trovati in modo empirico, a seconda del tipo d'immagine bisognerà impostare soglie più o meno grandi, ma anche più o meno distanti fra loro.

In figura 3.8 è illustrata l'applicazione del *Sobel operator*, notare come il risultato sia molto più pulito di quello illustrato in figura 3.7.



Figura 3.8: Esempio di applicazione del *Canny Edge Detector*

3.2.8 Hough circle transform

Prima di poter spiegare questa tecnica si devono introdurre alcuni concetti come la *Hough Line Transform*, lo *Hough Parameter Space* e la parametrizzazione raggio-angolo di una retta. Le trasformate di Hough sono largamente usate come metodi di estrazione delle caratteristiche permettono di rilevare semplici figure come linee e cerchi. Il loro obiettivo è raggruppare correttamente un insieme di pixel in modo che l'insieme raffiguri la forma che si vuole trovare. Una delle problematiche principali è la possibilità che nell'immagine la forma da rilevare sia discontinua o deformata.

Esploriamo, facendo riferimento a quanto illustrato in [15], come il metodo di *Hough Line Transform* permette di trovare una linea retta in un'immagine. Definiamo tre punti $P_i = (x_i, y_i)$ con $i = 1, 2, 3$ in modo che siano allineati, cioè che esista una retta alla quale appartengano tutti. Sappiamo che per ciascun punto ci sono infinite rette che soddisfano $y_i = mx_i + c$, ma che soltanto una di queste rette passa anche per gli altri due punti. L'equazione appena descritta è in funzione di m e c , riscrivendola come

$$c = y_i - mx_i \quad \text{con} \quad i = 1, 2, 3 \quad (3.7)$$

risulta più facile immaginarla (fissato un i) come una retta in uno spazio in cui c è sull'asse delle ordinate e m su quello delle ascisse. Lo spazio appena descritto prende il nome di *Hough Parameter Space*, in cui ogni punto (m, c) descrive una retta nello spazio di partenza. Nello spazio dei parametri di Hough, le tre rette della formula 3.7 si devono necessariamente incontrare in esattamente un punto, sia (m_0, c_0) . Questo è dovuto al fatto che sappiamo che deve esistere una coppia di parametri per cui la retta $y = m_0x + c_0$, nello spazio originale, passa per tutti i P_i . L'idea alla base della *Hough Line Transform* è quella di trovare quello che soddisfa il maggior numero di equazioni, definito un range di valori per m e per c . In pratica ad ogni cella nello spazio $m-c$ viene associato un intero che indica quante delle rette di equazione 3.7 passano per quella cella.

Ci si accorge facilmente che questo metodo non ci permette di rilevare rette parallele all'asse delle y ; infatti, bisognerebbe avere un parametro m con valore che tende all'infinito. Per risolvere questa problematica viene usata la parametrizzazione raggio-angolo. Come si vede in figura 3.9a, ogni retta viene identificata da una coppia univoca (r, θ) . Nella coppia r indica la distanza dall'origine al punto più vicino della retta, mentre θ indica l'angolo tra il segmento appena generato e l'asse delle x . Ogni retta viene identificata dall'equazione:

$$r = x_i \cos \theta + y_i \sin \theta \quad \text{con } i = 1, 2, 3 \quad (3.8)$$

ottenibile con semplici calcoli geometrici. Questo significa che il fascio di rette nello spazio di partenza verrà descritto da curve nello spazio dei parametri di Hough con assi $\theta-r$, come si vede in figura 3.9b. Valutando le celle nello stesso modo di prima si ottiene la retta passante per i tre punti, ma identificata dalla coppia (r_0, θ_0) .

Vediamo ora come si può modificare la tecnica appena descritta per rilevare cerchi anziché linee rette. L'idea principale ruota sempre attorno al concetto di voto di celle nello spazio dei parametri.

In questo caso si sfrutta il fatto che, in una circonferenza c di raggio r , ogni punto è equidistante dal centro. Per ogni punto P_i di c può essere tracciata una circonferenza c_i di centro P_i e raggio r . Sappiamo che tutte le circonferenze c_i si incontrano in un unico punto: il centro di c , poiché è l'unico equidistante da tutti i centri delle c_i .

Si ricorda che l'equazione di una circonferenza di centro (a, b) e raggio r è

$$(x - a)^2 + (y - b)^2 = r^2 \quad (3.9)$$

Supponiamo di conoscere il raggio r_0 della circonferenza che vogliamo rilevare e di avere un insieme di punti P_i che sappiamo appartenere alla circonferenza. Lo spazio dei parametri $a-b$ ci permette di creare le circonferenze centrate

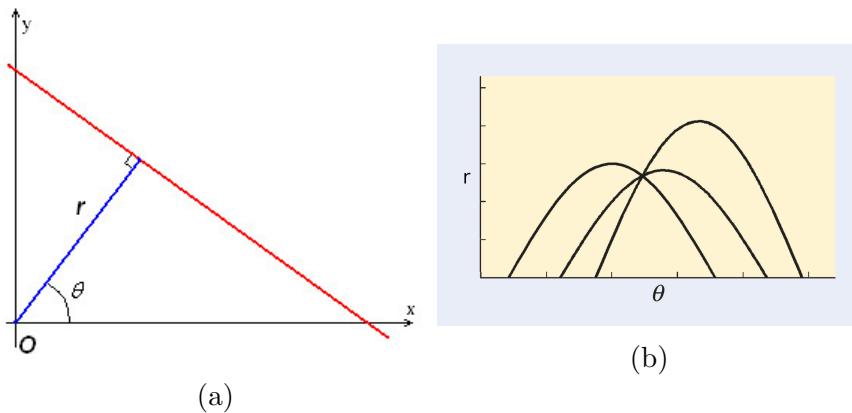


Figura 3.9: Parametrizzazione raggio-angolo e rappresentazione dello spazio dei parametri di Hough

nei P_i e di assegnare un punteggio alto alla cella in (a_0, b_0) perché votata da molte circonferenze. In figura 3.10 si vede la griglia su cui vengono votate le circonferenze.

Se anche il raggio fosse incognito allora lo spazio dei parametri di Hough avrebbe tre dimensioni ed ogni punto sarebbe una tripla (a, b, r) appartenente alla superficie di un cono.

Le implementazioni della *Hough Circle Transform* sfruttano una matrice di supporto, chiamata accumulatore, che corrisponde allo spazio dei parametri di Hough. Ciascun elemento è un intero con valore iniziale zero, che verrà incrementato di uno per ogni linea passante per quelle coordinate.

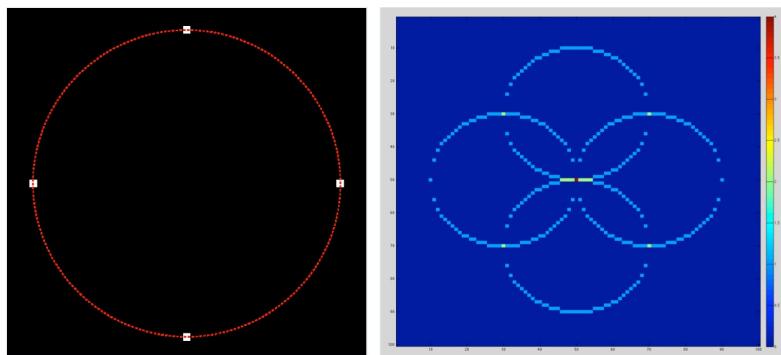


Figura 3.10: Circonferenza e spazio dei parametri di Hough per le circonferenze con raggio dato

3.3 Applicazione degli algoritmi descritti

Si fa notare che prima di ottenere immagini come quella illustrata in figura 3.11 e soprattutto prima di essere sicuri che questo è il tipo di immagine migliore per i nostri scopi, sono stati necessari numerosi esperimenti e tentativi. In particolar modo è stato importante capire che tipo di informazione potesse essere gestita interamente dalla rete neurale e quale invece dovesse essere rimossa o modificata. Lo scopo del nostro *pre-processing* è quello di fornire un'immagine contenente solo l'informazione necessaria e sufficiente per poter discriminare un pezzo conforme da uno scarto.

Nonostante in questa sezione vengano mostrate solo immagini di carcasse conformi, il processo iterativo appena descritto è stato eseguito confrontando costantemente immagini conformi e scarto. Immagini scarto verranno mostrate nella prossima sezione, a loro interamente dedicata.

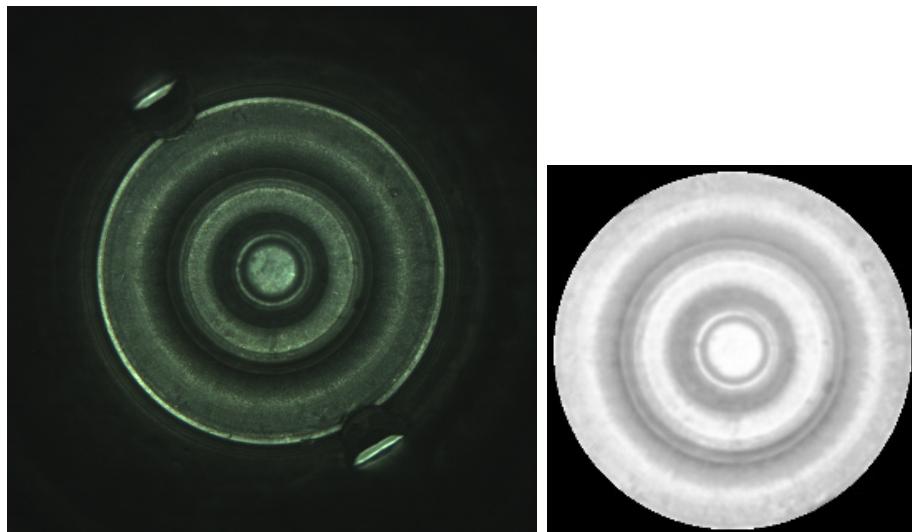


Figura 3.11: Un conforme prima e dopo il *pre-processing*

1 Centramento con Hough circle transform

Sappiamo che le immagini del *dataset* non sono perfettamente centrate, per questo motivo si è deciso di utilizzare la tecnica della *Hough Circle Transform* per ottenere un'approssimazione del centro della carcassa. Avendo le coordinate di quest'ultimo si può riallineare il pezzo con il centro dell'immagine. Ottenere un algoritmo che fosse capace di centrare correttamente tutte le immagini, non è stato facile, ma può essere riassunto nei seguenti passaggi:

- creazione di una copia dell'immagine;
- conversione della copia in scala di grigi;
- equalizzazione dell'istogramma della copia;
- ottenimento di alcuni cerchi candidati tramite trasformate di Hough applicate alla copia, qui si sfrutta l'informazione circa le dimensioni tipiche della circonferenza dello scalino più grande;
- viene effettuata la media tra i centri dei cerchi ottenuti nel passaggio precedente;
- l'immagine originale viene traslata in modo che il centro stimato si trovi in mezzo all'immagine.

Nella figura 3.12 sono *illustrate* l'immagine prima e dopo il centramento ed il cerchio rilevato.

2 Equalizzazione

Per gestire la variazione di luminosità tra immagini si utilizza *histogram equalization*. In questo modo le immagini in bianco e nero delle carcasse risultano molto più simili tra di loro e si vanno a mitigare i fastidiosi effetti di sovra illuminazione.

D'ora in poi verranno manipolate immagini in scala di grigi. Questa scelta, validata dagli esperimenti, nasce principalmente considerando il fatto che le immagini sono in falsi colori. Mantenendo il *dataset* a colori manterremmo dell'informazione che non ci è di alcuna utilità, ma soprattutto richiederemmo alla nostra rete neurale uno sforzo maggiore. La rete, infatti, dovrebbe imparare a gestire non solo le forme geometriche del pezzo, ma anche le sue colorazioni.

Nella figura 3.13 sono illustrate due carcasse conformi con luminosità differenti, affiancate dalla loro trasformazione.

3 Smooth dell'immagine

Utilizzando un filtro gaussiano è stato rimosso il rumore principale, ossia quello dovuto a graffi ed effetto “sale e pepe”. È stato fondamentale definire il filtro in modo che non fosse troppo aggressivo altrimenti ci sarebbe stato il rischio di perdere anche l'informazione relativa alla colla.

In figura 3.14 si mostra l'effetto dell'applicazione del *kernel*.

4 Mascheramento

L'area che nell'immagine corrisponde alla parete verticale della carcassa, ed in particolar modo alle due balze, deve essere rimossa. Come abbiamo già sottolineato, la posizione delle balze non è correlata alla presenza della colla; mentre si suppone che la colla non si presenti soltanto sulla parete verticale, ma che colli almeno fino a raggiungere il gradino più ampio.

Nella figura 3.15 è illustrato un conforme prima e dopo il mascheramento.

5 Crop

A questo punto l'immagine contiene una grande porzione di pixel di colore nero, a causa del mascheramento. Possiamo effettuare un'operazione di *crop* sapendo che il pezzo è centrato nell'immagine e conoscendo il raggio dell'area nera. Osservando figura 3.16 ci si accorge che dopo il ritaglio l'immagine trasporta, in proporzione, molta più informazione.

6 Resizing

Come ultima cosa viene effettuato un *resizing* dell'immagine. Ogni elemento del *dataset* ha originariamente una dimensione di 896x896 *pixel*, dopo le trasformazioni effettuate abbiamo un'immagine di dimensione circa 580x580 *px*. Si ricorda che molte architetture di reti neurali accettano immagini di dimensione 224x224. È stato verificato se questa dimensione fosse accettabile anche nel caso di questo *dataset*: risulta che mantenere l'immagine ad una dimensione maggiore non comporta alcun vantaggio, mentre ridurla al di sotto dei 200 *pixel* di lato rende la classificazione difficile anche per un essere umano.

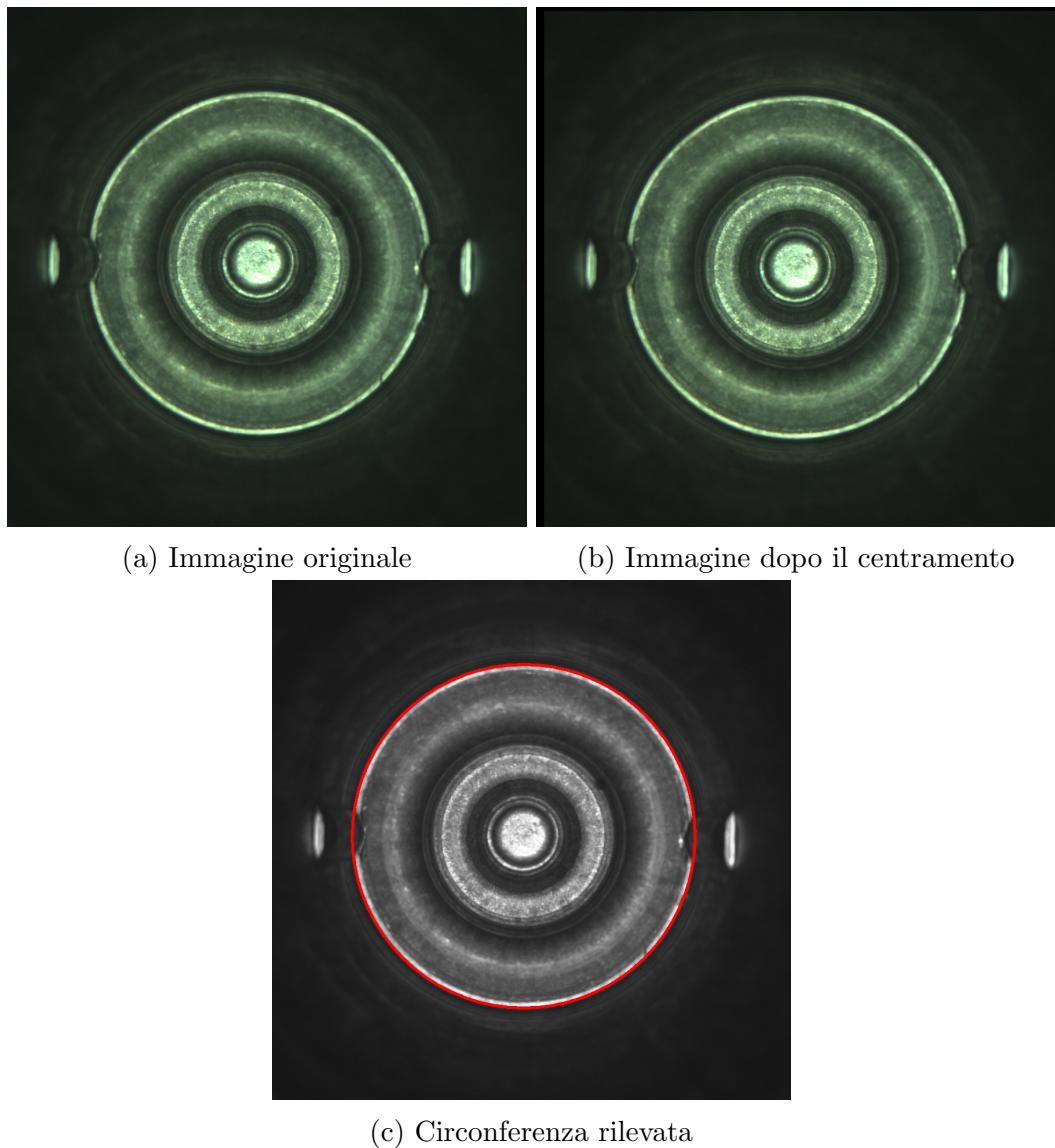
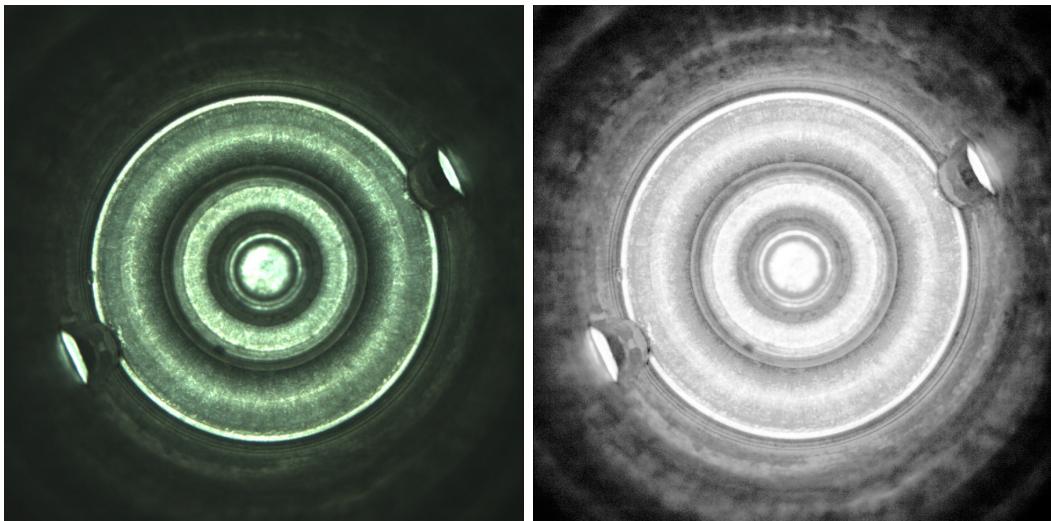
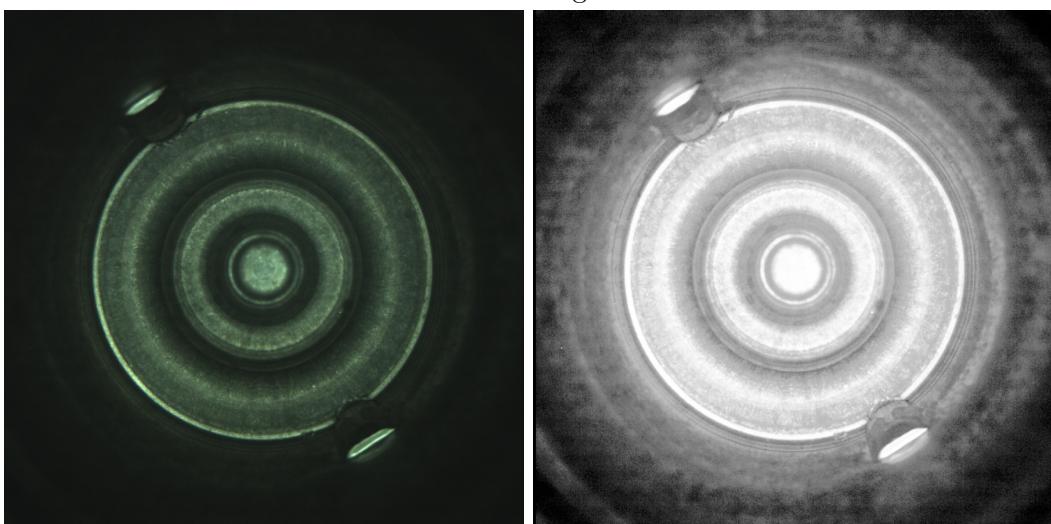


Figura 3.12: Esempio di centramento



(a) Immagine originale con alta luminosità (b) Risultato dell'equalizzazione dell'immagine con alta luminosità



(c) Immagine originale con bassa luminosità (d) Risultato dell'equalizzazione dell'immagine con bassa luminosità

Figura 3.13: Esempio di equalizzazione

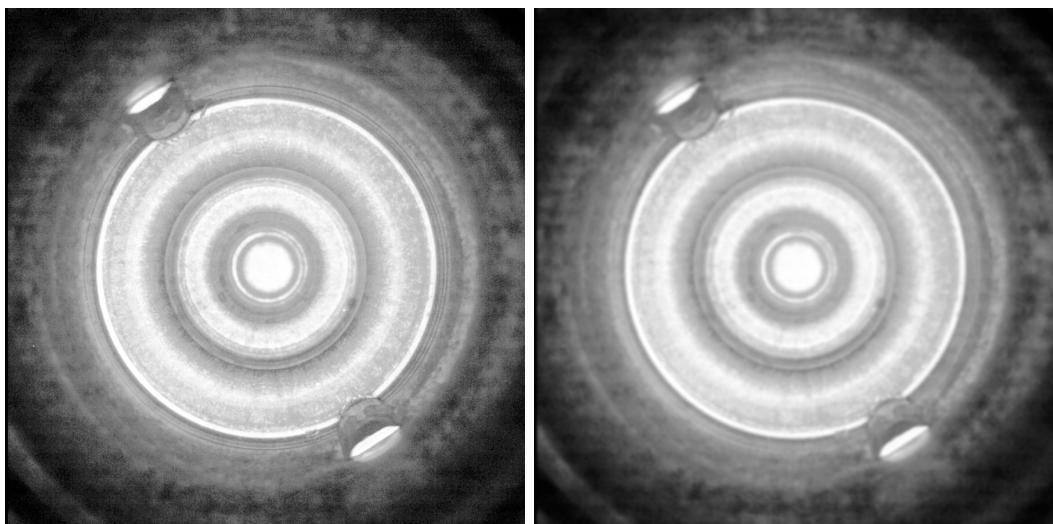


Figura 3.14: Immagine prima e dopo l'applicazione del filtro di Gauss

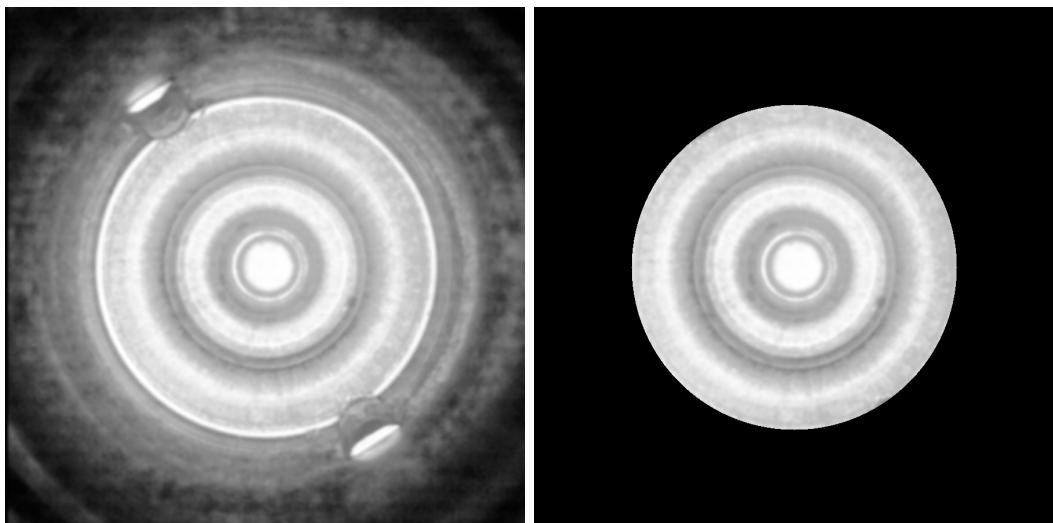


Figura 3.15: Immagine prima e dopo il mascheramento

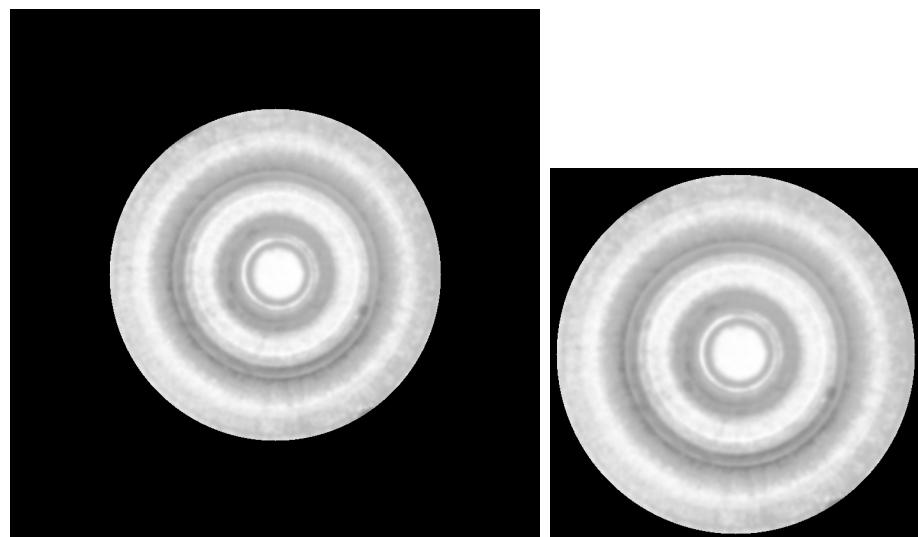


Figura 3.16: Immagine prima e dopo l'operazione di *crop*

3.4 Data Augmentation

Il *Data Augmentation* è una tecnica che permette di aumentare le dimensioni del *dataset*, generando nuovi elementi con caratteristiche verosimili. Ciò risulta molto facile quando si manipolano immagini. Queste possono essere facilmente traslate, ruotate o specchiate così da ottenere immagini differenti dall'originale, ma che contengono dell'informazione molto vicina a quella di partenza. L'idea alla base della di queste trasformazioni è quella di incentivare la rete neurale a riconoscere gli oggetti a prescindere dalla loro orientazione.

Nel nostro caso si è deciso di ruotare i conformi con angoli multipli di 45°. Così facendo le macchioline ed i graffi compariranno in modo più uniforme, essendo distribuiti su più immagini. Dopo questa operazione il numero di immagini conformi a nostra disposizione è pari a 13752, ossia 8 immagini per ciascuno dei 1719 campioni di partenza.

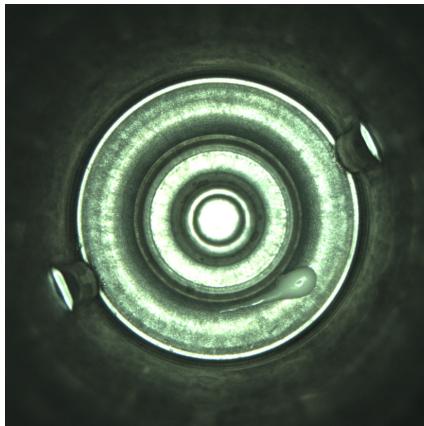
3.4.1 Generazione scarti sintetici

L'operazione di rotazione potrebbe essere applicata anche alle immagini scarto, ma si è preferito optare per una manipolazione più complessa.

Sappiamo che le immagini degli scarti sono state raccolte in un arco di tempo maggiore rispetto a quello dei conformi. Ciò può essere verificato confrontando alcune immagini delle due classi, prendiamo in esempio figura 3.1 e figura 3.2, di pagine 13 e 14 rispettivamente. Si può notare che alcuni scarti presentano degli anelli di colore più scuro, come se la superficie fosse più sporca. Questa caratteristica non compare in nessuna immagine della classe conforme e sappiamo non essere in relazione con la presenza di colla. Alcuni esemplari appartengono a lotti di carcasse molto più vecchi di quelli fotografati nei conformi, per questo motivo si è cercato di ricreare degli scarti che fossero più vicini ai pezzi recenti. Si è prestata particolare attenzione a non generare immagini irreali, perché avrebbero invalidato qualsiasi tipo di risultato. Infatti, allenare un modello su un *dataset* che non rispecchia la realtà renderebbe impossibile prevedere che il suo comportamento a livello applicativo.

In figura 3.17 è illustrato uno scarto, il ritaglio della sua colla ed il risultato dell'applicazione di questa su due conformi. Innanzitutto, tutti i tipi di colla sono stati ritagliati manualmente con un programma di manipolazione di immagini digitali. Per ogni scarto si è ottenuta un'immagine come quella riportata in figura 3.17b. Notare come sia stato importante mantenere l'informazione riguardo la posizione della colla: certe forme sono dovute al fatto che la colla è scivolata lungo i gradini del pezzo. Si è preferito evitare di posizionare queste colle in zone prive di gradini o che avrebbero fatto sì che la colla avesse altre conformazioni.

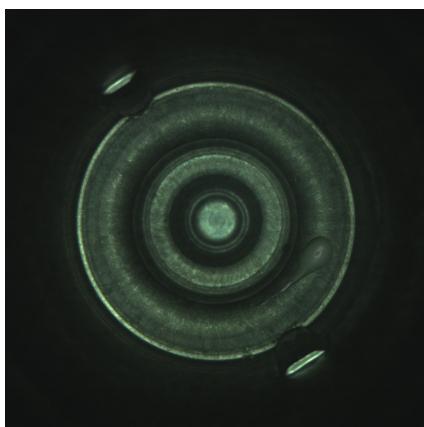
I ritagli appena creati non possono essere semplicemente applicati sui conformi, basti pensare a come risulterebbe una colla fotografata in condizioni di grandi luminosità su una foto molto scura. Per questo motivo la luminosità del ritaglio viene adattata a quello della zona che verrà coperta. La figura 3.17c e la figura 3.17d mostrano come questo accorgimento dia risultati assolutamente verosimili: gli scarti sintetici sono stati ottenuti a partire da due conformi distinti su cui è stata applicata la colla in figura 3.17b. Per scrupolo è stato anche effettuato uno *smooth* lungo il contorno della colla applicata, così da rendere più dolce la transizione tra il fondo della carcassa e il corpo della colla. Si può dire che gli scarti sintetici siano indistinguibili dagli scarti originali.



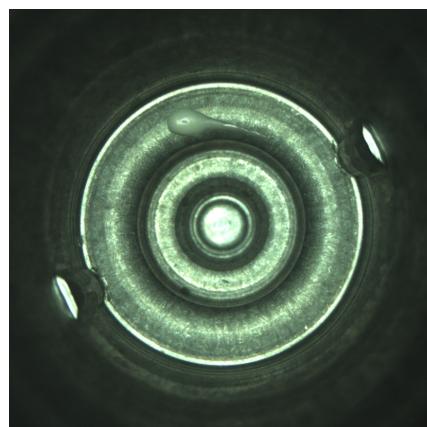
(a) scarto



(b) Colla ritagliata



(c) scarto sintetico



(d) scarto sintetico

Figura 3.17: scarto e scarti sintetici a confronto

Capitolo 4

Gli Autoencoder

In questo capitolo verranno descritti la struttura ed il funzionamento degli autoencoder. La parte centrale del capitolo è dedicata alle modalità con cui una rete neurale viene allenata.

4.1 La struttura di un autoencoder

Un *autoencoder*(AE) è un particolare tipo di rete neurale. Può essere diviso in due componenti: il primo viene chiamato *encoder*, mentre il secondo prende il nome di *decoder*. Lo scopo dell'*encoder* è codificare il dato in ingresso in una versione compressa. Sia n la dimensionalità dell'*input*, quando il dato raggiunge il collo di bottiglia dell'AE, ovvero la parte centrale della rete, ha raggiunto il livello di massima compressione, sia m la sua nuova dimensionalità con $m < n$. Lo spazio m -dimensionale in cui l'informazione è stata mappata viene chiamato spazio latente o spazio nascosto. Ora il dato può essere decompresso dal *decoder*. In questo modo verrà riportato alle sue dimensioni originali, cioè mappato nello spazio n -dimensionale di partenza.

Visto dall'esterno, il compito di un *autoencoder* è quello di ritornare un valore il più simile al dato in ingresso. Durante l'allenamento, fissato m con un valore che dipende dalla forma della rete, si vuole trovare il miglior spazio latente possibile; dove con migliore si intende quello spazio di cardinalità m che permette di mantenere tutte le informazioni caratterizzanti dell'*input*. Riformulando quanto detto: un *autoencoder* può essere visto come una funzione f rassomigliante la funzione identità, ma al cui intero c'è un vincolo tale da rendere il compito di restituzione dell'*input* non banale.

Come viene spiegato da Andrew Ng in [16], gli *autoencoder* fanno parte degli algoritmi di apprendimento non supervisionato: cioè di quella classe di algoritmi, in contrapposizione a quelli ad apprendimento supervisionato, che

non ha bisogno di dati etichettati. Anzi, vengono usati proprio per trovare nuovi *pattern* e correlazioni tra gli elementi del *dataset*, come gli algoritmi *K-Means* e *DBSCAN*, entrambi di *clustering*, permettono di raggruppare nuvole di punti con caratteristiche simili. Un AE è non supervisionato perché non conosciamo a priori la forma che verrà data allo spazio latente.

Gli *autoencoder* più semplici sono composti da due strati di neuroni densamente connessi: il primo funge da *encoder* ed ha un numero di unità pari alla dimensione dello spazio latente; il secondo ha tante unità quanto la dimensione dell'*input*, quindi funge da *decoder*. L'architettura appena descritta può essere osservata in figura 4.1, notare la tipica forma a clessidra.

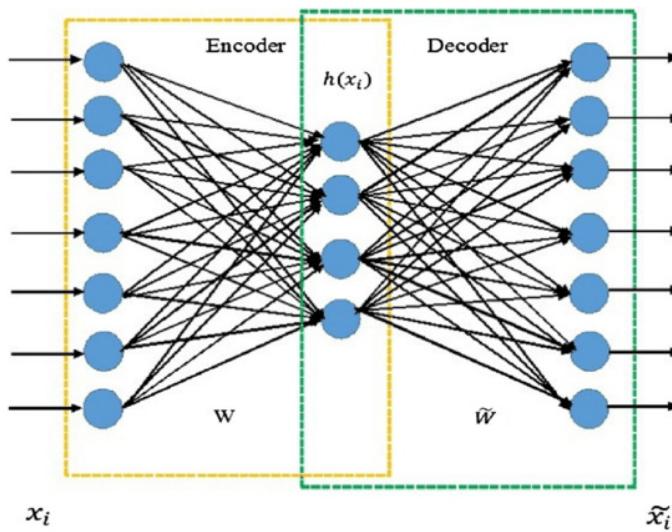


Figura 4.1: Architettura di un semplice *autoencoder*

Facendo sempre riferimento a [16], un neurone è un'unità computazionale che prende in *input* un vettore x di elementi x_i con $i = 1, 2, 3, \dots, n$ e che ritorna

$$h_{W,b}(x) = f(W^T x) = f\left(\sum_{i=1}^n W_i x_i + b\right)$$

dove $h_{W,b}(x)$ è un generatore di ipotesi non-lineare, con parametri W e b che possono adattarsi al *dataset*. I pesi del neurone vengono salvati in W , vettore con tanti elementi quanti quelli di x . Il valore b viene detto *bias* ed è una quantità che verrà sempre sommata al risultato del prodotto riga per colonna tra W e x . La funzione f è chiamata funzione di attivazione. La funzione

sigmoidea viene utilizzata spesso come funzione di attivazione, è definita come:

$$f(z) = \frac{1}{1 + \exp(-z)}$$

Notare come $f(z)$ sia una funzione da \mathbb{R} in $[0, 1]$, ciò risulta utile soprattutto se il compito del neurone è dividere gli *input* in due classi distinte, cioè 0 e 1, basterà verificare se $f(z) > 0.5$. In caso affermativo l'*input* verrà associato alla classe con etichetta 1, altrimenti a 0. Un'altra funzione di attivazione, molto utilizzata nei nodi interni delle reti, viene chiamata *ReLU* ed è definita come:

$$f(z) = \max(0, z)$$

La funzione *ReLU*, che significa *Rectified Linear Unit*, è stata pensata per simulare ciò che avviene in un vero neurone: fino a che il segnale in ingresso non supera una certa soglia il neurone rimane inattivo, ciò corrisponde a valori di z inferiori o pari a zero; appena il segnale diventa abbastanza intenso, allora il neurone inizia a trasmettere. All'aumentare della potenza del segnale in ingresso corrisponde un aumento proporzionale anche in uscita.

Quando uno strato della rete è composto da più neuroni, ciascuno di essi riceverà una copia di x e ritornerà un *output* in base ai propri W e b . Dato che il numero di connessioni cresce rapidamente, essendo pari a $n * m$ per ogni strato (con n la dimensione del vettore in ingresso ed m di quello in uscita), questi strati vengono chiamati densamente connessi o, più semplicemente, densi.

Dato che l'*input* di una funzione d'attivazione è il risultato del prodotto vettoriale tra x e i pesi del neurone, bisogna chiarire come questi pesi vengano corretti. In altre parole bisogna definire in che modo la rete venga allenata.

Facendo riferimento a quanto spiegato in [17], si consideri una rete il cui scopo è determinare se in un'immagine è raffigurato un gatto. Il caso affermativo corrisponde a 1, quello negativo a 0. Se si effettua una linearizzazione dell'immagine in un vettore x , è possibile fornire il vettore appena creato in ingresso alla rete neurale in figura 4.2. Il *dataset* sarà quindi formato da vettori x etichettati con valori $y \in \{0, 1\}$. Si fa notare che l'allenamento di una rete neurale può essere ricondotto ad un problema di minimizzazione, in cui si vuole minimizzare le predizioni sbagliate. L'algoritmo con cui si allenano le reti neurali viene chiamato *backpropagation* e può essere diviso in due momenti: nel primo i calcoli vengono effettuati da sinistra a destra, così da ottenere la predizione \hat{y} ; nel secondo, dopo aver calcolato l'errore della predizione, i pesi vengono aggiornati partendo da destra. In questo esempio lo scarto verrà calcolato usando la funzione di costo logaritmica, definita come:

$$L(\hat{y}, y) = -\ln(\hat{y}) - (1 - y)\ln(1 - \hat{y}) \quad (4.1)$$

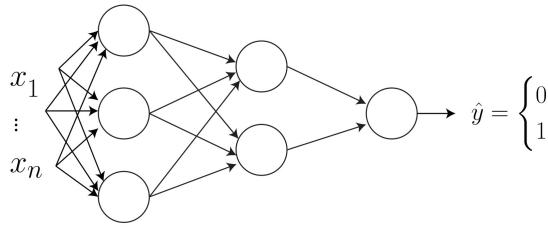


Figura 4.2: Semplice rete neurale

Si può notare come l'equazione 4.1 tenda a 0 quando i valori di \hat{y} e y sono simili, mentre tenderà a valori molto grandi se la predizione non è corretta; infatti $L(\hat{y}, y) = +\infty$. A questo punto i pesi di ogni *layer* l possono essere aggiornati con:

$$W^{[l]} = W^{[l]} - \alpha \frac{\partial L}{\partial W^{[l]}} \quad (4.2)$$

$$b^{[l]} = b^{[l]} - \alpha \frac{\partial L}{\partial b^{[l]}} \quad (4.3)$$

Il parametro α prende il nome di tasso di apprendimento, in inglese *learning rate*, mentre $\partial L / \partial W^{[l]}$ e $\partial L / \partial b^{[l]}$ sono dei gradienti. In pratica si sta applicando l'algoritmo di *gradient descent* ad ogni neurone della rete. L'intuizione con cui quest'ultimo opera è sfruttare la derivata di una funzione $f(x)$ per avvicinarsi al suo minimo al crescere delle iterazioni. Se il segno della derivata $f'(x)$ è negativo significa che il valore di f , calcolato nell'immediato intorno destro di x , sarà minore di $f(x)$. Viceversa, se la derivata ha segno positivo, f sarà più piccola per valori a sinistra di x .

Nell'equazione 4.2 si può vedere come il valore del gradiente vada a modificare i pesi, proprio con le modalità appena descritte. Ora anche lo scopo del parametro α risulta più chiaro: definisce quanto aggiornare i pesi del neurone. Trovare un adeguato tasso di apprendimento è fondamentale se si vuole alleare correttamente una rete. Si vede chiaramente come valori troppo piccoli di α fanno in modo che i pesi non vengono aggiornati affatto, mentre valori troppo grandi potrebbero portare l'algoritmo di *backpropagation* a non convergere.

Chiarito il modo in cui la rete apprende, resta da specificare il modo in cui il *dataset* venga fornito alla rete durante l'allenamento. Un allenamento è composto da svariate epoche ed ogni epoca è suddivisa in *batch*. Un *batch* è un gruppo di elementi, questi gruppi sono creati in modo da formare una partizione del *dataset*. Dopo che la rete ha elaborato i dati contenuti in un singolo *batch* viene utilizzato l'algoritmo di *backpropagation* per aggiornare i pesi. Un'epoca si conclude soltanto quando questo procedimento è stato

effettuato per tutti i gruppi. Quindi ad ogni epoca corrisponde la predizione e l'aggiornamento dei pesi per ciascun elemento del *dataset*. Il numero delle epoche e la dimensione dei *batch* devono essere determinati in modo empirico.

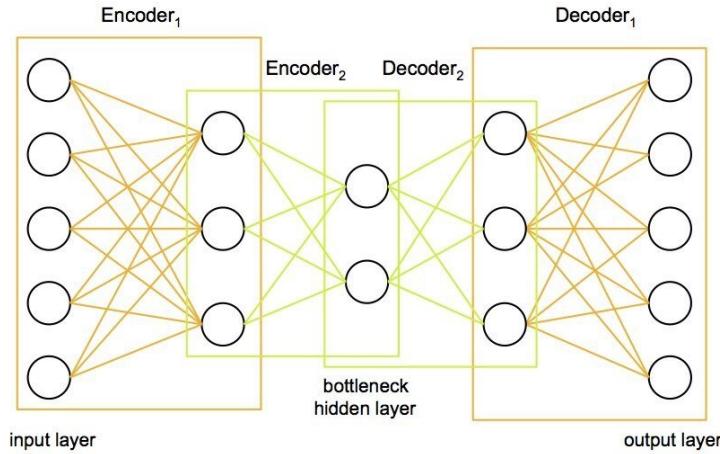


Figura 4.3: Architettura di un *stacked autoencoder*

Gli *autoencoder* come quello in figura 4.1 hanno dato risultati equiparabili, se non migliori, a quelli ottenuti con metodi di *dimensionality reduction* classici (*cfr.* [16],[19]). Essendo gli AE delle reti neurali, è possibile aggiungere vari strati densi, come si vede in figura 4.3. Questi *autoencoder* prendono il nome di AE multi-strato o *stacked autoencoder* ed hanno capacità astrattive maggiori. Allo stesso modo è anche possibile aggiungere strati convolutivi, convolutivi trasposti oppure di *down* e di *up-sampling*.

Strati Convolutivi Un *convolutional layer* è uno strato in cui i pesi hanno la forma di un filtro convolutorio, in questo modo verrà mantenuta anche dell'informazione spaziale. In pratica la rete si adatta in modo che i filtri lascino passare, o mettano in evidenza, soltanto quelle caratteristiche utili allo svolgimento del compito. Come viene mostrato molto bene in [18], questo tipo di convoluzioni è una generalizzazione parametrica delle convoluzioni descritte a pagina 17.

Innanzitutto è bene elencare i parametri che caratterizzano uno strato di questo tipo, per semplicità si suppone che le immagini ed i *kernel* utilizzati siano sempre quadrati:

- n indica il lato dell'immagine in ingresso;
- k indica il lato del filtro. Si ricorda che i valori all'interno del filtro sono i pesi che dovranno essere corretti;

- p indica quanto *zero padding* aggiungere all’immagine in ingresso. Con *zero padding* si intende l’aumentare la dimensione dell’*input* aggiungendo righe e colonne di zeri. Il risultato dell’applicazione del *padding* con $p = 1$ può essere osservato in figura 4.4b;
- s indica il passo, in inglese *stride*, con cui il filtro viene mosso sull’immagine in ingresso. In figura 4.4 sono riportati i risultati che si ottengono con passi differenti;
- con ch_{in} si specifica quanti sono i canali in ingresso, ad esempio se l’immagine in *input* è in scala di grigi si avrà $ch_{in} = 1$;
- con ch_{out} si specifica quanti sono i canali in uscita.

L’equazione 4.4 mostra la relazione che intercorre tra i vari parametri dello strato convolutivo e la dimensione m dell’immagine in uscita.

$$m = \frac{n + 2p - f}{s} + 1 \quad (4.4)$$

Si fa notare che le convoluzioni descritte a pagina 17 equivalgono ad impostare, con k dispari, i parametri $p = \lfloor k/2 \rfloor$, $s = 1$, $ch_{in} = 1$ (oppure 3 a seconda del caso) e $ch_{out} = 1$. Infatti, l’immagine in uscita che si ottiene, ad esempio con l’operatore Sobel, ha dimensioni pari a quelle dell’immagine in ingresso.

Osservando le architetture delle reti convolutive VGG [5] e ResNet [4] ci si accorge che, fatta eccezione per i primi *layer*, gli strati hanno un numero di canali in ingresso ed in uscita dell’ordine delle centinaia. Nel caso del primo *layer* dell’architettura VGG11 si ha $ch_{in} = 3$ e $ch_{out} = 64$, mentre per il secondo si ha $ch_{in} = 64$ e $ch_{out} = 128$. Una delle motivazioni è che avere più canali significa avere più *kernel*, quindi anche più possibilità di apprendimento. Infatti, prendendo in considerazione un solo strato convolutivo, il numero di *kernel* è dato da $n_k = ch_{in} * ch_{out}$, mentre il numero di parametri modificabili durante l’allenamento è $n_k * k * k$.

Strati di Down-Sampling È stato dimostrato empiricamente che inserire degli strati il cui compito è ridurre le dimensioni dell’*input*, aiuta le reti convolutive ad ottenere risultati migliori. Infatti, sia l’architettura VGG che quella ResNet sfruttano ampiamente un tipo di *layer* chiamato *Max-Pooling*. Quest’ultimo, osservabile in figura 4.5, assomiglia ad un *convolutional layer* ma non ha parametri apprendibili. Infatti, il filtro ritorna semplicemente il massimo dell’area su cui è stato posizionato. La dimensione del *kernel* e lo *stride* scelto definiscono la dimensione dell’*output*. Solitamente uno strato di questo

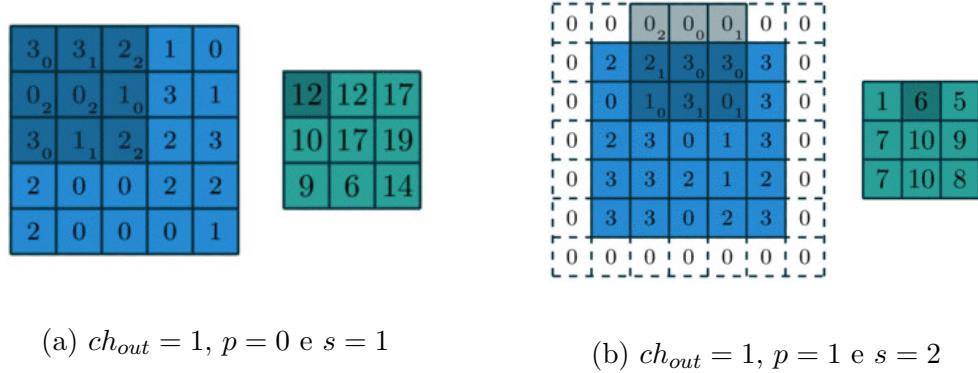


Figura 4.4: Esempio di strati convolutivi

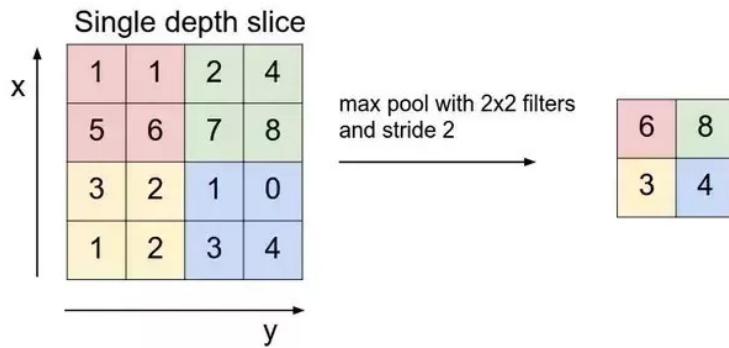


Figura 4.5: Esempio di strato max-pooling

tipo viene utilizzato per dimezzare n , cioè è ottenibile impostando $k = 2$ e $s = 2$.

Fin'ora sono stati descritti gli strati comuni a tutte le reti convolutive, ora verranno illustrati due *layer* elusivi degli *autoencoder*.

Strati di Up-Sampling Come il nome suggerisce, l'*Up-Sampling layer* effettua un'operazione opposta a quella di uno strato di *Down-Sampling*. Nello specifico aumenta le dimensioni dell'*input* riempendo gli spazi creati con medie tra gli elementi esistenti oppure ripetendoli.

Strati Convolutivi Trasposti Lo scopo di questi strati è simulare l'opposto di un'operazione convolutiva: se una convoluzione riduce la dimensione del dato in ingresso, una convoluzione trasposta deve aumentarla. Il modo più

semplice per ottenere questo risultato è effettuare una convoluzione con un *zero padding* sufficientemente grande, come si vede in figura 4.6.

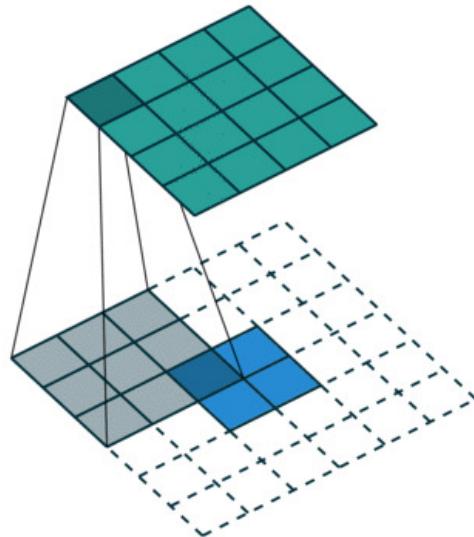


Figura 4.6: Esempio di strato convolutivo trasposto

spostare qua architettura

Capitolo 5

Risultati Ottenuti

In questo capitolo verranno illustrati gli obiettivi che si vogliono raggiungere con gli *autoencoder*, le metriche con cui sono stati valutati. Ed infine il *post-processing* applicato alle immagini ricostruite.

5.1 Obiettivo

In figura 5.1 sono riportate tre immagini per chiarire in che modo gli *autoencoder* sono stati sfruttati per classificare Conformi e Scarti. La figura 5.1a illustra uno Scarto. L’immagine riportata in figura 5.1b è quella che si vorrebbe ottenere dall’AE a partire dallo Scarto appena illustrato. Notare come si vorrebbe che il pezzo fosse riprodotto il più fedelmente possibile, ma che l’informazione della colla venisse rimossa. In questo modo sarebbe possibile effettuare una differenza *pixel* per *pixel* tra immagine in ingresso ed immagine in uscita (detta anche ricostruita) ottenendo così un risultato simile a quello in figura 5.1c.

Nel caso migliore possibile la classificazione verrebbe effettuata verificando se nell’immagine-differenza tutti i valori sono zero. Ossia l’immagine in ingresso appartiene alla classe Conforme ed è stata ricostruita alla perfezione. Dato che ci si aspetta che l’*autoencoder* non ricostruisca la colla nell’immagine in uscita, nell’immagine-differenza ci sarà un’area di *pixel* con valori in assoluto maggiori di zero.

È impossibile che l’*autoencoder* raggiunga una precisione così alta, infatti è molto più verosimile che l’immagine ricostruita sia soltanto un’approssimazione dell’immagine in ingresso. Si ricorda che molto probabilmente l’AE rimuoverà tutte quelle caratteristiche particolari di un pezzo (graffi, macchie, . . .), perché sono rumore rispetto ad una carcassa media.

Quindi si può affermare che la classificazione è divisa in due parti: nella prima l'immagine viene elaborata dall'*autoencoder*; nella seconda l'immagine in ingresso e quella in uscita vengono confrontate, questa parte prende il nome di *post-processing*. L'effettiva classificazione viene eseguita in quest'ultima parte.

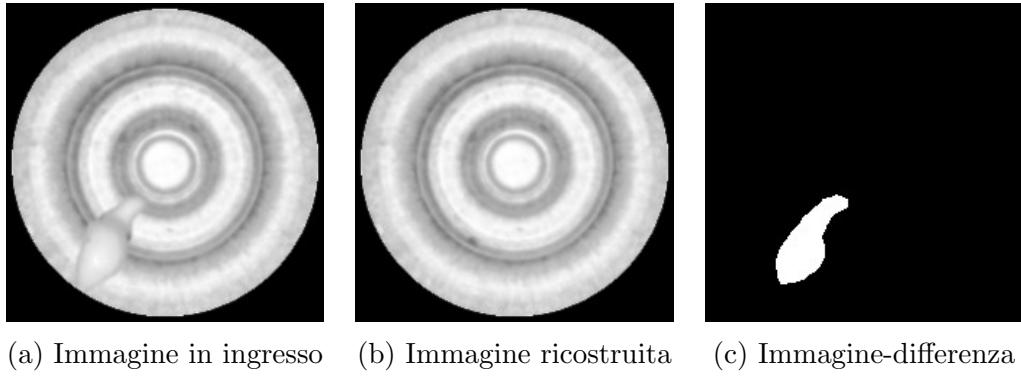


Figura 5.1: Simulazione di un risultato ottimale

5.2 Metriche di valutazione

Definire delle metriche per stabilire quali architetture preformassero meglio è stata una parte critica. Infatti, gli *autoencoder* potevano essere confrontati solo tramite due criteri: uno numerico, ossia la funzione di costo ottenuta durante l'allenamento, ed uno qualitativo, cioè osservare la qualità generale delle immagini generate. Purtroppo nessuna delle due metriche permette di ottenere delle percentuali esatte sull'accuratezza delle classificazioni. Tali dati numerici sono stati ottenuti soltanto quando le immagini-differenza sono risultate soddisfacenti e si è quindi potuto sviluppare l'algoritmo di *post-processing*.

5.3 Il modello

spostare dettagli dell'archittura il autoendoer

lasciare qua i dettagli sull'addestramento

L'architettura illustrata in figura 5.2¹ ha dato i risultati migliori. Elenchiamo le sue caratteristiche principali:

¹L'immagine è stata generata usando <https://github.com/HarisIqbal88/PlotNeuralNet>

- l'immagine in ingresso è in scala di grigi e di lato 200 *pixel*;
- ogni strato convolutivo o convolutivo trasposto ha filtri di dimensione 5x5, ha *stride* pari a 1, non ha nessun *padding* e tutti (tranne il primo e l'ultimo strato) hanno $ch_{in} = ch_{out} = 32$. Inoltre tutti i *layer* utilizzano la funzione di attivazione *ReLU*, fatta eccezione per l'ultimo in cui è presente la funzione sigmoidea. Infatti, le immagini, prima di essere passate alla rete, vengono normalizzate in $[0, 1]$, ciò facilita i calcoli effettuati all'interno dell'*autoencoder*. L'output della rete, appartenente a $[0, 1]$, verrà mappato nuovamente nell'intervallo $[0, 255]$.
- tutti i *max-pool layer* hanno sia il filtro che il passo pari a 2;
- il primo strato denso prende un vettore di $24 * 24 * 32 = 18432$ valori e lo mappa in uno spazio 2000-dimensionale. Questa corrisponde alla massima compressione dell'informazione;
- il secondo strato denso effettua l'operazione opposta, mappando il vettore dello spazio latente in uno che possa avere le dimensioni di $24 * 24 * 32$.

Complessivamente la rete deve imparare 73832000 parametri. Di questi metà si trovano nell'*encoder* e metà nel *decoder*. Questo valore potrebbe sembrare grande ma, osservando che la VGG11 [5] ha più di 130 milioni di parametri, risulta ragionevole. Si fa notare che la maggior parte dei parametri è contenuto nei due strati densi. Sappiamo che il numero di parametri di uno strato denso è dato da $n*m$, con n ed m le dimensioni dei vettori in ingresso ed in uscita. Nel nostro caso abbiamo due strati da $18432 * 2000 = 36864000$ parametri l'uno. I quattro *layer* convolutivi e convolutivi trasposti da 32 canali in ingresso ed in uscita hanno 25600 parametri ciascuno.

L'allenamento del modello in figura 5.2 è stato effettuato sui Conformi del *dataset*, processati come descritto a pagina 32 e seguenti. In questa fase non sono stati utilizzati né gli Scarti né gli Scarti Sintetici.

Nel grafico in figura 5.3 si può notare come la funzione di costo scenda in modo repentino nelle prime cinque epoche per poi stabilizzarsi. Il tasso di apprendimento è stato gestito dinamicamente: se al passare delle epoche il valore della funzione di costo non cala, allora il tasso di apprendimento viene diviso per un fattore dieci.

Ogni cinque epoche è stata generata un'immagine con alcuni Conformi in ingresso ed in uscita. È interessante notare, osservando figura 5.4, come le prime ricostruzioni siano tutte uguali. Nella figura sono riportati, in alto, i sei esemplari dati in *input* alla rete, mentre in basso sono mostrate le loro ricostruzioni. Probabilmente questo è dovuto al fatto che i pesi si distribuiscono

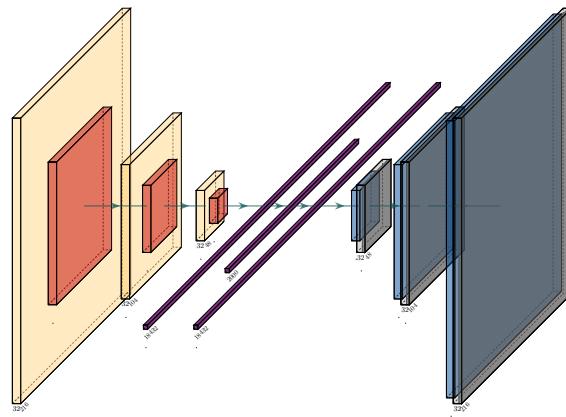


Figura 5.2: L'architettura della rete proposta

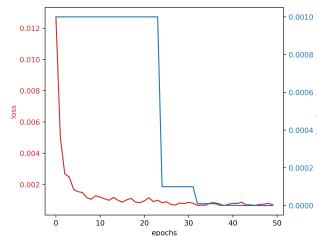


Figura 5.3: Andamento della *loss* e del *learning rate* al passare delle epoche

fin da subito in modo da catturare le proprietà comuni a tutte le immagini, ad esempio l'area nera della maschera e gli anelli concentrici. In figura 5.5 è riportato un altro gruppo di Conformi assieme alle ricostruzioni relative all'epoca numero quarantacinque. Ci si accorge subito che le immagini in uscita risultano meno sgranate e che in alcune di esse sono state generate perfino macchie ed aree più scure. Come ci si poteva aspettare la superficie delle immagini in *output* risulta molto più lisca e priva della maggior parte dei graffi.

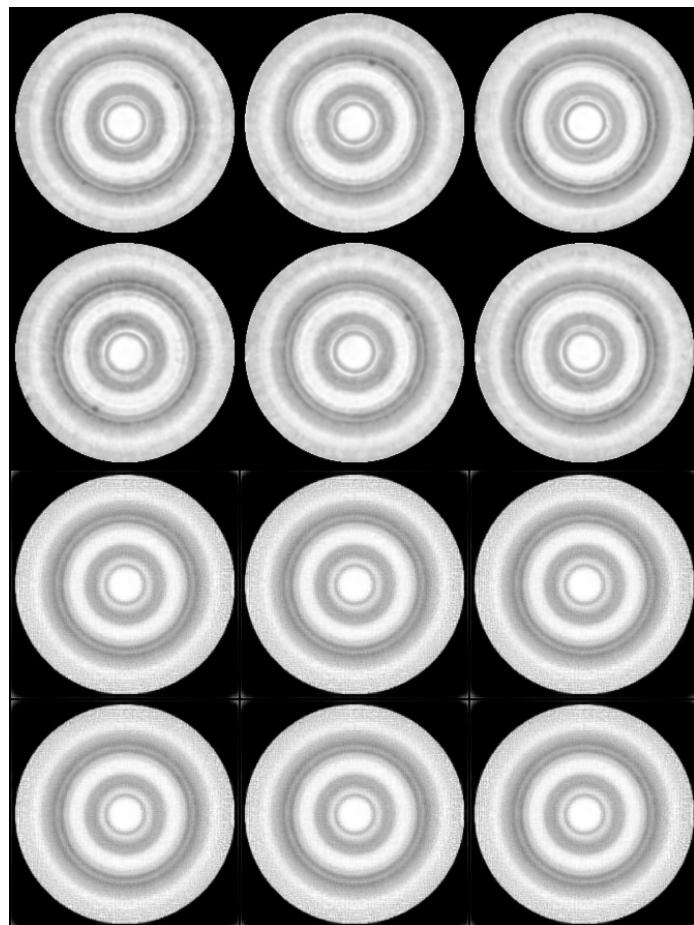


Figura 5.4: In alto 6 immagini Conformi in *input*, in basso la loro ricostruzione alla prima epoca

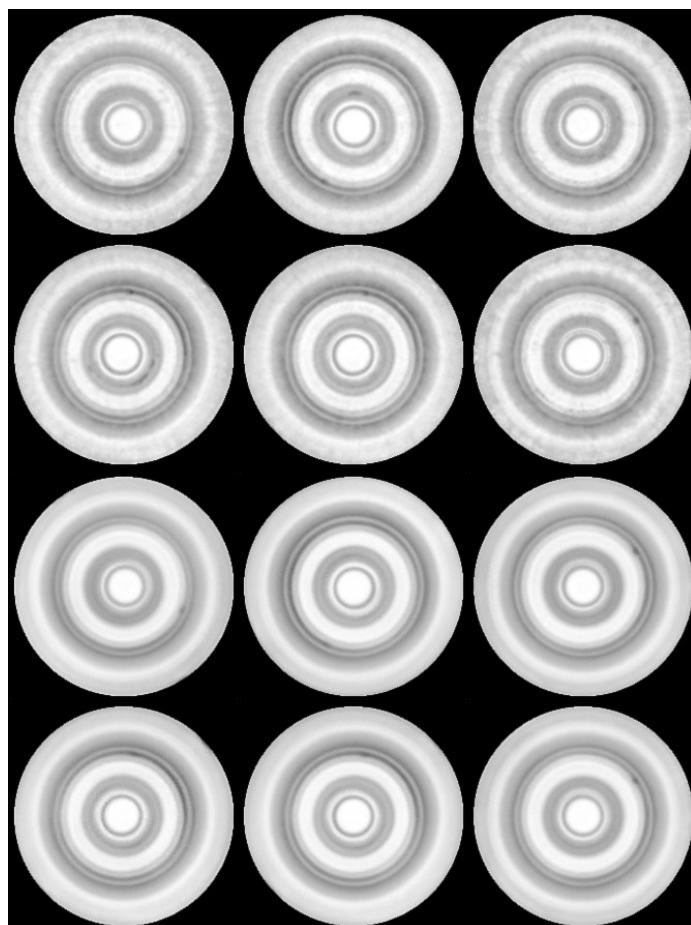


Figura 5.5: In alto 6 immagini Conformi in *input*, in basso la loro ricostruzione alla 45-esima epoca

5.4 Post-Processing

spostare Post-processing nella parte del dataset

Dopo aver osservato la precisione con cui i Conformi vengono riprodotti bisogna verificare come l’AE si comporti con le immagini della classe Scarto. In figura 5.6 si possono osservare alcuni esempi di Scarti all’ingresso e all’uscita dell’*autoencoder*. Notare come nelle prima coppia la colla sia stata rimossa molto bene e la carcassa ricostruita abbia le caratteristiche di quella originale. Purtroppo nella seconda coppia si vede come le zone più scure tipiche di alcune carcasse con colla, causino la creazione di rumore nell’immagine in *output*. Proprio questo motivo ha portato alla creazione degli Scarti Sintetici. Si fa subito presente che gli Scarti Sintetici sono stati generati a partire da un gruppo di Conformi differente dall’insieme di immagini usate durante l’allenamento.

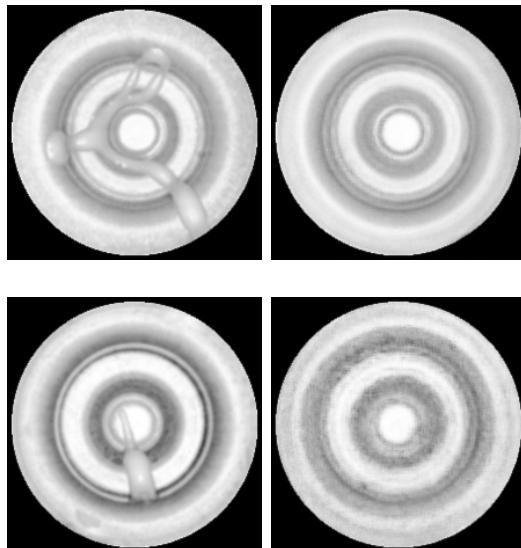


Figura 5.6: Esempi di Scarti e loro ricostruzione

L’algoritmo di *post-processing* è stato sviluppato usando immagini-differenza generate da Scarti Sintetici, perché risultano più pulite. Inoltre in questo modo ci si aspetta che siano più verosimili a quelle che si otterrebbero durante la messa in opera del sistema.

In figura 5.7 si possono vedere l’immagine di partenza (in alto a sinistra), l’immagine ricostruita dall’*autoencoder* (in alto a destra), il risultato del post-processing della differenza fra le due (in basso a destra) e la predizione finale. L’algoritmo di *post-processing*, ottenuto con metodi empirici, opera in questo modo:

- viene calcolato il valore assoluto della differenza *pixel* per *pixel* tra le immagini d'*input* e d'*output*. Il risultato di questo passaggio non è stato mostrato, perché i valori non sono molto grandi. Ciò significa che l'immagine, principalmente nera, contiene alcune aree con *pixel* di colore grigio scuro;
- viene effettuata una doppia sogliatura sull'immagine-differenza. In questo modo tutti i *pixel* con valori sufficientemente grandi vengono ricolorati di bianco, mentre i restanti sono mappati a 0;
- viene utilizzato un algoritmo di rilevamento di macchie che sfrutta la differenza di gaussiane. Si effettuano varie convoluzioni con filtro di Gauss, in questo modo le macchie più piccole verranno sfumate sempre più fino a scomparire. Effettuando differenze tra immagini più o meno sfumate si può determinare se la macchia ha dimensioni sufficientemente grandi e stimarne il centro.

Si è verificato che questo tipo di manipolazione non generasse falsi positivi: i valori delle soglie e la dimensione minima delle macchie che si rilevano sono tali da rimuovere il rumore delle immagini-differenza.

Va detto che molti Scarti vengono predetti come tali anche a causa delle fasce più scure, che non permettono una ricostruzione fedele della carcassa oppure risultano grandi a sufficienza nell'immagine-differenza.

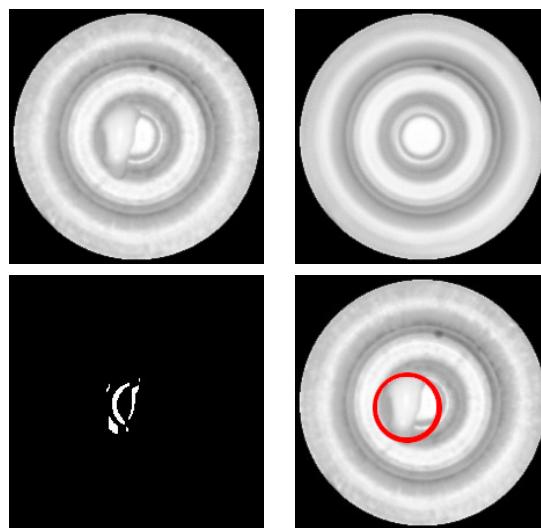


Figura 5.7: Post-processing applicato ad uno Scarto Sintetico

5.5 I Risultati

Riassumiamo ora brevemente i risultati ottenuti. Come si vede nella tabella 5.1, tutti i Conformi (che non sono stati usati durante l'allenamento) sono stati classificati correttamente. Viceversa non tutti gli Scarti e gli Scarti Sintetici sono stati classificati come tali. Per quanto riguarda i secondi va specificato che sono stati scelti a caso trenta Conformi e su ciascuno è stato applicato un ritaglio di colla. Osservando gli elementi predetti in modo scorretto e la relativa immagine-differenza ci si accorge che la colla può passare inosservata se:

- ha un colore molto vicino a quello della superficie della carcassa, quindi tende a corrispondere, nell'immagine-differenza, ad un area con valori in assoluto piccoli;
- si presenta come una striscia sottile.

Classe	Elementi	Predetti come KO	Percentuale
Conformi	314	0	0%
Scarti	30	28	93.3%
Scarti Sintetici	30	28	93.3%

Tabella 5.1: Predizioni sull'insieme di *Test*

Questi dati sono riportati anche sotto forma di matrice di confusione in tabella 5.2. Possiamo usare la tabella per ricavare le metriche di *accuracy*, *precision* e *recall* definite come:

$$\text{accuracy} = \frac{\text{TP} + \text{TN}}{\text{TP} + \text{TN} + \text{FP} + \text{FN}} = \frac{56 + 314}{56 + 314 + 4} = 0.989$$

$$\text{precision} = \frac{\text{TP}}{\text{TP} + \text{FP}} = \frac{56}{56} = 1.00$$

$$\text{recall} = \frac{\text{TP}}{\text{TP} + \text{FN}} = \frac{56}{56 + 4} = 0.933$$

Notiamo come la *accuracy*, che indica quante volte viene predetta la classe corretta, sia molto alta. Non ci sorprende che la *precision* sia così alta perché uno dei nostri obiettivi era evitare che venissero scartati Conformi. Infatti, si è mantenuto il numero di Falsi Positivi il più basso possibile. Il valore più basso

		Predizione		
		60	314	totale
Valore effettivo	p'	56 TP	4 FN	60
	n'	0 FP	314 TN	314
totale		56	318	

Tabella 5.2: Matrice di confusione per l'insieme di *Test*

è la *recall* ossia che percentuale degli Scarti è stati effettivamente riconosciuta, valore comunque soddisfacente perché corrisponde al 93.3%.

Per avere una stima sulle capacità della rete su un numero maggiore di Conformi si è deciso di effettuare delle predizioni anche sulle immagini usate durante l'allenamento. Generalmente le immagini-differenza della carcasse Conformi si presentano quasi interamente nere. Le poche macchie bianche sono di dimensioni ridotte e distanti tra loro. Il principale motivo che causa una predizione errata è il fatto che il pezzo sia stato fotografato da una distanza maggiore della media. Questo significa che le balze non vengono coperte interamente dalla maschera e quindi, dato che l'AE non le può ricostruire, possono risultare come una macchia sufficientemente grande. Nella tabella 5.3 viene illustrato che il 98.9% dei Conformi viene identificato correttamente.

Classe	Elementi	Predetti come KO	Percentuale
Conformi	1375	15	1.1%

Tabella 5.3: Predizioni sull'insieme di *Train*

Capitolo 6

Conclusioni

Questa sezione la metterei nel capitolo precedente così da rimpolparlo. Nelle conclusioni bisogna dire in max una paginina che cosa è stato fatto non così in profondità come hai fatto qui... Bisogna ridire qual è l'obiettivo della tesi, descrivere brevemente il metodo utilizzato che si compone di una fase di pre-processing, autoencode e post-processing. Descrivere brevemente quali sono stati i risultati ottenuti e poi dedicare una piccola parte agli sviluppi futuri.

6.1 Commento Dei Risultati

Innanzitutto è doveroso confrontare i risultati ottenuti con gli obiettivi posti all'inizio di questo documento. Si ricorda che uno dei principali obiettivi era quello di non scartare più del 2% di esemplari Conformi. Osservando i risultati ottenuti si può vedere che, nonostante il campione a disposizione abbia una numerosità ridotta, nessun Conforme dell'insieme di *test* è stato classificato in modo scorretto. Questo significa che c'è una quantità di Falsi Positivi pari allo 0%. Verificando le capacità del sistema anche sull'insieme d'allenamento, risulta che lo 1.1% di elementi è stato classificato come Scarto. Tra questi si trovano principalmente foto scattate ad una distanza superiore alla media.

Va fatto notare che difficilmente un sistema è perfetto: infatti di solito richiedere un'alta capacità di riconoscimento per una classe porta ad un calo di accuratezza per un'altra. Quindi avere come obiettivo un sistema che lasci passare tutti i Conformi e allo stesso tempo sia chirurgico nell'identificazione degli Scarti significa porsi di fronte ad una sfida notevole. L'insieme di algoritmi descritto in questo documento riconosce uno Scarto con un'accuratezza del 93.3%. Purtroppo il numero limitato di esemplari per questa classe non ci

permette di tracciare delle percentuali che possano definirsi accurate, infatti quel 6.7% è rappresentato da appena quattro carcasse. Si suppone però che gli esemplari Scarto a nostra disposizione catturino sufficientemente bene la forma più frequente di colla, concludiamo che questo risultato sia soddisfacente.

Inoltre si ritiene necessaria un'ultima considerazione su adattabilità e flessibilità della soluzione proposta. Entrambi gli algoritmi di *pre* e *post-processing* hanno flessibilità ridotta perché sono stati creati ed impostati manualmente. È possibile che, al variare della calibratura del processo di raccolta immagini, si ottengano risultati inaspettati. Basti pensare che le dimensioni della maschera applicata in fase di *pre-processing* non sono adattative, ciò significa che, ad esempio, al variare della distanza della fotocamera dal fondo della carcassa, non si possa esser certi né che tutti i gradini siano visibili, né che le balze vengano nascoste completamente. Infatti, proprio quest'ultimo è il principale motivo di classificazione errata di alcuni Conformi. Va però fatto notare che sono algoritmi facilmente modificabili e che la loro calibratura potrebbe essere effettuata durante la calibratura del macchinario, dato che il risultato dipende da pochi parametri.

Per quanto riguarda l'*autoencoder*, il suo allenamento risulta rapido e richiede un numero esiguo di risorse. In particolar modo non è richiesto che il *dataset* sia etichettato a mano, pratica dispendiosa e prona ad errori. Inoltre, anche qualora l'insieme di immagini contenesse elementi della classe errata, ciò non sarebbe un problema, perché l'*autoencoder* si adatta agli elementi più frequenti. Di conseguenza queste caratteristiche permettono di creare nuovi *dataset* in un tempo molto ridotto rispetto ad altre tecniche.

In conclusione questa soluzione dimostra d'essere valida, permettendo di identificare Conformi e Scarti con una precisione soddisfacente. Purtroppo le limitazioni dei dati a nostra disposizione non ci permettono di affermare in maniera definitiva che la soluzione, così com'è stata presentata in questo documento, sia pronta per essere utilizzata in applicazioni reali. Bisogna però dire che questo modello, con le dovute modifiche agli algoritmi di *pre* e *post-processing*, può essere facilmente adattato a problemi di natura simile a quello descritto, grazie alla flessibilità dell'*autoencoder*.

Bibliografia

- [1] Y. LeCun, C. Cortes and C.J.C. Burges, *The MNIST Database of handwritten digits*. URL: <http://yann.lecun.com/exdb/mnist/>.
- [2] *The CIFAR-10 dataset*. URL: <https://www.cs.toronto.edu/~kriz/cifar.html>.
- [3] *ImageNet*. URL: <http://www.image-net.org/>.
- [4] K. He, X. Zhang, S. Ren and J. Sun, *Deep Residual Learning for Image Recognition*. (2015).
- [5] K. Simonyan and A. Zisserman , *Very Deep Convolutional Networks for Large-Scale Image Recognition*. (2015). URL: <https://neurohive.io/en/popular-networks/vgg16/>.
- [6] *Kernel (image preprocessing)*. URL: [https://en.wikipedia.org/wiki/Kernel_\(image_processing\)](https://en.wikipedia.org/wiki/Kernel_(image_processing)).
- [7] DeepAI, *What is Feature Extraction*. URL: <https://deepai.org/machine-learning-glossary-and-terms/feature-extraction>.
- [8] *CCIR 601 Standard*. URL: https://en.wikipedia.org/wiki/Rec._601.
- [9] Wikipedia, *Linear Interpolation*. URL: https://en.wikipedia.org/wiki/Linear_interpolation.
- [10] Wikipedia, *Bilinear Interpolation*. URL: https://en.wikipedia.org/wiki/Bilinear_interpolation.
- [11] Wikipedia, *Histogram Equalization*. URL: https://en.wikipedia.org/wiki/Histogram_equalization.
- [12] *Histogram Equalization*. URL: https://www.math.uci.edu/icamp/courses/math77c/demos/hist_eq.pdf.

- [13] Wikipedia, *Gaussian Blur*. URL: <https://en.wikipedia.org/wiki/Gaussian.blur>.
- [14] Wikipedia, *Sobel Operator*. URL: https://en.wikipedia.org/wiki/Sobel_operator.
- [15] P.E. Hart, *How the Hough Transform Was Invented*. IEEE SIGNAL PROCESSING MAGAZINE (2015). URL: <https://pdfs.semanticscholar.org/f7e8/cbca97de34fd3695e538e164a1b40d27b04e.pdf>.
- [16] A. Ng, *Sparse Autoencoder*. CS294A Lecture Notes (2011). URL: https://web.stanford.edu/class/cs294a/sparseAutoencoder_2011new.pdf.
- [17] A. Ng and K. Katanforoosh, *Deep Learning* CS229 Lecture Notes URL: http://cs229.stanford.edu/notes/cs229-notes-deep_learning.pdf.
- [18] Theano, *Convolution Arithmetic Tutorial*. URL: http://deeplearning.net/software/theano_versions/dev/tutorial/conv_arithmetic.html#no-zero-padding-unit-strides-transposed.
- [19] D. Oehm, *PCA vs Autoencoders for Dimensionality Reduction*. R-bloggers (2018). URL: <https://www.r-bloggers.com/pca-vs-autoencoders-for-dimensionality-reduction/>.
- [20] O. Ronnenberger, P. Fischer and T. Brox, *U-Net: Convolutional Networks for Biomedical Image Segmentation*. (2015). URL: <https://arxiv.org/pdf/1505.04597.pdf>,