



UNIVERSITÀ  
DEGLI STUDI  
FIRENZE

Scuola di Scienze Matematiche, Fisiche e Naturali  
Corso di Laurea in Informatica

Tesi di Laurea

INDIVIDUAZIONE DI IMMAGINI ALTERATE  
DA UNA TELECAMERA RGB MAL  
FUNZIONANTE ATTRAVERSO UN AGENTE  
ADDESTRATO

DETECTION OF FAILED IMAGES FROM AN  
RGB CAMERA THROUGH TRAINED  
AGENTS

PIETRO BERNABEI

Relatore: *Andrea Ceccarelli*

AA 19/20

Pietro Bernabei: *Individuazione di immagini alterate da una telecamera RGB mal funzionante attraverso un agente addestrato*, Corso di Laurea in Informatica, © AA 19/20

---

## INDICE

---

1	Introduzione	5
1.1	Motivazioni	5
1.2	Obiettivo	5
1.3	Organizzazione del lavoro	6
2	Fondamenti teorici	7
2.1	Sistemi critici	7
2.1.1	Dependability	7
2.1.2	Le Minacce: guasti, errori e fallimenti	8
2.1.3	Gli attributi della dependability	10
2.1.4	I mezzi per ottenere la dependability	11
2.2	Artificial Intelligence	12
2.2.1	Agente razionale, misura di prestazione, ambiente	
		13
2.2.2	Agenti che apprendono	14
2.2.3	Neural Network	15
2.3	Autonomous Driving	16
2.3.1	Livelli di automazione della guida	17
2.3.2	Visione artificiale	18
2.3.3	Reti neurali applicate all'autonomus driving	20
2.3.4	Classificatori	21
2.3.5	Object Recognition	22
2.4	Autonomus driver simulator	22
2.4.1	CARLA	23
3	Costruzione del dataset	25
3.1	Definizione delle classi	25
3.2	Acquisizione	26
3.3	Sporcatura	27
3.3.1	Classi di guasto	27
3.4	Suddivisione del dataset	36
4	Costruzione del detector	41
4.1	Convolutional Neural Network	41
4.1.1	L'architettura	42
4.1.2	Convolutional layer	42
4.1.3	Funzione di attivazione Relu	43
4.1.4	Pooling layer	43

2 Indice

4.1.5	Dense layer	44
4.1.6	Struttura della rete sviluppata	44
5	Esecuzione e risultati	47
5.1	Addestramento del rete convulzionale	47
5.2	Testing della rete	48
6	Conclusioni e lavori futuri	53
7	A Manuale utente	59
7.1	Acquisizione	59
7.2	Sporcatura e suddivisione del dataset	59
7.3	Training e testing: Tensorflow	60

---

## ELENCO DELLE FIGURE

---

Figura 1	Catena di guasto errore fallimento (figura ripresa da [3])	9
Figura 2	Esempio di agente razionale	14
Figura 3	classificazione SAE	18
Figura 4	Schema di posizionamento dei sensori su veicolo a guida autonoma	19
Figura 5	Adversial Examples	20
Figura 6	Rappresentazione per confronto tra Golden Run e Blur	28
Figura 7	Rappresentazione per confronto tra Golden Run e Black	28
Figura 8	Rappresentazione per confronto tra Golden Run e Brightness	29
Figura 9	Rappresentazione per confronto tra Golden Run e Greyscale	30
Figura 10	Rappresentazione per confronto tra Golden Run e Broken Lens	30
Figura 11	Rappresentazione per confronto tra Golden Run e Dirty Lens	31
Figura 12	Rappresentazione per confronto tra Golden Run e Rain	31
Figura 13	Rappresentazione per confronto tra Golden Run e Condensation	32
Figura 14	Rappresentazione per confronto tra Golden Run e Icelens	33
Figura 15	Rappresentazione per confronto tra Golden Run, 50 death pixel e 200 death pixel	34
Figura 16	Rappresentazione per confronto tra Golden Run e Banding	35
Figura 17	Rappresentazione per confronto tra Golden Run e No noise reduction	35
Figura 18	Rappresentazione per confronto tra Golden Run e No sharpness	36

4 Elenco delle figure

Figura 19	Rappresentazione per confronto tra Golden Run e No Demosaicing	36
Figura 20	Differenza tra Golden Run e No Chromatic Aberration	37
Figura 21	Esempio di image set per l'agente All	39
Figura 22	Rappresentazione delle strutture dei dataset	40
Figura 23	Esempio del funzionamento di un Convolutional Layer.	43
Figura 24	Esempio di pooling con stride e dimensione del filtro pari a due.	44
Figura 25	Risultati fase di Testing:MCC	51
Figura 26	Tabella dei success rate (ripresa da [14])	54

# 1

---

## INTRODUZIONE

---

### 1.1 MOTIVAZIONI

Nel giro di un paio di secoli il mondo ha cambiato le sue abitudini repentinamente; i popoli sono passati dal viaggiare in groppa a un cavallo, alla groppa di una macchina; dall'essere il conducente, al condotto. Le macchine a guida autonoma parziale e totale stanno diventando ogni giorno che avanza realtà. Questa rivoluzione sta permeando il nostro stile di vita diventando un sogno sempre più accessibile a tutti. A causa della presenza sempre più impellente di queste macchine nella vita di tutti i giorni, sono stati definiti degli standard sempre più stringenti al fine di garantire la continuità del servizio, l'assenza di possibili errori e malfunzionamenti. Come per un essere umano che soffre di miopia, guidare senza occhiali è pericoloso; anche per il sistema di guida autonoma guidare con le telecamere affette da guasti porta a incidenti. Come verrà espresso in seguito, negli attuali sistemi, le telecamere ricoprono un ruolo fondamentale nel momento decisionale della guida autonoma e un loro malfunzionamento nel processo di acquisizione si ripercuote nel sistema decisionale, causando errori non molto graditi al guidatore.

### 1.2 OBIETTIVO

Nella seguente tesi si propone un detector, ovvero un sistema software, in grado di rilevare nel flusso di immagini generate dalla telecamera RGB (in ambito automotive) la presenza di guasti nel sistema di acquisizione, come congelamento, pixel bruciati, e tanti altri; il detector poi notificherà i possibili esiti al sistema decisionale del mezzo, prevenendo lo sviluppo di fallimenti.

### 1.3 ORGANIZZAZIONE DEL LAVORO

Come verrà esposto in seguito, un veicolo a guida autonoma presenta una suite di sensori, attraverso i quali "vede" l'ambiente circostante. Basandosi su queste percezioni, il sistema decisionale guida il mezzo in sicurezza. Un particolare elemento di questa suite di sensori sono le fotocamere RGB le quali risultano essere le più impiegate e importanti fra tutti gli altri componenti; difatti un loro guasto comporta un fallimento nel sistema decisionale. Il detector è posto tra il sistema di acquisizione della fotocamera RGB e il sistema decisionale, proprio al fine di rilevare la presenza di guasti nel sensore: il suo scopo è comunicarli al sistema decisionale che compirà di conseguenza le sue scelte.

Il detector descritto nella seguente tesi impiega una particolare forma di intelligenza artificiale: la rete neurale convoluzionale.

Come verrà esposto meglio in seguito, una Convolutional Neural Network è una forma di supervised learning, particolarmente efficiente in ambienti grafici, soprattutto per la classificazione di immagini o identificazione di elementi al loro interno. Gli agenti supervisionati, e quindi anche le CNN, richiedono una fase di training, nella quale viene definita la loro conoscenza di base (knowledge base), su cui prenderanno le loro decisioni. Premesso ciò, lo sviluppo del detector si divide in:

- Definizione e generazione del dataset
- Creazione del modello della ConvNet
- Training del modello
- Testing del modello

# 2

---

## FONDAMENTI TEORICI

---

### 2.1 SISTEMI CRITICI

Ogni giorno le persone usano infrastrutture, mezzi, telecomunicazioni e servizi di qualsivoglia, affidandosi totalmente al loro funzionamento, alla loro continuità, dando per scontato che non possano subire guasti o malfunzionamenti, poiché qualora questi avvenissero, il risultato sarebbe devastante per tutto il nostro sistema di vita. Queste componenti sono definite **sistemi critici** e il loro corretto funzionamento dipende dalla nostra capacità di analizzarne gli aspetti quantitativi. Essi sono relativi sia a caratteristiche prestazionali, quali velocità di elaborazione o altre misure di efficienza, sia a caratteristiche di sicurezza, disponibilità o affidabilità. Gli aspetti quantitativi nel loro insieme devono dimostrare e convincere circa l'adeguatezza dei nostri manufatti al fine di svolgere compiti sempre più critici e delicati per i quali li utilizziamo.

#### 2.1.1 *Dependability*

La **dependability** è una delle proprietà fondamentali dei sistemi informatici insieme a funzionalità, usabilità, performance e costo. Per fornirne una prima definizione, è necessario illustrare i concetti di [3]:

- **servizio:** Il servizio fornito da un sistema è il comportamento del sistema stesso, così come viene percepito dai suoi utenti.
- **utente:** Un utente di un sistema è un altro sistema che interagisce attraverso l'interfaccia del servizio.
- **funzione di sistema:** La funzione di un sistema rappresenta che cosa ci attendiamo dal sistema; la descrizione della funzione di un sistema è fornita attraverso la sua specifica funzionale. Il servizio è detto corretto se realizza la funzione del sistema.

Quindi **dependability** è la capacità di un sistema di fornire un servizio su cui è possibile fare affidamento in modo giustificato [9]. La dependability presenta tre diversi concetti di cui è necessario tenere conto [9]:

- **Le minacce o impedimenti alla dependability.** Gli impedimenti sono le cause potenziali di comportamenti non previsti.
- **Gli attributi della dependability.** Gli attributi ci permettono di esprimere e verificare il livello di dependability richiesto od ottenuto.
- **I mezzi per ottenere la dependability.** I mezzi sono le tecniche che permettono di ottenere comportamenti corretti, nonostante il verificarsi degli impedimenti.

### 2.1.2 *Le Minacce: guasti, errori e fallimenti*

Si definisce come [9]:

- **guasto:** la causa accertata o ipotizzata di un errore, derivante da malfunzionamenti di componenti, interferenze ambientali di natura fisica, sbagli dell'operatore o da una progettazione fallace.
- **errore:** è la parte dello stato del sistema che può causare un successivo fallimento; in alternativa si definisce errore la manifestazione di un guasto all'interno di un programma o di una struttura dati.
- **fallimento:** di sistema è un evento che occorre quando un errore raggiunge l'interfaccia di servizio, alterando il servizio stesso. Quando un sistema viola la sua specifica di servizio si dice che è avvenuto un fallimento; il fallimento è quindi una transizione da un servizio corretto a un servizio non corretto. La transizione inversa, da un servizio non corretto ad uno corretto, è detta ripristino.

**Il guasto** può rimanere dormiente per un certo periodo, fino alla sua attivazione. L'attivazione di un guasto porta ad un errore, che è la parte dello stato del sistema che può causare un successivo fallimento. I guasti di un sistema possono essere classificati secondo diversi punti di vista, ad esempio fisico, logico e di interazione. Un'altra suddivisione può essere fatta in base alla natura del guasto: un guasto può essere intenzionale o accidentale, malizioso oppure non malizioso; ed ancora in base alla

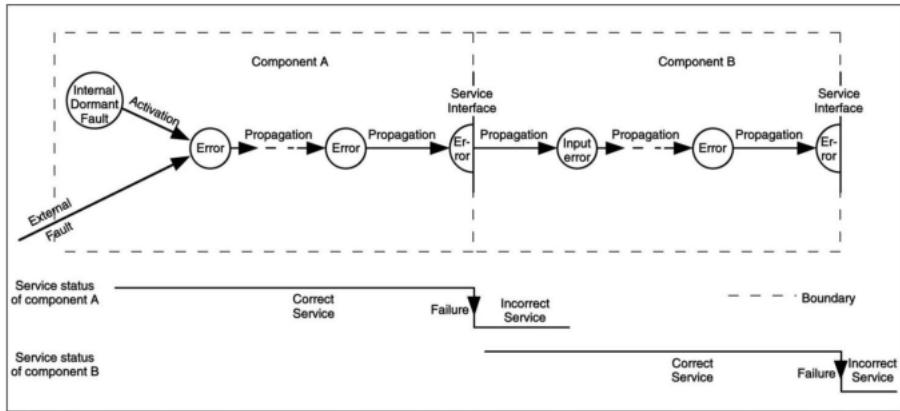


Figura 1: Catena di guasto errore fallimento (figura ripresa da [3])

persistenza dove abbiamo guasti permanenti, transienti ed intermittenti. Per una tassonomia completa si rimanda a [3].

**Il fallimento** di un componente si verifica quando il servizio fornito devia dalla sua specifica: si verifica nel momento in cui un errore del componente si manifesta alla sua interfaccia, e diventa quindi un guasto per il sistema. Il fallimento è quindi l'effetto, osservabile esternamente, di un errore nel sistema; gli errori sono in stato latente fino a che non vengono rilevati e/o non producono un fallimento.

La deviazione dal servizio corretto può assumere diverse forme, che vengono chiamate modi di fallimento e sono classificati secondo la loro gravità (severity).

**Chain Fault-Error-Failure** Un sistema è formato da un insieme di componenti che interagiscono tra loro, perciò lo stato del sistema è l'insieme degli stati dei suoi componenti. Un guasto causa inizialmente un errore nello stato di uno (o più) componenti, ma il fallimento del sistema non si verifica fino a quanto l'errore non raggiunge l'interfaccia del servizio. La propagazione di errori può permettere ad un errore di raggiungere l'interfaccia di servizio. Questo insieme di meccanismi costituisce la catena di impedimenti **guasto-errore-fallimento** (fault-error-failure).

La propagazione all'interno di un componente (propagazione interna) è causata dal processo di elaborazione: un errore viene successivamente trasformato in altri errori. La propagazione da un componente A verso un componente B che riceve un servizio da A (propagazione esterna) (Fig:1) avviene quando un errore raggiunge l'interfaccia di servizio del compo-

nente A. A questo punto, il servizio che B riceve da A diventa non corretto e il fallimento di A appare a B come un guasto esterno, e si propaga come un errore all'interno di B (Fig:1). Per esempio, nel caso di un veicolo a guida autonoma, un guasto nella lente o nel sensore della fotocamera, genera un errore. Quando esso raggiunge l'interfaccia di servizio genera un fallimento per la fotocamera. Il sistema di visione artificiale (B), vede il fallimento della fotocamera(A) come un guasto, giacchè non eroga il servizio correttamente. Il guasto della componente di acquisizione genera un errore interno al sistema, che come viene dimostrato in [14], porta al fallimento del sistema decisionale della macchina a guida autonoma. Il detector proposto si pone tra la fotocamera e il sistema decisionale al fine di intercettare il fallimento della fotocamera, impedendo così il fallimento del sistema decisionale che avrebbe conseguenze catastrofiche sui passeggeri e su tutto ciò che li circonda.

### 2.1.3 *Gli attributi della dependability*

Il concetto di dependability è la sintesi di più attributi che forniscono misure quantitative o qualitative del sistema:

- **Affidabilità (reliability):** è la capacità del sistema di erogare un servizio corretto in modo continuo; misura la fornitura continua di un servizio corretto.
- **Manutenibilità (maintainability):** è la capacità del sistema di subire modifiche e riparazioni; misura il tempo necessario per ristabilire un servizio corretto.
- **Disponibilità (availability):** è la prontezza del sistema nell'erogare un servizio corretto; misura la fornitura di servizio corretto, rispetto all'alternanza fra servizio corretto e non corretto.
- **Confidenzialità (confidentiality):** è l'assenza di diffusione non autorizzata di informazioni; misura l'assenza di esposizione non autorizzata di informazione.
- **Integrità (integrity):** descrive l'assenza di alterazioni improprie del sistema; misura l'assenza di alterazioni improprie dello stato del sistema.
- **Sicurezza (safety)** è l'assenza di conseguenze catastrofiche sugli utenti e sull'ambiente circostante.

- **Sicurezza (security)** è vista come la contemporanea esistenza di availability solo per gli utenti autorizzati, confidentiality, e integrity, dove per “improprie” si intende “non autorizzate” [4].

Nella loro definizione, la disponibilità e la affidabilità evidenziano la capacità di evitare i fallimenti, mentre la safety e la security evidenziano la capacità di evitare specifiche classi di fallimenti come ad esempio fallimenti catastrofici e accesso non autorizzato alle informazioni. Se i modi di fallimento previsti sono specificati e limitati, si parla di sistemi **fail-controlled**; un sistema i cui fallimenti sono limitati soltanto all'interruzione del servizio sono chiamati **fail-stop** o **fail-silent**. I sistemi **fail-safe**, invece, sono quelli per cui i fallimenti possibili sono solamente fallimenti non catastrofici.

#### 2.1.4 *I mezzi per ottenere la dependability*

Lo sviluppo di sistemi dependable richiede l'utilizzo combinato di quattro tipologie di tecniche:

- **prevenzione dei guasti**, per prevenire l'occorrenza o introduzione di guasti nel sistema;
- **tolleranza ai guasti**, per erogare un servizio corretto anche in presenza di guasti;
- **rimozione dei guasti**, per ridurre il numero o la gravità dei guasti;
- **previsione dei guasti**, per stimare il numero di guasti presenti nel sistema, la loro incidenza futura, o le loro probabili conseguenze.

**PREVENZIONE DEI GUASTI:** La “Fault Prevention” viene effettuata ricorrendo a tecniche e processi di controllo di qualità sia durante la progettazione del software che durante la produzione dei componenti hardware.

**TOLLERANZA AI GUASTI:** La “Fault Tolerance” mira a preservare l'erogazione di un servizio corretto in presenza di guasti attivi. Essa viene solitamente implementata tramite rilevazione di errori (error detection) e conseguente recupero dello stato del sistema (system recovery). In particolare, la rilevazione degli errori origina un segnale di errore all'interno del sistema; esistono due classi di tecniche di rilevazione di errori:

**concurrent error detection** viene effettuata durante l'erogazione del servizio, **preemptive error detection** viene effettuata quando l'erogazione del servizio è sospesa e controlla la presenza di errori latenti e guasti dormienti. Il recupero dello stato del sistema trasforma uno stato che contiene uno o più errori attivi (ed eventualmente guasti), in uno stato che non contiene errori rilevati e guasti che possono essere nuovamente attivati. Il recovery consiste in **error handling** e **fault handling**. **Error handling** elimina gli errori dallo stato del sistema mentre **Fault handling** impedisce che i guasti che sono stati localizzati vengano nuovamente attivati.

**RIMOZIONE DEI GUASTI:** La **Fault Removal** viene effettuata sia durante la fase di sviluppo, che durante la vita operazionale del sistema. La rimozione dei guasti è uno degli obiettivi del processo di verifica e validazione (V&V). La verifica è un processo attraverso cui si determina se il sistema soddisfa alcune proprietà determinate dalle specifiche o imposte all'inizio della fase di sviluppo; se così non è si cerca di individuare il guasto che impedisce di soddisfare tali proprietà e lo si corregge. La validazione consiste invece nel controllare se il sistema soddisfa le proprie specifiche e se le specifiche descrivono adeguatamente la funzione intesa per il sistema.

La rimozione dei guasti durante la sua vita operazionale è manutenzione correttiva o preventiva. La manutenzione correttiva ha l'obiettivo di rimuovere guasti che sono stati segnalati come la causa di uno o più errori, la manutenzione preventiva cerca di scovare e rimuovere i guasti prima che causino degli errori durante la normale operazione del sistema.

**PREVISIONE DEI GUASTI:** La **Fault Forecasting** è condotta effettuando una valutazione del comportamento del sistema rispetto all'occorrenza e attivazione dei guasti. La valutazione può essere di due tipi: qualitativa, che mira ad identificare, classificare e valutare i modi di fallimento o le combinazioni di eventi che porterebbero ad un fallimento del sistema; quantitativa (o probabilistica), che mira a valutare in termini probabilistici il grado con cui alcuni attributi vengono soddisfatti dal sistema; questi attributi sono in questo caso visti come misure.

## 2.2 ARTIFICIAL INTELLIGENCE

Nel pensiero comune, quando si idealizza l'intelligenza artificiale, la si immagina un'entità in grado di pensare, capace di sostituirci. In realtà,

l'intelligenza artificiale è quella branca dell'Informatica che, dato un determinato problema, definisce agenti in grado di trovare in modo efficace le migliori soluzioni. Quindi il campo di applicazione dell'intelligenza artificiale non è unico, ma è suddiviso in sottodiscipline, dove si spazia da aree più generali come l'apprendimento e la percezione, ad altre più specializzate come il gioco degli scacchi e la dimostrazioni di teoremi matematici, per arrivare ad altre ancor più complesse e complete come la guida autonoma.

### 2.2.1 *Agente razionale, misura di prestazione, ambiente*

Nello specifico l'intelligenza artificiale si occupa di progettare **agenti razionali**, che posti in un **ambiente**, riescano a massimizzare la propria **misura di prestazione**[8].

Un **agente** è definito come un sistema che percepisce il suo ambiente attraverso dei sensori e agisce su di esso mediante attuatori. Per farsi un'idea, si può assumere che l'essere umano sia un agente che, fornito di sensori, come occhi, orecchie e altri organi, percepisce l'ambiente circostante e che, attraverso altrettanti organi, come mani, gambe e bocca (attuatori), interagisce con l'ambiente o altri agenti. La macchina a guida autonoma, in quanto agente razionale, possiede sensori quali: telecamere, sensori a infrarossi e radar, attraverso i quali "vede". Per quanto concerne invece l'agire detiene: motore, freni, e tanti altri attuatori.

Un altro aspetto che accomuna tutti gli agenti sono le **percezioni**, termine usato per indicare gli input percettivi dell'agente in un dato istante [8]. La totalità delle percezioni generate dall'agente in tutta la sua storia, è definita invece come **sequenza percettiva**, dove in generale la scelta dell'azione di un agente in un qualsiasi istante può dipendere dall'intera sequenza percettiva osservata fino a quel momento.

In termini matematici la rappresentazione del comportamento di un agente è descritto dalla **funzione agente**, che descrive la corrispondenza tra una qualsiasi sequenza percettiva e una specifica azione. Se la funzione agente è la rappresentazione matematica dell'agente, con il termine **programma agente** si indica la sua implementazione concreta in esecuzione sull'architettura dell'agente [8].

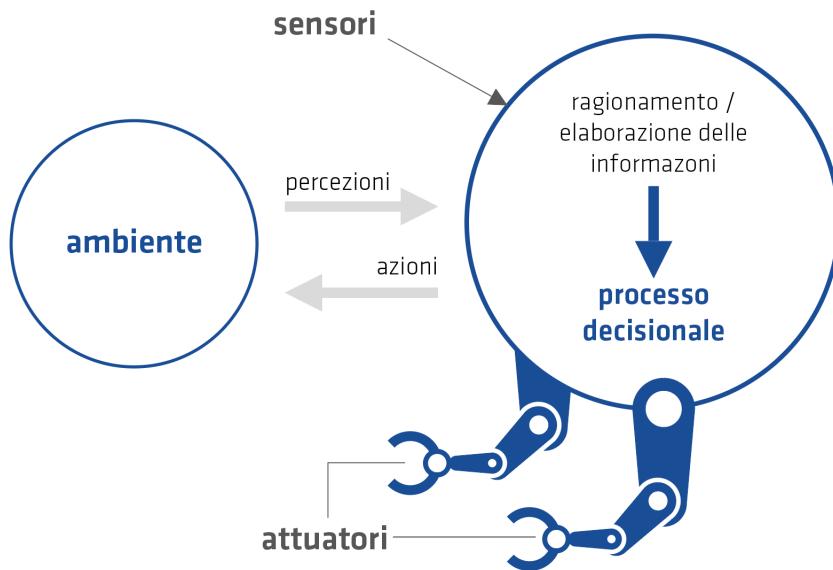


Figura 2: Esempio di agente razionale

### 2.2.2 Agenti che apprendono

Si definisce un agente che apprende, un'entità che migliora le proprie prestazioni nelle attività future dopo aver effettuato le osservazioni sul mondo [8]. Questo processo di apprendimento è decisivo per un agente in quanto [8]:

- il programmatore non può anticipare tutte le possibili situazioni che l'agente si potrà trovare davanti;
- il programmatore non può anticipare tutti i cambiamenti nel tempo;
- il programmatore spesso non sa come programmare loro la soluzione a un problema.

Asserito ciò, come esistono agenti diversi per problemi diversi, esistono apprendimenti differenti per ogni agente, che possono essere applicati alle varie componenti che lo costituiscono. La classificazione dei suddetti apprendimenti si basa su quattro fattori: la componente che viene migliorata, la rappresentazione usata per i dati e i componenti, la conoscenza a priori, e per ultimo i feedback disponibili da apprendere. Su quest'ultimo fattore sono stati sviluppati tre diversi paradigmi di apprendimento:

- **Unsupervised learning:** l'agente apprende patterns dall'input senza feedback espliciti.
- **Reinforcement learning:** l'agente apprende tramite una serie di rinforzi positivi o negativi (premi o punizioni).
- **Supervised learning:** all'agente sono fornite coppie di valori input-output, dalle quali apprende una funzione che mappi il collegamento tra i suddetti valori.

Nello specifico, il compito di un apprendimento supervisionato di: dato un training set di N esempi della forma input-output:

$$(x_1, y_1), (x_2, y_2), \dots (x_n, y_n)$$

(dove ogni valore  $y$  è generato da un funzione sconosciuta  $y = f(x)$ ), è scoprire una funzione  $h$  che approssimi la funzione  $f$ . Il learning problem si può definire in due modi a seconda dell'output  $y$ : nel caso in cui  $y$  appartenga a un insieme finito di valori il problema è definito **classificazione** (binaria o booleana, qualora si abbiano due sole scelte), invece, qualora  $y$  sia un numero, è definito come **regressione**.

### 2.2.3 Neural Network

Una particolare forma di supervised learning, come ci suggerisce il titolo del capitolo, sono l'**artificial neural network** o più brevemente **neural network**. La **rete neurale** è composta da un insieme di nodi o unità, connesse da link orientati [8]. La topologia e le proprietà dei nodi determinano le proprietà della rete.

**NEURONE:** Ciascun neurone o nodo ha un insieme di input e produce un singolo output che può essere inviato a un gruppo di altri neuroni. Questo si attiva quando una combinazione lineare dei suoi input superano la sua threshold. L'output dei neuroni finali sarà il risultato dello scopo della rete.

**Funzione di attivazione:** In generale un neurone calcola una somma pesata dei suoi input:

$$in_j = \sum_{i=0}^n w_{i,j} a_i$$

a questa è applicata la **funzione di attivazione**  $g$ , dalla quale si ottiene l'output [8]:

$$a_j = g(in_j) = g\left(\sum_{i=0}^n w_{i,j} a_i\right)$$

**COLLEGAMENTI:** I neuroni sono interconnessi tra di loro, al fine di propagare il risultato della funzione di attivazione di un nodo padre a un nodo figlio. Oltre a propagare il risultato tra neuroni, il collegamento ha anche un peso numerico  $w_{i,j}$  associato, il quale determina la forza e il segno della connessione[8].

**FEED-FORWARD NETWORK** Definito il modello matematico dei neuroni, questi si interconnettono attraverso i collegamenti a formare una rete.

Esistono due principali tipologie di network.

La prima è **feed-forward network**, la quale prevede connessioni in una sola direzione, formando un DAG (Directed Acyclic Graph). Ogni nodo riceve gli input dai nodi superiori e invia gli output a nodi inferiori; non sono previsti cicli. Questa tipologia di rete rappresenta una funzione del corrente input, senza però mantenere al suo interno nessun tipo di stato se non i pesi dei collegamenti.

Per quanto concerne la seconda tipologia si ha il **recurrent network**: esso prevede che i suoi neuroni usino il loro output come feedback per i loro stessi input. In questa maniera i livelli di attivazione della rete formano un sistema dinamico che raggiunge uno stato stabile oppure esibisce oscillazioni o ancora comportamenti caotici. Questa caratteristica permette alla rete di supportare una memoria a breve termine.

### 2.3 AUTONOMOUS DRIVING

Nell'ultimo decennio, il settore dell'automotive ha vissuto una continua evoluzione. In particolar modo, le grandi aziende automobilistiche e molti gruppi di ricerca si sono mossi verso l'automazione della guida ed il miglioramento delle infrastrutture già esistenti. I motivi di questo interesse del settore verso la guida autonoma sono vari:

- riduzione del numero di incidenti(anche fino al 90%),
- riduzione dei consumi,

- riduzione delle emissioni di CO<sub>2</sub>,
- riduzione di tutte le varie problematiche della sicurezza stradale.

### 2.3.1 *Livelli di automazione della guida*

La SAE International, un ente di standardizzazione automobilistica, nel 2014 pubblica un nuovo standard internazionale, il J3016 ("Levels of Driving Automation"). In questo sistema di classificazione, all'aumentare del livello di automazione, la responsabilità del conducente passa dalla guida all'attività di supervisione.

Lo standard suddivide l'automazione in 6 livelli: dal livello 0, in cui è assente qualsiasi tipologia di automazione, al livello 5, dove il sistema guida in maniera completamente autonoma. Per una esposizione più completa si rimanda alla figura 3.

- Livello 0: veicoli privi di qualsiasi sistema di automazione.
- Livello 1: veicoli che mettono a disposizione dell'autista almeno un sistema di assistenza (per esempio: l'assistenza alla frenata).
- Livello 2: veicoli che combinano due o più sistemi avanzati di assistenza alla guida, come l'Adaptive Cruise Control, Lane-Keeping Assist, e l'Automatic Emergency Braking. Tutti questi rientrano sotto la dicitura ADAS (Advanced Driver-Assistance Systems).
- Livello 3: veicoli definiti a guida semi-autonoma, in quanto sono in grado di gestire situazioni variegate, anche al di fuori di un contesto autostradale. Essi si prendono la responsabilità di gestire tratti del tragitto, richiedendo però alla fine degli stessi la ripresa del controllo da parte del conducente. Qualora questo non avvenisse, arrestano in sicurezza il veicolo.
- Livello 4: veicoli definiti a guida autonoma, dotati di un set di tecnologie in grado di procedere per lunghi tragitti, superando una varietà di ostacoli (come i caselli autostradali). Il loro funzionamento tuttavia è limitato da fattori quali:
  - una determinata area geografica,
  - condizioni meteo avverse,
  - una velocità massima, e così via.

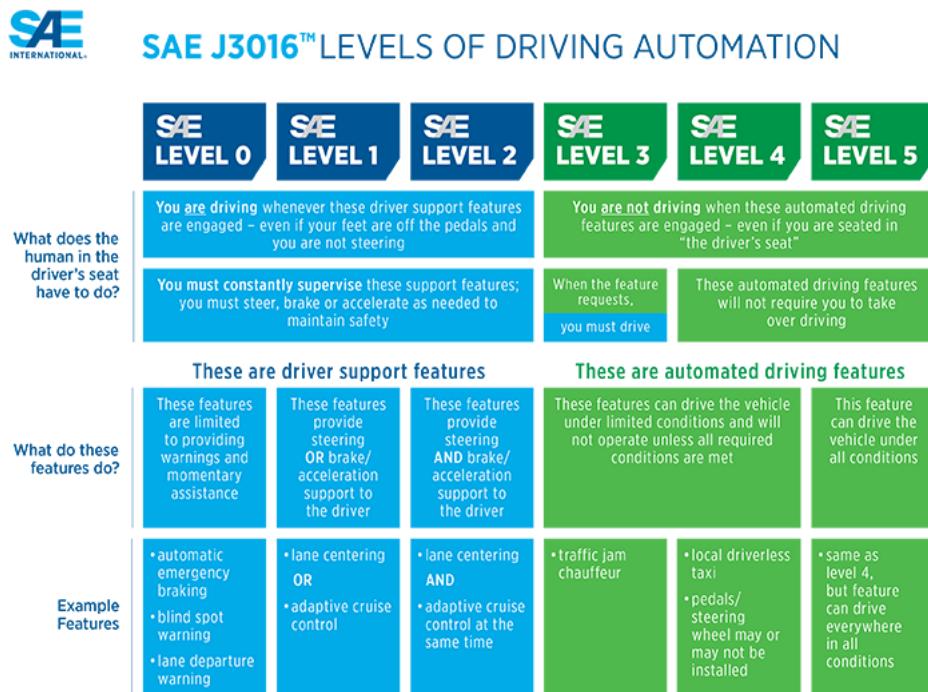


Figura 3: classificazione SAE

- Livello 5: veicoli privi di conducente e limitazioni (a dispetto del livello 4), capaci di procedere in qualsiasi condizione e luogo.

### 2.3.2 Visione artificiale

Come definito nella sezione precedente dedicata all’Intelligenza Artificiale, ogni agente acquisisce una sensazione, la elabora prendendo una decisione e la esegue tramite gli attuatori. Applicando questo concetto all’Autonomus Driving, anche i veicoli hanno una serie di sensori che permettono loro di prendere coscienza dell’ambiente circostante in cui sono immersi. Le principali tecnologie che compongono la visione artificiale avanzata dei veicoli a guida autonoma sono[13]:

- Fotocamere
- Radar
- Lidar
- GPS

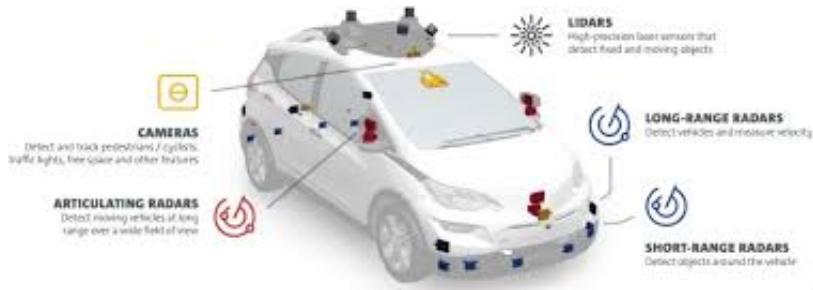


Figura 4: Schema di posizionamento dei sensori su veicolo a guida autonoma

come si può constatare nella Figura 4. Tutte insieme forniscono un modello del ambiente circostante (3D) attraverso immagini bidimensionali (2D). Lo scopo della visione artificiale è proprio quella di riprodurre la vista umana. Fra tutte le tecnologie, le fotocamere risultano essere le componenti più presenti e sfruttate. Alla luce della proponderante importanza delle fotocamere e del loro corretto funzionamento, il lavoro di tesi è incentrato sullo sviluppo di agenti addestrati nell'individuare il fallimento delle stesse (ovvero quando le immagine catturate sono fallaci).

**LIDAR - Light Detection And Ranging:** E' un sensore che attraverso l'emissione e ricezione di segnali luminosi, laser, ottiene informazioni fisiche dell'ambiente e cinematiche del veicolo. Il LIDAR è solito essere installato sul tettuccio del veicolo.

**RADAR - RAdio Detection And Ranging:** E' un sistema che utilizza onde radio trasmesse nell'ambiente, al fine di raccogliere informazioni sugli ostacoli intorno al veicolo e aumentare la consapevolezza della locazione degli altri veicoli presenti. Questo sensore controlla le distanze dalle altre auto e indica al veicolo autonomo di accelerare o rallentare a seconda del comportamento degli altri conducenti [13].

**Videocamera** Le telecamere sono necessarie nel sistema di trasporto intelligente affinché vengano riconosciuti: gli ostacoli, la loro posizione nello spazio e relativa velocità. Le immagini bidimensionali derivanti da una singola fotocamera o le mappe 3D risultanti dall'utilizzo della doppia fotocamera, individuano in modo stereoscopico lo spazio disponibile per i movimenti autonomi del veicolo. Queste immagini o mappe vengono utilizzate per estrarre informazioni quantitative dalle scene e per tracciare

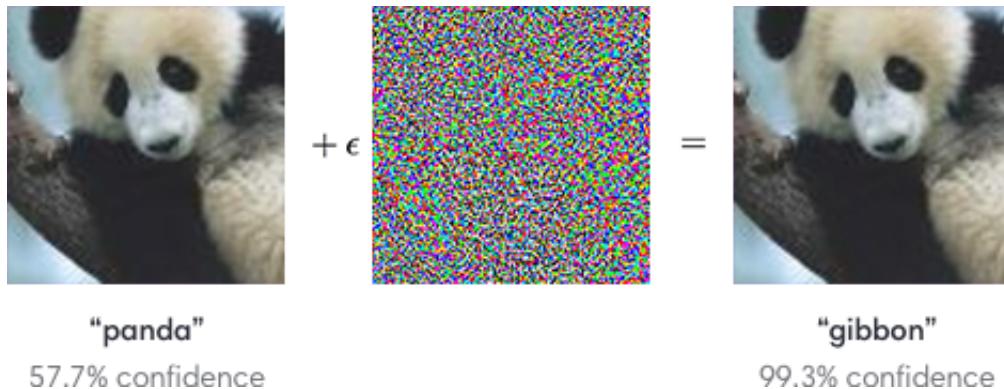


Figura 5: Adversarial Examples

gli obiettivi del veicolo. Le immagini sono segmentate in un certo numero di pixel. Ogni pixel viene elaborato e memorizzato, il che richiede un'elevata velocità di calcolo ed un elevato spazio di archiviazione [13].

**GPS - Global Positioning System** Il Sistema di Posizionamento Globale, attraverso una rete dedicata di satelliti artificiali in orbita, fornisce a un terminale mobile o ad un ricevitore GPS informazioni sulle sue coordinate geografiche e sul suo orario in quasi tutte le condizioni atmosferiche ed ovunque non vi siano ostacoli che potrebbero impedirne l'invio e la ricezione dei segnali (edifici molto alti, gallerie, ecc.), generando errori dell'ordine di metri [14].

### 2.3.3 Reti neurali applicate all'autonomus driving

Le Reti Neurali sono oggi ampiamente accettate come soluzioni dominanti per la visione artificiale, per il riconoscimento vocale e per l'elaborazione del linguaggio naturale. Nel campo della visione artificiale, le Neural Network elaborano le informazioni ottenute dai sensori e ne estraggono informazioni utili al fine di prendere decisioni riguardo il movimento, garantendo la sicurezza dei viaggiatori e degli utenti esterni al veicolo. Questo sistema presenta comunque delle criticità (come una non corretta interpretazione dell'ambiente a causa di guasti) che minano la sicurezza dei passeggeri, ma anche tutto ciò che circonda il veicolo.

Per quanto riguarda le prestazioni misurate su questa tecnologia, sono stati registrati punteggi di accuratezza spesso corrispondenti a quelli di un individuo umano, su benchmark chiave. Ben diversi, invece, sono stati i punteggi osservati nei casi peggiori [14]. Con casi peggiori si vuole

intendere quei casi definiti Adversarial Examples [6] o input perturbati anche solo leggermente, che dal punto di vista umano risultano sempre comprensibili, mentre da quello della macchina, cambiano totalmente di significato.

Un esempio di questo è rappresentato dalla Figura 5. Questa mostra una rete neurale allenata nel riconoscimento di animali, che all'applicazione di un semplice filtro cambia totalmente risposta alla classificazione. Al contrario, la percezione tramite l'occhio umano non rileva variazioni nel significato dell'immagine.

Naturalmente, se queste reti vengono utilizzate su veicoli che devono muoversi in un ambiente che presenta un'eterogeneità di soggetti e comportamenti, si parla di contesti nei quali affidabilità, sicurezza e tutti gli altri attributi che definiscono la dependability di un sistema sono al primo posto in ordine di importanza.

#### 2.3.4 *Classificatori*

Un'attività importante delle reti neurali nella guida autonoma è il processo di classificazione, ovvero l'identificazione di particolari caratteristiche intrinseche di un insieme di immagini o fotogrammi sulle quali prendere delle decisioni. Come detto precedentemente, le reti neurali presentano delle criticità qualora lavorino su immagini trasformate spazialmente [6]. Con il termine "trasformate spazialmente" si intende racchiudere tutti i possibili cambiamenti che un'immagine può subire: come ad esempio può avvenire ruotandola, tagliandola, ridimensionandola ecc. Queste sono trasformazioni naturali e sono utilizzate in vari contesti:

- verificare il livello di completezza di un classificatore,
- allenare una rete neurale,
- ricreare scenari o situazioni con cui il veicolo su cui è installata una determinata rete può avere a che fare.

Con il termine "trasformata spazialmente" quindi, non ci si deve immaginare una modifica profonda del fotogramma in questione, ma anzi, basta anche solo una piccola rotazione, di meno di un grado, per far sì che il classificatore confonda, ad esempio, un revolver con una trappola per topi [6]. La qual cosa, pensando ad un utilizzo anche diverso da quello che se ne fa su un veicolo a guida autonoma, può far sorgere molte preoccupazioni e domande.

### 2.3.5 *Object Recognition*

In generale ci si riconduce al problema dell'Object Recognition o riconoscimento artificiale degli oggetti. Questo richiede che ci sia una nozione di somiglianza visiva difficile da far apprendere ad un sistema: significherebbe catturare e insegnare ad una macchina la nozione di percezione umana [6].

Tuttavia, oggi, le DNN (Deep Neural Network) hanno raggiunto prestazioni all'avanguardia, talvolta competitive con la percezione umana. Una delle sfide principali nella guida autonoma è il cambiamento ambientale che un qualsiasi sistema autonomo può incontrare, ovvero le differenti distanze e angolazioni con cui viene percepito l'ambiente circostante dai sistemi di visione artificiale variano ogni volta che il veicolo si muove nello spazio [15]. Purtroppo, nel contesto della guida autonoma anche un piccolo errore di valutazione nella navigazione potrebbe causare incidenti altamente devastanti.

## 2.4 AUTONOMUS DRIVER SIMULATOR

Lo sviluppo della guida senza conducente è un argomento attuale e di grande interesse su cui molte società automobilistiche stanno investendo, poiché promette di essere un mercato da miliardi di dollari. E' altrettanto vero però che la guida in ambiente cittadino è una sfida molto più grande, a causa del maggiore grado di variabilità degli scenari che si possono presentare e degli oggetti che devono essere riconosciuti. E' quindi necessario ancora molto lavoro prima che sia disponibile un sistema in grado di muoversi da solo in questi ambienti senza pericolo e in modo efficiente.

Nell'ottica dello sviluppo di un sistema di guida autonoma efficiente e sicuro si rende necessario lo sviluppo di metodi e strumenti per il miglioramento dell'efficienza del prodotto e per la validazione dello stesso. Tra le tecniche utilizzate per automobili a guida autonoma ci sono metodi di apprendimento automatico che richiedono una gran quantità di immagini; inoltre sia per lo sviluppo che per la validazione si deve considerare che esistono infiniti scenari (situazioni) che si possono presentare e ai quali il sistema progettato deve essere in grado di ovviare nel modo più sicuro. Per quanto detto lo sviluppo e la validazione delle auto a guida autonoma si configurano come molto complicati e costosi visto che è necessario effettuare un gran numero di prove in diversi scenari. Per aiutare a risolvere il problema dei costi dello sviluppo con testing in

ambiente fisico e della richiesta di dati che abbiamo presentato sono stati sviluppati dei simulatori che permettono di sperimentare il software in un ambiente virtuale.

Tra gli ambienti virtuali possiamo annotare: NVIDIA DRIVE Constellation, AirSim, e CARLA Simulator. Questi simulatori riproducono scenari di guida cittadina in cui è possibile testare un'auto a guida autonoma che raccoglie dati dall'ambiente. Per le considerazioni fatte in precedenza, un simulatore si presenta come un alleato fondamentale nel processo di sviluppo e di validazione. Infatti l'alta avidità di dati da parte degli algoritmi di machine learning impiegate dal sistema di guida autonoma viene compensato dalla capacità del sistema simulatore di generare centinaia di migliaia di riproduzioni, nello specifico quelle fisicamente e eticamente più difficili da riprodurre nello sviluppo su strada.

#### 2.4.1 CARLA

Tra quelli citati in precedenza, il simulatore impiegato per il progetto è CARLA. Un simulatore grafico open source costituito da un'architettura client-server scalabile, dove il server è responsabile di tutto ciò che riguarda la simulazione dell'ambiente in tutti i suoi aspetti: rendering del sensore, calcolo della fisica, aggiornamenti sullo stato del mondo, sui suoi attori e molto altro. Mentre il lato client consiste in una somma di moduli client che controllano la logica degli attori sulla scena e impostano le condizioni del mondo. Per il progetto è stato utilizzato la versione 0.8.4 di CARLA.

**SENSORI** Come in un veicolo reale, CARLA predispone una suite di sensori preimpostasti, fotocamere RGB e pseudo sensori che forniscano ground-truth depth e ground-truth semantic segmentation. La loro posizione, numero e direzione possono essere specificate dal client che nel caso del progetto è stato impostato a una sola fotocamera; la sua posizione è stata impostata in maniera tale da simulare la prima persona all'interno del veicolo, anzichè in terza persona da impostazioni di default (esterna al veicolo).



# 3

---

## COSTRUZIONE DEL DATASET

---

Un agente che impiega il paradigma del supervised learning richiede un dataset per formare la propria conoscenza di base del mondo che lo circonda. Nel caso di un classificatore di immagini la componente principale di questa mole di dati sono appunto le immagini. A ciascuna di esse sarà applicato un filtro che simuli un particolare guasto per poi essere suddivise in tre tipologie di dataset diverse, uno per ogni fase del procedimento di training e testing della rete. In maniera più schematica la creazione del dataset si basa sulla seguente procedura:

- Definizione delle classi
- Acquisizione immagini
- Sporcatura immagini
- Suddivisione del dataset

Ciascuno step è illustrato in dettaglio nei paragrafi seguenti.

### 3.1 DEFINIZIONE DELLE CLASSI

Il problema della classificazione delle immagini è quel problema che si occupa di suddividere gli elementi in classi o categorie. Le classi o etichette (label) sono gruppi di elementi che hanno caratteristiche comuni. Per il progetto, ciò che accomuna gli elementi di una classe sono il guasto che li affligge. Per quanto concerne le tipologie di guasto su cui l'agente è stato allenato sono state acquisite dallo studio [14]. Nello studio citato sono state considerate diverse tipologie di guasti alla fotocamera per studiarne le ripercussioni all'interno di un sistema self driving.

Le classi prese in considerazione per la costruzione dei vari dataset su cui sarà allenato l'agente sono diciotto. Di queste la prima, Golden Run, rappresenta le immagini pulite, le successive sedici rappresentano

ognuna il singolo guasto e l'ultima, All, contiene contemporaneamente tutti i sedici guasti presi in considerazione.

- Golden Run
- Blur
- Black
- Brightness
- 50 death pixel
- 200 death pixel
- Nodemos
- No Noise
- Sharpness
- Brokenlen
- Icelens
- Banding
- Greyscale
- Condensation
- Dirty Lens
- No Chromatic Aberration
- Rain
- All

### 3.2 ACQUISIZIONE

Definite le classi, la fase successiva nella costruzione dei dataset è l'acquisizione delle immagini. Per questa fase entra in gioco la piattaforma CARLA, presentata nella sezione 2.4.1, con il suo sistema Client-Server per le simulazioni. Più dettagliatamente come client per le simulazioni è stato impiegato uno script del progetto github [18] sviluppato proprio per l'acquisizione di immagini durante l'esecuzione. Avviato il lato server, il lato client è stato impostato per generare 524 simulazioni diverse, ovvero 524 ambienti generati casualmente con condizioni atmosferiche e punti di generazione del veicolo diverse. La maggiore differenziazione nei dati a disposizione ha permesso di non impostare nessuna condizione all'ambiente riducendo la possibile deriva in uno stato di overfitting dell'agente durante il training.

In ciascuna delle simulazioni il veicolo si muove nelle vie della città seguendo percorsi casuali definiti secondo le politiche interne di CARLA. Durante ogni tragitto il Client è stato impostato per acquisire 300 immagini dal sensore della fotocamera frontale della vettura (nel formato .png e ciascuna dalle dimensioni di  $800 \times 600$  pixel) occupando un spazio totale di archiviazione di circa 80 GB. Si precisa che la versione 0.8.4 presenta dei bug relativi alla lettura del file di settings, quindi per l'impostazione del client è stato necessario modificare direttamente i valori interni al programma.

Si rimanda alla sezione Manuale Utente per la visione delle modifiche e i codici utilizzati per i due programmi.

### 3.3 SPORCATURA

Ottenute queste grande mole di immagini pulite, prive di qualsiasi errore, la fase denominata "sporcatura" prevede l'applicazione dei guasti alle immagini; guasti che sono simulati da filtri ripresi dal progetto github [16] i quali sono stati riadattati per l'uso del progetto. L'adattamento è stato eseguito al fine di riunire in un unico codice tutti i diversi script sviluppati nel citato progetto github permettendo così un automatizzazione del processo di modifica delle immagini e di formazione dei dataset. Questo adattamento rientra sotto il programma `carla_image_modify.py` del progetto github [17]. Quest'ultimo è stato sviluppato in python 3.7 al fine di gestire tutto il processo di creazione del dataset, in particolare la gestione della modifica delle immagini attraverso i vari algoritmi per la simulazione dei guasti e la loro copia all'interno delle varie sottocartelle del dataset. La gestione della ripartizione delle immagini da parte di `carla_image_modify.py` sarà mostrato nella sezione 3.4. Di seguito sarà mostrato brevemente, per ciascuna classe di fallimento presa in considerazione, il sistema di sporcatura delle immagini e un confronto tra una immagine pulita e una affetta da guasto.

#### 3.3.1 *Classi di guasto*

I guasti presi in considerazione per il progetto sono stati recuperati da uno studio precedente svolto sull'analisi del impatto di possibili criticità della videocamera in ambito self-driving [14]. In questo studio sono stati analizzati un vasto numero di fallimenti della fotocamera e come essi si ripercuotano sulle capacità decisionali del sistema di self driving. Di seguito verranno esposti i vari fallimenti presi in considerazione. Per una esposizione più esaustiva si rimanda a [14].

**GOLDEN RUN** La Golden Run rappresenta la classe di immagini che non simulano alcun fallimento ovvero immagini che sarebbero prodotte dal sensore funzionante. La sua presenza all'interno del dataset, come già detto, è necessaria in quanto coadiuva l'agente nel riconoscere se il sensore presenta o meno un guasto .

**BLUR** Per Blur si intende l'effetto fotografico in cui l'obiettivo della fotocamera non mette a fuoco l'ambiente circostante. Per la simulazione dell'effetto è stato utilizzata la funzione blur di CV2 impostando un filtro di dimensioni 5x5. Variando le dimensioni di quest'ultimo il grado di

sfuocatura può essere regolato. Un esempio del concetto appena spiegato è visibile alla Figura 6.

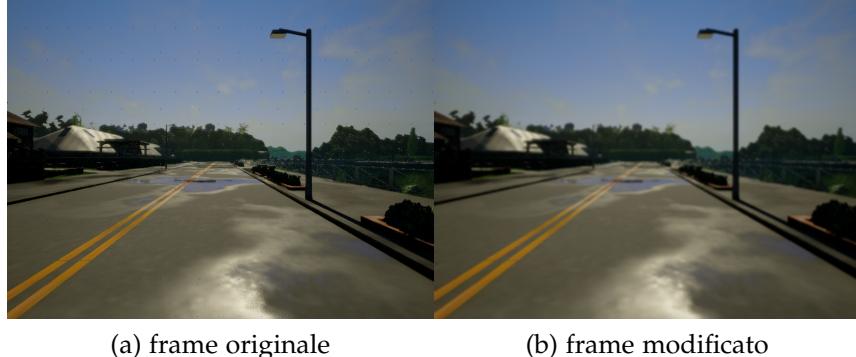


Figura 6: Rappresentazione per confronto tra Golden Run e Blur

**BLACK** Black è il guasto che rappresenta il limite massimo della rottura delle componenti dell’obiettivo [14], come allo stesso modo lo è White. Quest’ultimo non è stato però preso in considerazione ritenendo che i due guasti possano essere assimilabili per la loro somiglianza. Entrambi scaturiscono da un guasto all’otturatore o al diaframma dell’obiettivo: essi sono coloro che gestiscono il sistema di esposizione del sensore alla luce. Nel caso di un’eccessiva esposizione si avranno immagini bianche mentre, nel caso opposto scaturiranno immagini completamente nere. Il fallimento è stato riprodotto operando sulle immagini a livello di singolo pixel ponendo i tre valori che rappresentano i singoli canali RGB uguali a zero. Un esempio di ciò è visibile alla Figura 7.

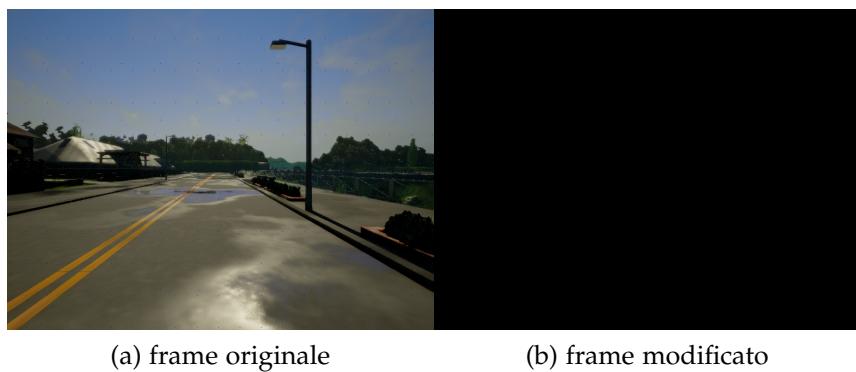


Figura 7: Rappresentazione per confronto tra Golden Run e Black

**BRIGHTNESS** Mentre il fallimento White è scaturito da un'eccessiva esposizione del sensore alla luce, Brightness si pone come una via di mezzo tra il normale comportamento e il fallimento precedente. Per ottenere questo effetto è stata applicata all'immagine la funzione `ImageEnhance.Brightness()` con un fattore di illuminazione pari a 3.5. Il fattore è stato tenuto fisso per tutta la campagna sperimentale ma può essere regolato al fine di ottenere un livello di luminosità diverso (arrivando a degenerare nel fallimento White). Un esempio del fallimento è visibile nella Figura 8.

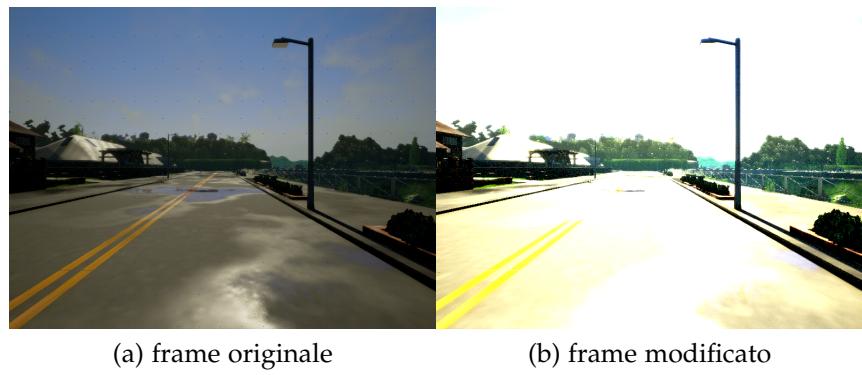


Figura 8: Rappresentazione per confronto tra Golden Run e Brightness

**GREY SCALE** Il Grey Scale è un guasto che inficia il filtro di Bayer (quest'ultimo è il dispositivo dedito all'acquisizione dei valori RGB per i vari pixel). Il guasto si ripercuote così sulle immagini "colorandole" di grigio. Per modificare i dati è stata utilizzata la funzione `cvtColor` della libreria cv2 (usando come parametro `cv2.COLOR_BGR2GRAY`). Gli effetti di questa funzione sono visibili alla Figura 9.

**BROKEN LENS** Il Broken Lens è un guasto inteso come una vera e propria rottura di una o più lenti dell'obiettivo, le quali a seconda della frattura riportata riducono la qualità e la correttezza delle informazioni ottenibili. Per la simulazione del guasto è stato usato un set di 15 immagini sviluppate appositamente per l'emulazione di una frattura delle lenti. Il set è stato fornito dal progetto github [16]. Queste emulazioni vengono apposte sull'immagine attraverso la funzione `blend` della libreria PIL. Il prodotto di questa operazione è risultato essere troppo scuro; è stato fornito quindi un aumento della luminosità attraverso la funzione



Figura 9: Rappresentazione per confronto tra Golden Run e Greyscale

Brightness di ImageEnhance (con un valore pari a 1.6). Un esempio di questo è visibile alla Figura 10.

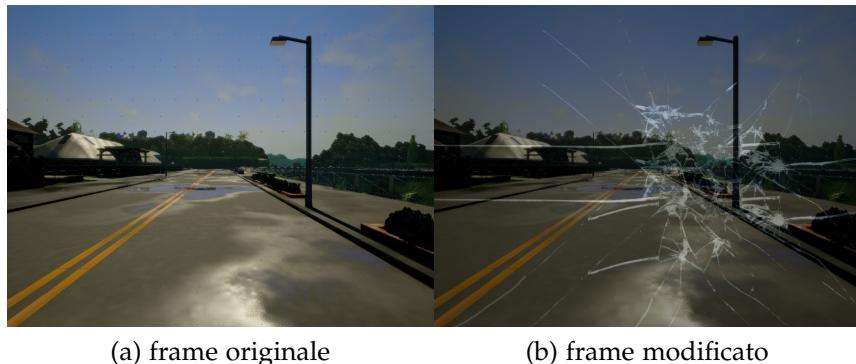


Figura 10: Rappresentazione per confronto tra Golden Run e Broken Lens

**DIRTY LENS** Come si deduce dal nome, questo guasto è dovuto alla presenza di polvere sulla lente dell'obiettivo. Polvere che può essere interna o esterna alla stessa lente e che quindi richiederà una diversa tipologia di manutenzione a seconda del caso. Ai fini del progetto non sono stati distinti i due casi poiché le immagini generate non risultavano particolarmente differenziate. La presenza di pulviscolo più o meno grande rimane impressa nell'immagine riducendo le informazioni ottenibili dal sensore. Come per Broken Lens, è stata utilizzata la medesima procedura per la modifica dell'immagine; attraverso la funzione blend della libreria PIL sono state applicate le immagini sviluppate nel progetto github [16]. I prodotti di questa operazione risultano essere troppo scuri per una emulazione realistica del guasto. Per aumentare la loro luminosità è stato

utilizzata la funzione Brightness di ImageEnhance (con un valore pari a 1.6). Un esempio del guasto è visibile alla Figura 11.

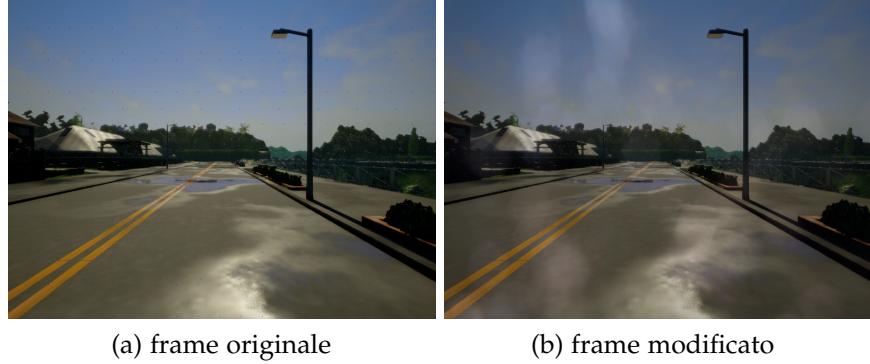


Figura 11: Rappresentazione per confronto tra Golden Run e Dirty Lens

**RAIN** Con questo termine si intende il fallimento in cui sulla lente della videocamera sono presenti diverse piccole macchie, per lo più di colore bianco, dovute al depositarsi di gocce d'acqua sulla lente esterna. Per la riproduzione del guasto è stata utilizzata la funzione paste di PIL sovrapponendo le immagini rappresentanti la pioggia sull'immagine pulita. Le immagini sovrapposte per la simulazione del guasto sulla lente sono state anch'esse riprese dal progetto github [16]. Un esempio del guasto è visibile alla Figura 12.



Figura 12: Rappresentazione per confronto tra Golden Run e Rain

**CONDENSATION** Quando la temperatura dell'aria esterna si riduce bruscamente la condensa può apparire sulle lenti dell'obiettivo della

telecamera. La condensa (o umidità) provoca il degrado delle immagini e, qualora penetrasse all'interno del dispositivo, anche il degrado delle parti elettroniche. "Condensation" fa sì che l'immagine venga acquisita nel modo corretto sebbene ogni fotogramma potrebbe riportare delle pesanti modifiche dovute agli aloni sulle lenti. La condensa, infatti, può formarsi anche all'interno delle lenti dell'obiettivo. Questo può provocare degli annebbiamenti nei fotogrammi o determinare la completa inutilità dell'immagine acquisita. Come per Rain, il processo di sporcatura delle immagini segue lo stesso algoritmo, utilizzando la funzione `paste` di PIL per la sovrapposizione del filtro associato al guasto sull'immagine pulita. Per una raffigurazione delle guasta si rimanda alla Figura 13.

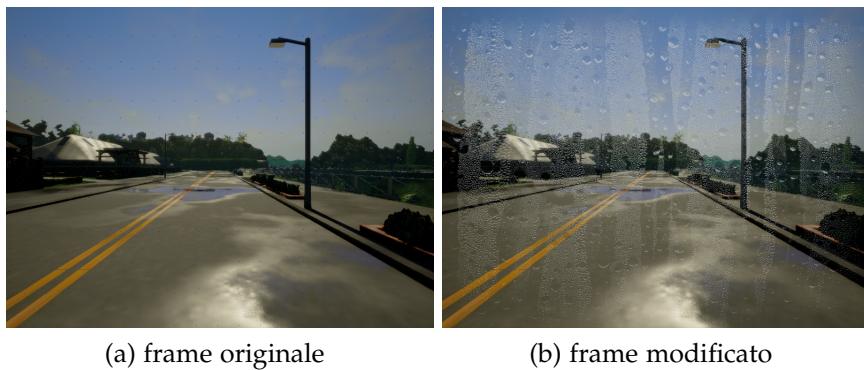


Figura 13: Rappresentazione per confronto tra Golden Run e Condensation

**ICE LENS** Ice Lens è il guasto che si presenta in casi di temperature particolarmente basse; esso può scaturire dal possibile congelamento e delle componenti interne (riducendone le funzionalità), e della lente (formando aloni gelati ed impedendo così l'acquisizione di immagini chiare). Simile agli altri due guasti collegati all'acqua, il guasto Ice è stato riprodotto attraverso la funzione `paste` della libreria PIL. La Figura 14 mostra le differenze tra la Golden Run e un'immagine affetta da Ice.

**DEAD PIXEL** Dead Pixel è un malfunzionamento che a differenza dei precedenti non è scaturito da un funzionamento errato della lente, bensì dal sensore. Le immagini sono acquisite correttamente, ma presentano pixel difettati. Uno, cinquanta, duecento, sparsi, concentrati, a strisce, a reticolo: questi sono solo alcuni dei modi in cui questo guasto si può presentare, e come si vedrà in seguito ognuno di essi incide sulla quantità di informazioni ottenibili dalle singole immagini. Alla luce di questo gran

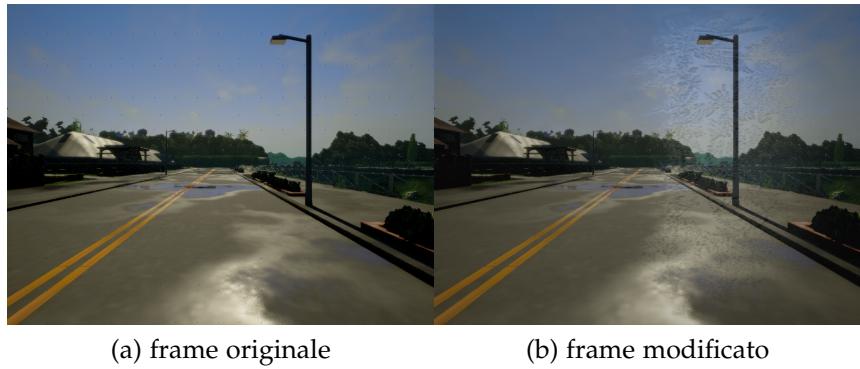


Figura 14: Rappresentazione per confronto tra Golden Run e Icelens

numero di tipologie di Dead Pixel sono state prese in considerazione due sole varianti: 50 death pixel e 200 death pixel (nelle quali i due numeri rappresentano il numero i pixel morti). In particolare nelle immagini i pixel morti sono distribuiti equamente su tutta la superficie del frame seguendo uno schema a reticolo. Per la loro simulazione si prevede di modificare il colore del singolo pixel annerendolo. Per le due rappresentazioni si rimanda alla Figura 15 consigliando di porre attenzione sull'orizzonte per apprezzare meglio le differenze.

**BANDING** Il Banding è un fallimento che si presenta nell'immagine sotto forma di righe orizzontali parallele oppure ove il colore di sfondo degrada nei toni più chiari. Il primo fenomeno è scaturito da una scarsa qualità del sensore mentre il secondo dalla mancanza di toni a disposizione per ricreare una gradazione uniforme del colore. Per ottenere l'effetto di Banding l'algoritmo sviluppato prevede l'applicazione di un filtro all'immagine pulita attraverso la funzione `blend` della libreria PIL. Un esempio dell'effetto del guasto sulle immagini è visibile nella Figura 16.

**NO NOISE REDUCTION** Con questo particolare fallimento il dispositivo cattura l'immagine, ma nelle fasi di elaborazione per la riduzione del rumore si ha un errore che determina la non corretta sanitizzazione del fotogramma (totale o parziale). Questo procedimento, qualora l'algoritmo si interrompa durante l'esecuzione, porta al blocco del dispositivo non fornendo così le informazioni al computer centrale e minando la sicurezza dei passeggeri. Per la riproduzione del fallimento è impiegata la funzione `random.normal()` della libreria numpy all'immagine, il quale genera una

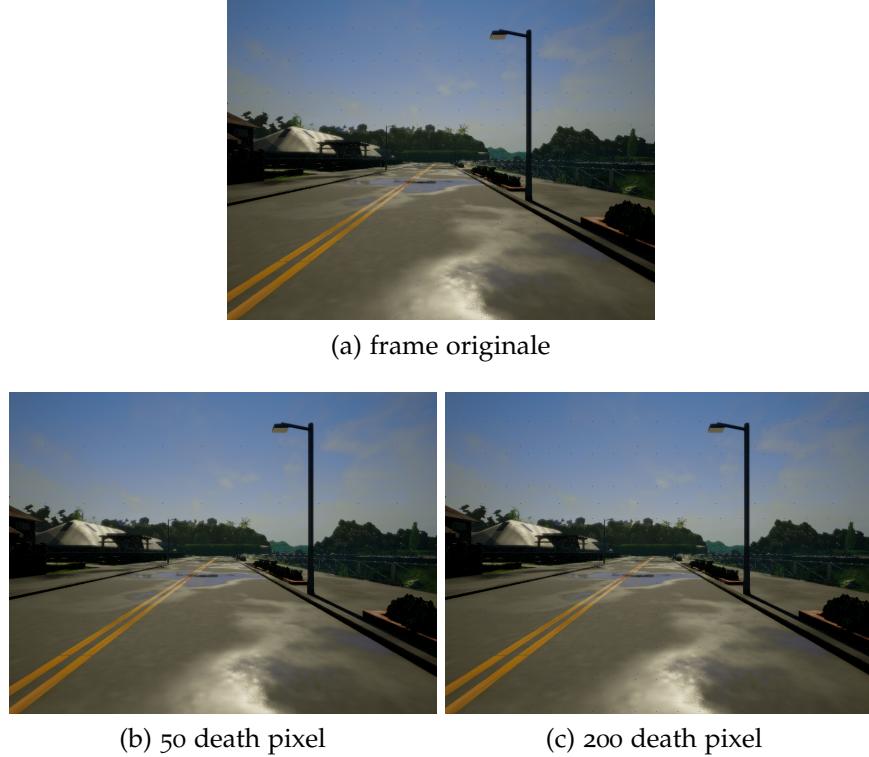


Figura 15: Rappresentazione per confronto tra Golden Run, 50 death pixel e 200 death pixel

distribuzione normale parametrizzata. Questa viene in seguito applicata all’immagine ottenendo così un risultato simile alla Figura 17.

**NO SHARPNESS** Per No Sharpness si intende quel guasto che scaturisce nelle fasi di elaborazione delle immagini catturate, più precisamente nella fase di correzione della nitidezza (in pratica la capacità di un apparecchio fotografico di identificare e definire il limite di separazione tra due aree contigue che abbiamo diversa luminosità e/o colore [14]). La nitidezza è da intendere come la chiarezza dei bordi e dei confini degli elementi che compongono un’immagine [14]. Il malfunzionamento è stato riprodotto grazie alla funzione Sharpness di ImageEnhance (con un fattore di nitidezza impostato a  $-3.5$ ). Quest’ultimo è regolabile al fine di ottenere diverse gradazioni. La Figura 18 mostra l’effetto del guasto sull’immagine.

**NO DEMOSAICING** No Demosaicing rappresenta il fallimento in cui l’immagine rimane grezza (o RAW), ovvero è impossibilità nel ricostruire

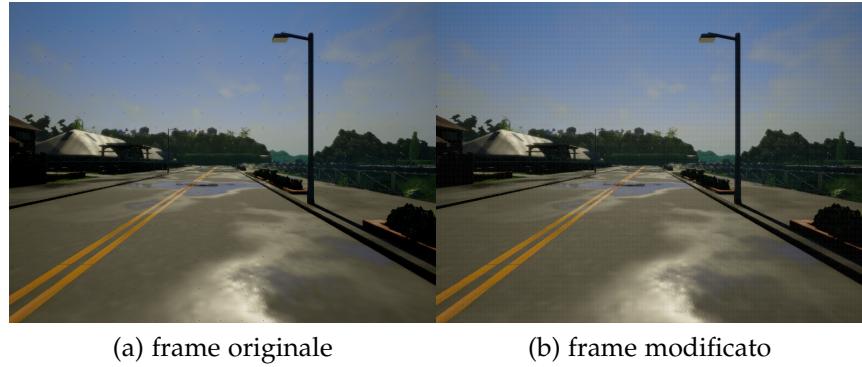


Figura 16: Rappresentazione per confronto tra Golden Run e Banding

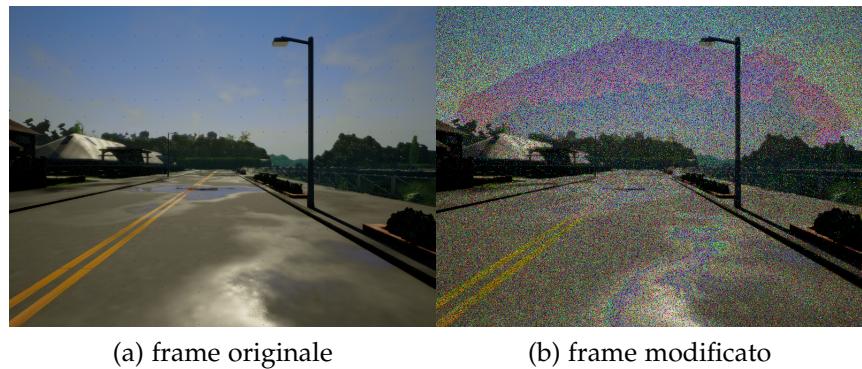


Figura 17: Rappresentazione per confronto tra Golden Run e No noise reduction

tramite tecniche di interpolazione tutti i colori dell’immagine. L’algoritmo sviluppato fornisce l’effetto descritto generando un array di dimensioni doppie rispetto a quelle dell’immagine originale, inoltre mappa al suo interno i valori RGB dell’immagine pulita in accordo al pattern BGGR. Infine converte il risultato della mappatura con la funzione cvtColor di CV2 (con parametro cv2.COLOR\_BGR2RGB) nel risultato finale. Il risultato di questo procedimento è visibile alla Figura 19.

**NO CHROMATIC ABERRATION** Con No Chromatic Aberration si vuole intendere la mancata elaborazione (totale o parziale) del fotogramma acquisito per la rimozione dell’aberrazione cromatica. Questo si traduce in immagini che presentano ai bordi dei soggetti aloni colorati (viola per lo più) e una sorta di sfocatura generale. L’effetto è sviluppato attraverso un algoritmo prodotto dallo studio di [14]. Per un esempio si rimanda alla Figura 20.

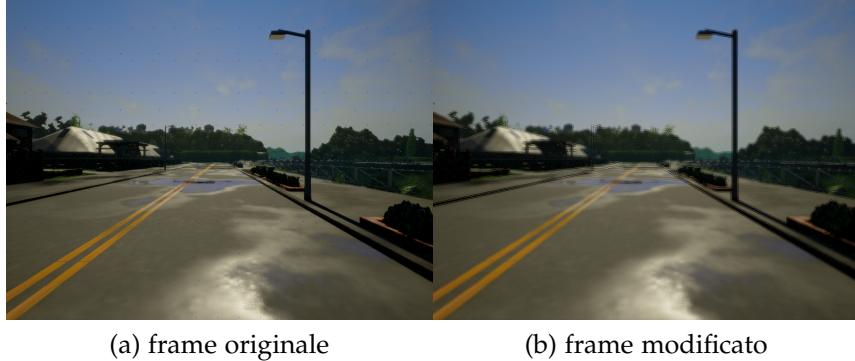


Figura 18: Rappresentazione per confronto tra Golden Run e No sharpness

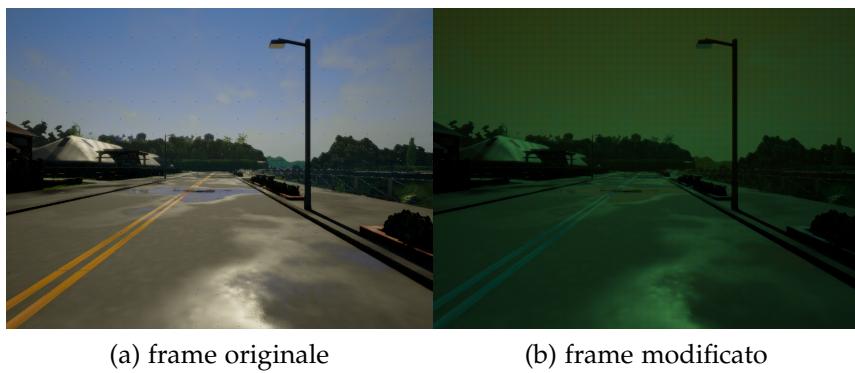


Figura 19: Rappresentazione per confronto tra Golden Run e No Demosaicing

**ALL** La seguente classe non rappresenta un singolo guasto ma contemporaneamente tutti i diversi fallimenti sopracitati. Quindi all'interno dei suoi dataset ogni fallimento è rappresentato da una certa percentuale di elementi, il più possibile equa. Per la sua realizzazione sono stati semplicemente applicate tutte le varie funzioni che abbiamo descritto, senza sovrapporle.

#### 3.4 SUDDIVISIONE DEL DATASET

Data questa esaustiva descrizione dei fallimenti sono stati costruiti 17 dataset diversi, uno per ciascun fallimento. Come mostrato dalla Figura 22a, per il training set e per il validation set sono stati predisposti rispettivamente 240000 e 60000 elementi. All'interno dei set gli elementi sono equamente suddivisi nelle due etichette che li compongono (Modified e Original). Per quanto riguarda i test set questa suddivisione non è così

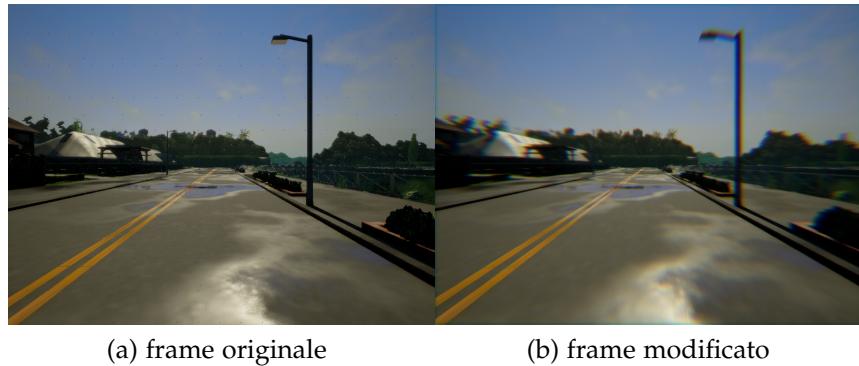


Figura 20: Differenza tra Golden Run e No Chromatic Aberration

netta; per i fallimenti singoli i test set definiti sono due: uno simmetrico, composto da Golden Run-Fallimento, e l’altro, asimmetrico, con lo stesso numero di elementi del precedente ma diversamente distribuiti tra le due etichette presenti (per apprezzare meglio questo concetto si rimanda alla Figura 22a). Oltre tutto gli elementi appartenenti al test set asimmetrico non sono solo Golden Run- Singolo fallimento, bensì risultano sporcati da tutti i fallimenti. Il set asimmetrico è stato sviluppato per testare quanto l’agente allenato su un singolo fallimento lo riconosca in mezzo a tutti gli altri.

Per quanto concerne l’agente sviluppato per la detection di tutti i guasti, anch’esso presenta set con le stesse dimensioni dei precedenti set, ma al suo interno la suddivisione è differente (essa è come mostrata nella Figura 21); è stato impostato un unico test set simmetrico composto da 7200 immagini con etichetta Golden Run e 7200 immagini sporcate con tutti i vari fallimenti. In questo caso la mancanza del test set asimmetrico è avvalorata dal fatto che sarebbe una ridondanza. Una rappresentazione della suddivisione logica dei dati nei dataset per una generica classe di fallimento è mostrata nella Figura fig. 22.

Il programma sviluppato per la gestione della ripartizione delle immagini è il già citato `carla_image_modify.py` del progetto [github](#) [17]. Per semplicità il programma sopracitato gestisce tutto il processo di sporcatura e distribuzione delle immagini nelle cartelle dedicate, non elemento per elemento, ma a blocchi... o meglio episodi. Questa scelta è visibile sui test set della classe All dove il numero di elementi all’interno delle due sezioni oscilla di una quantità pari a trecento immagini (essi compongono un singolo episodio). Questa scelta di elaborare le immagini non singolarmente, ma a blocchi, è nata dal sistema di memorizzazione

delle immagini acquisite da CARLA che salva il materiale in cartelle rappresentanti i singoli episodi di simulazione. Ciò fa sì che le immagini sporcate da un determinato guasto non siano la totalità delle immagini a disposizione diviso il numero dei guasti, ma il numero di episodi diviso il numero dei guasti.

Il programma presenta due routine diverse per la creazione dei dataset a seconda del tipo di classe che deve gestire. Di seguito sono descritte:

- **Singola classe di guasto:** i guasti sono impiegati singolarmente nella costruzione del train, validation e test set sulla singola classe, scrivendo un'immagine sia nella sua versione pulita, sia in quella "sporca" all'interno delle rispettive sottocartelle del set. Le immagini pulite sono immagazzinate nella sezione "Original" mentre le modificate nella sezione "Modified" del set. Per il test set su tutte le classi invece il set sviluppato è asimmetrico; per questo nella sezione "Original" oltre a comparire la versione pulita di tutte le immagini è presente per ogni singolo guasto (diverso da quello della classe in esecuzione) un episodio di elementi sporcati dai guasti stessi. Nella sezione "Modified" dello stesso set sono presenti solo le immagini sporcate dal guasto a cui appartiene il dataset.
- **Classe ALL:** Per questa classe il programma impiega tutti i guasti nella creazione del dataset. Come mostrato in Figura 22b il guasto non presenta il test set asimmetrico come per i singoli guasti. Per la costruzione dei set il procedimento prevede che gli elementi di un episodio siano sporcati da un singolo guasto e siano salvati nella loro forma originale nella cartella "Original"; diversamente la forma sporca è salvata nella cartella "Modified". Questo sistema permette così che ciascun guasto sia rappresentato all'interno dei vari set garantendo però che l'immagine si presenti solamente come Golden Run e sporca da un unico guasto.

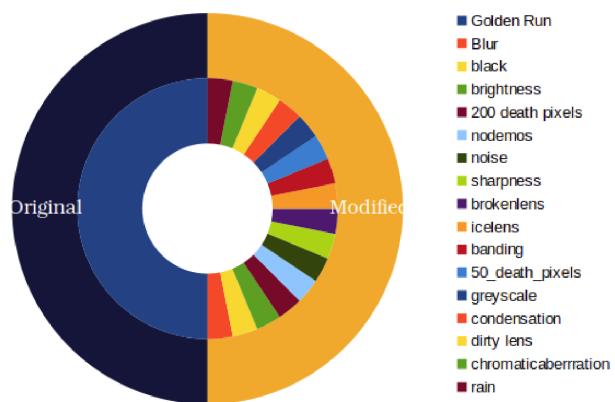
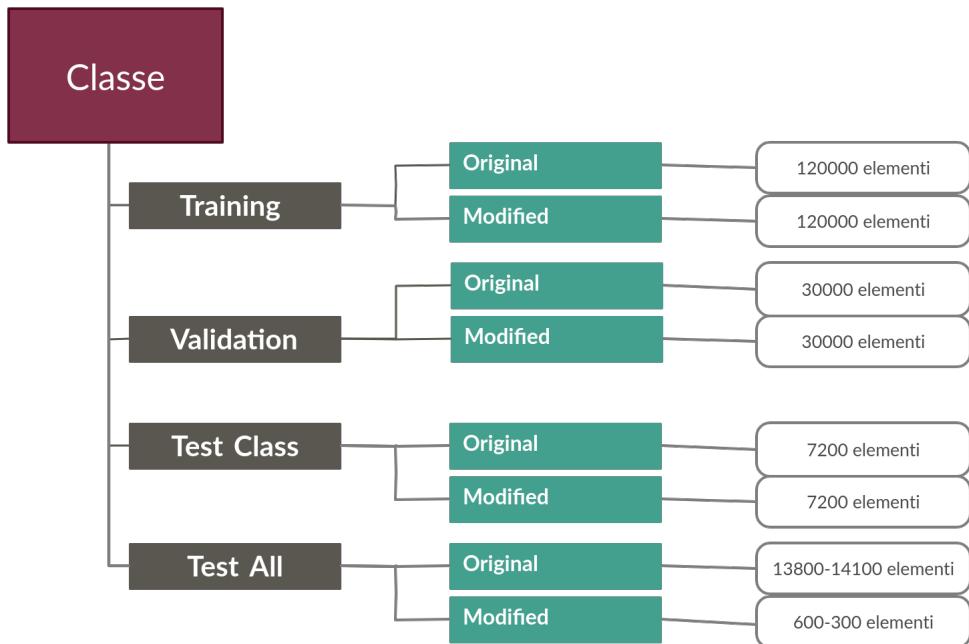
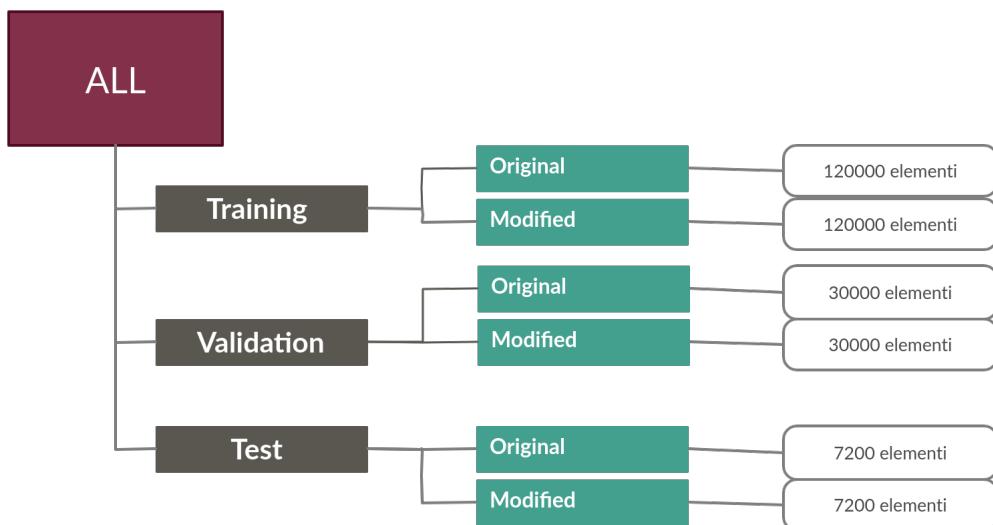


Figura 21: Esempio di image set per l'agente All



(a) Struttura del dataset di una generica classe di fallimento



(b) Struttura del dataset della etichetta All

Figura 22: Rappresentazione delle strutture dei dataset

# 4

---

## COSTRUZIONE DEL DETECTOR

---

Lo scopo del detector è quello di individuare all'interno del flusso di immagini della fotocamera la presenza di distorsioni, ovvero riuscire a prevenire il fallimento del sistema decisionale a causa di un guasto della fotocamera stessa. Per perseguire il suo scopo il detector deve essere in grado di classificare ciascun'immagine del flusso, destreggiandosi tra immagini pulite (ovvero immagini che non generano fallimenti nel sistema) e immagini sporche (generate da un guasto). Questa distinzione binaria tra le due sole tipologie di immagini scaturisce dal fatto che non è richiesta l'identificazione precisa di un singolo guasto (multiclassificatore); si richiede al contrario la sola distinzione tra elementi buoni e elementi cattivi. L'elemento classificatore è svolto da una rete neurale convoluzionale o ConvNet. Come si evince dalla denominazione, la ConvNet è una tecnologia basata sulle reti neurali più generiche; essa è già ampiamente impiegata nei sistemi di riconoscimento grafico come classificatori, object detection e vocal recognition.

### 4.1 CONVOLUTIONAL NEURAL NETWORK

Una rete neurale convoluzionale è una rete neurale di tipo feed-forward network dove la strato convoluzionale compare almeno una volta.

**Input** L'input di una Convolutional Neural Network o ConvNet è un'immagine, o meglio una matrice di valori di ogni singolo pixel occupante una precisa posizione all'interno dell'immagine. Nel caso di immagini RGB queste sono descritte da un terza di matrici dell'intensità dei colori primari (rosso, verde, blu); nel caso di immagini in bianco e nero queste sono descritte da una singola matrice. Le matrici che compongono le immagini identificano i **canali** (o profondità) della rete. In generale la ConvNet opera su strutture a griglia contraddistinte da relazioni

spaziali tra pixel, ereditate da uno strato al successivo tramite valori che descrivono piccole regioni locali dello strato precedente. L'insieme di matrici degli strati nascosti (hidden layer) ottenuto dalla convoluzione, o da altre operazioni, è definito feature map (o activation map). I parametri addestrabili invece sono tensori denominati filtri o kernel [7].

#### 4.1.1 *L'architettura*

Una CNN è composta da una successione di strati che si possono ripetere. Gli strati principali sono [7]:

- convolutional layer;
- activation layer;
- pooling layer;
- dense layer;

Di ciascuno sarà data una breve descrizione in seguito.

#### 4.1.2 *Convolutional layer*

Per quanto concerne l'analisi delle immagini il ConvLayer ha dimostrato di avere molto successo nell'ottenere buoni risultati. Successo ottenuto data la sua capacità di estrarre le piccole caratteristiche che compongono un'immagine. Per estrarle viene applicato un kernel (o filtro) di dimensioni  $[n \times n \times \text{canale}]$  in ogni possibile posizione dell'immagine coprendola interamente; inoltre è computato il prodotto scalare tra il filtro stesso e la matrice corrispondente del volume di input avente uguale dimensione. È possibile visualizzare la convoluzione come una sovrapposizione del kernel sull'immagine in input [12]. Il risultato di questa operazione è una nuova matrice (feature map) di dimensioni pari o minori rispetto alla matrice in ingresso. In particolare le dimensioni della feature map dipende da due fattori, lo stride e il padding. Lo stride indica il numero di pixel di cui la matrice si sposta dopo ogni operazione mentre il padding definisce il comportamento del kernel sui bordi della matrice a cui è applicata [1]. Nella Figura 23 è mostrato un esempio del suo funzionamento. Questa sua capacità di estrarre le caratteristiche da una matrice in input, combinata con la sua multipla presenza nella rete, permette alla ConvNet di distinguere forme sempre più complesse. Questo risultato è dovuto alla

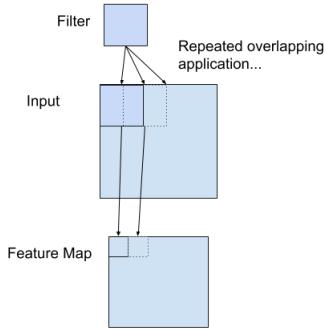


Figura 23: Esempio del funzionamento di un Convolutional Layer.

capacità del filtro di modificarsi al fine di apprendere le caratteristiche dei pattern delle immagini. Infatti i filtri dei primi strati individuano forme primitive mentre quelli successivi imparano a distinguere forme sempre più grandi e complesse; il processo richiede così lunghe sequenze di blocchi di strati per studiare un'immagine intera [7].

#### 4.1.3 Funzione di attivazione *Relu*

L'operazione di attivazione non-lineare segue l'operazione di convoluzione. Per ogni strato la funzione di attivazione genera uno strato di eguale dimensione di valori limitati da soglie. In quanto semplice mapatura uno-a-uno dei valori di attivazione la funzione ReLU non altera l'impronta spaziale dello strato.

#### 4.1.4 Pooling layer

Il pooling è una tecnica che viene utilizzata per estrarre le informazioni più utili da un'area in uno strato convoluzionale e quindi ridurre la quantità di informazioni da analizzare nella fase successiva. L'area di input del livello di raggruppamento è chiamata livello di filtro ed è l'equivalente del livello del campo ricettivo del ConvLayer. Il parametro stride nel pooling layer, oltre a comportarsi allo stesso modo del omonimo del Convlayer, è importante per tre motivi: il primo è la riduzione dell'impronta spaziale delle mappe di attivazione, il secondo è un certo grado di invarianza alla traslazione e il terzo è un incremento del campo ricettivo [12]. Esistono diversi tipi di pooling, ma la variante che ha dimostrato avere maggior successo è il max pooling; esso infatti estrae il massimo valore contenuto in matrici quadrate di ogni mappa di attivazione e produce un altro

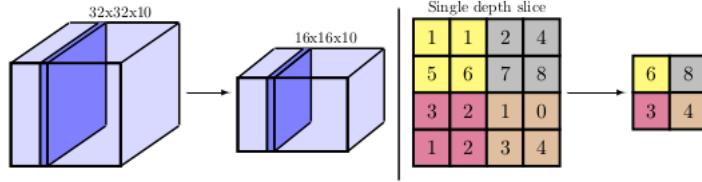


Figura 24: Esempio di pooling con stride e dimensione del filtro pari a due.

hidden layer di eguale profondità. Un altro motivo del loro successo è il grado di non-linearietà e invarianza alla traslazione che introducono [12]. Quest'ultima variante sarà utilizzata nel progetto.

Un esempio di come può essere utilizzato il max pooling è mostrato nella Figura 24, dove vengono utilizzati uno stride e un filtro di dimensioni  $2 \times 2$ .

#### 4.1.5 Dense layer

Il dense layer connette ogni feature map in input con la corrispondente feature map in output. Presenta la struttura di una rete feed-forward tradizionale e aumenta a dismisura il numero di parametri addestrabili per effetto del numero di connessioni. Ad esempio, se due strati completamente connessi hanno 4096 unità ciascuno, il numero di connessioni (quindi di parametri) sarà superiore a 16 milioni. [12]

#### 4.1.6 Struttura della rete sviluppata

La rete costruita per il progetto è stata sviluppata nel programma tensor.py scritto in Python nella versione 3.6 impiegando principalmente la libreria TensorFlow-GPU nella sua versione 1.14.

Per la costruzione della rete sono stati tenuti di conto due aspetti diversi: uno volto alle dimensioni della rete (ovvero definendo una struttura tale da garantire dei risultati promettenti) mentre l'altro volto a minimizzare le sue dimensioni (al fine di non gravare troppo sulla esigua capacità della RAM dedicata alla GPU). Dopo attente ricerche la struttura che ha mostrato un bilanciamento tra i due aspetti espressi in precedenza è quella mostrata nella Tabella 1. In particolare per tutti gli strati convoluzionali è stato utilizzato un padding SAME e come funzione di attivazione, ReLu. L'utilizzo del opzione SAME per il padding fa sì che le dimensioni della feature map in output sia identica alle dimensioni della feature map in input. Per la compilazione della rete invece è stato utilizzato come otti-

mizzatore "adam" e come metrica sia per il training che per la validation, Accuracy, mentre come loss function, BinaryCrossentropy. Per il testing è stato utilizzata una metrica diversa, data la presenza di un dataset asimmetrico per il quale Accuracy come metrica non risulta ottimale. Per questo la metrica utilizzata è stata la MCC, ovvero MatthewsCorrelationCoefficient, più adatta a dataset asimmetrici. Quest'ultima è direttamente calcolabile dalla confusion matrix secondo la formula:

$$MCC = \frac{TP \times TN - FP \times FN}{\sqrt{(TP + FP)(TP + FN)(TN + FP)(TN + FN)}} \quad (4.1)$$

Per quanto concerne Accuracy è una metrica presente ufficialmente in Tensorflow mentre MCC è sviluppata nativamente solo per Tensorflow 2.x in librerie di terze parti. Al fine di calcolare la MCC nella fase di testing sono state utilizzate le seguenti metriche per compilare la rete: `True_positive`, `True_negative`, `False_positive`, `False_negative`, ottenendo così una Confusion Matrix.

N.	Layer	Dimensioni filtro	Output Shape
0	Input	-	800X600X3
1	Conv2d	3X3	800X600X16
2	ReLU	-	800X600X16
3	Max pooling2d	-	400X300X16
4	Conv2d	3X3	400X300X32
5	ReLU	-	400X300X32
6	Max pooling2d	-	200X150X32
7	Conv2d	3X3	200X150X64
8	ReLU	-	200X150X64
9	Max pooling2d	-	100X75X64
10	Flatten	-	48000
11	Dense	-	512
12	Dense	-	1

Tabella 1: Struttura della rete sviluppata



# 5

---

## ESECUZIONE E RISULTATI

---

In questo capitolo saranno mostrati i risultati ottenuti in questo progetto. Si specifica che tutto il lavoro svolto è stato eseguito sul server `lutezio@math.unifi.it`. Le componenti del server sono le seguenti:

- Processore: 2 CPU Xeon X5650
- Memoria fisica: 48 GB di RAM
- Memoria secondaria: 2TB
- Scheda grafica: NVidia RTX 2080
- Memoria fisica GPU 8 GB di Ram dedicata
- Sistema Operativo: Ubuntu 18.04 LTS

`tensor.py` è il programma sviluppato per la costruzione della rete, il training e il testing degli agenti. Questo è scritto in Python nella versione 3.7 e svolge tutte le azioni sopra citate utilizzando le funzioni presenti nella libreria TensorFlow-GPU nella sua versione 1.14.

### 5.1 ADDESTRAMENTO DEL RETE CONVULUZIONALE

Come già espresso in precedenza per ciascuna classe sono state eseguite cinque epoch con batch di dimensione pari a 4 (dimensioni forzate a causa delle caratteristiche della RAM della GPU dedicata). Le metriche impiegate per questa fase si ricordano essere Accuracy e BinaryCrossentropy. I risultati della fase di training e validation ottenuti per ciascun agente sono mostrati nella Tabella 2. Si specifica che sia per la fase di training che per le fasi di validation è stato impostato lo shuffle; il mescolamento è stato attivato in maniera tale da somministrare i dati in ordine casuale al fine di ridurre la possibilità di generare overfitting negli agenti.

Per il codice specifico utilizzato in questa fase si rimanda alla sezione del Manuale Utente.

	Train		Validation	
Classi	train loss	train acc	val loss	val acc
Blur	2,26E-12	1	1,97E-12	1
black	0	1	3,92E-36	1
brightness	2,26E-16	1	1,97E-16	1
50 death pixels	0,6932	0,5	0,6932	0,5
200 death pixels	0,6932	0,5	0,6932	0,5
nodemos	9,84E-20	1	2,06E-26	1
noise	2,42E-13	1	4,18E-17	1
sharpness	0,0015	1	2,93E-27	1
brokenlens	5,93E-16	1	4,30E-16	1
icelens	0,0089	0,9995	0,0017	0,9996
banding	0,0132	0,9995	0,0026	0,9992
greyscale	2,26E-22	1	2,22E-22	1
condensation	1,03E-19	1	1,73E-17	1
dirty lens	0,0157	0,9985	0,028	0,9977
chromaticaberration	2,96E-17	1	0	1
rain	1,58E-16	1	5,03E-28	1
all	0,0996	0,9358	0,0965	0,9337

Tabella 2: Tabella dei risultati della fase di training e validation

## 5.2 TESTING DELLA RETE

Ottenuti i 17 agenti istruiti, la fase finale del progetto è verificare la loro effettiva efficacia sulle due tipologie di test set. Per gli agenti dedicati al rilevamento di un singolo guasto i test sono effettuati su due set diversamente composti: uno simmetrico e uno asimmetrico. Come già mostrato nella sezione 3.4 il set simmetrico presenta la stessa struttura del rispettivo train set e validation set mentre il test set asimmetrico presenta una struttura diversa. In quest'ultima l'etichetta Original è composta sia da immagini pulite sia da immagini sporcate dai vari guasti (diversi da quello dell'agente da testare); mentre l'etichetta Modified di dimensioni

minori rispetto alla precedente è composta solamente da immagini affette dal guasto che caratterizza l'agente.

Per quanto concerne l'agente allenato nell'identificare tutte le classi di guasto è definito solo un test set simmetrico strutturato alla stessa maniera degli altri suoi set.

Per quanto riguarda il testing la funzione utilizzata è stata evalute della libreria di Tensorflow. La funzione testa il modello su un set di dati e restituisce i valori della funzione di loss e della metrica. Nella Tabella 3 sono mostrati i valori della confusion matrix e della MCC ottenuti dalla fase di testing. Nella Figura 25 è mostrato un grafico riassuntivo dei valori della metrica MCC. Si ricorda che la metrica MCC è una funzione che ha per codominio valori compresi tra  $-1$  e  $1$ .

Tabella 3: Tabella: Confusion Matrix e MCC ottenute dalla fase di testing

classi	simmetrico						asimmetrico			
	tn	tp	fn	fp	mcc	tn	tp	fn	fp	mcc
Blur	7200	7200	0	0	1	404	600	0	13397	0,04
Black	7200	7200	0	0	1	13800	600	0	0	1
Brightness	0	7200	0	7200	0	868	600	0	12932	0,05
50 death pixels	0	7200	0	7200	0	0	300	0	14100	0
200 death pixels	0	7200	0	7200	0	0	600	0	13800	0
Nodemos	7200	7200	0	0	1	13200	600	0	600	0,69
Noise	7200	7200	0	0	1	13200	600	0	600	0,69
Sharpness	6859	7200	0	341	0,95	12175	600	0	1625	0,49
Brokenlens	7200	7200	0	0	1	13800	600	0	0	1
Icelens	6041	5741	1459	1159	0,64	11215	278	22	2885	0,25
Banding	695	7181	19	6505	0,22	2907	300	0	11193	0,07
Greyscale	5670	7200	0	1530	0,81	600	300	0	13500	0,03
Condensation	7200	7171	29	0	1	14053	200	100	47	0,73
Dirty lens	4833	3552	3648	2367	0,17	9295	54	246	4805	-0,05
Chromaticaberration	7200	7200	0	0	1	13500	300	0	600	0,56
Rain	7166	5534	1666	34	0,78	13992	197	103	108	0,64
All	4	7198	2	7196	0,01					

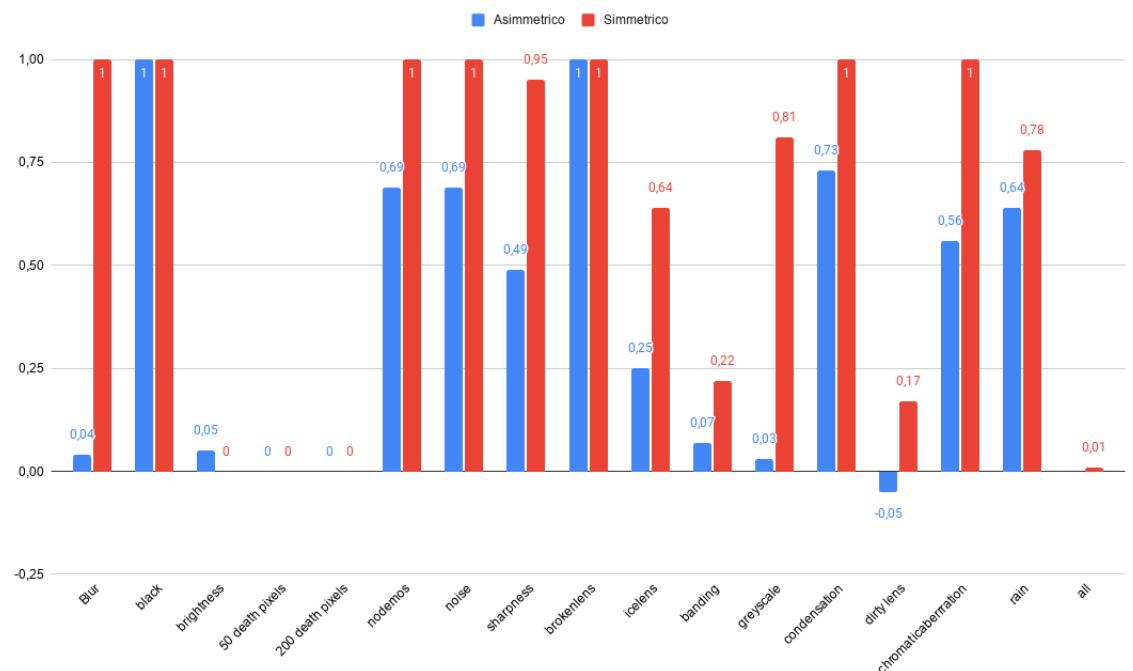


Figura 25: Risultati fase di Testing:MCC



# 6

---

## CONCLUSIONI E LAVORI FUTURI

---

Dai risultati ottenuti nella Figura 25 un agente basato sulla Convolutional Neural Network (come soluzione software per la detection del fallimento del sensore della fotocamera) risulta essere una valida soluzione. Questa considerazione scaturisce dal fatto che semplicemente con solo tre fasce di strati, ognuna composta da Conv2d e Maxpooling (e solo 5 epoch), l'agente presenta un'ottima capacità di classificazione nei casi in cui la fotocamera subisca un solo tipo guasto; inoltre mostra altrettante buone capacità nei casi in cui il flusso di immagini, tra sensore e sistema decisionale, presenti più di un guasto. Prima di andare a commentare i vari risultati è necessario presentare lo studio su cui è basata la scelta dell'impiego di un agente per la detection del fallimento della fotocamera in un sistema self driving.

Lo studio [14] ha evidenziato come una serie di fallimenti della fotocamera nel sistema self driving incida sulle capacità della componente decisionale nello svolgere le sue attività. I risultati ottenuti da questo studio sono mostrati alla seguente Tabella 26. Nella Figura 26, viene sintetizzato il success rate ottenuto da tre diversi tipi di test (turn, straight e navigation) per le varie configurazioni di fallimento. Questi tre test sono caratterizzati nel seguente modo:

- Straigth road: La posizione di destinazione B si trova diritta davanti alla posizione di partenza A.
- Turn road: La posizione di destinazione B è a un giro di distanza da la posizione di partenza A.
- Navigation road: Non ci sono restrizioni sulla posizione del file posizione di destinazione B rispetto alla posizione di partenza A; ciò si traduce in corse di maggiore distanza e più svolte [14].

Questi test sono basati sul benchmark *corl2017*. Questo benchmark è composto da multiple runs nelle quali un veicolo raggiunge una destina-

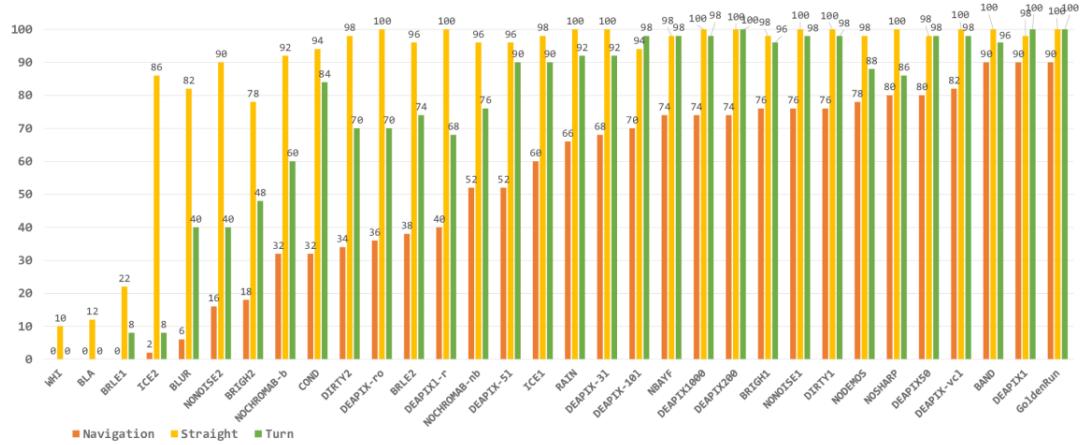


Figura 26: Tabella dei success rate (riprresa da [14])

zione B da un punto A in un tempo preciso. Quest'ultimo è impostato calcolando il tempo medio che un veicolo a 10 km/h impiega a percorre l'intero tragitto che congiunge il punto A al punto B. La locazione dei punti A e B definisce il diverso tipo di test, come definito sopra [14]. Sulla base dei dati esposti in Figura 26, ciò che traspare è che l'efficacia di alcuni agenti rispetto ad altri risulta avere un peso maggiore (riuscendo a identificare i fallimenti che gravano di più sul sistema decisionale del veicolo).

Dai risultati del mio lavoro gli agenti che presentano le migliori prestazioni sono Black e Brokenlens avendo ottenuto ottimi punteggi in entrambi i test set. Capacità così elevate date probabilmente dalla particolare caratterizzazione dell'immagine da parte del guasto.

**BLUR** Con valori ottimi nel riconoscere le immagini pulite da quelle affette da Blur, l'agente presenta invece delle prestazioni peggiori con le immagini affette da più guasti. Oltre che dal valore del MCC, le basse capacità sono mostrate dalla sua Confusion Matrix ove risulta che abbia classificato più o meno tutte le foto come sporcate dal suo guasto. Ciò che non si può estrapolare da questi dati è se nelle 404 immagini classificate come True negative ovvero non Blur, siano presenti immagini pulite.

**BLACK E BROKEN LENS** I due agenti istruiti sui seguenti guasti hanno ottenuto i migliori risultati in entrambi i test, per tutte le metriche, rendendoli così degli ottimi candidati al loro impiego in un sistema self driving.

**BRIGHTNESS** Il seguente agente mostra uno dei risultati più particolari dell’intera campagna sperimentale. Durante la fase di training presenta dei valori, Binary Crossentropy per la loss function e Accuracy per la metrica, incoraggianti con un loss value vicino allo zero e un accuratezza pari ad 1. Ciò che è stato rilevato dalla fase di testing è: una totale incapacità nel distinguere nel primo test la differenza tra immagini pulite e sporche classificando il tutto come immagini sporche, mentre nel secondo test i risultati sono stati leggermente diversi, ma comunque tali da non garantire allo stato attuale il suo utilizzo come detector in qualsiasi sistema dotato di visione artificiale.

**50 DEATH PIXELS E 200 DEATH PIXELS** I due agenti sono analizzati insieme poichè mostrano gli stessi risultati in ogni fase. Risultati non sono molto soddisfacenti in quanto mostrano una completa inefficienza nel riconoscere il proprio fallimento. Inefficienza scaturita con molta probabilità dalla poca differenziazione tra l’immagine pulita e l’immagine guasta. Le immagini impiegate presentano 800 pixel×600 pixel, per un totale di 480000 pixel, di cui nel caso dei guasti solo 50 e 200 sono anneriti. Numeri che come è stato dimostrato nel progetto [14], non impattano in maniera pesante sul sistema decisionale (nella Figura 26 i valori di cui tenere conto sono quelli etichettati come deapix50 e deapix200). Una modifica che potrebbe essere testata per il miglioramento di questi due agenti potrebbe essere quella di incrementare il numero di hidden layer al fine di aumentare la peculiarità di analisi dell’immagine da parte della rete.

**NO DEMOSAICING, NO NOISE, CHROMATIC ABERRATION** Gli agenti in questione mostrano nella fase di training valori ottimi con una accuratezza pari a 1 e una loss pari a 0. Valori che si ripresentano ottimi nel caso del test simmetrico ma non altrettanto nel test asimmetrico. Nell’ultimo caso la particolarità di entrambi è che classificano in maniera perfetta i loro guasti e quasi la totalità delle altre immagini. L’unica eccezione per entrambi è la classificazione di 600 elementi segnati come False Positive. Di quest’ultimi non si può identificare in maniera assoluta la loro classe di appartenenza. Ciò che risulta però particolare è che quel preciso numero di elementi classificati erroneamente hanno le stesse dimensioni degli elementi che una classe di guasto ha all’interno del set asimmetrico. La domanda che può sorgere da questa coincidenza è se questi tre agenti siano in grado di identificare altri guasti simili al loro oppure no, presentando così una loro maggiore versatilità nell’uso.

**SHARPNESS** Agente che come i tre precedenti mostra ottime prestazioni nel test simmetrico, mentre nel test asimmetrico presenta degli errori nella classificazione di alcune immagini.

**CONDENSATION** Un altro agente che ha riportato risultati particolari è Condensation. Presenta dei piccoli errori nella classificazione di alcune immagini guaste nel test simmetrico, mentre sbaglia 100 elementi modificati dal suo guasto nel test set asimmetrico. Questi cento elementi non sono stati identificati, ma potrebbero rivelare la mancanza dell'agente nel classificare una particolare variante del guasto. Affermazione avvalorata dal fatto che il numero delle varianti di Condensation prese in considerazione per tutti i set sono tre e, poiché il numero degli elementi sporcati nel test set asimmetrico da Condensation sono 300, ogni variante presenta 100 elementi. La suddetta teoria può essere verificata solamente sviluppando ulteriori test su set differentemente strutturati.

**RAIN** L'agente che ha presentato una variabilità minore tra test set simmetrico e asimmetrico è il Rain che risulta essere buono in entrambe le situazioni. La teoria esposta per l'agente Condensation non è ugualmente applicabile all'agente Rain in quanto il numero di varianti prese in considerazione in questo caso sono 5 (per variante si intende un determinato filtro applicato alle immagini).

**ICELENS** Icelens è un agente che presenta dei buoni risultati nel riconoscere i propri guasti sia in ambiente mono-guasto sia in presenza di più malfunzionamenti; potrebbe essere migliorato in seguito aumentando il tempo di training e il numero di hidden layer (ricordando che l'agente è stato allenato su tre diverse varianti del suo guasto).

**BANDING** Per quanto concerne il Banding i risultati del training e quelli dei test risultano essere discordanti. Nella fase di training e validation come ogni altro agente mostra delle performance particolarmente alte, mentre nella fase di testing, sia simmetrico che asimmetrico, i risultati ottenuti mostrano una particolare incapacità dell'agente nel distinguere gli elementi. Esso classifica tutti gli elementi come immagini guaste (allo stesso modo dell'agente Death Pixel). Comportamento prodotto probabilmente dalla poca differenziazione tra immagine Golden Run e Banding (per un esempio del guasto si rimanda alla Figura 16). È da notare come questo guasto non influisca in maniera particolare nel sistema decisionale di un veicolo self driving, come mostrato nella Figura

26 all’etichetta "BAND".

**GREY SCALE** Grey Scale si presenta con ottime prestazioni nel test simmetrico, al contrario nel test asimmetrico mostra un deficit. Questa mancanza è particolare in quanto non riesce a classificare tutte le immagini pulite ma solamente una piccola parte di esse. Sarebbe interessante poter classificare a quali classi di guasto appartengano gli elementi di questo piccolo raggruppamento.

**DIRTY LENS** Ultimo degli agenti a singolo guasto, Dirty Lens mostra uno dei risultati peggiori per la metrica MCC in entrambi i test. La considerazione da tenere di conto nel giudizio dell’agente è l’alto numero di varianti (pari a 36) utilizzate per tutti i suoi dataset. Queste variazioni hanno sicuramente inficiato le prestazioni attuali dell’agente, ma approntando un aumento degli hidden layer e del tempo di allenamento si può arrivare a una miglioramento.

**ALL** L’agente All sviluppato per la detenction di tutti i guasti presi in considerazione non presenta degli ottimi risultati nel suo unico test (simmetrico). La sua capacità classificatrice così bassa è dovuta con molta probabilità all’alta variabilità dovuta a tutti i guasti. Essa può essere migliorata andando a ridurre il numero di guasti da identificare, rimuovendo tutti quelli che non hanno un grosso impatto sul sistema decisionale e tutti quelli di cui neanche l’agente addestrato sul singolo guasto è riuscito a dare un’identificazione (Death Pixel). Un’altra alternativa per l’aumento delle prestazioni, che non esclude la precedente, è l’implementazione di ulteriori hidden layer (Conv2d e Maxpooling). Questo fa sì che la capacità dell’agente di estrarre i particolari identificativi del guasto dall’immagine migliori. Infine l’applicare il callbacks EarlyStopping della libreria di Keras nella fase di training risulta essere una valida soluzione. Con questa feature, impostando un limite alto di epoch eseguibili, il sistema termina l’allenamento quando esegue un numero di epoch pari al limite imposto o quando una metrica smette di migliorare dopo un numero di iterazioni. In questa maniera si ottiene una rete allenata al massimo delle sue capacità con una probabilità ridotta di cadere in uno stato di overfitting.

**CONCLUSIONI** Concludendo, la tecnologia delle Convolutional Neural Network ha mostrato ottimi risultati nel classificare i guasti sia in ambienti più controllati, ovvero mono guasto, sia in ambienti multi guasto.

Questi risultati suggeriscono che è potenzialmente possibile impiegare gli agenti nel flusso di informazioni tra la fotocamera e il sistema decisionale, individuando determinati guasti prima che portino al fallimento dell'intero sistema. Questo fatto non esclude che molti altri agenti testati ma non perfetti potrebbero essere impiegati nei sistemi dopo un miglioramento delle loro prestazioni; esso potrebbe essere raggiunto tramite un aumento della durata del training impiegando come funzione di terminazione, EarlyStopping. L'aumento delle dimensioni della rete migliorerebbe la precisione. Un futuro sviluppo di questa ricerca può essere:

- studiare tutti i guasti che non sono stati presi in considerazione in questo progetto
- verificare se alcuni guasti possono essere individuati anche da altri agenti non allenati nel riconoscerli
- sviluppare agenti in grado di riconoscere immagini multiguasto.

Queste sono solo alcuni dei possibili sviluppi applicabili agli agenti nella detection dei fallimenti della videocamera.

# 7

---

## A MANUALE UTENTE

---

### 7.1 ACQUISIZIONE

Per l'acquisizione delle immagini è stato utilizzato CARLA nella sua versione 0.8.4 e il progetto github [18]. Si riportano i comandi utilizzati...  
Lato Server:

---

```
DISPLAY=:21 ./CarlaUE4.sh -opengl -world-port=10500 -quality=low
```

---

Lato Client:

---

```
python3 client_example.py --images-to-disk --location=_out -p=10500
```

---

Si precisa che al momento della scrittura della tesi nella versione 0.8.4 di CARLA il sistema di configurazione del Client attraverso il file .ini non era funzionante. Per questo è stato necessario aprire il file `client_example.py` presente di default all'interno della cartella `PythonClient` e modificare i campi interessanti. Nel caso attuale sono stati modificati il numero degli episodi che sarebbero stati eseguiti poi dal sistema e la posizione della camera RGB all'interno del veicolo. A ogni modo per visionare il file `client_example.py` modificato e usato per il progetto, si rimanda alla pagina github [17]. Comunque si precisa inoltre che una run da 500 episodi da 300 immagini ciascuno occupa all'incirca 80 GB di memoria.

### 7.2 SPORCATURA E SUDDIVISIONE DEL DATASET

Il programma principale di questa fase è `carla_image_modify` sviluppato in python 3.6. Al suo interno sono presenti solamente due semplici funzioni, `manage_image` e `modify_photo`. La prima gestisce l'intero processo: permette di definire il numero di episodi da dedicare a ciascuna delle tre tipologie di set, gestisce l'apertura e il caricamento delle immagini dai vari episodi e li passa alla fase di sporcatura. Quest'ultimo processo è

attuato tramite la funzione `open_cv2()` del file `modify_photo.py` la quale carica tutte le immagini presenti in una determinata cartella a sua volta attraverso la funzione `imread()` della libreria CV2. La seconda funzione invece gestisce la fase di sporcatura: richiama le varie funzioni che simulano i guasti contenuti nel file `python modify_photo.py`.

Il programma `modify_photo.py` sviluppato in Python nella versione 3.6 definisce per ogni guasto una funzione da applicare alle immagini. Per tutti quei guasti che richiedono l'uso della funzione `blend` è stato definita la funzione `overlap` a cui è possibile passare come parametro un fattore di "illuminazione" per regolare la luminosità delle immagini. Per quanto riguarda tutte quelle funzioni che impiegano la funzione `paste` sempre della libreria PIL è definita la funzione `paste_image()`. Tutti gli altri guasti sono stati ripresi dal progetto [github](#) [16]. Nel caso in cui fosse necessaria l'introduzione di nuovi guasti si consiglia la modifica del dizionario `dispatcher` definito nel fondo del file, inserendo il nome della funzione e una stringa che lo identifichi per tutto il programma. Al momento della scrittura della funzione simulatrice di un guasto non vi è alcuna limitazione, l'unica accortezza di cui ci si raccomanda è il rispetto dello standard tenuto nel definire l'intestazione delle funzioni. L'ultimo programma riguarda la creazione delle cartelle secondo la struttura nella Figura 22a e nella Figura 22b, questa è gestita dal programma `python manager_of_path.py`. Si ricorda che il contenuto di questi programmi è visibile nel progetto [github](#) [17].

### 7.3 TRAINING E TESTING: TENSORFLOW

Il programma sviluppato per la gestione di tutti gli aspetti riguardanti l'agente rientra sotto il programma `tensor.py`. Come punto di partenza viene definito il comportamento del sistema nei confronti della RAM della GPU, essendo questa molto golosa di risorse. Il comportamento è impostato con la seguente dicitura:

---

```
config = tf.ConfigProto()
    config.gpu_options.allow_growth = True # dynamically grow the
        memory used on the GPU
    config.log_device_placement = True # to log device placement (on
        which device the operation ran)
    sess = tf.Session(config=config)
```

---

Ciò comporta che l'acquisizione da parte del sistema della RAM agisca secondo la politica di occupare una quantità di memoria crescente tale

da soddisfare la richiesta. Per maggiore sicurezza è stato impostato attraverso callback il salvataggio dei pesi del modello. I checkpoint sono stati eseguiti ogni cinque epoch.

Per il caricamento di ciascun dataset è stata impiegata la funzione `flow_from_directory` della classe `ImageDataGenerator`. Si precisa che per il training set sia per il validation set è stato attivato lo shuffle degli elementi.

Un esempio del codice utilizzato è mostrato in seguito:

---

```
train_image_generator = ImageDataGenerator(rescale=1. / 255)
train_data_gen =
    train_image_generator.flow_from_directory(batch_size=batch_size,directory=mp.get
    IMG_WIDTH),class_mode='binary')
```

---

Il modello, centro di tutto il progetto, è stato sviluppato scegliendo il modello sequenziale con la seguente struttura:

---

```
model = Sequential([
    Conv2D(16, 3, padding='same', activation='relu',
           input_shape=(IMG_HEIGHT, IMG_WIDTH, 3)),
    MaxPooling2D(),
    Conv2D(32, 3, padding='same', activation='relu'),
    MaxPooling2D(),
    Conv2D(64, 3, padding='same', activation='relu'),
    MaxPooling2D(),
    Flatten(),
    Dense(512, activation='relu'),
    Dense(1)
])
```

---

Per la compilazione sono state definite invece due diversi tipologie: la prima per il training e validation, la seconda per i due test set:

---

```
model.compile(optimizer='adam',
              loss=tf.keras.losses.BinaryCrossentropy(from_logits=True),
              metrics=['accuracy'])

model.compile(optimizer='adam',
              loss=tf.keras.losses.BinaryCrossentropy(from_logits=True),
              metrics=[tf.keras.metrics.TruePositives(),
                      tf.keras.metrics.TrueNegatives(),
                      tf.keras.metrics.FalsePositives(),
                      tf.keras.metrics.FalseNegatives()])
```

---

---

Infine per il training la funzione utilizzata è stata fit() impostata nella seguente maniera:

```
#total_train =numero di elementi nel training set, 240000
#total_val=numero di elementi nel validation set, 60000
#batch_size=4
model.fit(
    train_data_gen,
    steps_per_epoch=total_train // batch_size,
    epochs=epochs,
    validation_data=val_data_gen,
    validation_steps=total_val // batch_size,
    callbacks=[cp_callback]
)
```

---

mentre per la fase di testing è stata utilizzata la funzione evaluate() nella seguente maniera:

```
#test_data_gen=generatore per il test set
ls, tp, tn, fp, fn=model1.evaluate(test_data_gen, verbose=0)
```

---

---

## BIBLIOGRAFIA

---

- [1] Mattsson, Niklas. Classification Performance of Convolutional Neural Networks. (2016). (Cited on page 42.)
- [2] Guyon, Isabelle. A scaling law for the validation-set training-set size ratio. *AT&T Bell Laboratories*, 1(11),(1997). Citeseer.'
- [3] Avizienis, Algirdas and Laprie, J-C and Randell, Brian and Landwehr, Carl. Basic concepts and taxonomy of dependable and secure computing. *IEEE transactions on dependable and secure computing*, 1(1), pp. 11–33,(2004). IEEE. (Cited on pages 3, 7, and 9.)
- [4] Nicol, David M and Sanders, William H and Trivedi, Kishor S. Model-based evaluation: from dependability to security. *IEEE Transactions on dependable and secure computing*, 1(1), pp. 48–65,(2004). IEEE. (Cited on page 11.)
- [5] Smith, RM and Trivedi, Kishor S. and Ramesh, AV. Performability analysis: measures, an algorithm, and a case study. *IEEE Transactions on Computers*, 37(4), pp. 406–417,(1988). IEEE.
- [6] Engstrom, Logan and Tran, Brandon and Tsipras, Dimitris and Schmidt, Ludwig and Madry, Aleksander. Exploring the landscape of spatial robustness. *International Conference on Machine Learning*, pp. 1802–1811,(2019). (Cited on pages 21 and 22.)
- [7] Torresin, Luca. Sviluppo ed applicazione di reti neurali per segmentazione semantica a supporto della navigazione di rover marziani. (2019). (Cited on pages 42 and 43.)
- [8] Russell, Stuart J and Norvig, Peter Intelligenza artificiale. Un approccio moderno. 1,(2005). Pearson Italia Spa. (Cited on pages 13, 14, 15, and 16.)
- [9] Bondavalli, Andrea L'analisi quantitativa dei sistemi critici: Fondamenti e Tecniche per la Valutazione-Analitica e Sperimentale-di Infrastrutture Critiche e Sistemi Affidabili. (2011). Società Editrice Esculapio. (Cited on page 8.)

- [10] CENELEC, EN 50126-railway applications: The specification and demonstration of reliability, availability, maintainability and safety (rams) European Committee for Electrotechnical Standardization, 6,(1999).
- [11] Aghdam, Hamed Habibi and Heravi, Elnaz Jahani Guide to convolutional neural networks. New York, NY: Springer, 10, pp. 978–973,(2017). Springer.
- [12] Aggarwal, Charu C and others Neural networks and deep learning. (2018). Springer. (Cited on pages 42, 43, and 44.)
- [13] Das, Plaban. Risk analysis of autonomous vehicle and its safety impact on mixed traffic stream. (2018). (Cited on pages 18, 19, and 20.)
- [14] Secci, Francesco and Ceccarelli, Andrea On failures of RGB cameras and their effects in autonomous driving applications. *arXiv preprint arXiv:2008.05938*,(2020). (Cited on pages 4, 10, 20, 25, 27, 28, 34, 35, 53, 54, and 55.)
- [15] Eykholt, Kevin and Evtimov, Ivan and Fernandes, Earlence and Li, Bo and Rahmati, Amir and Xiao, Chaowei and Prakash, Atul and Kohno, Tadayoshi and Song, Dawn Robust physical-world attacks on deep learning visual classification. *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 1625–1634,(2018). (Cited on page 22.)
- [16] Secci, Francesco. Python\_Image\_Failures (2020). [https://github.com/francescosecci/Python\\_Image\\_Failures](https://github.com/francescosecci/Python_Image_Failures). (Cited on pages 27, 29, 30, 31, and 60.)
- [17] Bernabei, Pietro. Carla\_Detector\_Cam\_Malfunction (2020). [https://github.com/BernabeiPietro/Carla\\_Detector\\_Cam\\_Malfunction](https://github.com/BernabeiPietro/Carla_Detector_Cam_Malfunction). (Cited on pages 27, 37, 59, and 60.)
- [18] alirezahappy. Carla\_Script (2018). [https://github.com/alirezahappy/Carla\\_Script](https://github.com/alirezahappy/Carla_Script). (Cited on pages 26 and 59.)