



Tesi di Laurea

Pietro Bernabei - matricola:6291312

Anno Accademico 2019/20

Contents

1	Introduzione	3
1.1	Motivazioni	3
1.2	Obiettivo	3
1.3	Organizzazione del lavoro	3
2	Fondamenti teorici	4
2.1	Sistemi Critici	4
2.1.1	Dependability	5
2.1.2	Le Minacce: guasti, errori e fallimenti	6
2.1.3	Gli attributi della dependability	8
2.1.4	I mezzi per ottenere la dependability	9
2.2	Artificial Intelligence	13
2.2.1	Agente razionale,Misura di prestazione,Ambiente	13
2.2.2	Agenti che apprendono	14
2.2.3	Neural Network	15
2.2.4	Convulutional Neural Network	17
3	Costruzione del dataset	22
4	Costruzione del detector	24
4.1	Addestramento del rete convulzionale	25
5	Esecuzione e risultati	25
6	Conclusioni e lavori futuri	25
7	A Manuale utente	25

1 Introduzione

1.1 Motivazioni

Nel giro di un paio di secoli il mondo, è passato dal viaggiare in groppa a un cavallo, alla groppa di una macchina, dal essere il conducente, al condotto. Le macchine a guida autonoma parziali e totali stanno diventando ogni giorno che avanza, realtà. Questa rivoluzione sta permeando il nostro stile di vita, diventandone dipendenti a tal punto da richiedere che questi siano continuativi, privi da possibili errori o 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 con guasti porta a incidenti. Come verrà espresso in seguito, le telecamere negli attuali sistemi, ricoprono un ruolo decisivo nel momento decisionale della guida autonoma, e un loro malfunzionamento nel processo di acquisizione ripercuote nel sistema decisionale, errori non molto graditi al guidatore.

1.2 Obiettivo

Nella seguente tesi si propone un sistema software, detto Detector, in grado di rilevare, nel flusso di immagini generate dalla telecamera di un mezzo a guida autonoma, la presenza di malfunzionamenti nel sistema di acquisizione, come congelamento, pixel bruciati, e tanti altri, per poi notificare i possibili esiti al sistema decisionale del mezzo.

1.3 Organizzazione del lavoro

Avendo definito il detector, come un sistema in grado di rilevare una variazione malevola nel flusso di immagini acquisite dalla fotocamera,

questo dovrà essere in grado di classificare l'immagine sottoposta a decisione. Per questo la soluzione descritta,impiega una particolare forma di intelligenza artificiale,la rete neurale convuluzionale (CNN)... Come verrà esposto meglio in seguito, una convulutional neural network è una particolare forma di supervised learning, particolarmente efficiente in ambienti grafici, soprattutto per la classificazione di intere immagini o identificazione di elementi al loro interno. I supervised learning, e quindi anche CNN richiedono una fase di training, nella quale viene definita la propria conoscenza di base (Knowledge base), sulla quale si baserà per prendere le decisioni. Detto ciò lo sviluppo del detector si divide in:

- Definizione e generazione Dataset
- Creazione CNN
- Training CNN

2 Fondamenti teorici

2.1 Sistemi Critici

Ogni giorno, una persona usa infrastrutture, mezzi, telecomunicazioni, e servizi di qualsiasi genere, affidandosi totalmente al loro funzionamento, alla loro continuità, dando per scontato che non possano subire guasti o malfunzionamenti, perchè ove questi avvengano il risultato sarebbe devastante per tutto il nostro sistema di vita. Queste componenti si definiscono come **Sistemi Critici**, e il loro corretto funzionamento, dipende dalla nostra capacità di analizzarne aspetti quantitativi relativi sia a caratteristiche prestazionali quali velocità di elaborazione o altre misure di efficienza, sia caratteristiche di sicurezza, disponibilità o affidabilità che dimostrino e ci convincano della adeguatezza

dei nostri manufatti per i 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.

Detto ciò nella sua definizione originale **dependability** è la capacità di un sistema di fornire un servizio su cui è possibile fare affidamento in modo giustificato.[4]

Una definizione alternativa, che stabilisce un criterio per decidere se un determinato servizio è dependable, definisce la **dependability** di un sistema come la capacità di evitare fallimenti che siano più frequenti e più severi del limite accettabile [3].

Ciò di cui la definizione precedente non tiene conto, è che il comportamento del nostro sistema, sia sempre definito nella sua totalità, senza ambiguità e precisa. Per questo la dependability può essere vista come una misura di quanta fiducia possiamo riporre in modo giustificato sul servizio erogato dal sistema stesso.[4]

Un'esposizione sistematica dei concetti relativi alla dependability consiste di tre parti:

- **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:[4]

- **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 susseguente 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 persistenza dove abbiamo guasti permanenti, transienti ed intermittenti. Per una tassonomia completa si rimanda a [3].
(**riguarda**)

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 possono venire classificati secondo la loro gravità (severity).

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.

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):** 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.
- **La sicurezza (safety)** è poi l'assenza di conseguenze catastrofiche sugli utenti e sull'ambiente circostante. La safety può essere vista come l'affidabilità del sistema .
- **sicurezza (security)** può quindi essere vista come la contemporanea esistenza di availability solo per gli utenti autorizzati, confidentiality, e integrity, dove per “improprie” si intende “non autorizzate” [5].

Ciascuno di questi attributi può essere più o meno importante in base all'applicazione: la disponibilità del servizio è sempre richiesta, anche se può variare sia l'importanza relativa che il livello quantitativo richiesto mentre, la affidabilità, la safety, la confidenzialità e gli altri attributi possono essere richiesti o meno. 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.

I requisiti di dependability di un sistema sono forniti attraverso una descrizione degli obiettivi richiesti per uno o più degli attributi sopra descritti, rispetto alle modalità di fallimento previste per il sistema. 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 e può assumere tre forme: rollback, dove la trasformazione consiste nel ritornare ad uno stato in cui si trovava il sistema prima della rilevazione dell'errore; rollforward, dove si porta il sistema in uno stato del tutto nuovo, e compensation, dove lo stato contiene abbastanza ridondanza per eliminare la parte erronea. Fault handling impedisce che i guasti che sono stati localizzati vengano nuovamente attivati, attraverso quattro fasi:

- **fault diagnosis** identifica l'origine della causa degli errori, in termini di locazione e tipo;

- **fault isolation** isola logicamente o fisicamente il componente, impedendogli di partecipare all'erogazione del servizio, trasformando il guasto in un guasto dormiente;
- **system reconfiguration** che riconfigura il sistema, ad esempio attivando componenti di riserva o ridistribuendo il carico tra i componenti funzionanti;
- **system reinitialization**, che esegue i controlli e gli aggiornamenti necessari in seguito alla nuova configurazione.

Solitamente l'attività di fault handling è seguita da azioni di manutenzione correttiva, che rimuovono i guasti isolati dal fault handling, ad esempio sostituendo un componente segnalato come guasto. Il fattore che distingue la fault tolerance dalla manutenzione (maintenance) è che quest'ultima richiede l'intervento di un agente esterno.

Rimozione dei guasti La “Fault Removal” viene effettuata sia durante la fase di sviluppo, che durante la vita operativa 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. Le tecniche di verifica possono essere classificate in base alla necessità di esercitare il sistema. La verifica di un sistema senza la sua esecuzione è una verifica statica, altrimenti è una verifica dinamica. La rimozione dei guasti durante la sua vita operativa 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. Alcuni metodi di analisi sono specifici per una valutazione qualitativa o quantitativa, mentre altri possono essere utilizzati per entrambi i tipi di analisi. I due approcci principali per il fault forecasting di tipo probabilistico sono la modellizzazione e il testing. Questi approcci sono complementari: la costruzione di un modello del sistema richiede delle informazioni su alcuni processi di base del sistema, che possono essere acquisite tramite testing. Generalmente, un sistema eroga diversi servizi, e spesso esistono due o più modi di erogazione del servizio, ad esempio da servizio a pieno regime, a servizio di emergenza. Questi modi distinguono la qualità o completezza del servizio erogato. Misure di dependability collegate alla qualità del servizio erogato (performance) vengono solitamente riassunte nella nozione di performability [7]. I due principali approcci alla previsione dei guasti quantitativa sono la costruzione di modelli e la loro soluzione (analitica o tramite simulazione), in cui il comportamento del sistema è riprodotto tramite un modello (tipicamente un modello stato-transizione), e la osservazione e la valutazione (anche tramite test specifici) sperimentale. Le attività di fault injection sono un esempio di valutazione sperimentale del sistema ai fini della fault forecasting, in quanto permette di esaminare l’evoluzione e le

conseguenze dei guasti in un sistema.

2.2 Artificial Intelligence

Nel pensiero comune quando si pensa all'intelligenza artificiale, si immagina un'entità in grado di pensare, che prima o poi ci sostituirà. Nella realtà l'intelligenza artificiale è quella branca dell'Informatica che dato un determinato problema, definisce degli agenti che trovano in modo efficace le migliori soluzioni. Quindi il campo di applicazione dell'intelligenza artificiale non è unico ma è suddiviso in sottodiscipline, dove si va 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.

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**. [6] 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, sente l'ambiente circostante, e attraverso altrettanti organi, come mani, gambe e bocca, interagisce con l'ambiente o altri agenti. (attuatori) Un agente fisico, come una macchina a guida autonoma, ha sensori come: telecamere, sensori a infrarossi, radar e tanti altri, mentre come attuatori: motori, freni, e tanti altri. Un'altra cosa che accomuna tutti gli agenti, sono le **percezioni**, termine usato per indicare gli input percettivi dell'agente in dato istante. [6] 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. [6]

2.2.2 Agenti che apprendono

Si definisce un agente sta imparando se migliora le proprie prestazioni nelle attività future dopo aver effettuato le osservazioni sul mondo. [6] E perchè deve imparare? Un agente deve imparare per tre motivi:

- 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 programmatori spesso non sanno come programmare loro la soluzione a un problema.

[6]

Detto ciò, come esistono agenti diversi, per problemi diversi, esistono apprendimenti diversi per ogni agente, che possono essere applicati alle varie componenti dell'agente. Una loro classificazione si basa su 4 fattori:

- **componente** dell'agente migliorata.
- **rappresentazione** usata per i dati e i componenti
- **prior knowledge** dell'agente presente.

- **feedback** disponibili da apprendere.

Esistono tre grosse tipologie di feedback che determinano i tre principali tipi di apprendimento:

unsupervised learning , dove un agente apprende patterns dall'input senza feedback espliciti.

reinforcement learning , dove un agente apprende da una serie di rinforzi, punizioni o premi.

supervised learning , dove ha un agente sono forniti coppie di valori input-output e apprende una funzione che mappi dall'input all'output. Nello specifico, il compito di un apprendimento supervisionato è quello che 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 una funzione sconosciuta $y=f(x)$, questa scopre una funzione h che approssima la funzione f . Il Learning problem si può definire in due maniere a seconda dell'output y : nel caso in cui y è uno di un insieme finito di valori il problema è definito **classificazione** (binaria o booleana, nel caso di due sole scelte), mentre se y è un numero si definisce come **regressione**

2.2.3 Neural Network

Una particolare forma di supervised learning, come ci suggerisce il titolo del seguente capitolo, sono le artificial neural network o più brevemente neural network. La **rete neurale** è composta da un insieme di nodi o unità connesse da link orientati.[6] La topologia e le proprietà dei nodi determina 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 hard o soft threshold. L'output dei neuroni finali sarà il risultato della task.

Collegamenti Un link tra una unità i e un unità j serve a propagare l'attivazione a_i da i a j . Oltretutto ogni collegamento ha anche un peso numerico $w_{i,j}$ associato, il quale determina la forza e il segno della connessione.

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 deriva l'output:

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

Nel caso in cui la funzione di attivazione è una hard threshold, l'unità è chiamato **perceptron**, nel caso in cui questa sia una funzione logica, si usa il termine **sigmoid perceptron**.

feed-forward network Definito il modello matematico di ogni neurone, questi si conetteranno insieme, a formare una rete. Esistono due principali tipologie: la **feed-forward network**, prevede connessioni solo in una 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 mantenere al suo interno, nessun tipo di stato se non i pesi dei collegamenti. Il **recurrent network**, prevede che i suoi neuroni usino il proprio output come feedback per il proprio input, così che i livelli di attivazione della rete formano un sistema dinamico che raggiunge uno stato stabile o esibiscono oscillazioni o addirittura comportamenti caotici. Questa caratteristica, permette alla rete di supportare un memoria a breve termine.

2.2.4 Convolutional Neural Network

Una rete neurale convuluzionale è una rete neurale di tipo feed-forward network dove la strato convuluzionale compare almeno una volta. L'input di una convolutional neurale network o ConvNet, è una immagine, ossia una matrice di valori di ogni singolo pixel occupante una precisa posizione all'interno dell'immagine. Nel caso di immagini RGB, questa sarà descritta da un terna 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** per la rete. Questa con le dimensioni dell'immagine, (height e width) compongono l'input della rete.

L'architettura Le reti neurali convoluzionali operano 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), risultato della convoluzione o di altre operazioni, è definita feature map o activation map, i parametri addestrabili sono tensori denominati filtri o kernel. Una CNN è composta da un insieme di strati che si susseguono, illustrati in dettaglio nei paragrafi seguenti. Gli strati principali sono:

- convolutional layer;

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

[8]

Convolutional layer La convoluzione è l'operazione fondamentale delle ConvNet. Essa dispone un filtro in ogni possibile posizione dell'immagine, coprendola interamente, e computa il prodotto scalare tra il filtro stesso e la matrice corrispondente del volume di input, avente eguale dimensioni. È possibile visualizzare la convoluzione come una sovrapposizione del kernel sull'immagine in input (o strato nascosto). [1] Un filtro è caratterizzato dai seguenti iperparametri (hyperparameters):

- altezza F_q
- larghezza F_q
- profondità d
- numero q

Solitamente i filtri hanno forma quadrata e profondità uguale a quella dello strato al quale sono applicati, nel caso di immagini RGB, 3. Il numero di possibili allineamenti tra filtro e immagine definisce altezza e larghezza della successiva feature map. Dal hidden layer q_1 , il filtro produce uno nuovo hidden layer q_2 $L_{q+1} = L_q - F_q + 1$ $B_{q+1} = B_q - F_q + 1$ Ad esempio, per immagini di dimensioni $32 * 32$, un filtro $5 * 5$ genera uno strato nascosto $28 * 28$. È opportuno fare distinzione tra profondità del filtro e profondità di strato nascosto/mappa di attivazione: la prima, d è la stessa dello strato al quale è applicato, la

seconda deriva invece dal numero di filtri applicati. Il numero di filtri è un iperparametro definito sulla base della capacità di distinguere forme sempre più complesse che si vuole conferire alla rete. I filtri sono dunque i componenti a cui saranno associate le caratteristiche dei pattern delle immagini. I filtri dei primi strati individuano forme primitive, quelli successivi imparano a distinguere forme sempre più grandi e complesse. Una proprietà della convoluzione è l'**equivarianza alla traslazione**: immagini traslate sono interpretate allo stesso modo e i valori delle mappa di attivazione traslano con i valori di input [8]. Ciò significa che forme particolari generano feature maps simili, indipendentemente dalla loro collocazione nell'immagine. Una delle proprietà della convoluzione è la seguente: Una convoluzione sullo strato q incrementa il campo recettivo di una feature dallo strato q allo strato $q + 1$. In altre parole, ogni valore della mappa di attivazione dello strato successivo cattura una regione spaziale più ampia del precedente. Feature maps degli strati catturano aspetti caratteristici di regioni via via maggiori e questo è il motivo per cui le CNN possono essere definite “profonde”: per studiare l'intera immagine sono necessarie lunghe successioni di “blocchi” di strati. In figura, un esempio dell'operazione di convoluzione.

Padding L'operazione di convoluzione comporta una contrazione dello strato q rispetto a $q + 1$ ed una conseguente perdita di informazioni. Il problema può essere arginato utilizzando il cosiddetto padding, una tecnica che prevede l'aggiunta di $\frac{(F_{meno1})}{2}$ pixel ai bordi delle mappe di attivazione per mantenere l'impronta spaziale. Ovviamente, per non alterare l'informazione, ai pixel sono assegnati valori nulli. Il risultato [1] è un incremento delle dimensioni (altezza e larghezza) del volume di input di $F - 1$, esattamente la quantità di cui è ridotto a seguito della convoluzione. Essendo il prodotto zero, le regioni esterne soggette a padding non contribuiscono al risultato finale del prodotto

scalare. Ciò che invece accade è permettere al filtro convoluzionale di scavalcare i bordi dello strato e computare il prodotto scalare solamente per le celle di valori diversi da 0. Questa tipologia di padding è definita “half-padding” in quanto circa metà del filtro oltrepassa i bordi, quando collocato alle estremità. L’half-padding è utilizzato per mantenere il “footprint” spaziale. Quando il padding non è utilizzato, si parla semplicemente di valid-padding e nella pratica non dà buoni risultati per il seguente motivo: mentre con l’half-padding le celle ai bordi contribuiscono all’informazione, nel caso di valid-padding, queste non vedono il passaggio del filtro e sono sotto-rappresentate. Un’altra forma di padding è il full-padding, con il quale si lascia che il filtro esuli completamente dal layer andando ad occupare celle di soli zeri. Così facendo si incrementa l’impronta spaziale dello strato, allo stesso modo in cui il valid-padding la riduce.[1] Strides Un filtro convoluzionale computa il prodotto scalare in ogni singola posizione dello strato di input ma è altresì possibile limitare la computazione ad un numero inferiore di posizioni, facendo uso dello stride S . La convoluzione è allora applicata alle posizioni $1, S + 1, 2S + 1$ etc., lungo entrambe le dimensioni. L’output avrà altezza e larghezza, rispettivamente: $L_{q+1} = L_q - F/S + 1$ $B_{q+1} = B_q - F/S + 1$ Ne consegue che lo stride comporta una riduzione delle dimensioni di un fattore di circa $1/S$, e dell’area di S alla seconda. Generalmente si usano valori limitati a 1 o 2 mentre con stride maggiori si punta alla riduzione della richiesta di memoria. Tramite lo stride è possibile catturare pattern complessi in vaste porzioni di immagine, con risultati simili a quelli prodotti dal max-pooling. In genere le dimensioni delle immagini in input sono ridotte a $L = B$, per evitare complicazioni nella definizione degli iperparametri. Per quanto concerne la convoluzione, il numero di filtri è solitamente potenza di 2 per facilitare la computazione, lo stride di 1 o 2, le dimensioni del filtro di 3 o 5. Filtri piccoli significano reti più profonde e più performanti. Ogni filtro convoluzionale è infine associato ad un bias: dato un filtro p , ed il layer q , bias è indicato con $b(p, q)$. Il

bias è un fattore di moltiplicazione della mappa di attivazione e la sua presenza incrementa il numero di parametri di un'unità. Il numero di features in ogni strato sarà dunque $1 + L \times B \times d$. Come tutti gli altri parametri, il valore del bias è definito tramite retropropagazione in fase di addestramento. [1]

Funzione di attivazione Relu L'operazione di attivazione non-lineare segue l'operazione di convoluzione. Per ogni strato $L_q \times B_q \times d$, la funzione di attivazione genera uno strato di eguale dimensione $L_q \times B_q \times d$ di valori limitati da soglie: in quanto semplice mappatura uno-a-uno dei valori di attivazione, la funzione ReLU non altera l'importa spaziale dello strato. L'attivazione avviene tramite funzioni matematiche. Mentre in passato la tangente iperbolica, la funzione sigmoide, softsign godevano di ampia diffusione, ora sono limitate a reti non-profonde e ormai rimpiazzate dalla funzione di attivazione ReLU (Rectified Linear Unit). Il motivo principale è che in reti neurali profonde, il gradiente di queste tre funzioni di attivazione si annulla durante la retropropagazione ed impedisce all'algoritmo di proseguire con l'addestramento. Inoltre, la funzione ReLU è computazionalmente molto più efficiente. ReLU è definita come segue:

ReLU è di fatto una funzione di rimozione di valori negativi rivelatasi efficace ed efficiente: la sua derivata in \mathbb{R}^+ è sempre 1 e non satura in \mathbb{R} ; in altre parole, il codominio della funzione è $[0, \infty)$. In corrispondenza dell'origine, la funzione non approssima l'identità e genera invece un gradiente elevato impedendone la scomparsa. [2]

Pooling layer Il max-pooling estrae il massimo valore contenuto in matrici $P \times P$ di ogni mappa di attivazione e produce un altro hidden layer di eguale profondità. Anche in questo caso, come per la convoluzione, si fa uso di stride: se lo stride è 1 lo strato 2 così generato avrà dimensioni: $L_2 = L_1 - P + 1$ $B_2 = B_1 - P + 1$ d_2

= d1. Per stride maggiori di 1, come solitamente si usa, altezza e larghezza saranno, rispettivamente: $L2 = L1 - P/S + 1$ $B2 = B1 - P/S + 1$ Rispetto alla convoluzione, il pooling è effettuato al livello di ciascuna mappa di attivazione, quindi il loro numero rimane alterato e l'output è uno strato della stessa profondità (ma di altezza e larghezza differenti). Una configurazione tipica è dimensione 2x2 e stride 2: così facendo non c'è sovrapposizione tra regioni.

L'uso dello stride nel pooling 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. Si osservi che per ridurre l'impronta spaziale possono essere utilizzati unicamente strati convoluzionali con stride maggiori di 1. Nonostante ciò, si preferisce tuttora utilizzare max-pooling o qualche altra variante dato il grado di non-linearità e invarianza alla traslazione che introducono.[1]

dense layer Connette ogni feature in input con la corrispondente feature 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. [1]

3 Costruzione del dataset

Siccome il detector impiega la CNN, come strumento per l'identificazione di guasti nel sensore della telecamera del veicolo, è necessario istruire il sistema alla loro identificazione. Per fare questo, è necessario predisporre un dataset su cui istruire il sistema. Per costruire questo, si necessita l'acquisizione di immagini simili a quelle ottenibili da una telecamera posto su una vettura. Nel caso della seguente tesi, il clas-

sificatore è di tipo binario, quindi classifica le immagini tra pulite o sporche, quindi a sua volta il dataset sarà suddiviso in due classi o label.

Dataset

CARLA CARLA è un software open source di simulazione per la ricerca di guida autonoma. Il simulatore CARLA è costituito da un'architettura client-server scalabile. Il server è responsabile di tutto ciò che riguarda la simulazione stessa: rendering del sensore, calcolo della fisica, aggiornamenti sullo stato del mondo e sui suoi attori e molto altro. Poiché mira a risultati realistici, la soluzione migliore sarebbe eseguire il server con una GPU dedicata, soprattutto quando si ha a che fare con l'apprendimento automatico. Il lato client consiste in una somma di moduli client che controllano la logica degli attori sulla scena e impostano le condizioni del mondo. Come predetto il detector ivi esposto impiega come sistema classificatore una rete neurale convulzionale, ed essendo questo una forma di apprendimento supervisionato, richiede una fase di training prima del suo utilizzo.

Acquisizione La scelta di utilizzare una soluzione software per la simulazione di una

Detto ciò, la prima cosa che è necessario predisporre è il dataset di dati, da fornirgli. Nel caso di questa tesi, i detector sono implementati attraverso singoli classificatori binari, dove ciascuno è stato allenato ad apprendere singoli casi di malfunzionamento. Come spiegato precedentemente, le reti neurali prevedono due diversi dataset: il training set e il validation set. Entrambi suddivisi in golden run e sporcate.

Quindi il dataset avrà la seguente forma, generale a tutti i malfunzionamenti presi in considerazione:

- Training set

- Originale
 - Sporcata
- Validation set
 - Originale
 - Sporcata

Dovendo ottenere un simil risultato, si è proceduto con le due seguenti fasi:

- Acquisizione;
- Sporcatatura;

Acquisizione La prima fase del processo di costruzione del dataset prevede l'acquisizione di immagini pulite dal simulatore CARLA. Attraverso il seguente software github, è stato possibile acquisire le immagini senza errori. Sono state avviate 500 simulazioni diverse, dove da ciascuna sono state acquisite 300 immagini .png nella forma (800*600)

Sporcatatura La seconda fase del processo di costruzione, prevede la "sporcatatura" delle immagini salvate nella prima fase, immagazzinandole in maniera tale da essere poi usate dalla rete neurale. Per "sporcatatura" di una immagine, è inteso l'applicazione di un effetto all'immagine che simuli un guasto al sensore ottico della vettura. Per fare questo è stato usato il progetto github "progetto secci".

4 Costruzione del detector

Per la costruzione del detector, è stata implementata una rete neurale convuluzionale.

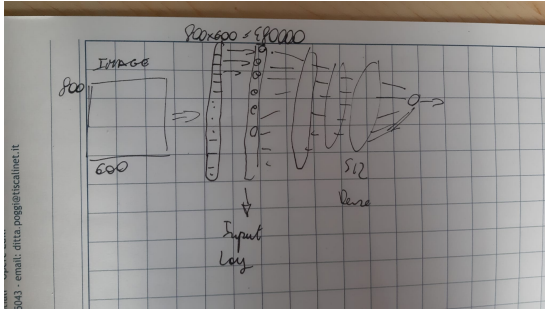


Figure 1: CNN detector

Come si nota dalla figura 1, questa è composta da un insieme di layer. Il primo layer ha un è un conv2D layer, di ben $800 \times 600 \times 3 = 1440000$ neuroni. Le dimensioni di questo primo sono date dalla necessità di poter elaborare un immagine per intero, e infatti $(800, 600)$ sono le dimensioni di larghezza e altezza delle immagini, e quindi il numero di pixel che compongono l'immagine sono $800 \times 600 = 480000$, l'ultimo fattore è dato dalla composizione interna di un pixel. Questo rappresenta il numero di canali dei colori, ed essendo immagini rgb (red, green, blue) sono 3. Definito l'input del primo layer e quindi dell'intera rete, si può passare all'output.

4.1 Addestramento del rete convuluzionale

5 Esecuzione e risultati

6 Conclusioni e lavori futuri

7 A Manuale utente

References

- [1] Charu C Aggarwal et al. *Neural networks and deep learning*. Springer, 2018.
- [2] Hamed Habibi Aghdam and Elnaz Jahani Heravi. “Guide to convolutional neural networks”. In: *New York, NY: Springer* 10 (2017), pp. 978–973.
- [3] Algirdas Avizienis et al. “Basic concepts and taxonomy of dependable and secure computing”. In: *IEEE transactions on dependable and secure computing* 1.1 (2004), pp. 11–33.
- [4] Andrea Bondavalli. *L’analisi quantitativa dei sistemi critici: Fondamenti e Tecniche per la Valutazione-Analitica e Sperimentale-di Infrastrutture Critiche e Sistemi Affidabili*. Società Editrice Esculapio, 2011.
- [5] David M Nicol, William H Sanders, and Kishor S Trivedi. “Model-based evaluation: from dependability to security”. In: *IEEE Transactions on dependable and secure computing* 1.1 (2004), pp. 48–65.
- [6] Stuart J Russell and Peter Norvig. *Intelligenza artificiale. Un approccio moderno*. Vol. 1. Pearson Italia Spa, 2005.
- [7] RM Smith, Kishor S. Trivedi, and AV Ramesh. “Performability analysis: measures, an algorithm, and a case study”. In: *IEEE Transactions on Computers* 37.4 (1988), pp. 406–417.
- [8] Luca Torresin. “Sviluppo ed applicazione di reti neurali per segmentazione semantica a supporto della navigazione di rover marziani”. In: (2019).