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

Tesi di Laurea

ATTACCHI VERSO SISTEMI DI APPRENDIMENTO IN AMBITO AUTONOMOUS DRIVING: STUDIO E IMPLEMENTAZIONE IN AMBIENTI SIMULATI

ADVERSARIAL ATTACKS TOWARDS MACHINE LEARNING SYSTEMS IN AUTONOMOUS DRIVING:STUDY AND IMPLEMENTATION IN A SIMULATED ENVIRONMENT

NICCOLÓ PIAZZESI

Relatore: *Andrea Ceccarelli*

Anno Accademico 2019-2020

INDICE

ELENCO DELLE FIGURE

ELENCO DELLE TABELLE

"Inserire citazione"
— Inserire autore citazione

INTRODUZIONE

Il mondo odierno è ormai pervaso dall'Intelligenza Artificiale. Uno dei settori di maggior interesse per la ricerca in questo ambito è indubbiamente quello delle cosidette *Self Driving Car*, ovvero le macchine a guida autonoma. Grandi aziende quali *Google* e *Tesla* hanno già sviluppato dei propri modelli(Figura 1 e 2) e l'interesse per questo tipo di veicoli è sempre in maggior crescita. Per poter funzionare correttamente questi veicoli acquisiscono informazioni dall'ambiente circostante sotto forma di immagini. Queste immagini vengono classificate da un sistema interno che, in base alle informazioni ricevute, decide l'azione da compiere(sterzare, accelerare, frenare ecc.). Anni ed anni di sviluppo e ricerca hanno reso sempre più affidabili e sicuri questi sistemi ma restano comunque presenti delle vulnerabilità. Una vulnerabilità molto importante sono i cosiddetti *Adversarial attacks*. Il meccanismo di questi attacchi è molto semplice:
alle immagini raccolte dal sistema viene applicata una modifica impercettibile a occhio umano ma in grado di causare un errore di classificazione che può portare, ad esempio, una macchina ad accelerare quando dovrebbe frenare. La ricerca su questi tipi di attacchi quindi è fondamentale per garantire l'affidabilità dei veicoli a guida autonoma. Lo scopo di questa tesi è lo studio e l'implementazione di *Adversarial Attacks* contro modelli di guida autonoma in ambiente simulato. Il lavoro è cosi suddiviso:

- **Capitolo 1:** Fondamenti teorici alla base dei sistemi intelligenti
- **Capitolo 2:** Fondamenti di guida autonoma

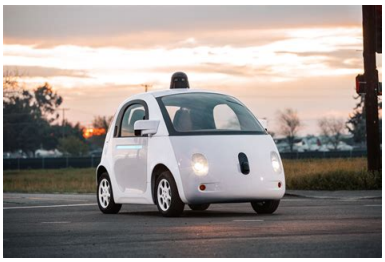

Figura 1: self driving car di Google

Figura 2: self driving car di Tesla

- **Capitolo 3:** Dependability e security
- **Capitolo 4:** Strumenti utilizzati: in particolare vengono presentati *l'Adversarial Robustness Toolbox* e il simulatore *Carla*
- **Capitolo 5:** Il lavoro svolto:attacchi scelti e motivazioni, implementazione e risultati
- **Capitolo 6:** Conclusioni e possibili sviluppi futuri

1

SISTEMI INTELLIGENTI

1.1 INTELLIGENZA ARTIFICIALE

Dare un'unica definizione di intelligenza artificiale risulta estremamente difficile a causa della vastità e dell'interdisciplinarietà dell'argomento. Soltanto nella figura 3 ne troviamo addirittura otto , tutte valide, ma forse la descrizione migliore per i nostri scopi è quella data dal padre di questa disciplina, John Mccarthy: " L'Intelligenza Artificiale è la scienza volta alla creazione di macchine *intelligenti*, più nello specifico di *programmi intelligenti*." [2].

1.1.1 *Agenti e ambienti*

Il concetto di macchina intelligente può essere ulteriormente astratto dall'idea di **agente computazionale intelligente**. Esaminiamo adesso questa definizone. Un **agente** è un'entità che compie azioni e percepisce informazioni da un ambiente.

Un'agente si comporta in modo **intelligente** quando:

``The exciting new effort to make computers think ... *machines with minds*, in the full and literal sense" (Haugeland, 1985)  ``The automation of activities that we associate with human thinking, activities such as decision-making, problem solving, learning ..." (Bellman, 1978)	``The study of mental faculties through the use of computational models" (Charniak and McDermott, 1985)  ``The study of the computations that make it possible to perceive, reason, and act" (Winston, 1992)
``The art of creating machines that perform functions that require intelligence when performed by people" (Kurzweil, 1990)  ``The study of how to make computers do things at which, at the moment, people are better" (Rich and Knight, 1991)	``A field of study that seeks to explain and emulate intelligent behavior in terms of computational processes" (Schalkoff, 1990)  ``The branch of computer science that is concerned with the automation of intelligent behavior" (Luger and Stubblefield, 1993)

Figura 3: Possibili definizioni di Intelligenza artificiale[1]

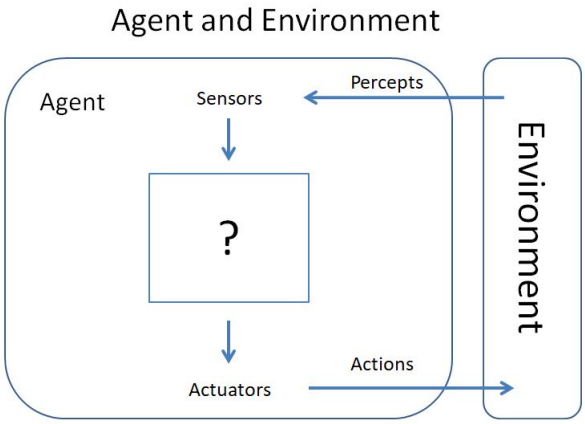

Figura 4: Agente computazionale[1]

- le sue azioni sono appropriate alle circostanze e ai suoi scopi
- È flessibile per ambienti e scopi dinamici
- impara dall'esperienza
- fa le scelte **giuste** date le sue limitazioni percettive e computazionali. Un agente tipicamente non può osservare l'ambiente direttamente; ha una memoria e un tempo per agire molto limitati.

Un agente **computazionale** è un agente le cui decisioni possono essere descritte in termini di una computazione. Questo significa che, una decisione può essere scomposta in una serie di operazioni primitive implementabili in un dispositivo fisico.[3].

1.1.2 *Intelligenza e performance*

Abbiamo detto che un agente è intelligente quando compie le *scelte giuste*. Ma cosa significa fare la scelta giusta? Rispondiamo a questa domanda in un modo molto banale: considerando le *consequenze* del comportamento di un agente. Un agente si muove all'interno di un ambiente e compie una sequenza di azioni in base alle informazioni che riceve. Queste azioni causano variazioni allo stato dell'ambiente. Se la variazioni è desiderabile, l'agente ha fatto le scelte giuste. La nozione di desiderabilità è catturata da una **misura di prestazione(performance measure)** che valuta una qualunque sequenza di azioni. Ovviamente

PEAS description of the task environment for an automated taxi

Agent Type	Performance Measure	Environment	Actuators	Sensors
Taxi Driver	safe, fast, legal, comfortable, maximum profit, getting to correct destination	roads, other traffic pedestrians customers	steering, accelarator, brake, signal, horn, display	cameras, sonar, speedometer, GPS, odometer, accelerometer, engine sensors, keyboard

Figura 5: descrizione PEAS dell'ambiente di lavoro di un taxi automatico

non esiste una performance measure globale; solitamente viene costruita appositamente per uno specifico problema. Le specifiche dell'agente, dell'ambiente e della misura di prestazioni vengono raggruppate nell'**ambiente di lavoro(task environment)**, attraverso la descrizione PEAS(Performance,Environment,Actuators,Sensors).[1] La figura 5 riassume un esempio di descrizione PEAS dell'ambiente di lavoro di un taxi.

1.1.3 *Razionalità*

Ciò che abbiamo descritto fin'ora ci permette di verificare in modo quantitativo l'intelligenza di un agente. Espandiamo questa idea introducendo la nozione di **razionalità**. Cosa sia razionale in qualsiasi momento dipende da quattro elementi:

- la misura di prestazioni
- la conoscenza a priori dell'agente

- le azioni che l'agente può compiere
- la sequenza di percezioni fino a quel momento

Questo ci porta alla definizione di **Agente Razionale**:

per ciascuna sequenza di percezioni, un agente razionale seleziona un'azione che massimizza il valore atteso della performance measure, data l'evidenza fornita dalla sequenza di percezioni e dalla conoscenza a priori dell'agente

La razionalità non significa **onniscienza**. Un agente deve essere in grado di prendere decisioni anche quando non ha a disposizione tutte le informazioni necessarie a compiere l'azione perfetta. Un agente razionale massimizza *il valore atteso* della performance basandosi sulla sua conoscenza pregressa dell'ambiente, ma non sempre un ambiente è completamente osservabile. Per questo motivo sono fondamentali la fase di raccolta delle informazioni(**information gathering**) e di apprendimento(**learning**). Per poter migliorare le prestazioni, un agente deve essere in grado di raccogliere la massima quantità possibile di informazioni e di aggiungere tali informazioni alla propria base di conoscenza. Questo permette una scelta sempre migliore delle azioni da compiere.[1]

1.1.4 *Scopo dell'IA*

Lo scopo scientifico dell'IA è quello di studiare i principi e i meccanismi che rendono il comportamento intelligente possibile sia nei sistemi naturali che in quelli artificiali. Lo scopo ingegneristico di questa disciplina è la costruzione di dispositivi fisici in grado di comportarsi in modo intelligente. Da queste basi si sono sviluppati un innumerevole quantità di branche, tra cui le più importanti sono:[2]

- **logical AI**
- **search**
- **knowledge and reasoning**
- **planning**
- **learning from experience**
- **genetic programming**

In figura 6 vengono riportate alcune delle applicazioni di maggior successo dei *Sistemi Intelligenti*. Il settore delle Self driving Car si basa in particolare su **Machine Learning** applicato alla **Computer Vision**.

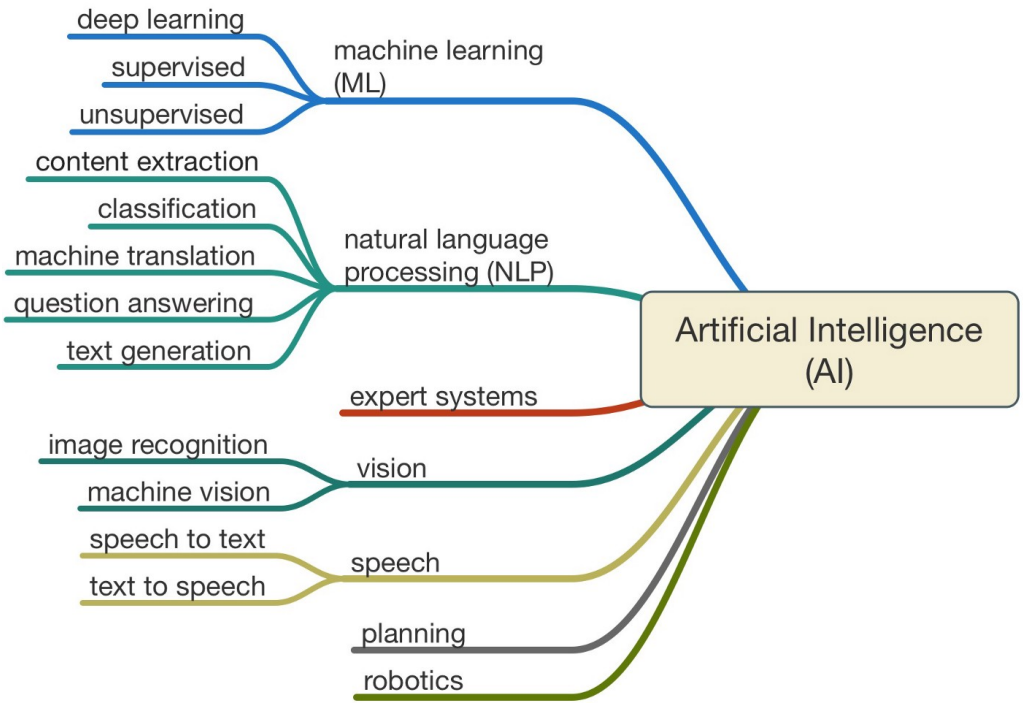

Figura 6: applicazioni dell'IA[4]

1.2 MACHINE LEARNING

Il **Machine Learning(ML)** è una branca dell'intelligenza artificiale dedicata allo studio e allo sviluppo di algoritmi che migliorano le prestazioni attraverso l'esperienza[5]. L'abilità di un algoritmo di migliorare dall'esperienza automaticamente è fondamentale, perchè è impossibile scrivere programmi che sappiano a priori tutte le possibili situazioni in cui un agente potrebbe ritrovarsi. Inoltre l'ambiente in cui l'agente è inserito può mutare nel tempo[1]. Consideriamo il caso dei mercati finanziari. Un programma creato per prevedere i prezzi delle azioni di domani deve essere in grado di adattarsi alle variazioni improvvise causate ad esempio da una crisi globale.

1.2.1 *Problemi di apprendimento ben definiti*

Iniziamo ad approfondire i concetti di apprendimento automatico considerando alcuni task di apprendimento. Più precisamente:

Definizione 1.1. Si dice che un programma **impara** dall'esperienza E rispetto a una classe di tasks T e una misura di prestazioni P, se la sua

prestazione nei tasks in T, misurata da P, migliora con l'esperienza E

Dalla definizione 1.1 possiamo specificare diversi problemi di apprendimento, ad esempio[6]:

- **Task T**: giocare a scacchi
- **Misura di prestazioni P**: percentuale di partite vinte contro gli avversari
- **Esperienza E**: giocare partite di prova contro se stesso

Ogni problema di learning consiste nel prendere la conoscenza a priori e i dati ricevuti(l'esperienza E) e trasformarli in una rappresentazione interna utilizzata da un agente per prendere decisioni. La rappresentazione può corrispondere con i dati stessi ricevuti ma di solito è una sintesi compatta e significativa. In definitiva, per specificare una determinata tecnica di apprendimento è quindi necessario affrontare le seguenti[3] questioni:

- **Task** Un task di apprendimento è una qualsiasi attività che può essere appresa da un agente
- **Feedback** Durante l'apprendimento a un agente viene fornito un riscontro in base alla correttezza delle azioni svolte. Il riscontro può essere un premio o una punizione. In base ad esso un agente modifica le proprie azioni migliorando coì la propria esecuzione su un determinato task.
- **Rappresentazione** Come detto in precedenza l'esperienza deve influenzare la rappresentazione interna di un agente. Gran parte del Machine Learning è focalizzato nel contesto di una specifica rappresentazione(es. reti neurali)
- **Online e Offline** Nel learning offline, tutti i training examples sono disponibili prima dell'azione di un agente. Nel learning online, gli esempi vengono ricevuti durante l'esecuzione.
- **Misura di Successo** Per sapere se un agente ha effettivamente imparato, è necessaria una misura di successo. La misura NON riguarda le prestazioni sui dati di addestramento, bensì le prestazioni su nuove esperienze.
- **Bias** Con il termine *bias* si intende la tendenza a preferire un'ipotesi rispetto a un'altra, concetto fondamentale nel processo di scelta

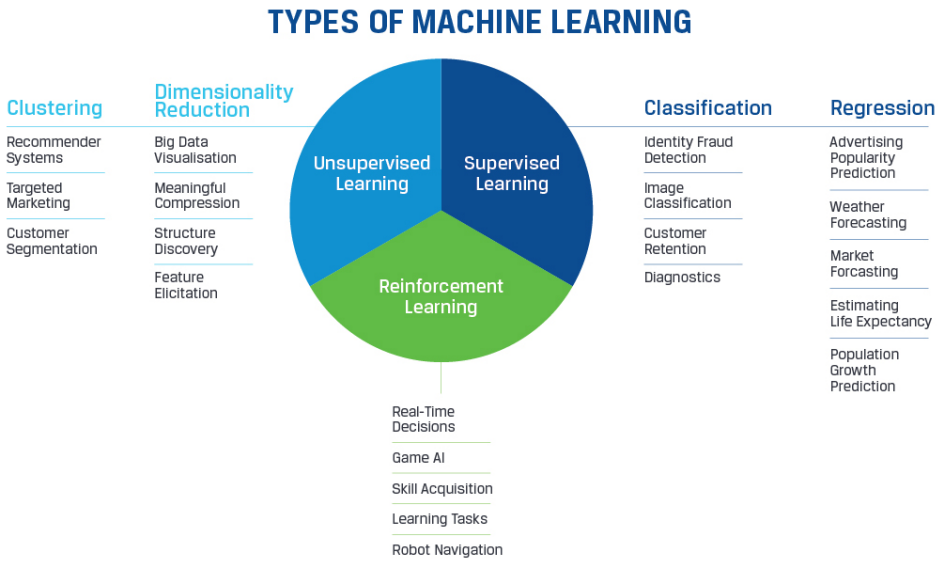

Figura 7: tre maggiori tipi di learning[7]

- **Noise** Una delle proprietà più importanti per un algoritmo di apprendimento è la capacità di gestione di dati condizionati. Infatti nella pratica i dati possono spesso risultare imperfetti o in alcuni casi incompleti .
- **Interpolazione e Estrapolazione** L'interpolazione riguarda le previsioni tra i casi per i quali si possiedono dei dati(ad esempio approssimare una funzione dati alcuni valori di essa), l'estrapolazione comporta invece previsioni che vanno oltre i dati conosciuti.

1.2.2 *Forme di Learning*

Gli algoritmi di Machine Learning sono suddivisi in diverse classi, a seconda del risultato desiderato. In figura 7 vengono mostrate le tre maggiori classi. Nel contesto di questa tesi, siamo interessati al **Supervised Learning**, in particolare ad algoritmi di **Deep Supervised Learning** basati su **Reti Neurali Convoluzionali**.

1.2.3 *Deep Supervised Learning*

La forma più comune di Machine Learning è il supervised learning. Consideriamo il caso in cui si voglia addestrare un classificatore che

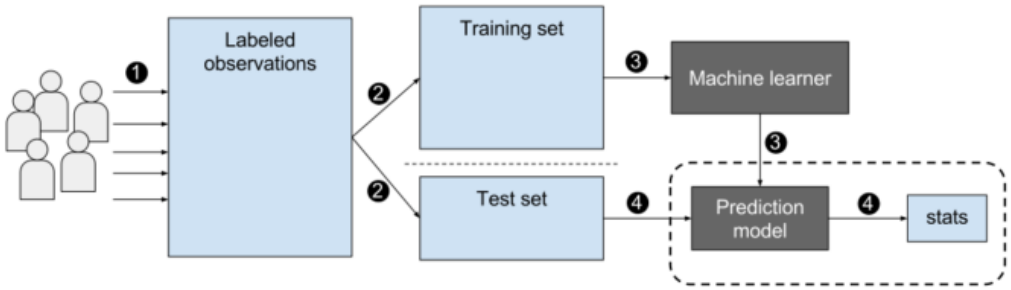

Figura 8: Supervised Learning[8]

sappia riconoscere i diversi tipi di segnali stradali. Si colleziona un dataset composto da immagini di segnali stradali, ciascuno di esso con la rispettiva categoria(stop, limite di velocità divieto di sosta ecc.). Queste immagini compongono il **training set**. Durante l'apprendimento, vengono mostrate alla macchina le immagini del training set. Per ciascuna immagine l'algoritmo produce un output della forma di un vettore di punteggi, uno per ogni categoria. Si calcola una funzione che misura l'errore(**loss function**) fra l'output prodotto e l'output desiderato. In base all'errore commesso l'algoritmo modifica dei paramentri interni, detti pesi, in modo da ridurlo. Per aggiustare il vettore dei pesi in modo corretto , l'algoritmo di apprendimento utilizza una tecnica detta *Gradient Descent*. Ad ogni errore commesso, il vettore dei pesi viene aggiustato nella direzione di massima decrescita dell'errore, fino a raggiungere un minimo locale. La fase di addestramento termina quando si raggiunge il minimo errore possibile, calcolato come la media della funzione di errore su ogni immagine del dataset. Terminato il training, la prestazione del classificatore viene misurata su un differente dataset di immagini, detto **test set**. Questo serve a verificare la capacità di generalizzazione della macchina, andando a controllare la correttezza degli output su input sconosciuti.

Una buona fetta delle applicazioni pratiche utilizza classificatori lineari su caratteristiche modellate a mano. Un classificatore binario calcola una somma pesata delle componenti del cosidetto **feature vector**. Se la somma è sopra una certa soglia, l'input è classificato in una certa classe. I classificatori lineari suddividono lo spazio degli input in semplici sottoregioni dette iperpiani, ma contesti come il riconoscimento di immagini o il riconoscimento vocale hanno bisogno di modelli più complessi, capaci di rilevare minuscole variazioni su caratteristiche importanti(**selettività**) e di ignorare anche grandi variazioni su caratteristiche irrilevanti al con-

testo, come ad esempio una rotazione dell'immagine(**invarianza**). La scelta delle caratteristiche rilevanti è quindi una fase fondamentale del processo di learning. Tradizionalmente, la scelta delle features veniva costruita direttamente dagli sviluppatori, il che comportava una grande richiesta di abilità ingegneristica e di competenza nel dominio di lavoro. Grazie al Deep Learning il processo di estrazioni di features rilevanti viene automatizzato con una procedura di apprendimento a scopo generale. Un'architettura di deep learning è formata da uno stack multilivello di moduli elementari, tutti(o la maggior parte) sottoposti a learning, e la maggior parte dei quali calcola una mappatura non lineare tra input-output. Ciascun livello aumenta sia la selettività che l'invarianza della rappresentazione. Maggiore è la profondità dello stack, maggiore è la complessità della funzione di scelta, capace di rilevare i più piccoli cambiamenti a dettagli rilevanti[9].

1.2.4 *Deep FeedForward Neural Networks*

Le reti neurali feedforward sono uno dei modelli computazionali di maggior successo nell'ambito del deep learning. Lo sviluppo in questo campo è stato influenzato dallo studio sulla struttura del cervello umano e sulle modalità di trasmissioni di informazioni tra neuroni. L'obiettivo di una rete feedforward è l'approssimazione di una funzione $f*$ che, nel caso di un classificatore mappa un vettore di feature **x** a una categoria **y**. Una rete feedforward definisce una funzione $y = f(x;\theta)$ e apprende il valore di θ che permette la miglior approssimazione possibile. Questo tipo di reti viene detto **feedforward** perchè i dati passano dall'input x, alle computazioni che definiscono f, e infine all'output y. Non vi sono cicli nei quali risultati di calcoli interni sono reinseriti all'interno della rete. Quando le reti includono anche cicli vengono dette **reti ricorrenti**. Uno schema di una rete feedforward è presentato in figura 9. I livelli intermedi sono detti **hidden(nascosti)** perchè nel **training set** non vengono esplicitati gli output desiderati per ciascuno di questi livelli. Ogni livello nascosto rappresenta una funzione e la funzione totale è data dalla composizione di queste funzioni. Per esempio, se abbiamo 3 livelli la funzione completa sarà $f(x) = f^3(f^2(f^1(x)))$. Il singolo livello è composto da molte **unità** che agiscono in parallelo, ciascuna rappresentante una funzione da vettore a scalare[10].

La generica i-esima unità riceve un vettore di input dalle unità del livello precedente, indicati con X_j. X_j viene trasmesso all'unità opportu-

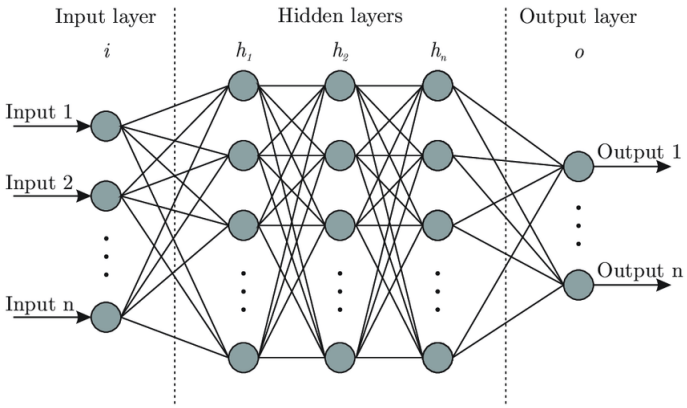

Figura 9: schema base di una rete neurale[11]

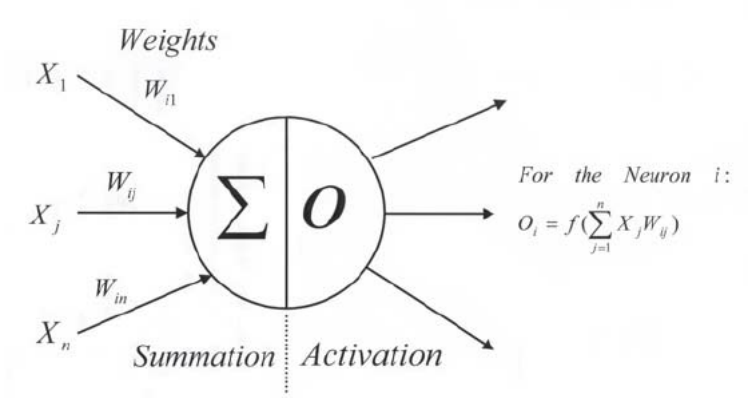

Figura 10: un "neurone" artificiale [13]

namente moltiplicato da un peso W_{ij}. Gli input ricevuti dall'i-esima unità costituiscono lo stato di attivazione, rappresentato dalla sommatoria

$$\sum_{j} W_{ij} X_j$$

. L'output prodotto dall'unità viene calcolato attraverso la funzione f, detta funzione di attivazione. Il compito della funzione di attivazione è quello di controllare se lo stato di attivazione supera una certa soglia. Se questo avviene l'unità trasmette alle unità del livello successivo[12].

1.2.5 *Learning nei neural network*

Il learning di una rete neurale avviene solitamente(nel caso dell'object recognition) in modalità supervisionata. Si vuole determinare un vettore

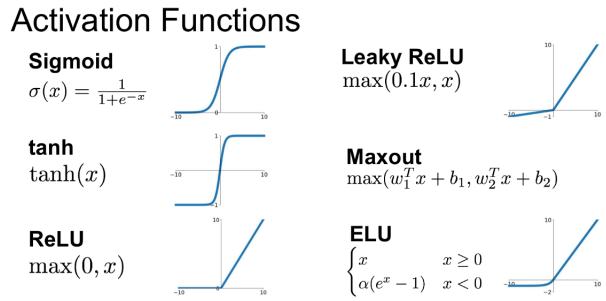

Figura 11: Alcune funzioni di attivazione[14]

dei pesi $\vec{w}$ che permetta alla rete di produrre l'output corretto per ciasun esempio nel training set. L'idea di base è quella di usare il gradient descent ma in una rete multilivello il problema è ulteriormente complicato dalla presenza degli **hidden layers**. Mentre nell'output layer l'errore che si commette in ciascuna fase di addestramento è esplicito, l'errore presente nei livelli nascosti non è immediato da calcolare, in quanto nel training set non sono presenti i valori che i nodi nascosti devono assumere. Il problema viene risolto trasmettendo a ritroso l'errore commesso dall'output layer ai livelli nascosti, con un algoritmo detto di **back-propagation**.

procedure TRAIN
  $X \leftarrow$ Training Data Set of size mxn
  $y \leftarrow$ Labels for records in X
  $w \leftarrow$ The weights for respective layers
  $l \leftarrow$ The number of layers in the neural network, 1...L
  $D_{ij}^{(l)} \leftarrow$ The error for all l,i,j
  $t_{ij}^{(l)} \leftarrow 0$. For all l,i,j
  For $i = 1$ to m
    $a^l \leftarrow feedforward(x^{(i)}, w)$
    $d^l \leftarrow a(L) - y(i)$
    $t_{ij}^{(l)} \leftarrow t_{ij}^{(l)} + a_j^{(l)} \cdot t_i^{l+1}$
  if $j \neq 0$ **then**
    $D_{ij}^{(l)} \leftarrow \frac{1}{m} t_{ij}^{(l)} + \lambda w_{ij}^{(l)}$
  else
    $D_{ij}^{(l)} \leftarrow \frac{1}{m} t_{ij}^{(l)}$
    where $\frac{\partial}{\partial w_{ij}^{(l)}} J(w) = D_{ij}^{(l)}$

Figura 12: pseudocodice della back-propagation[15]

1.2.6 *Reti Convoluzionali*

Nel contesto dell'object-detection le reti più utilizzate sono le reti convoluzionali. Le reti convoluzionali sono progettate per processare dati

in forma matriciale, come immagini e audio. La loro architettura è strutturata in una serie di fasi. Le prime fasi sono composte da due tipi di layer: **convolutional layers** e **pooling layers.** Le unità in un layer convoluzionale sono organizzate in feature maps,addette a riconoscere alcune caratteristiche di un input. Ciascuna unità è collegata a un sottoinsieme di unità delle feature maps nei precedenti livelli attraverso un gruppo di pesi detto **filter bank** e utilizza una funzione di attivazione non lineare come una ReLU. Tutte le unità di una feature map condividono la stessa filter bank, e diverse feature maps usano diverse filter bank. L'aggregazione di unità in feature maps è motivata dalla struttura dell'input. Nei dati in forma matriciale, gruppi di valori locali sono spesso fortemente correlati, formando pattern locali che vengono facilmente riconosciuti. La condivisione di pesi tra unità diverse è spiegata notando come un pattern possa apparire in qualsiasi parte dell'immagine, e l'utilizzo di una filter bank garantisce invarianza alla località. Passando ai pooling layer essi hanno il compito di "fondere" features semanticamente simili,tipicamente calcolando il massimo tra un gruppo di unità di una feature map. Questo causa una riduzione nelle dimensioni dell'input che viene poi passato ai layer successivi, La riduzione delle dimensione è fondamentale per garantire efficienza computazionale e per aumentare l'invarianza alle piccole perturbazioni. Due o più fasi di convoluzione e pooling sono sono seguite da altri layer convoluzionali e completamente connessi. Le filter banks vengono addestrate con algoritmi di backpropagation.[9]

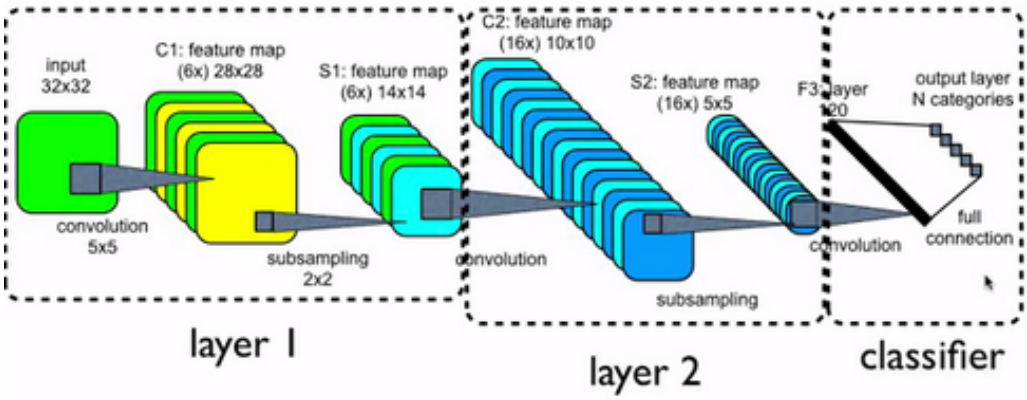

Figura 13: rete neurale convoluzionale

1.2.7 *Adversarial Examples*

Nonostante gli ottimi risultati raggiunti, le reti neurali sono state dimostrate essere vulnerabili ai cosiddetti *adversarial examples*. Gli adversarial examples sono input intenzionalmente modificati che risultano essere molto simili agli input originali per gli esseri umani, ma che causano decisioni scorrette da parte dei modelli. La vulnerabilità delle reti a questo tipo di attacchi rappresenta un grande rischio di sicurezza nelle applicazioni pratiche come i veicoli a guida autonoma. Studiarli è quindi necessario per identificare le limitazioni degli algoritmi di learning attuali, dando così spunti per migliorare la robustezza dei modelli.

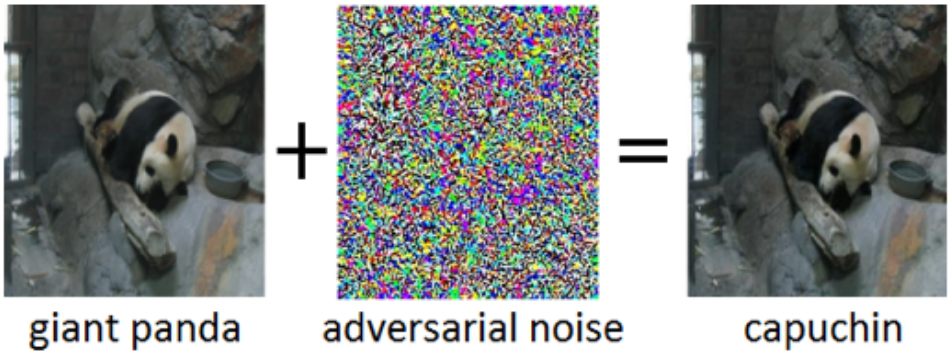

Figura 14: Adversarial example

2

FONDAMENTI DI GUIDA AUTONOMA

Lo sviluppo tecnologico nel campo dell'intelligenza artificiale ha permesso una grande evoluzione nel campo della guida autonoma. L'interesse per questo settore è guidato dai seguenti vantaggi che questi veicoli possono potenzialmente portare[16]:

- **Riduzione degli incidenti** La maggior parte degli incidenti stradali(fino al 90%)sono dovuti a errori umani. Eliminarli significherebbe ridurre notevolmente il numero di infortuni e salvare delle vite.
- **Riduzione del traffico** Grazie al controllo automatico si possono evitare le situazioni di congestione che avvengono naturalmente sulle strade. Oltre a migliorare il deflusso veicolare, questo porterebbe anche una riduzione delle emissioni di CO_2 e del consumo di benzina, causate dalle macchine ferme nel traffico.
- **Migliore capacità stradale** L'ottimizzazione del trafico potrebbe portare un notevole incremento nella capacità autostradale. La capacità di monitorare in modo costante l'ambiente circostante e reagire istantaneamente renderebbe più sicuro viaggiare a velocità maggiori e con meno spazio tra ciascun veicolo.
- **Accessibilità** Attualmente molte persone anziane e/o con disabilità sono escluse dal viaggio in automobile in solitaria. Le self-driving car rappresentano un'opportunità di indipendenza per queste categorie, che potrebbero così muoversi senza dover ricorrere a soluzioni esterne.

Attualmente però non si puo parlare ancora di guida completamente autonoma, ma di sistemi di guida assistita. Restano ancora molte problematiche aperte, soprattutto da un punto di vista legale[18] ed etico[19]. Per questo, e per i problemi di *dependability* illustrati in seguito, lo sviluppo di veicoli autonomi è in molti casi ancora in fase prototipale. La ricerca è quindi fondamentale per portare a miglioramenti significativi.

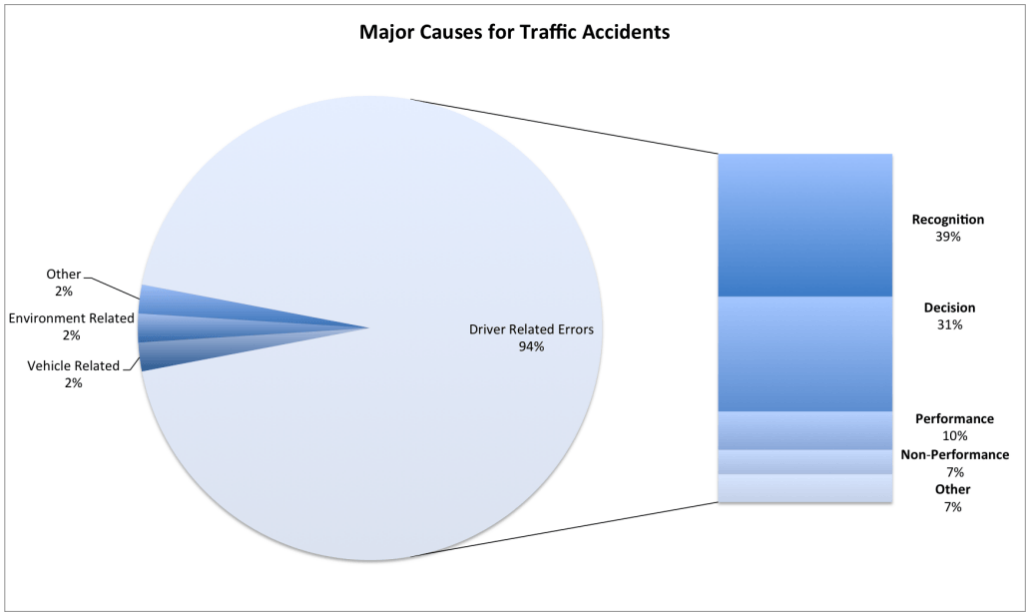

Figura 15: Maggiori cause di incidenti stradali ad Austin tra il 2005 e il 2007[17]

2.1 LIVELLI DI AUTOMAZIONE

Nel 2014 la **SAE international** ha pubblicato lo standard J3016, col quale vengono definiti sei liveli di autonomia, in base a quanto un guidatore deve intervenire.

Livello 0

Nessun tipo di automazione. Il controllo è completamente affidato al conducente.

Livello 1: Assistenza alla guida

Il conducente si occupa di tutti gli aspetti della guida. Viene supportato da sistemi elettronici che possono segnalare la presenza di pericoli attraverso segnali visivi e acustici. Vengono classificati di livello 1 anche i veicoli con una singola funzione di assistenza automatizzata(ad esempio il controllo automatico della velocità).

Livello 2: Automazione Parziale

Il conducente deve mantenere il controllo del veicolo in ogni istante ma le varie funzioni vengono automatizzate da due o più sistemi avanzati di

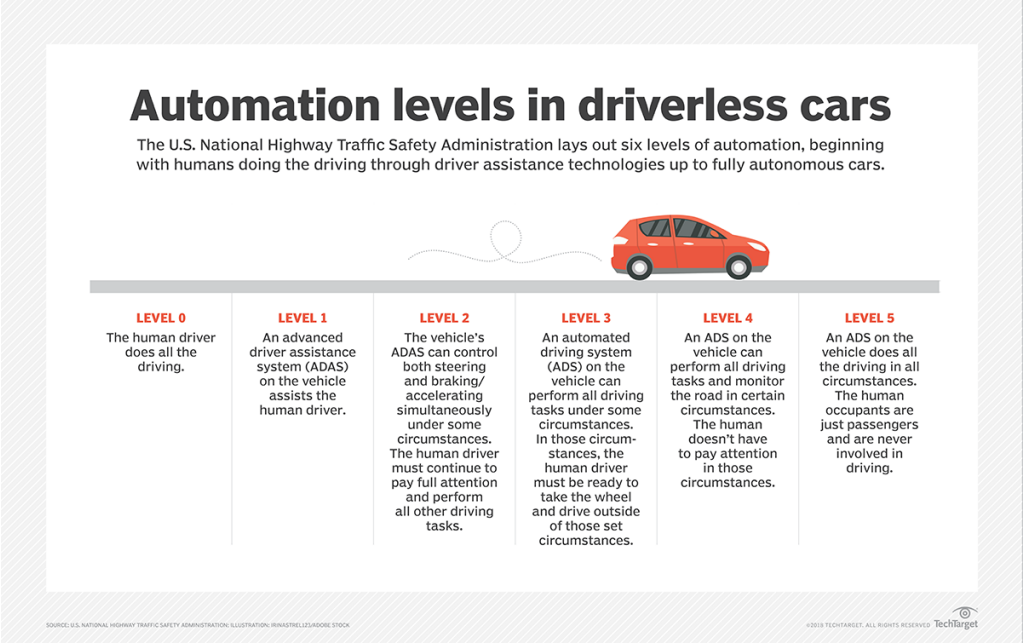

Figura 16: livelli di automazione[20]

assistenza alla guida(**sistemi ADAS**). Questi sistemi includono:

- Adaptive Cruise Control- controllo automatico della velocità
- Lane-Keeping Assist - controllo dello sterzo per impedire l'uscita dalla corsia
- Automatic Emergency Braking - controllo del freno per le situazioni di emergenza

Livello 3: Automazione condizionale

La guida è completamente automatizzata ma non in tutti i momenti del viaggio. Il conducente viene avvisato quando deve riprendere il controllo ed è tenuto a rispondere in modo appropriato. Se questo non avviene, il sistema arresta il veicolo nel modo più sicuro possibile.

Livello 4: Alta Automazione

Il viaggio è completamente gestito dal sistema. Il conducente non è tenuto a intervenire in nessun momento. La completa autonomia è garantita però a patto che non esistano limitazioni come ad esempio condizioni metereologiche fortemente avverse.

Livello 5: Automazione totale

Sistema completamente autonomo. Non vi è alcuna limitazione ed è tutto gestito dal sistema interno in qualunque momento.

2.1.1 *La situazione attuale*

Al momento in commercio esistono veicoli fino al secondo livello(Tesla AutoPilot, Cadillac Super Cruise, Volvo Pilot Assist ecc.). I veicoli di livello 3 e 4 sono ancora in fase prototipale ma aziende come Honda e Audi promettono di avere modelli in commercio già dal 2021[21]. Il livello 5 sembra ancora lontano, ma il ritmo di sviluppo attuale fa ben sperare per i prossimi anni.

2.2 VISIONE ARTIFICIALE

La visione dell'ambiente circostante è la questione focale nello sviluppo di un veicolo a guida autonoma. Per poter "vedere", una macchina utilizza diversi sensori:

- Camere
- Radar
- Lidar

Figura 17: i sensori più utilizzati[22]

2.2.1 *Camere*

Le videocamere utilizzate sono profondamente diverse dalle normali videocamere in vendita. Sono progettate in modo da avere un campo

visivo molto ampio, migliore sensibilità anche a bassa luminosità, bassa risoluzione(migliori performance) e per funzionare ad alte e basse temperature. Restano però limitate in situazioni di bassa visibilità.

2.2.2 *Radar*

Il compito dei radar è principalmente quello di rilevare ostacoli e situazioni di pericolo imminente. In base alle informazioni raccolte l'auto accelera o frena. I radar si rilevano utili anche nella fase di parcheggio.

2.2.3 *Lidar*

I lidar sono dispositivi il cui funzionamento è simile a quello dei radar. La differenza principale sta nel segnale utilizzato: i radar usano onde radio mentre i lidar usano impulsi luminosi. I lidar sono particolarmente utili per la rilevazione di pedoni e per riconoscere i segnali stradali.

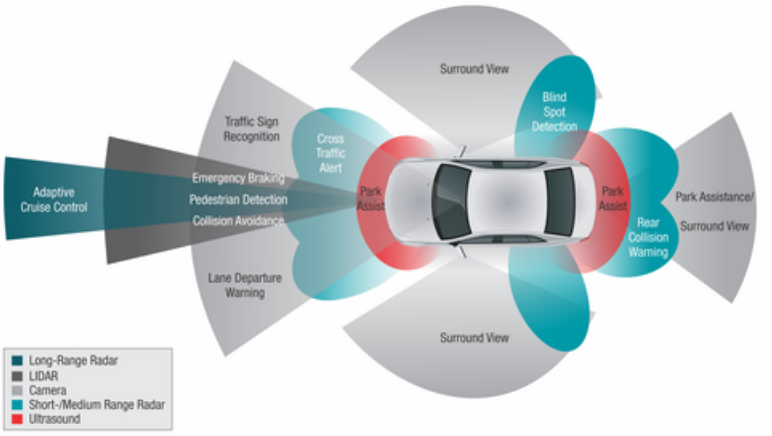

Figura 18: sensori in una macchina autonoma[23]

2.2.4 *Dai sensori agli attuatori*

Tutti i sensori sono fondamentali perchè ciascuno di esso compensa i punti deboli dell'altro. Una videocamera non funziona bene in condizioni di nebbia intensa, ma un radar sì. Un radar però non riconosce se un semaforo è verde o rosso, mentre una camera sì. Nella figura 18 sono mostrate le funzioni svolte da ciascun sensore.

Le informazioni raccolte dai sensori vengono passate al "livello" sottostante, nel quale il sistema interno sceglie la prossima azione da compiere. Il modello decisionale più utilizzato è una rete neurale convoluzionale. Il deep learning si è dimostrato estremamente potente in quest'ambito, fornendo ottimi risultati per l'object recognition. L'azione scelta viene passata agli attuatori(freno, sterzo, acceleratore) che provvedono ad eseguirla.

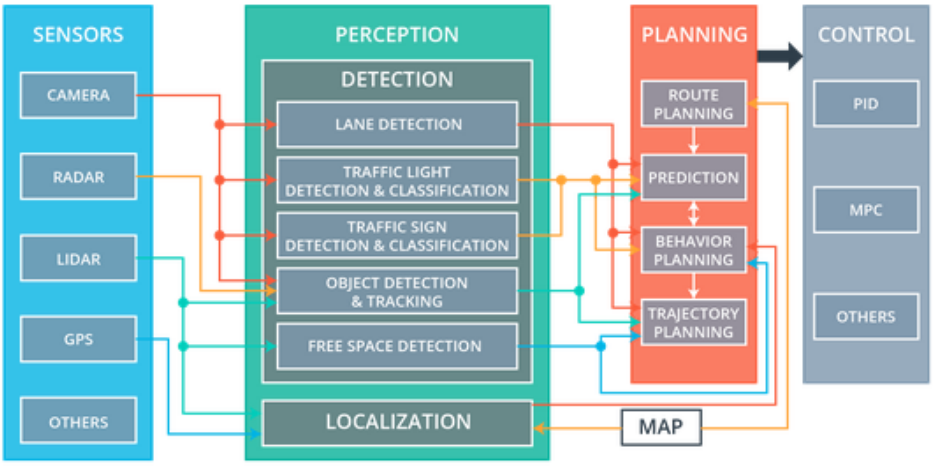

Figura 19: Il processo decisionale[24]

3

DEPENDABILITY E SECURITY

[25] I veicoli a guida autonoma appartengono alla categoria dei sistemi critici. Un sistema critico è un sistema il cui malfunzionamento può provocare danni considerati inacettabili. Questi includono danni a oggetti di valore, danni ambientali e nei casi piu gravi, il ferimento o addirittura la morte delle persone. Per garantire che un tale sistema operi nel modo più sicuro possibile è necessario analizzare tutti i fattori che possono portare a un fallimento irreversibile.

La **dependability** è la capacità di un sistema di fornire un servizio sul quale è possibile fare affidamento in modo giustificato. Essa viene suddivisa in 3 categorie:

- Attributi
- Minacce
- Mezzi di Raggiungimento

3.1 ATTRIBUTI

La dependability comprende i seguenti attributi:

- **availability**: disponibilità del servizio corretto
- **reliability**: stabilità del servizio corretto
- **safety**: assenza di conseguenze catastrofiche sull'utente e sull'ambiente
- **integrity**: assenza di alterazioni improprie al sistema
- **maintainability**: capacità di subire modifiche e riparazioni

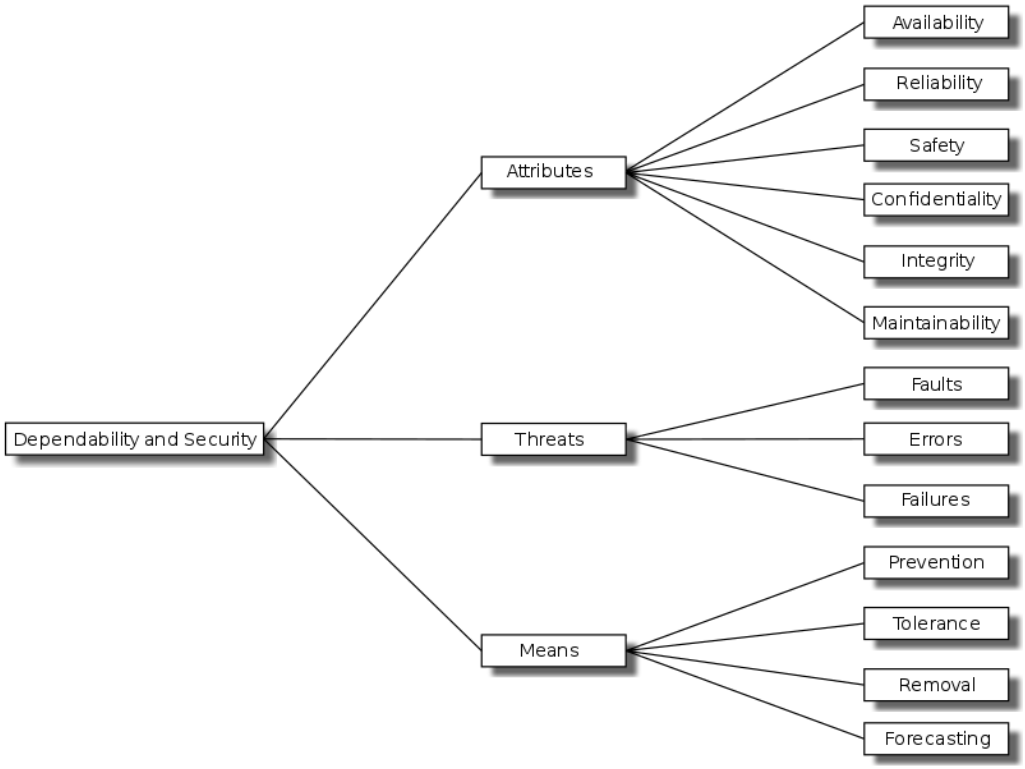

Figura 20: tassonomia della dependability[26]

Quando si considera la **security** è necessario specificare un ulteriore attributo: la confidentiality, ovvero la mancata divulgazione di informazioni non autorizzata. La security è composta da confidentiality, integrity e availability.

3.2 MEZZI DI RAGGIUNGIMENTO

Nel corso degli anni si sono sviluppate molte metodologie per raggiungere dependability e security. Tali metodologie possono essere raggruppate in quattro macrocategorie:

- **fault prevention**: mezzi per prevenire l'occorrenza e l'introduzione di guasti
- **fault tolerance**: mezzi per evitare fallimenti di servizio quando sono presenti dei guasti
- **fault removal**: mezzi per ridurre numero e gravità dei guasti

- **fault forecasting**: mezzi per stimare numero, incidenza futura e probabili conseguenze di guasti

Fault prevention e fault tolerance portano al conseguimento della dependability mentre fault removal e fault forecasting sono i mezzi di validazione

3.3 MINACCE

Le maggiori minacce per la dependability sono i **guasti**(faults in inglese). I guasti hanno varie cause e causano gli **errori**. Un errore può causarne altri fino a propagarsi al di fuori dei confini del sistema. Quando ciò avviene, si verifica un **fallimento**. Un fallimento è la situazione in cui il servizio fornito è diverso dal servizio corretto.

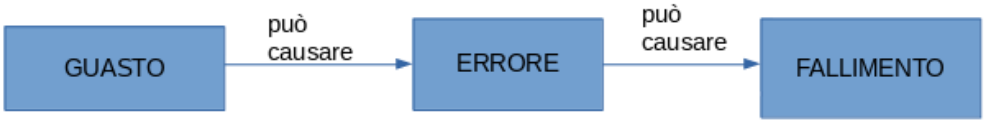

Figura 21: Catena guasto-errore-fallimento

3.3.1 *Tipi di guasti*

I guasti vengono classificati secondo 8 parametri, i quali creano le classi di guasto elementari, mostrate in figura 22. I diversi tipi di guasto sono raggruppati in tre categorie:

- **Development faults**: i guasti che avvengono in fase di sviluppo
- **Physical faults**: i guasti che riguardano l'hardware
- **Interaction faults**: tutti i guasti causati dall'interazione con l'ambiente esterno

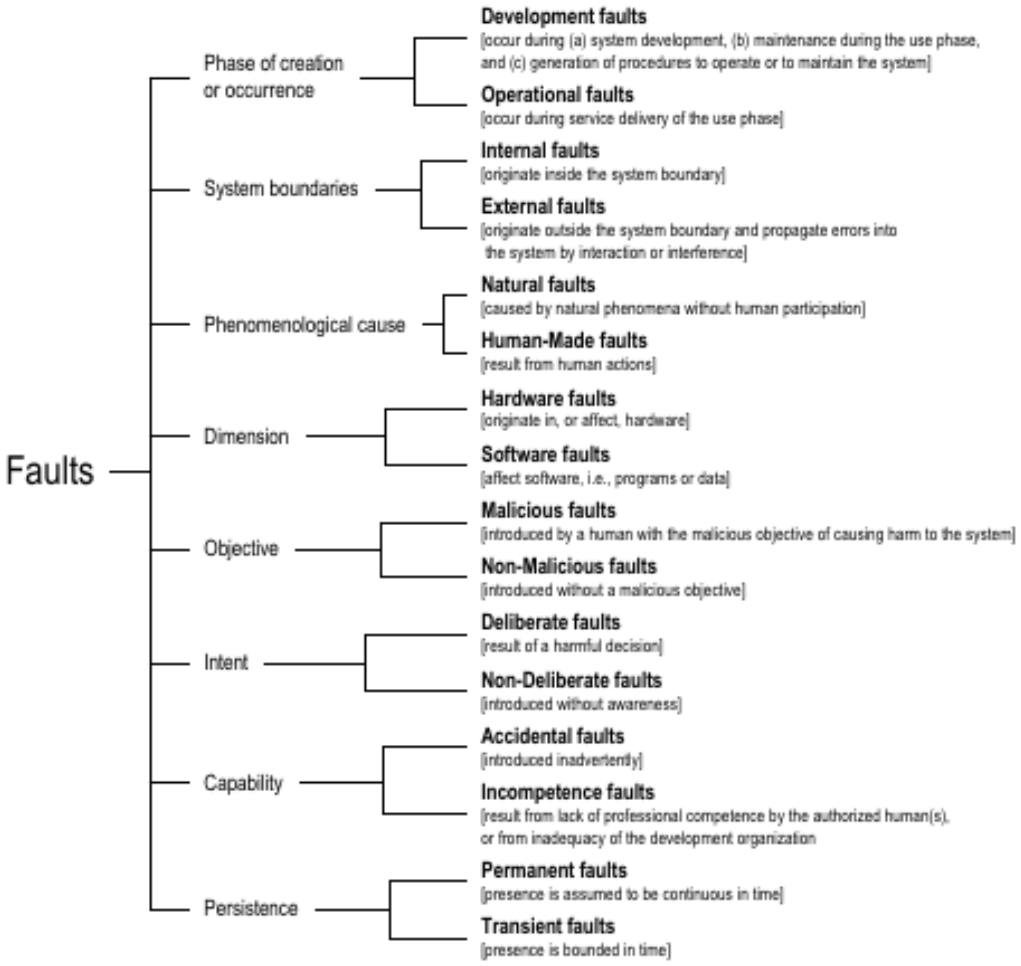

Figura 22: tassonomia dei guasti[25]

Guasti causati dall'azione umana

I guasti sul quale ci concentriamo sono quelli di origine umana . Questi guasti possono essere causati da una mancanza di azioni ove necessarie(omission faults), oppure da azioni che si rilevano essere sbagliate(commision faults). Vengono suddivisi in due tipi, sulla base dell'*obiettivo* dello sviluppatore o dell'essere umano che lo causa:

- guasti involontari, causati accidentalmente senza lo scopo di arrecare danno
- guasti maligni, causati in modo volontario per arrecare danni al sistema durante l'uso

I guasti maligni hanno come scopo la negazione del servizio(denial of service), l'accesso a informazioni confidenziali o la modifica impropria di un sistema. Vengono raggruppati in due classi:

- **guasti a livello logico**: comprendono virus, worms, bombe logiche ecc.
- **tentativi di intrusione**, anche usando mezzi fisici

In entrambi i casi si sfrutta una vulnerabilità di un sistema per ottenerne il pieno controllo(exploit). La vulnerabilità solitamente è a livello software ed è causata dall'azione involontaria degli sviluppatori.

4

STRUMENTI UTILIZZATI

4.1 CARLA

[27] La ricerca per lo sviluppo della guida autonoma in ambiente urbano è ostacolato da costi infrastrutturali e dalle difficoltà logistiche. È estremamente difficile avere anche solo un veicolo(che sarebbe comunque insufficiente) a disposizione per il testing e la validazione. Inoltre alcuni scenari di system verification sono troppo pericolosi per essere eseguiti nel mondo reale. Si pensi ad esempio allo scenario in cui un pedone attraversa improvvisamente la strada davanti alla macchina. Per questi motivi il training e la validazione vengono spesso fatti in ambiente simulato. Un simulatore molto adatto a questo scopo, e quello che è stato utilizzato in questo lavoro è **Carla**. Carla è un simulatore open source per la guida in ambiente urbano. È stato interamente sviluppato per supportare l'addestramento, la prototipazione e la validzione di modelli di guida autonoma, includendo percezione e controllo. Carla è una piattaforma libera. Tutti i modelli sono stati creati da zero e sono liberamente accessibili. Include diversi tipi di layout urbani, di modelli per veicoli,edifici,pedoni,segnali stradali ecc. La simulazione supporta una moltitudine di sensori differenti e possono essere specificate un ampio numero di condizioni ambientali.

4.1.1 *Il motore della simulazione*

L'engine interno è sviluppato sulla base dell'Unreal Engine 4(UE4). Fornisce una fisica reale, npc che seguono una logica elementare e un'ottima qualita di rendering. Il simulatore provvede una semplice interfaccia tra il mondo e un agente che interagisce con esso. Questa funzionalità è implementata con un'architettura server-client. Il server esegue la simulazione e il rendering. Il client comunica con il server attraverso un'API imple-

Figura 23: Alcune delle condizioni metereologiche disponibili in Carla

mentata in python. L'API permette l'interazione tra l'agente autonomo e il server attraverso i socket. Il client invia comandi e meta comandi e riceve dati dai sensori. I comandi sono sterzo, accelerazione e frenata. I meta comandi sono usati per modificare le proprietà del mondo(es. per generare veicoli e pedoni).

4.1.2 *L'ambiente*

L'ambiente è composto da modelli 3d statici(edifici, vegetazione, segnali stradali, infrastruttre) e modelli dinamici(veicoli, pedoni.). Le dimensioni dei modelli sono quanto più possibile realistiche. Il comportamento degli NPC è stato implementato per mantenere un alto livello di realismo. I veicoli non controllabili sono basati sul modello standard di UE4(PhysXVehicles). I pedoni si muovono nelle strade seguendo seguendo una mappa di navigazione specifica. Sono incoraggiati a muoversi nei marciapiedi e sulle strisce pedonali ma possono attraversare la strada in qualsiasi istante. Per quanto riguarda illuminazione e condizioni atmosferiche si può scegliere tra 18 diverse combinazioni. Le combinazioni sono date da due possibili condizioni di luminosità(mezzogiorno e tramonto) e nove condizioni metereoligiche. Quest'ultime differiscono per copertura del cielo, livello di precipitazioni e la presenza o meno di

pozze sull'asfalto.

4.1.3 *Sensori*

I sensori presenti in carla attualmente sono camere RGB e pseudo sensori che forniscono ground truth depth e ground truth semantic segmentation. Quest'ultimo riconosce 12 categorie semantiche:strada, delimitatore di corsia, segnale stradale, marciapiede, recinzione, palo, muro, edificio, vegetazione veicolo, pedone e altro. In aggiunta alle misure sensoriali,

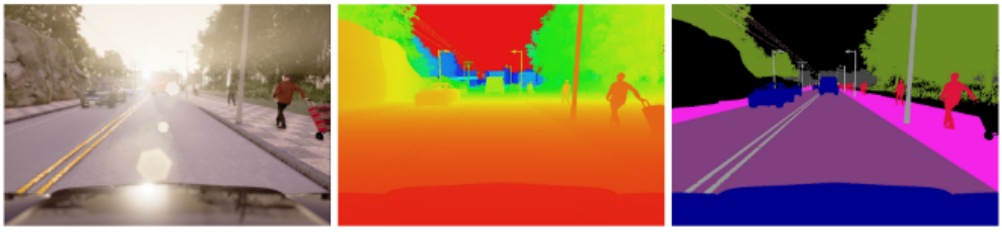

Figura 24: Alcuni sensori disponibili:

CARLA provvede anche una serie di misure associate allo stato dell'agente e al rispetto delle regole stradali. Le misure sullo stato dell'agente riguardano posizione e orientamento del veicolo, velocità, accelerazione, e danni accumulati da collisioni. Per quanto riguarda le regole stradali vengono riportati stato dei semafori e limite di velocità nella posizione attuale della macchina.

4.1.4 *Guida Autonoma in CARLA*

CARLA supporta lo sviluppo, l'addestramento e l'analisi delle prestazioni di sistemi di guida autonoma. Attualmente il simulatore è stato usato per valutare tre approcci:

- Modular pipeline
- Imitation Learning
- Reinforcement Learning

Con tutti gli approcci l'agente interagisce con l'ambiente in passi temporali discreti. Ad ogni istante l'agente riceve un'osservazione O e produce un'azione A. L'azione è un vettore tridimensionale che rappresenta lo sterzo, l'accelerazione e il freno. L'osservazione O è formata da una tupla

di input sensoriali monodimensionali(velocità), bidimensionali(GPS) e multidimensionali(immagini rgb, mappe di profondità). Inoltre in tutti gli approcci viene usato anche un tragitto definito da un planner topologico di alto livello.

Modular Pipeline

La guida è divisa in tre sottosistemi:

- perception
- planning
- continous control

PERCEPTION Per la percezione viene usata una rete di segmentazione semantica basata su RefineNet. La rete viene addestrata per classificare ogni pixel di un'immagine in una delle seguenti categorie:

- strada
- marciapiede
- delineatore di corsia
- oggetto dinamico
- oggetto statico

Il training set è composto da 2500 immagini etichettate prodotte direttamente all'interno del simultatore Viene inoltre usato un classificatore binario addestrato su 500 immagini per misurare la probabilità di trovarsi a un incrocio.

PLANNING Il local planner controlla la navigazione di basso livello generando un insieme di waypoints: punti che rappresentano la posizione e l'orientamento desiderati negli istanti successivi. Il pianificatore è implementato come una macchina a stati. Gli stati possibili sono:

- seguire la strada
- girare a sinistra
- girare a destra

- incrocio
- arresto di emergenza

Le transizioni avvengono sulla base dei dati raccolti dal modulo di percezioni e dalle informazioni topologiche generate dal global planner.

CONTINOUS CONTROL controlla gli attuatori per eseguire l'azione voluta. Viene utilizzato un controller PID(proportional-integral-derivative). Ciascun controller riceve posizione, velocità e lista di waypoints e compie sterzata, accelerazione o frenata.

Imitation Learning

Vengono utilizzati dei tragitti registrati di guidatori umani nella città di addestramento. Il dataset è composto da tuple formate da osservazione, azione e comando. I comandi vengono forniti dai guidatori durante la fase di data collectioni. Sono stati usati 4 comandi: segui la strada, vai dritto al prossimo incrocio, vai a destra al prossimo incrocio e vai a sinistra al prossimo incrocio. Le osservazione sono immagini raccolte dalla camera anteriore. E stata aggiunta una perturbazione nel dataset per rendere più robusto il learning. Il dataset è usato per addestrare un DNN per prevedere l'azione dell'esperto data l'osservazione e il comando.

Reinforcement Learning

Una rete viene addestrata sulla base di un segnale di ricompensa, senza utilizzare traiettorie raccolte da umani. Viene usato l'algoritmo AC3 per la navigazione goal directed. In ogni episodio l'agente deve raggiungere un obiettivo, guidato da comandi di alto livello. L'episodio termina se si raggiunge il gol, se scade il tempo o se avviene una collisione. La ricompensa è una somma pesata di cinque termini: velocità, distanza dall'obiettivo, danno da collisione, salita sul marciapiede e invasione di corsia.

4.2 ADVERSARIAL ROBUSTNESS TOOLBOX

[28] L'Adversarial Robustness Toolbox(ART) è una libreria Python che aiuta sviluppatori e ricercatori nella difesa dei modelli di machine learning dagli adversarial examples, rendendoli più sicuri ed affidabili. Creare tali difese significa certificare e verificare la robustezza con metodi come il preprocessing degli input, l'aumento del training set con esempi adversarial e fare uso di metodologie di rilevamento di input modificati. L'ART fornisce degli attacchi grazie ai quali valutare la sicurezza dei modelli testati. Nella libreria sono presenti interfacce standard per le librerie di machine learning più popolari. L'architettura dell'ART rende semplice combinare i vari approcci ed è progettato sia per i ricercatori che vogliono eseguire esperimenti su larga scala di benchmarking di attacchi e difese, sia per gli sviluppatori che vogliono rilasciare applicazioni che fanno uso di machine learning in modo sicuro.

4.2.1 *La struttura della libreria*

L'ART è diviso in vari sottomoduli:

- art.attacks
- art.classifiers
- art.defences
- art.detection
- art.metrics
- art.poison_detection

art.attacks

Implementazione degli attacchi presenti nella libreria. Gli attacchi sono divisi in tre categorie:**Evasion Attacks**, **Poisoning Attacks** e **Extraction Attacks**.

EVASION ATTACKS Si applica una modifica impercettibile all'input per causare la misclassificazione del modello attaccato. Si distinguono in attacchi whitebox e attacchi blackbox a seconda che i parametri del modello siano conosciuti o meno.

- Threshold Attack
- Pixel Attack
- HopSkipJump Attack
- High Confidence Low Uncertainty adversarial examples
- Projected Gradient Descent
- NewtonFool
- Elastic net attack
- Spatial transformation attack
- Query-efficient black-box attack
- Zeroth-order optimization attack
- Decision-based attack / Boundary attack
- Adversarial patch
- Decision tree attack
- Carlini & Wagner(C&W) L_2 and L_inf attacks
- Basic iterative method
- Jacobian saliency map
- Universal perturbation
- DeepFool
- Virtual adversarial method
- Fast gradient method

POISONING ATTACKS Si modifica il training set per ridurre l'accuratezza del processo di learning. Il poisoning include la modifica degli esempi nel dataset, l'iniezione di dati "maligni" o la modifica delle etichette.

- Poisoning Attack on SVM
- Backdoor Attack

EXTRACTION ATTACKS Si cerca di sviluppare un modello sulla base di un modello proprietario e chiuso, "rubando" il comportamento del modello attaccato.

- Functionally Equivalent Extraction
- Copycat Confidence
- KnockoffNets

art.classifers

Implementazione delle interfacce per usare librerie di machine learning esterne insieme all'ART. Ciascuna libreria ha la propria classe wrapper. In ogni classe vengono forniti i metodi per addestrare e testare i vari modelli. Le librerie attualmente supportare sono: Tensorflow, Keras, PyTorch, MXNet, Scikit-learn, XGBoost,LightGBM,CatBoost e Gpy.

art.defences

Implementazione delle difese presenti nell'ART, suddivise in preprocessor, postprocessor, trainer e transformer.

- Preprocessor
  - Thermometer encoding
  - total variance minimization
  - PixelDefend
  - Gaussian data augmentation
  - Feature squeezing
  - Spatial smoothing
  - JPEG compression
  - Label smoothing
  - Virtual adversarial training
- Postprocessor
  - Reverse Sigmoid
  - Random Noise
  - Class labels

  - High Confidence
  - Rounding
- Trainer
  - Adversarial training
  - Adversarial training Madry PGD
- Transformer
  - Defensive Distillation

art.detection e art.poison_detection

Implementazione delle metodologie di rilievo degli adversarial examples e del data poisoning.

- adversarial detection
  - rilevatore base degli input
  - rilevatore addestrato su uno specifico livello
  - rilevatore basato su Fast Generalized Subset Scan
- poisoning detection
  - rilevazione basata sull'analisi dell'attivazione
  - rilevazione basata sulla provenienza dei dati

art.metrics

Definizione di alcune metriche per verificare, validare e certificare sicurezza e robustezza dei modelli in analisi.

- Clique Method Robustness Verification
- Randomized smoothing
- CLEVER
- Loss sensitivity
- Empirical robustness

5

INIEZIONE DI ATTACCHI IN CARLA

Il lavoro svolto è stato il seguente: l'individuazione di attacchi dell'ART interessanti da analizzzare e implementare, iniettandoli all'interno di un modello preesistente di guida autonoma che gira su CARLA. In questo capitolo vengono descritti:

- il modello scelto
- gli attacchi scelti
- iniezione e risultati

5.1 IL MODELLO SCELTO

Per semplicità implementativa è stato scelto un modello preaddestrato. La scelta è ricaduta su Learning By Cheating(LBC) [29]. LBC utilizza l'imitation learning come approccio e divide la fase di addestramento in due PARTi:

- nella prima viene addestrato un agente privilegiato sulla base di traiettorie di esperti. Questo agente ha accesso a tutte le informazioni dell'ambiente circostante(layout ambientale, posizione di ogni altro partecipante). Agisce sull'ambiente con una vista dall'alto(birdview).
- nella seconda fase l'agente privilegiato fa da supervisore per l'addestramento di un agente sensorimotore. Questo ha accesso soltanto alle informazioni dei propri sensori(una camera RGB frontale). Viene addestrato per imitare l'agente privilegiato.

L'apprendimento è stato suddiviso in due fasi per separare due compiti: imparare ad *agire*, compito dell'agente privilegiato e imparare a *vedere*, compito dell'agente sensorimotore. Tale suddivisione porta i seguenti vantaggi:

- poichè l'agente privilegiato utilizza una rappresentazione compatta dell'ambiente, apprende più velocemente e generalizza meglio
- la supervisione fornita dall'agente privilegiato è di gran lunga più forte rispetto alle traiettorie originali, perchè può essere interrogato su qualsiasi stato dell'ambiente
- l'agente privilegiato è di tipo white box e il suo stato interno può essere analizzato in qualsiasi istante

Nella figura 25 è mostrata l'architettura dei due agenti. Concentrandosi sull'agente sensorimotore l'input viene passato a una rete convoluzionale che produce una serie di waypoints nella camera frontale. Gli waypoints sono selezionati sulla base del comando scelto e vengono passati al controllore di basso livello che produce un valore di sterzata, accelerazione o freno

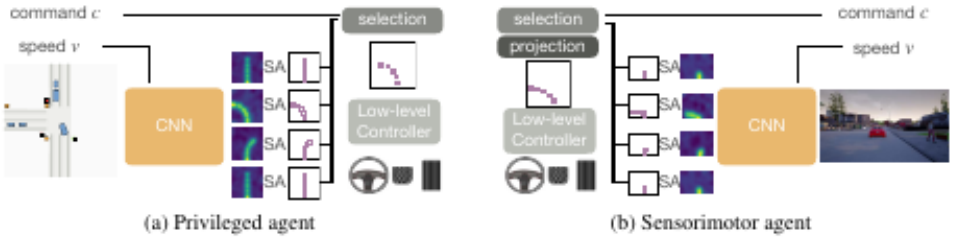

Figura 25: architettura dei due agenti[29]

5.2 GLI ATTACCHI SCELTI

Tra gli attacchi presenti nell'ART sono stati analizzati solamente gli evasion attacks per i seguenti motivi:

- utilizzando un modello preaddestrato ha poco senso andare a implementare i poisoning attacks, in quanto essi agiscono durante la fase di training
- gli extraction attacks sono stati esclusi in quanto LearningByCheating è open-source mentre questo tipo di attacchi sono rilevanti quando si ha a che fare con algoritmi proprietari

Tra gli attacchi analizzati ne abbiamo individuati tre di particolare interesse:

- Adversarial Patch
- Spatial Transformation
- HopSkipJump
- Basic Iterative Method
- NewtonFool
-

5.2.1 *Adversarial Patch*

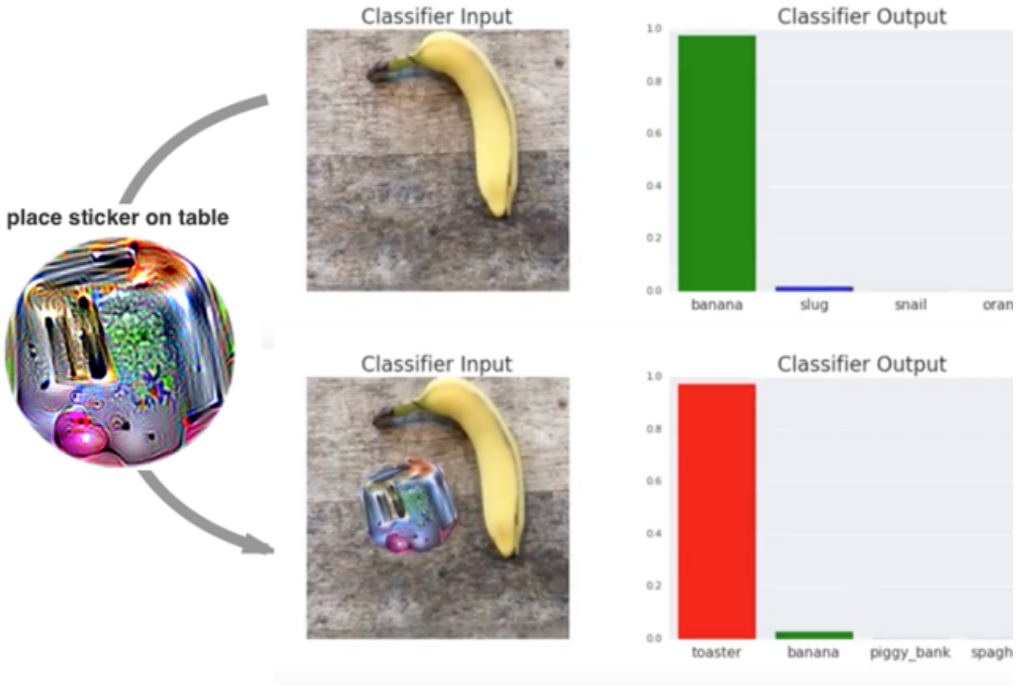

Figura 26: adversarial patch

[30] In questo attacco si genera una patch che, quando entra nel campo visivo di un classificatore, causa un'errore di misclassificazione. Lo sviluppo dell'attacco è indipendente dalla immagine su cui viene generata. La patch generata può essere applicata direttamente alle immagini ma l'utilizzo più interessante è senza dubbio la stampa e l'applicazione di tale patch ad oggetti fisici. In questo caso la patch potrebbe essere distribuita attraverso internet, stampata, e utiilizzata da un qualsiasi attaccante.

Questo attacco è profondamente atipico rispetto ai normali evasion attacks. La perturbazione è infatti di grandi dimensioni e questo potrebbe sembrare controintuitivo rispetto a ciò che sappiamo di tali attacchi. Ma il vantaggio principale sta proprio nella sua natura unica: una grande perturbazione è "resistente" anche rispetto ai normali metodi di difesa che si concentrano sulle piccole perturbazioni, ma che possono essere annullate in casi così estremi.

5.2.2 *Spatial Transformation*

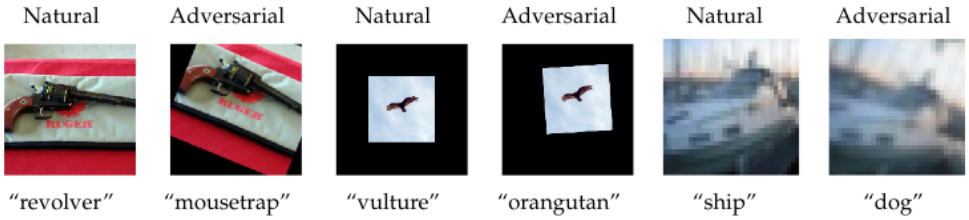

Figura 27: esempi di trasformazioni avversarie

[31]La perturbazione generata è di tipo spaziale. Si cercano i parametri $(\delta u, \delta v, \theta)$ per cui un'immagine ruotata di un angolo θ e traslata di $(\delta u, \delta v)$ pixel viene classificata in modo errato, massimizzando la loss function. Questo tipo di perturbazioni può essere generato in modo "maligno", ma può anche essere causato da perturbazioni naturali(gli oggetti reali non sempre appaiono perfettamente centrati).

5.2.3 *HopSkipJump*

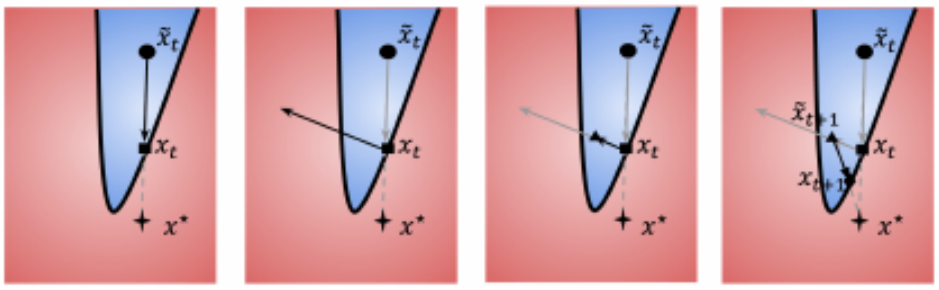

Figura 28: funzionamento dell'hopskipjump

È un attacco blackbox e necessita solo delle classi di output del modello. Parte da una grande perturbazione e punta a ridurla al minimo mantenendo comunque l'errore di classificazione. La perturbazione viene ridotta attraverso diverse iterazioni di ricerca binaria. Ciascuna iterazione individua una nuova perturbazione valida, più "piccola" della precedente.

5.2.4 *Basic Iterative Method*

[32]Versione iterativa del Fast Gradient Method. Tale metodo genera una perturbazione risolvendo il problema di massimizzare la loss function con la seguente equazione:

$$X^{adv} = X + \epsilon sign(\nabla_X J(X, y_{true}))$$

Nel BIM questa operazione viene ripetuta per un numero arbitrario di iterazioni, andando a clippare la perturbazione generata ad ogni passo in modo da restare all'interno dei limiti predisposti dall'iperparametro ϵ.

$$X_0^{adv} = X, X_{N+1}^{adv} = Clip_{X,\epsilon}\{X_N^{adv} + \alpha sign(\nabla_X J(X_N^{adv}, y_{true}))\}$$

α indica di quanto è stato modificato un pixel ad ogni passo(in questo caso $\alpha = 1$)

5.2.5 *NewtonFool*

[33] Si assuma che il classificatore $F(x)$ sia della forma $argmax_l F_s(x)$ e che l'output F_s sia disponibile all'attaccante. Se $F(x_0) = l \in C$ per un qualche input x_0, allora F_s^l è la maggiore probabilità in $F_s(x_0)$. L'attacco consiste nel trovare un piccolo $\mathbf{d}$ per cui $F_s^l(x_0 + d) \approx 0$. L'equazione viene risolta usando il metodo di Newton per la risoluzione di equazioni non lineari, da cui il nome.

5.2.6 *Fattibilità degli attacchi scelti su veicoli a guida autonoma*

Nel contesto di questo lavoro iniettare un attacco si riduce a modificare il codice del modello LearningByCheating, di cui abbiamo l'accesso completo. Nel mondo reale invece, di solito è necessario prima assumere il controllo della parte del veicolo che ci permetta di applicare gli attacchi

scelti. Per quattro dei cinque attacchi scelti(hopskipjump,spatial transformation, basic iterative method, newtonfool), iniettarli in un veicolo significa ottenere il controllo del canale di comunicazione fra camera e l'unità di elaborazione centrale, nella quale è presente il modello decisionale. Così facendo si potrebbero modificare gli input prima che essi entrino all'interno del modello stesso. Questo potrebbe essere realizzato attraverso degli script che sfruttino possibili vulnerabilità del protocollo di comunicazione usato o errori di programmazione. Per l'attacco Adversarial Patch invece non c'è bisogno di ottenere il controllo diretto del veicolo. Come detto in precedenza, un attaccante potrebbe generare una patch in modo indipendente(o addirittura scaricarla da internet) e applicarla sugli oggetti riconosciuti abitualmente da un veicolo(segnali stradali, semafori, fiancate di altre auto ecc.).

5.3 INIEZIONE DEGLI ATTACCHI

LearningByCheating fornisce un modulo(run_benchmark) grazie al quale valutare un agente sensorimotore(benchmark_agent) su una serie di percorsi prestabiliti. In questi percorsi l'agente deve arrivare a un punto della mappa preciso entro un tempo limite. Per ciascun percorso viene tenuto traccia dei semafori rossi ignorati, delle invasioni di corsia e delle collisioni. Inoltre viene registrato un video che mostra il veicolo mentre percorre il tragitto. Nel video vengono mostrati anche gli waypoints generati dalla rete neurale in ogno momento. In 5.1 viene definito il loop principale di un percorso. Ad ogni "istante"(in realtà *env.tick()* aspetta l'esecuzione di un'azione) l'agente riceve le osservazioni dai sensori(*env.get_observations()*), le passa alla rete neurale che decide l'azione(*agent.run_step(observations)*) e applica l'azione restituita attraverso i controllori PID(*env.apply_control(control)*). Il percorso termina se il veicolo arriva a destinazione, scade il tempo, oppure avviene una collisione. L'ultima condizione non era presente nel codice originale ma è stata aggiunta per risparmiare tempo sulla singola run nel caso in cui avvenga un incidente. Per essere efficaci gli attacchi devono modificare gli input **prima** che arrivino alla rete, quindi la riga *agent.run_step(observations)* è stata individuata come il punto giusto nel quale iniettare gli attacchi prescelti

run_step è definito nella classe ImageAgent del modulo *image*(5.2).

La perturbazione dell'immagine avviene in:

```
_rgb= self.attack.generate(x=_rgb.cpu())
```

```
    _rgb = torch.FloatTensor(_rgb)
```

Il metodo *generate* attua la modifica vera e propria dell'immagine. Il campo *self.attack* rappresenta l'attacco iniettato. L'iniezione avviene nelle due righe aggiunte al costruttore:

```
    self.adv =
        load_model('/home/piazzesi/Desktop/carla_lbc/ckpts/image')
    self.attack = load_attack(self.adv, 'hopskipjump')
```

Le funzioni load_model e load_attack sono state implementate nel modulo attack5.3. Si nota come la definizione degli attacchi corrisponde semplicemente a una chiamata di un costruttore. L'implementazione vera e propria è fornita direttamente dai moduli dell'ART.

Le ultime modifiche sono state attuate nel metodo forward della classe ImagePolicyModelSS5.4. Questa classe è l'implementazione della rete neurale e in forward avviene il processo decisionale. forward viene anche usato nel metodo *generate* per realizzare la perturbazione. Il problema è che *generate* prende come unico parametro l'immagine RGB mentre forward usa anche velocità e comando. La soluzione scelta è stata la seguente: la velocità e il comando non vengono passati a forward attraverso i parametri di input, ma vengono recuperati dalle variabili globali *_speed* e *_command* definite in *run_step*. Usare variabili globali non è sicuramente una pratica di buona programmazione, ma ha permesso l'esecuzione degli attacchi senza problemi. Infine , sempre per garantire la compatibilità tra ART e modello è stata aggiunta la riga:

```
    location_pred = torch.reshape(location_pred,(1,10))
```

Listing 5.1: loop base di una run

```
while env.tick():
      observations = env.get_observations()
      control = agent.run_step(observations)
      diagnostic = env.apply_control(control)

      _paint(observations, control, diagnostic, agent.debug, env,
          show=show)

      diagnostic.pop('viz_img')
      diagnostics.append(diagnostic)
```

```
      if env.is_failure() or env.is_success() or env.collided==True:
         result['success'] = env.is_success()
         result['total_lights_ran'] =
             env.traffic_tracker.total_lights_ran
         result['total_lights'] = env.traffic_tracker.total_lights
         result['collided'] = env.collided
         result['t'] = env._tick
         break

   return result, diagnostics
```

Listing 5.2: ImageAgent

```
import math

import numpy as np

import torch
import torch.nn as nn
import sys
from . import common
from .agent import Agent
from .controller import CustomController, PIDController
from .controller import ls_circle
from .attack import load_model, load_attack
from benchmark import run_benchmark
CROP_SIZE = 192
STEPS = 5
COMMANDS = 4
DT = 0.1
CROP_SIZE = 192
PIXELS_PER_METER = 5




class ImageAgent(Agent):
   def __init__(self, steer_points=None, pid=None, gap=5,
       camera_args={'x':384,'h':160,'fov':90,'world_y':1.4,'fixed_offset':4.0},
       **kwargs):
      super().__init__(**kwargs)
```

```
        self.fixed_offset = float(camera_args['fixed_offset'])
        print ("Offset: ", self.fixed_offset)
        w = float(camera_args['w'])
        h = float(camera_args['h'])
        self.img_size = np.array([w,h])
        self.gap = gap
        self.adv =
            load_model('/home/piazzesi/Desktop/carla_lbc/ckpts/image')
        self.attack = load_attack(self.adv, 'hopskipjump')
        if steer_points is None:
            steer_points = {"1": 4, "2": 3, "3": 2, "4": 2}
        if pid is None:
            pid = {
                "1" : {"Kp": 0.5, "Ki": 0.20, "Kd":0.0}, # Left
                "2" : {"Kp": 0.7, "Ki": 0.10, "Kd":0.0}, # Right
                "3" : {"Kp": 1.0, "Ki": 0.10, "Kd":0.0}, # Straight
                "4" : {"Kp": 1.0, "Ki": 0.50, "Kd":0.0}, # Follow
            }
        self.steer_points = steer_points
        self.turn_control = CustomController(pid)
        self.speed_control = PIDController(K_P=.8, K_I=.08, K_D=0.)
        self.engine_brake_threshold = 2.0
        self.brake_threshold = 2.0
        self.last_brake = -1

    def run_step(self, observations, teaching=False):
        global _speed, _command
        rgb = observations['rgb'].copy()
        speed = np.linalg.norm(observations['velocity'])
        _cmd = int(observations['command'])
        command = self.one_hot[int(observations['command']) - 1]
         _rgb = self.transform(rgb).to(self.device).unsqueeze(0)
        _speed = torch.FloatTensor([speed]).to(self.device)
        _command = command.to(self.device).unsqueeze(0)
        _rgb= self.attack.generate(x=_rgb.cpu())
        _rgb = torch.FloatTensor(_rgb)
        if self.model.all_branch:
            model_pred, _ = self.model(_rgb, _speed, _command)
        else:
            model_pred = self.model(_rgb, _speed, _command)
        model_pred = torch.reshape(model_pred,(5,2))
        model_pred = model_pred.squeeze().detach().cpu().numpy()
```

```
pixel_pred = model_pred
# Project back to world coordinate
model_pred = (model_pred+1)*self.img_size/2
world_pred = self.unproject(model_pred)
targets = [(0, 0)]
for i in range(STEPS):
   pixel_dx, pixel_dy = world_pred[i]
   angle = np.arctan2(pixel_dx, pixel_dy)
   dist = np.linalg.norm([pixel_dx, pixel_dy])
   targets.append([dist * np.cos(angle), dist *
       np.sin(angle)])
targets = np.array(targets)
target_speed = np.linalg.norm(targets[:-1] - targets[1:],
    axis=1).mean() / (self.gap * DT)
c, r = ls_circle(targets)
n = self.steer_points.get(str(_cmd), 1)
closest = common.project_point_to_circle(targets[n], c, r)
acceleration = np.clip(target_speed - speed, 0.0, 1.0)
v = [1.0, 0.0, 0.0]
w = [closest[0], closest[1], 0.0]
alpha = common.signed_angle(v, w)
steer = self.turn_control.run_step(alpha, _cmd)
throttle = self.speed_control.step(acceleration)
brake = 0.0
# Slow or stop.
if target_speed <= self.engine_brake_threshold:
   steer = 0.0
   throttle = 0.0
if target_speed <= self.brake_threshold:
   brake = 1.0
self.debug = {
      # 'curve': curve,
      'target_speed': target_speed,
      'target': closest,
      'locations_world': targets,
      'locations_pixel': model_pred.astype(int),
      }
control = self.postprocess(steer, throttle, brake)
if teaching:
   return control, pixel_pred
else:
   return control
```

Listing 5.3: attack.py

```
import torch
import numpy as np
from art.classifiers import PyTorchClassifier
from art.attacks.evasion import SpatialTransformation, HopSkipJump,
    NewtonFool, BasicIterativeMethod
import bird_view.utils.bz_utils as bzu
import torch.nn as nn
import torch.optim as optim
from bird_view.models import image
from torchsummary import summary

def load_model(model_path):
''' Carica i pesi del modello preaddestrato e li inserisce nella
    classe PytorchClassifier, che fa da interfaccia tra modello e
    ART'''
   config = bzu.load_json('/'.join([model_path, 'config.json']))
   model_args = config['model_args']
   model_name = model_args['model']
   model_class = image.ImagePolicyModelSS
   model = model_class(**config['model_args'])
   model.load_state_dict(torch.load('/'.join([model_path,
      'model-10.th'])))
   criterion = nn.CrossEntropyLoss()
   optimizer = optim.Adam(model.parameters(),lr=0.0001)
   classifier = PyTorchClassifier(model=model,
      loss=criterion,optimizer=optimizer, input_shape=(3, 160,
      384),
   nb_classes=10,device_type='gpu')
   return classifier

def load_attack(classifier,attack):
   '''sceglie quale attacco iniettare a seconda della stringa
      passata in input'''
   attacks = {
   'hopskipjump':load_hopskip,
   'spatialtransformation':load_spatial,
   'newton':load_newton,
   'bid':load_bid
   }
   return attacks[attack](classifier)
```

```
def load_hopskip(classifier):
   return HopSkipJump(classifier=classifier,
                  targeted=False,
                  max_iter=0,
                  max_eval=1000,
                  init_eval=10)

def load_newton(classifier):
   return NewtonFool(classifier,max_iter=10)

def load_bid(classifier):
   return BasicIterativeMethod(classifier=classifier, max_iter=20,
      eps=0.2 )

def load_spatial(classifier):
   return SpatialTransformation(classifier=classifier,
                           max_translation=80,
                           num_translations=1,
                           max_rotation=230,
                           num_rotations=1)
```

Listing 5.4: ImagePolicyModelSS

```
class ImagePolicyModelSS(common.ResnetBase):
   def __init__(self, backbone, warp=False, pretrained=False,
       all_branch=False, **kwargs):
      super().__init__(backbone, pretrained=pretrained,
          input_channel=3, bias_first=False)

      self.c = {
            'resnet18': 512,
            'resnet34': 512,
            'resnet50': 2048
            }[backbone]
      self.warp = warp
      self.rgb_transform = common.NormalizeV2(
         mean=[0.485, 0.456, 0.406],
         std=[0.229, 0.224, 0.225]
      )

      self.deconv = nn.Sequential(
         nn.BatchNorm2d(self.c + 128),
         nn.ConvTranspose2d(self.c + 128,256,3,2,1,1),
```

```
            nn.ReLU(True),
            nn.BatchNorm2d(256),
            nn.ConvTranspose2d(256,128,3,2,1,1),
            nn.ReLU(True),
            nn.BatchNorm2d(128),
            nn.ConvTranspose2d(128,64,3,2,1,1),
            nn.ReLU(True),
        )

        if warp:
            ow,oh = 48,48
        else:
            ow,oh = 96,40

        self.location_pred = nn.ModuleList([
            nn.Sequential(
                nn.BatchNorm2d(64),
                nn.Conv2d(64,STEPS,1,1,0),
                common.SpatialSoftmax(ow,oh,STEPS),
            ) for i in range(4)
        ])

        self.all_branch = all_branch

    def forward(self, image, velocity=torch.FloatTensor([1.0]),
        command=torch.FloatTensor([[0,0,0,1]])):
        image= image.to(torch.device('cuda' if
            torch.cuda.is_available() else 'cpu'))

        if self.warp:
            warped_image = tgm.warp_perspective(image, self.M,
                dsize=(192, 192))
            resized_image = resize_images(image)
            image = torch.cat([warped_image, resized_image], 1)

        velocity = _speed
        command = _command
        image = self.rgb_transform(image)
        h = self.conv(image)
        b, c, kh, kw = h.size()
        # Late fusion for velocity
        velocity = velocity[...,None,None,None].repeat((1,128,kh,kw))
```

```
h = torch.cat((h, velocity), dim=1)
h = self.deconv(h)
location_preds = [location_pred(h) for location_pred in
    self.location_pred]

location_preds = torch.stack(location_preds, dim=1)

location_pred = common.select_branch(location_preds, command)
location_pred = torch.reshape(location_pred,(1,10))
if self.all_branch:
   return location_pred, location_preds

return location_pred
```

5.4 RISULTATI

Per confrontare gli attacchi è stato eseguito *run_benchmark* sulla suite di percorsi *regular*. Questa suite è composta da 4 mappe con determinate caratteristiche:

- NoCrashTown01-v3
  - numero di veicoli:20
  - numero di pedoni:50
- NoCrashTown02-v3
  - numero di veicoli:15
  - numero di pedoni:50
- NoCrashTown01-v4
  - numero di veicoli:20
  - numero di pedoni:50
- NoCrashTown02-v4
  - numero di veicoli:15
  - numero di pedoni:50

Ciascuna mappa prevede l'esecuzione di tre percorsi, ognuno con condizioni ambientali diverse. La valutazione degli attacchi si è svolta in due fasi:

- nella prima, i percorsi sono stati seguiti da un agente "puro", senza nessun attacco iniettato
- nella seconda fase, si eseguono le stesse run della prima fase, ma ogni volta con un diverso attacco iniettato.

I risultati prodotti dall'iniezione di ciascun attacco sono stati confrontati con quelli prodotti dalla run pura, in modo da valutarne l'efficacia. Il confronto è stato fatto sia in termini di percorsi portati a termine, sia andando ad analizzare e commentare i video prodotti.

5.4.1 *Run Pura*

5.4.2 *Iniezione di HopSkipJump*

5.4.3 *Iniezione di Spatial Transformation*

5.4.4 *Iniezione di Basic Iterative Method*

5.4.5 *Iniezione di NewtonFool*

BIBLIOGRAFIA

[1] Stuart J. Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach (3rd Edition)*. Prentice Hall, 2009. (Cited on pages 3, 8, 9, 10, 11, and 12.)

[2] John Mccarthy. What is artificial intelligence. 1998. (Cited on pages 8 and 11.)

[3] David Poole and Alan Mackworth. *Artificial Intelligence: Foundations of Computational Agents*. Cambridge University Press, Cambridge, UK, 2017. (Cited on pages 9 and 13.)

[4] Chethan Kumar. Artificial intelligence:definition, types, examples, tecnologies, 2018. (Cited on pages 3 and 12.)

[5] Wikipedia. Machine learning, 2020. (Cited on page 12.)

[6] Tom M. Mitchell. *Machine Learning*. McGraw-Hill, 1997. (Cited on page 13.)

[7] Glenn Doggett Julia Bonafede, Corey Cook. Artificial intelligence and its potential impact on the cfa institute code of ethics and standards of professional conduct, 2019. (Cited on pages 3 and 14.)

[8] Isha Salian. Supervize me: What's the difference between supervised, unsupervised, semi-supervised and reinforcement learning?, 2018. (Cited on pages 3 and 15.)

[9] Geoffrey Hinton Yann LeCun, Yoshua Bengio. Deep learning. *Nature*, pages 436,446, May 2015. (Cited on pages 16 and 19.)

[10] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. `http://www.deeplearningbook.org`. (Cited on page 16.)

[11] Facundo Bre, Juan Gimenez, and Víctor Fachinotti. Prediction of wind pressure coefficients on building surfaces using artificial neural networks. *Energy and Buildings*, 158, 11 2017. (Cited on pages 3 and 17.)

[12] Alessandro Mazzetti. *Reti Neurali Artificiali:introduzione ai principali modelli e simulazione su personal computer*. 1991. (Cited on page 17.)

[13] Ali Hasan, Hayder Al-Assadi, and Ahmad Azlan. *Neural Networks Based Inverse Kinematics Solution for Serial Robot Manipulators Passing Through Singularities*. 04 2011. (Cited on pages 3 and 17.)

[14] Shruti Jadon. Introduction to different activation functions for deep learning. *Medium*, 2018. (Cited on pages 3 and 18.)

[15] Hongquan Guo, Hoang Nguyen, Diep-Anh Vu, and Xuan-Nam Bui. Forecasting mining capital cost for open-pit mining projects based on artificial neural network approach. *Resources Policy*, 08 2019. (Cited on pages 3 and 18.)

[16] Pete Goldin. 10 advantagages of autonomous vehicles. *ITSdigest*, 2018. (Cited on page 21.)

[17] Causes of traffic accidents in austin, 2015. (Cited on pages 3 and 22.)

[18] Rob van der Heijden and Kiliaan van Wees. Introducing advanced driver assistance systems: Some legal issues. *European Journal of Transport and Infrastructure Research*, 1(3), 2001. (Cited on page 21.)

[19] Patrick Lin. *Why Ethics Matters for Autonomous Cars*, pages 69–85. Springer Berlin Heidelberg, Berlin, Heidelberg, 2015. (Cited on page 21.)

[20] Margaret Rouse. Self driving car, 2019. (Cited on pages 3 and 23.)

[21] new level 3 autonomous vehicles hitting the road in 2020. (Cited on page 24.)

[22] Paul McLellan. Automotive sensors:cameras,lidar, radar, thermal. *Breakfast Bytes Blog*, 2018. (Cited on pages 3 and 24.)

[23] Charles Murray. Automakers, suppliers ratchet up autonomous car programs. *DesignNews*, 2016. (Cited on pages 3 and 25.)

[24] Giuliano Giacaglia. Self driving cars. *Medium*, 2019. (Cited on pages 3 and 26.)

[25] A.Avizienis, J.C.Laprie, B.Randell, and C.Landwehr. Basic concepts and taxonomoy of dependable and secure computing. Technical report, Institute for Systems Research, 2004. (Cited on pages 3, 27, and 30.)

[26] Wikipedia. Dependability. (Cited on pages 3 and 28.)

[27] Alexey Dosovitskiy, German Ros, Felipe Codevilla, Antonio Lopez, and Vladlen Koltun. CARLA: An open urban driving simulator. In *Proceedings of the 1st Annual Conference on Robot Learning*, pages 1–16, 2017. (Cited on page 32.)

[28] Maria-Irina Nicolae, Mathieu Sinn, Minh Ngoc Tran, Beat Buesser, Ambrish Rawat, Martin Wistuba, Valentina Zantedeschi, Nathalie Baracaldo, Bryant Chen, Heiko Ludwig, Ian Molloy, and Ben Edwards. Adversarial robustness toolbox v1.2.0. *CoRR*, 1807.01069, 2018. (Cited on page 37.)

[29] Dian Chen, Brady Zhou, Vladlen Koltun, and Philipp Krähenbühl. Learning by cheating. In *Conference on Robot Learning (CoRL)*, 2019. (Cited on pages 3, 41, and 42.)

[30] Tom B. Brown, Dandelion Mané, Aurko Roy, Martín Abadi, and Justin Gilmer. Adversarial patch. *CoRR*, abs/1712.09665, 2017. (Cited on page 43.)

[31] Logan Engstrom, Dimitris Tsipras, Ludwig Schmidt, and Aleksander Madry. A rotation and a translation suffice: Fooling cnns with simple transformations. *CoRR*, abs/1712.02779, 2017. (Cited on page 44.)

[32] Alexey Kurakin, Ian J. Goodfellow, and Samy Bengio. Adversarial examples in the physical world. *CoRR*, abs/1607.02533, 2016. (Cited on page 45.)

[33] Uyeong Jang, Xi Wu, and Somesh Jha. Objective metrics and gradient descent algorithms for adversarial examples in machine learning. In *Proceedings of the 33rd Annual Computer Security Applications Conference*, ACSAC 2017, page 262–277, New York, NY, USA, 2017. Association for Computing Machinery. (Cited on page 45.)