

UNIVERSITÀ POLITECNICA DELLE MARCHE

FACOLTÀ DI INGEGNERIA



*Corso di Laurea Triennale in  
Ingegneria Informatica e dell'Automazione*

***Progettazione e Sviluppo di un algoritmo per il rilevamento  
dell'eye-blinking utilizzando immagini RGB***

***Design and Development of an algorithm to detect the  
eye-blinking from RGB images***

Relatore:  
DOTT. MANCINI ADRIANO

Laureando:  
ABBRUZZETTI MATTEO

ANNO ACCADEMICO 2020-2021



# Indice

<b>1</b>	<b>Introduzione</b>	<b>5</b>
1.1	Obiettivi . . . . .	9
1.2	Struttura della Tesi . . . . .	9
<b>2</b>	<b>Anche le macchine pensano</b>	<b>11</b>
2.1	Intelligenza Artificiale: quando è nata? . . . . .	11
2.2	Intelligenza Artificiale Forte e Debole . . . . .	14
2.2.1	L'Intelligenza Artificiale Debole . . . . .	14
2.2.2	L'Intelligenza Artificiale Forte . . . . .	14
2.3	Tecniche e modelli di apprendimento . . . . .	15
2.3.1	Il <i>Machine Learning</i> . . . . .	15
2.3.2	Il <i>Deep Learning</i> . . . . .	16
2.4	Rete Neurale . . . . .	17
2.5	Modello McCulloch-Pitts . . . . .	18
2.6	Back-Propagation . . . . .	19
2.6.1	Gradient Descent . . . . .	20
2.6.2	Valutazione degli iperparametri . . . . .	20
2.7	Reti Neurali Convoluzionali . . . . .	21
<b>3</b>	<b>Strumenti Utilizzati</b>	<b>25</b>
3.1	MediaPipe Face Mesh . . . . .	25
3.2	Dataset . . . . .	26
3.3	MobileNetV2 . . . . .	27
3.4	Librerie . . . . .	27
3.4.1	OpenCV . . . . .	27
3.4.2	Keras . . . . .	28
3.5	Linguaggi utilizzati . . . . .	28
3.6	Editor di Sviluppo . . . . .	29
3.7	VisualStudio Code . . . . .	29
3.8	Google Colab o Colaboratory . . . . .	30
3.8.1	TensorFlow . . . . .	31

---

<b>4</b>	<b>Sviluppo del Progetto</b>	<b>33</b>
4.1	Estrapolare dati da un video . . . . .	33
4.2	Approfondimento delle Classi . . . . .	34
4.2.1	myLandmarks.py . . . . .	34
4.2.2	facemesh.py . . . . .	35
4.2.3	evolution.py . . . . .	36
4.2.4	cnt.py . . . . .	38
4.3	Analisi dei risultati . . . . .	43
<b>5</b>	<b>Conclusioni</b>	<b>45</b>
5.1	Sviluppi Futuri . . . . .	45
	<b>Bibliografia</b>	<b>47</b>
	<b>Elenco delle figure</b>	<b>49</b>

# Capitolo 1

## Introduzione

Il presente lavoro di tesi è stato svolto in collaborazione con la collega Lucrezia Antenucci. Lo sviluppo del progetto è stato suddiviso in due parti: la dott.ssa Antenucci nel suo elaborato si è focalizzata sulle reti neurali e il loro addestramento, mentre in questa tesi verranno approfonditi gli aspetti relativi a Facemesh, in particolare, il riconoscimento e la cattura degli occhi e della bocca di un soggetto e di alcuni indici derivabili utili al riconoscimento dello stato (apertura / chiusura).

A partire dal 2001, in Italia, il numero di morti per incidenti stradali è sempre diminuito fino al 2013, per poi assestarsi intorno ai 3200-3400 morti l'anno. Nel 2020 si è avuta una diminuzione della mobilità stradale e, di conseguenza, anche del numero di incidenti a causa dei periodi di *lockdown* imposti dal governo italiano per limitare quanto più possibile la trasmissione della *malattia da nuovo coronavirus*. Nonostante la pandemia di *COVID-19* e i lunghi periodi di quarantena, in quell'anno è fallito l'obiettivo del decennio 2010-2020 di dimezzare il numero di vittime dovute agli incidenti stradali. Nell'ultimo anno del decennio, infatti, i 120.000 incidenti stradali con lesione a persone hanno provocato 2395 morti. C'è stata, dunque, una diminuzione del 30% rispetto alla media degli anni precedenti, ma risulta comunque non sufficiente e falsificata, appunto, dalla quarantena istituita a livello nazionale [1]. Nel rapporto Istat 2020, tra le cause principali degli incidenti stradali troviamo, al primo posto, la distrazione alla guida deriva in gran parte dall'utilizzo dei cellulari, seguita dal mancato rispetto della precedenza e dal raggiungimento di velocità troppo elevate.

Analizzando più approfonditamente il fenomeno della distrazione alla guida, possiamo suddividerlo in tre categorie: distrazioni visive, distrazioni manuali e distrazioni cognitive. Le distrazioni visive riguardano i casi in cui si distoglie lo sguardo dalla strada, come ad esempio quando si leggono messaggi di testo o si ricercano indicazioni stradali; le distrazioni manuali si hanno quando si tolgono le mani dal volante, per cercare e prendere qualcosa all'interno del veicolo, per usare un dispositivo manuale, per regolare la radio e per compiere molte altre azioni durante la guida; le distrazioni cognitive, infine, si verificano quando si pensa

ad altro e non si è interamente concentrati sulla guida, come quando si parla al telefono, si discute con un passeggero o si pensa al prossimo appuntamento [2].

Un'altra causa di molti incidenti, spesso mortali, è determinata dalla stanchezza al volante che è spesso provocata da mancanza di riposo, sonnolenza e lunghi viaggi notturni. Spesso molti guidatori proseguono il viaggio nonostante il sopraggiungere di chiari sintomi di affaticamento e stanchezza. Questo perché la maggior parte dei conducenti sottovaluta le limitazioni delle capacità di reazione dovute proprio alla stanchezza.

Per cercare di diminuire il numero di incidenti causati dalla distrazione alla guida e tenere alto il livello di attenzione del guidatore durante l'intero viaggio sono stati pensati e sviluppati nuovi metodi nell'ambito dell'Intelligenza Artificiale. I sistemi ADAS, *Advanced Driver Assistance System*, sono dei sistemi avanzati di assistenza alla guida che aiutano i conducenti e migliorano la sicurezza. All'interno di veicoli immatricolati negli anni recenti, sono già presenti sistemi di questo tipo e dal 2022 l'Unione Europea, attraverso il *Regolamento UE 2019/2144*, ha reso obbligatoria la presenza di alcuni di essi come il controllo adattivo della velocità e il rilevamento della stanchezza del conducente. Quest'ultimo, attraverso l'Intelligenza Artificiale, è in grado di imparare ed individuare i segnali del viso che indicano un abbassamento dell'attenzione del guidatore a causa, appunto, della stanchezza.

L'Intelligenza Artificiale, o semplicemente IA, è l'abilità di una macchina di sviluppare capacità umane quali il ragionamento, l'apprendimento, la pianificazione e la creatività. L'IA permette ai sistemi di capire il proprio ambiente, mettersi in relazione con quello che percepisce, risolvere problemi e agire verso un obiettivo specifico. I sistemi di IA sono capaci di adattare il proprio comportamento analizzando gli effetti delle azioni precedenti e lavorando in autonomia. Ci sono due tipi di Intelligenza Artificiale: software, come assistenti virtuali, software di analisi di immagini, motori di ricerca, sistemi di riconoscimento facciale e vocale, e intelligenza incorporata, che sono robot, veicoli autonomi, droni. L'Intelligenza Artificiale, quindi, è già applicata in diversi ambiti della nostra quotidianità senza rendercene conto. Alcuni esempi sono lo shopping in rete e le pubblicità, che utilizzano dati su acquisti e ricerche precedenti per suggerire prodotti che possono interessare all'utente, ricerche online, assistenti digitali personali, veicoli, traduzione automatica e case, città ed infrastrutture intelligenti[3].

Dunque possiamo definire l'Intelligenza Artificiale come la scienza che sviluppa l'architettura necessaria con lo scopo di far funzionare le macchine come il cervello umano, ovvero di creare dei computer che ragionino in modo simile, ma non uguale, all'essere umano. Ciò che mette in moto l'Intelligenza Artificiale è il *Machine Learning*. Il *Machine Learning* (o Apprendimento Automatico) è l'algoritmo che permette alle macchine di migliorarsi nel tempo aiutandole ad apprendere e migliorarsi in base all'esperienza e ai dati ricevuti. Intelligenza Artificiale e *Machine Learning* sono strettamente connessi tra loro, ma non sono la stessa cosa: il *Machine Learning* è un'applicazione dell'Intelligenza Artificia-

le. Ma più aumenta la complessità degli algoritmi per interpretare la realtà, più l'approccio all'Apprendimento Automatico si avvicina alla struttura delle reti neurali del cervello, diviene dunque un Apprendimento Profondo, o meglio noto come *Deep Learning* [4].

Recentemente è stato rivolto grande interesse al *Deep Learning* dovuto principalmente alle numerose conquiste in campo informatico, relative soprattutto alla sfera dell'hardware. Le architetture di *Deep Learning* sono state applicate nella *Computer Vision*, che ha lo scopo di riprodurre la vista umana e quindi essere in grado di rilevare oggetti, movimenti, azioni e stimare il posizionamento del corpo umano, ma anche nel riconoscimento automatico della lingua parlata, nell'elaborazione del linguaggio naturale, nel riconoscimento audio e nella bioinformatica, che cerca di spiegare fenomeni biologici attraverso l'utilizzo di strumenti informatici. Il *Deep Learning*, quindi, rappresenta una tecnica di Machine Learning che usa algoritmi in grado di simulare il cervello umano. Questi algoritmi si basano sullo sviluppo di reti neurali per apprendere e svolgere un'attività. Una rete neurale rappresenta un software che ha lo scopo di imitare il funzionamento del cervello umano consentendo ai programmi di riconoscere modelli e risolvere problemi.

Questa metodologia ha svariate applicazioni, ma tra le più importanti troviamo la diagnostica medica. Un dottore per poter fare una diagnosi deve avere conoscenze profonde ed esperienze pregresse, ebbene il *Deep Learning* può essere d'aiuto arricchendo, perfezionando e potenziando il bacino delle conoscenze del medico. Le macchine automatiche di traduzione esistono da tempo, ma commettono ancora molti errori e troppo spesso la traduzione non è perfetta. Grazie all'uso del *Deep Learning* questi limiti saranno superati in quanto la macchina sarà in grado di imparare gli errori frequenti, correggerli e quindi risultare più precisa. Ulteriori applicazioni recentemente sviluppate sono la colorazione delle immagini in bianco e nero, la generazione automatica di didascalie di immagini e la classificazione degli oggetti.

Inoltre, il *Deep Learning* trova applicazione nell'ambito della guida autonoma. Grazie all'aiuto di sensori e di telecamere capaci di elaborare le immagini, la guida autonoma consente di riconoscere gli ostacoli in entrambi i lati della carreggiata e di rilevare l'eventuale presenza di un pedone. Sarà capace di valutare un pericolo e di azionare in modo automatico il meccanismo di frenata. Queste auto, dunque, aiutano il conducente sotto diversi punti e, in un futuro prossimo, saranno in grado di circolare senza l'intervento umano e risulteranno essere anche più sicure. La computer vision in questo caso riproduce la vista umana, riconoscendo l'ambito nel quale si sta muovendo e fornendo tutte le indicazioni utili per muoversi in sicurezza.

Vi è grande interesse da parte delle aziende ad investire nel settore dell'automotive per svariate ragioni. La guida autonoma porterà sicuramente alla diminuzione degli incidenti in quanto i tempi di reazione di un veicolo autonomo sono inferiori rispetto all'uomo, garantendo dunque una maggiore affidabilità. Non

solo, questo, infatti, permetterebbe anche di ridurre la congestione stradale, di gestire efficientemente il flusso del traffico e, in aggiunta, c'è da considerare che i veicoli senza conducente possono procedere a velocità più elevate con minime possibilità di commettere errori, dunque il tempo dei viaggi sarebbe più breve. Ulteriore interesse da parte delle aziende del settore automobilistico riguarda lo sviluppo di metodi e strumenti che consentano non solo di riconoscere il grado di affaticamento e di distrazione del conducente, ma di avvisarlo in tempo dei pericoli a cui andrebbe incontro continuando il suo percorso alla guida. Questi dispositivi, presenti nella maggior parte dei casi nelle nuove autovetture, possono emettere un suono acustico oppure possono far comparire una scritta su un display in cui si invita l'autista a fare una pausa. Ad oggi possiamo trovare due tipologie di dispositivi [5]:

- Rilevatore di stanchezza a tempo. Questo rilevatore si basa su un timer e non misura effettivamente la stanchezza del guidatore. Se per un tempo di due ore consecutive non viene spento il motore dell'auto, il sistema ipotizza che il guidatore è stanco ed emette un segnale acustico accompagnato da una scritta sul display nella quale si consiglia la sosta. Il timer si azzerava quando il motore viene spento.
- Rilevatore di stanchezza reale. In questo caso viene effettivamente misurata la stanchezza del guidatore attraverso dei sensori che monitorano il volto del guidatore, analizzano il battito delle palpebre, tengono conto di eventuali sbadigli e di cambi di corsia senza aver inserito la segnalazione luminosa. Grazie a questi segnali il sistema valuta il grado di stanchezza del guidatore, ed eventuali consigli di sosta possono avvenire anche prima delle due ore alla guida.

Le autovetture di ultima generazione, sono equipaggiate di sensori tra cui un sistema di monitoraggio avanzato che, grazie all'uso di una telecamera montata sul cruscotto dell'auto rivolta verso il conducente, sono capaci di monitorare la sonnolenza o la distrazione del conducente ed emettere un avviso. La telecamera rileva gli occhi del conducente di giorno e di notte, anche se il conducente indossa degli occhiali da sole. Riesce a determinare se il conducente sta sbattendo le palpebre più del solito, se gli occhi si chiudono e se la testa si inclina in modo innaturale. Stabilisce se il conducente sta guardando la strada davanti a sé e se sta realmente prestando attenzione o se fissa la strada in modo distratto. Se il sistema ritiene che il conducente è distratto o assennato cerca di catturare la sua attenzione emettendo dei segnali audio, facendo accendere un indicatore sul cruscotto o facendo vibrare il sedile, e se i sensori esterni al veicolo ritengono che sta per avvenire una collisione, il sistema potrebbe applicare autonomamente i freni.



Lo scopo della seguente tesi è la progettazione e sviluppo di un algoritmo per il rilevamento della stanchezza di un conducente alla guida, in particolare nella realizzazione della parte software. L'oggetto principale su cui verte questa tesi è il rilevatore di stanchezza in tempo reale.

## 1.1 Obiettivi

Nel presente lavoro di tesi utilizziamo immagini RGB<sup>1</sup> e una *Neural Network*<sup>2</sup> per rilevare il livello di attenzione del guidatore in base a dei movimenti tipici dell'essere umano che si focalizzano sugli occhi, sulla bocca e sulla testa. Attraverso l'analisi dell'attenzione del guidatore, il sistema dovrà essere in grado di avvertire il guidatore nel caso in cui venga rilevato un suo livello di stanchezza pericolosamente elevato, in modo da prevenire eventuali incidenti causati da essa.

Per poter fare ciò, si utilizza il modulo *Mediapipe* sfruttando le potenzialità di *Facemesh*. A questi moduli si aggiunge la libreria *OpenCV* per lavorare e manipolare immagini e video [6]. Dopodiché andiamo ad addestrare una *Neural Network* per il riconoscimento del pattern, che ci serve ad identificare oggetti e segnali nei sistemi di controllo, facendo uso della libreria *Keras*.

Una *Neural Network* combina diversi livelli di elaborazione, utilizzando semplici elementi che operano in parallelo e sono ispirati ai sistemi nervosi biologici. In questo caso siamo nell'ambito della *Deep Learning* e della *Computer Vision*, che come già detto in precedenza, negli ultimi anni è sempre di più al centro dell'attenzione.

## 1.2 Struttura della Tesi

La parte essenziale del progetto di tesi è quella di sviluppare un sistema che riconosca, attraverso il movimento degli occhi, la stanchezza del guidatore in tempo reale per poi addestrarla attraverso una *Neural Network*.

Inizialmente si sviluppa una maschera del viso del guidatore utilizzando *Face-mesh* per poi evidenziare le parti che ci interessa analizzare con dei Landmarks. Attraverso un file *.csv*, creato manualmente, scorrendo *frame by frame* il video, segnaliamo il momento di chiusura e riapertura degli occhi con la posizione  $(x, y)$  in funzione dei frame che sono stati creati attraverso un file *.json*, intervallato in 3 istanti. In questo modo calcoliamo la velocità degli occhi per visualizzare la loro chiusura. Ulteriore step è quello di creare un bounding box per valutare l'eye-blinking e l'aspect ratio. Successivamente si sviluppa la *Neural Network*

---

<sup>1</sup>Dall'inglese Red Green Blue, è un modello di colori le cui specifiche sono state descritte nel 1936 dalla CIE (Commission internationale d'éclairage).

<sup>2</sup>modello di calcolo la cui struttura stratificata assomiglia alla struttura della rete di neuroni nel cervello, con strati di nodi connessi.

inizialmente testata su un Dataset già esistente, *MRL Eye Dataset* [7], che verrà spiegato in seguito. Andremo poi a testarlo con le immagini ricavate dal video in Facemesh valutando principalmente l'accuratezza ed infine lo valuteremo per tutta la durata del video.

Il presente lavoro di tesi è strutturato nei seguenti capitoli:

- introduzione;
- strumenti e metodi;
- analisi e sviluppo del modulo *software*;
- conclusioni e sviluppi futuri;
- appendice.

# Capitolo 2

## Anche le macchine pensano

L'Intelligenza Artificiale, in inglese *Artificial Intelligence* (AI), è il ramo della computer science che studia lo sviluppo di sistemi hardware e software dotati di capacità tipiche dell'essere umano ed in grado di perseguire autonomamente una finalità definita prendendo delle decisioni che, fino a quel momento, erano solitamente affidate agli esseri umani. Le capacità tipiche dell'essere umano riguardano, nello specifico, la comprensione ed elaborazione del linguaggio naturale e delle immagini, l'apprendimento, il ragionamento e la capacità di pianificazione e l'interazione con persone, macchine e ambiente. A differenza dei software tradizionali, un sistema IA non si basa sulla programmazione ma su tecniche di apprendimento: vengono cioè definiti degli algoritmi che elaborano un'enorme quantità di dati dai quali è il sistema stesso che deve derivare le proprie capacità di comprensione e ragionamento[8].

### 2.1 Intelligenza Artificiale: quando è nata?

In che anni si è cominciato a tentare di imitare la mente umana attraverso una macchina? Per come viene definita oggi, l'Intelligenza Artificiale nasce con l'avvento dei computer e la sua data viene fissata come il 1956. Proprio in quest'anno, infatti si parlò per la prima volta di Intelligenza Artificiale durante un convegno che si tenne ad Hanover, in America, e che vide la partecipazione di alcuni dei più importanti nomi di quella che sarebbe successivamente stata definita Intelligenza Artificiale, ma che allora veniva denominata Sistema Intelligente. Durante questo storico convegno, furono presentati alcuni programmi già capaci di effettuare alcuni ragionamenti logici, in particolar modo legati alla matematica. Il programma *Logic Theorist*, sviluppato da due ricercatori informatici, Allen Newell e Herbert Simon, era infatti in grado di dimostrare alcuni teoremi di matematica partendo da determinate informazioni [9].

In realtà è durante la Seconda Guerra Mondiale che nacque l'idea che l'intelligenza umana potesse essere simulata da una macchina. Ed è in quegli anni

che *Alan Turing*, uno dei più famosi informatici considerato uno dei padri dell'informatica moderna, pose le basi per concetti come calcolabilità e computabilità. In un articolo del 1950, *Computing Machinery and Intelligence* pubblicato sulla rivista *Mind*, descrisse un criterio per determinare se una macchina fosse intelligente o meno chiamato "*The Imitation Game*" o "*gioco dell'imitazione*". Questo metodo, anche noto con il nome "*Test di Turing*", si suddivide in due fasi [10].

Nella prima fase, tre persone, un uomo, una donna e un intervistatore (Fig. 2.1), sono in stanze separate e comunicano attraverso una telescrivente.

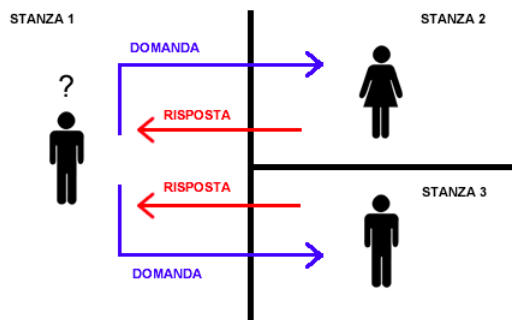


Figura 2.1: Rappresentazione della prima fase del *Test di Turing*

L'esaminatore pone delle domande alle due persone e, non vedendole, non sa se sta rivolgendo la domanda all'uomo o alla donna. Le domande hanno lo scopo di riuscire a stabilire chi sia l'uomo e chi sia la donna. I partecipanti al gioco ricevono la domanda e rispondono all'intervistatore in forma scritta, usando la telescrivente.

Fin qui il gioco sembra essere molto semplice e il problema di rapida soluzione. In realtà non è così perché i partecipanti al gioco possono anche mentire. I due partecipanti, infatti, hanno scopi diversi: un partecipante ha l'obiettivo di agevolare l'identificazione da parte dell'intervistatore, quindi è sincero nelle sue risposte; l'altro partecipante, invece, ha l'obiettivo di far sbagliare l'identificazione da parte dell'intervistatore, quindi fornisce risposte non veritiere.

Pertanto l'intervistatore non solo non sa chi sia l'uomo e chi la donna, ma non sa neanche chi dei due mente. Ripetendo in gioco  $N$  volte, l'intervistatore sbaglia il sesso dei partecipanti  $X$  volte, dunque il tasso di errore è pari a  $\frac{X}{N}$ .

Nella seconda fase del gioco sostituiamo uno dei due partecipanti con un computer (Fig. 2.2) e questa volta l'intervistatore deve capire se a rispondere è l'uomo o il computer.

Il processo è sempre lo stesso. L'intervistatore formula delle domande ai partecipanti tramite telescrivente e non può vederli. Non sa se sta parlando con due uomini o se uno di loro è una macchina. Alla fine del gioco dovrà identificare i partecipanti basandosi esclusivamente sulle loro risposte scritte. Il gioco si ripete

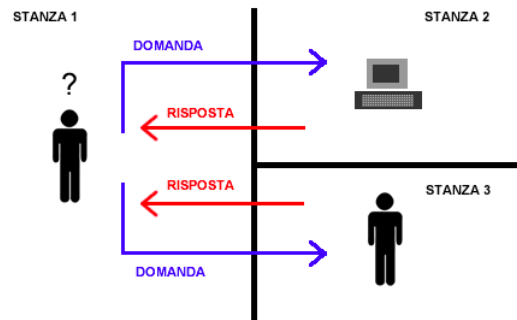


Figura 2.2: Rappresentazione della seconda fase del *Test di Turing*

$N$  volte e l'intervistatore sbaglia l'identificazione dei partecipanti per  $Z$  volte, ottenendo un tasso di errore percentuale pari a  $\frac{Z}{N}$ .

Il *Test di Turing* è superato se la percentuale di errore nel gioco in cui partecipa il computer è simile o inferiore a quella in cui partecipano solo gli umani, ovvero se  $\frac{X}{N} \simeq \frac{Z}{N}$ . Se il test è superato si può affermare che il computer è intelligente in quanto è indistinguibile dall'essere umano [11]. Il *Test di Turing* è da considerarsi come il manifesto dell'Intelligenza Artificiale.

In quel periodo storico vi furono notevoli scoperte scientifiche. Negli anni '40 i biologi svilupparono le prime teorie per spiegare come l'intelligenza e l'apprendimento fossero il risultato dei segnali trasmessi tra i neuroni del cervello umano. *Wiener* sviluppò le teorie cibernetiche di controllo e stabilità di reti elettriche, *Turing* elaborò la teoria del calcolo e *Shannon* la teoria dell'informazione [12]. Nel 1943 *McCulloch e Walter Pitts* [13] pubblicarono un lavoro in cui mostravano come un sistema di neuroni artificiali potesse essere in grado di eseguire funzioni logiche basilari. A partire dalle teorie di *McCulloch e Pitts*, *Rosenblatt* nel 1956 ideò la prima macchina in grado di simulare a livello software e hardware il funzionamento dei neuroni. Il sistema a cui diede vita è noto come *Perceptron*. Il *Perceptron* era un sistema dotato di due soli strati di neuroni: uno per l'input dei dati e l'altro per l'output dei risultati. *Rosenblatt* allora cercò di creare reti più complesse organizzandole in una gerarchia di molteplici strati: passando i dati da uno strato all'altro, sarebbe stato possibile creare *pattern* di pattern e risolvere problemi sempre più complessi. Nel 1969, *Minsky e Papert* [14] pubblicarono un saggio in cui dimostrarono che i network neurali fossero in grado di compiere solo alcune operazioni elementari, mentre non c'era speranza che portassero a termine compiti più complessi. Questo determinò la fine della ricerca sui *network neurali* fin quando nel 2012 [15] non nacque il *deep learning*.

## 2.2 Intelligenza Artificiale Forte e Debole

L'Intelligenza Artificiale è una disciplina moderna che riveste una importanza indubbia nel campo dell'informatica e non solo in quanto negli anni è stata fortemente influenzata da diversi settori quali la matematica, l'economia, le neuroscienze e la cibernetica.

Nel 1987 Somalvico [16], uno dei pionieri dell'intelligenza artificiale in Italia, ha definito l'Intelligenza Artificiale come quella disciplina che studia i fondamenti teorici, le metodologie e le tecniche che consentono di progettare sistemi hardware e sistemi di programmi *software* capaci di fornire all'elaboratore elettronico prestazioni che, a un osservatore comune, sembrerebbero essere di pertinenza esclusiva dell'intelligenza umana.

In realtà, non esiste una definizione univoca di IA e le interpretazioni possono variare a seconda della focalizzazione: da un lato, ci si può concentrare sui processi interni di ragionamento, dall'altro sul comportamento esterno dei sistemi, in linea di massima sempre prendendo come sorta di "misura di efficacia" la somiglianza o la vicinanza al comportamento umano. La comunità scientifica si è trovata d'accordo nel definire due differenti tipi di intelligenza artificiale, quella debole e quella forte.

### 2.2.1 L'Intelligenza Artificiale Debole

L'Intelligenza Artificiale debole racchiude al suo interno sistemi in grado di simulare alcune funzionalità cognitive dell'uomo senza tuttavia raggiungere le sue capacità intellettuali; si tratta, a grandi linee, di programmi di problem-solving in grado di replicare alcuni ragionamenti logici umani per risolvere problemi e prendere decisioni. Il suo obiettivo non è, quindi, quello di replicare completamente il funzionamento della mente umana. L'Intelligenza Artificiale Debole lavora indagando su casi simili, confrontandoli, elaborando possibili soluzioni e scegliendo quella più adeguata. In questo contesto entra in gioco la simulazione del comportamento umano. Le applicazioni basate sull'Intelligenza Artificiale Debole risultano essere perfette per suggerire all'uomo quali decisioni prendere poiché offrono più informazioni a supporto della propria scelta.

### 2.2.2 L'Intelligenza Artificiale Forte

La teoria dell'Intelligenza Artificiale Forte sostiene che le macchine siano in grado di sviluppare una coscienza di sé. La macchina non è solo uno strumento: se programmata opportunamente è essa stessa una mente dotata di una capacità cognitiva non distinguibile da quella umana. Alla base dell'Intelligenza Artificiale Forte ci sono una serie di programmi che attraverso una macchina vogliono riprodurre le prestazioni e le conoscenze delle persone esperte in un determinato campo. Dunque una macchina non solo può ragionare e risolvere problemi, ma

può diventare sapiente e autocosciente attuando processi di pensiero propri. Ci sono teorie che spingono alcuni scienziati ed esperti a ritenere che un giorno le macchine avranno una intelligenza propria, autonoma e probabilmente superiore a quella degli esseri umani. Questo potenziale momento è chiamato *Singularity*.

## 2.3 Tecniche e modelli di apprendimento

Prima della nascita dell'Intelligenza Artificiale, le macchine riuscivano ad eseguire delle istruzioni precise date in sequenza. Ciò che caratterizza l'Intelligenza Artificiale da un punto di vista tecnologico e metodologico è il metodo/modello di apprendimento con cui l'intelligenza diventa abile in un compito o azione. Questi modelli di apprendimento sono ciò che distinguono *Machine Learning* e *Deep Learning*.

### 2.3.1 Il *Machine Learning*

Il *Machine Learning* (o Apprendimento Automatico) è un sottoinsieme dell'Intelligenza Artificiale che si occupa di creare sistemi che apprendono in base ai dati che utilizzano. Questi sistemi “allenano” il software in modo che, correggendo gli errori, possa apprendere e svolgere in modo autonomo un compito o un'attività.

Il *Machine Learning* funziona in linea di principio sulla base di due distinti approcci che permettono di distinguere l'apprendimento automatico in due sottocategorie del ML a seconda del fatto che si diano al computer esempi completi da utilizzare come indicazione per eseguire il compito richiesto (apprendimento supervisionato) oppure che si lasci lavorare il software senza alcun “aiuto” (apprendimento non supervisionato).

Nell'apprendimento supervisionato si forniscono al computer dei set di dati come input e le informazioni relative ai risultati desiderati con l'obiettivo che il sistema identifichi una regola generale che colleghi i dati in ingresso con quelli in uscita in modo da poter poi riutilizzare tale regola per altri compiti simili.

Nell'apprendimento non supervisionato al sistema vengono forniti solo set di dati senza alcuna indicazione del risultato desiderato. Lo scopo di questo secondo metodo di apprendimento è “risalire” a schemi e modelli nascosti, ossia identificare negli input una struttura logica senza che questi siano preventivamente etichettati.

In realtà ci sono altri sottoinsiemi che consentono di fare un'ulteriore classificazione ancora più dettagliata del *Machine Learning* proprio in base al suo funzionamento, come ad esempio il *Machine Learning* con apprendimento per rinforzo e quello con apprendimento semi-supervisionato.

Nell'apprendimento per rinforzo il sistema deve interagire con un ambiente dinamico, che gli consente di avere i dati di input, e raggiungere un obiettivo, al raggiungimento del quale riceve una ricompensa, imparando anche dagli errori

identificati mediante punizioni. Il comportamento del sistema è determinato da una routine di apprendimento basata su ricompensa e punizione. Con un modello del genere, il computer impara per esempio a battere un avversario in un gioco (o a guidare un veicolo) concentrando gli sforzi sullo svolgimento di un determinato compito mirando a raggiungere il massimo valore della ricompensa; in altre parole, il sistema impara giocando (o guidando) e dagli errori commessi migliorando le prestazioni proprio in funzione dei risultati raggiunti in precedenza.

L'apprendimento semi-supervisionato si tratta di un modello ibrido dove al computer viene fornito un set di dati incompleti per l'apprendimento; alcuni di questi input sono dotati dei rispettivi esempi di output (come nell'apprendimento supervisionato), altri invece ne sono privi (come nell'apprendimento non supervisionato). L'obiettivo, di fondo, è sempre lo stesso: identificare regole e funzioni per la risoluzione dei problemi, nonché modelli e strutture di dati utili a raggiungere determinati obiettivi [17].

### 2.3.2 Il *Deep Learning*

Il *Deep Learning*, o apprendimento profondo, sottende a qualcosa di molto più ampio del semplice apprendimento su più livelli delle macchine. Il *Deep Learning* è una sotto-categoria del *Machine Learning* e indica quella branca dell'intelligenza artificiale che fa riferimento agli algoritmi ispirati alla struttura e alla funzione del cervello chiamate reti neurali artificiali.

Il *Deep Learning* [18] consente ai modelli computazionali composti da più livelli di elaborazione di apprendere rappresentazioni di dati con più livelli di astrazione. In particolare, scopre la struttura intricata in grandi set di dati usando l'algoritmo di *backpropagation* per indicare come una macchina dovrebbe modificare i suoi parametri interni che vengono utilizzati per calcolare la rappresentazione in ogni livello dalla rappresentazione nel livello precedente. Il *Deep Learning* sta facendo passi da gigante nel risolvere problemi aperti da anni e di interesse notevole per la comunità dell'Intelligenza Artificiale.

Il recente successo del *Deep Learning* è dovuto a fattori che hanno contribuito a colmare il vuoto in alcune aree che in passato non hanno permesso di raggiungere i risultati attesi come ad esempio l'incremento dei dati a disposizione, lo sviluppo di sistemi di calcolo parallelo altamente performanti basati su GP-GPU (General Purpose Graphics Processing Unit) e l'ottimizzazione dei metodi di addestramento delle reti neurali.

Tendenzialmente, si parla di *Deep Learning* e di reti neurali in modo interscambiabile, il che può creare confusione. Il termine "profondo" (deep) in *Deep Learning* si riferisce esclusivamente alla profondità dei livelli in una rete neurale. Una rete neurale che consiste in più di tre livelli può essere considerata un algoritmo di *Deep Learning*. Una rete neurale che ha solo due o tre livelli è soltanto una rete neurale di base.



## 2.4 Rete Neurale

Le Reti Neurali riflettono il comportamento umano, consentendo ai programmi informatici di riconoscere modelli e di risolvere problemi comuni nei campi dell'Intelligenza Artificiale, del *machine learning* e del *deep learning*.

Le Reti Neurali sono un sottoinsieme del *machine learning* e sono l'elemento centrale degli algoritmi di *deep learning*. La struttura delle Reti Neurali è ispirata al cervello umano: l'idea è di "imitare" il modo in cui i neuroni biologici si inviano segnali.

Le reti neurali fanno affidamento sui dati di addestramento per imparare e migliorare la loro accuratezza nel tempo. Una volta ottimizzati per l'accuratezza, questi algoritmi di apprendimento sono dei potenti strumenti nella computer science e nell'AI, consentendoci di classificare e organizzare in cluster i dati ad alta velocità.

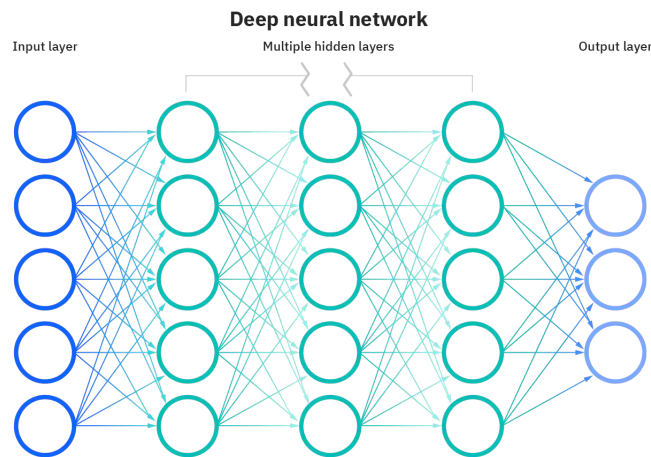


Figura 2.3: Schema di una Rete Neurale.

Le Reti Neurali sono composte da livelli di nodi che contengono un livello di input, uno o più livelli nascosti di input e di output (Fig. 2.3). Ciascun nodo si connette ad un altro e ha un peso e una soglia associati. Se l'output di qualsiasi singolo nodo è al di sopra della soglia specificata tale nodo viene attivato ed invia i dati al successivo livello della rete. In caso contrario non si passa alcun dato al livello successivo della rete.

Da un punto di vista matematico alla base delle Reti Neurali possiamo esprimere una funzione  $f(\mathbf{x})$  come composizione di altre funzioni  $g(\mathbf{x})$  che a loro volta possono essere espresse in funzioni più semplici. Dunque possiamo pensare ad una Rete Neurale come un insieme interconnesso di funzioni elementari in cui gli output sono gli input delle successive funzioni.

La prima rete neurale artificiale della storia è il Percettrone, che svolge il compito che il neurone svolge nelle reti neurali biologiche. Il percettrone nacque

da una prima bozza di neurone artificiale teorizzato da Warren McCulloch e Walter Pitts. In Figura 2.3 ogni pallino è un nodo che rappresenta un Percettrone. Questi nodi si comportano come funzioni che prendono  $n$  elementi in input e restituiscono solamente una singola uscita, che viene inviata come input per i Percettroni successivi. Per tal motivo parliamo di Reti Neurali Stratificate.

## 2.5 Modello McCulloch-Pitts

Come abbiamo già osservato in precedenza, il primo modello di Percettrone è dovuto a McCulloch e Pitts [13], e per tal motivo il modello è detto Modello di McCulloch-Pitts (Fig. 2.4).

Il modello del Percettrone McCulloch-Pitts rappresenta la prima base teorica di un neurone artificiale.

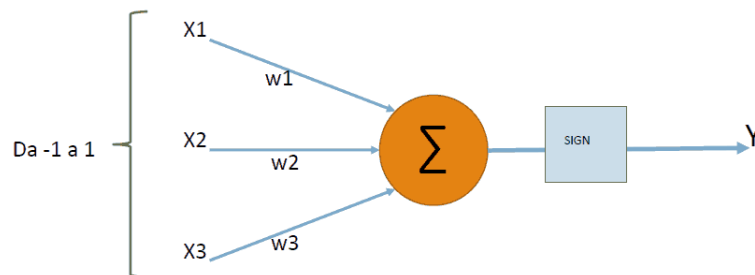


Figura 2.4: Modello McCulloch-Pitts, Struttura Percettrone.

La struttura del Percettrone è del tutto analoga a quella di un neurone biologico. Come input abbiamo i dendriti assieme alle sinapsi. Il nucleo e gli assoni costituiscono l'uscita del segnale dal neurone per passare ad interagire con altri neuroni.

Gli elementi che costituiscono il Percettrone sono:

- almeno due o più connessioni di input;
- un nucleo di calcolo o sommatore;
- un uscita o anche chiamata attivazione.

I valori di input e output possibili sono in virgola mobile e vanno da -1 a 1.

L'input del Percettrone è formato da un vettore di numeri reali in cui di volta in volta andremo ad inserire valori di input  $(x_1, x_2, \dots, x_n)$ . Inoltre avremo un ulteriore array dei pesi indicati con  $w_1, w_2, \dots, w_n$ . Per ottenere il vero e proprio dato eseguiremo una semplice operazione  $x_n * w_n$  per ogni  $x_n$  di input. Formalizzando il concetto avremo:

$$y = \left( \sum_{i=1}^n x_i w_i \right) + b \quad (2.1)$$

o analogamente :

$$y = \mathbf{w}^T \cdot \mathbf{x} + b,$$

dove con il simbolo  $*$  denotiamo il prodotto scalare.

Il termine  $b$  viene chiamato *bias* del perceptrone ed è considerato un peso a tutti gli effetti. Il suo compito è quello di modificare aumentando o diminuendo la soglia di attivazione che deriva dalle funzioni di attivazione ovvero si tratta di una funzione che invia un segnale solo se il livello di output del perceptrone supera una certa soglia (un esempio di funzione di attivazione è la funzione di Heaviside o meglio conosciuto funzione a gradino unitario).

Concludiamo generalizzando la formula (2.1) che rappresenta il nostro perceptrone. Sia  $\phi$  la funzione di attivazione, allora abbiamo

$$y = \phi \left( \sum_{i=1}^n x_i w_i \right) + b.$$

## 2.6 Back-Propagation

Le Reti Neurali sono in generale argomenti dell'Intelligenza Artificiale, e più precisamente del *Machine Learning* e *Deep Learning*, quindi possiamo aspettarci che tramite un corretto addestramento si ottengano dei risultati ottimi.

In questo lavoro ci si è occupati proprio di addestrare una rete neurale, infatti per farlo possiamo distinguere due sezioni distinte:

- *Model Evaluation*. Durante questa fase i pesi e i *bias* della Rete Neurale vengono mantenuti invariati e la rete effettua operazioni di predizione su una parte del dataset di riferimento. Rimandiamo al capitolo successivo per l'analisi dei Dataset. In questa sezione viene definita Batch ed è considerata una quantità arbitraria.
- *Model Training*. Una volta completata la fase di *Model Evaluation*, la rete ha a disposizione una serie di valori e predizioni desiderate. In questa fase gli input e gli output restano invariati e la rete effettua delle operazioni di aggiornamento dei pesi e *bias*.

Queste due operazioni vengono fatte fino alla fine del dataset di addestramento. A questo punto diremo che la rete neurale ha svolto un'epoca o epoch di addestramento. Il batch size, cioè il numero di esempi contenuti in ogni batch, e il numero di epoche di addestramento rappresentano due iperparametri che vanno ad influenzare il modo in cui la rete apprende e verranno trattati più nel particolare dopo aver descritto la *Gradient Descent*.

### 2.6.1 Gradient Descent

Nella fase di *Model Training* i pesi e i bias della Rete Neurale vengono aggiornati in modo da ottimizzare i risultati ottenuti. Questo processo viene detto *Gradient Descent* (o Discesa del Gradiente) ed è il metodo principale per ottimizzare le prestazioni di una rete neurale riducendo il tasso di perdita/errore della rete.

La *Gradient Descent* è un algoritmo di ottimizzazione che viene utilizzato per migliorare le prestazioni di una Rete Neurale apportando modifiche ai parametri della rete in modo tale che la differenza tra le previsioni della rete e i valori effettivi o previsti siano minimi.

### 2.6.2 Valutazione degli iperparametri

Nello studio della *back-propagation* abbiamo reso noti degli iperparametri i quali *Learning Rate*, la dimensione del batch, e il numero di epoche.

Analizziamo questi parametri ed indaghiamo in che modo influenzano l'apprendimento del modello.

- Il *Learning Rate* è un iperparametro che ci dice quanto velocemente la funzione di costo andrà a convergere o andrà verso il suo minimo.
- La dimensione del batch è un iperparametro che definisce il numero di campioni da elaborare prima di aggiornare i parametri del modello interno. Possiamo pensare un batch come un ciclo `for` che va a iterare uno o più campioni e fa previsioni. Alla fine del batch le previsioni vengono confrontate con le variabili di output previste e viene calcolato un errore. Un set di dati di training può essere suddiviso in uno o più batch. Quando la dimensione del batch è più di un campione e inferiore alla dimensione del set di dati di addestramento, l'algoritmo viene chiamato discesa del gradiente mini-batch.

– *Batch Gradient Descent*:

Dimensione del lotto = Dimensione del set di allenamento.

– *Stochastic Gradient Descent*:

Dimensione del lotto = 1.

– *Mini-Batch Gradient Descent*:

$1 < \text{Dimensione batch} < \text{Dimensione del set di allenamento}$ .

- Numero di Epoche è un iperparametro che definisce il numero di volte in cui l'algoritmo di apprendimento funzionerà attraverso l'intero set di dati di addestramento. Epoca o Epoch significa che ogni campione nel set di dati

di addestramento ha avuto l'opportunità di aggiornare i parametri del modello interno. un'epoca è composta da uno o più batch. Il numero di epoche troppo elevato può portare a problemi di *overfitting* (Fig. 2.5) ovvero la situazione in cui la rete diventa troppo sensibile alla variazione parametrica e di conseguenza perde accuratezza nell'applicazione. In caso opposto abbiamo *underfitting* ovvero l'eccessiva generalità della rete dovuta ad un allenamento su dataset ristretto ad un numero di epoche di allenamento insufficienti.

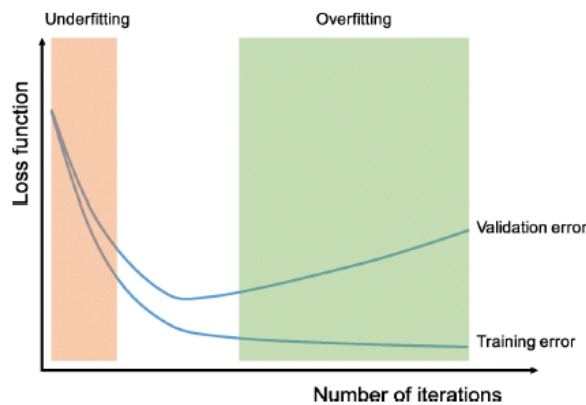


Figura 2.5: Zone di overfitting e underfitting in funzione del costo.

## 2.7 Reti Neurali Convoluzionali

Una tipologia di architettura di Rete Neurale è la Rete Neurale Convoluzionale.

Le *Convolutional Neural Network* (o Reti Neurali Convoluzionali) è un tipo di rete *feed-forward*<sup>1</sup>. Le Reti Neurali Convoluzionali sono una classe di Reti Neurali Artificiali diventate sempre più importanti nei vari compiti di visione artificiale.

La *Convolutional Neural Network* è progettata per apprendere automaticamente le gerarchie spaziali tramite la *back-propagation* utilizzando dei blocchi predefiniti come i livelli di pooling, i livelli di convoluzione e l'unità lineare rettificata (ReLU).

Possiamo affermare che la *Convolutional Neural Network* è un modello di apprendimento profondo per l'elaborazione dei dati e ha uno schema a griglia come le immagini.

Le *Convolutional Neural Network* (Fig. 2.6) sono costituite da neuroni che hanno pesi e bias apprendibili. Ogni neurone riceve alcuni input, esegue un prodotto scalare che lo trasforma in una funzione di attivazione. L'intera rete

<sup>1</sup>ispirata all'organizzazione della corteccia visiva

esprime un'unica funzione: dai pixel dell'immagine grezza su un'estremità, ai punteggi delle classi dall'altra.

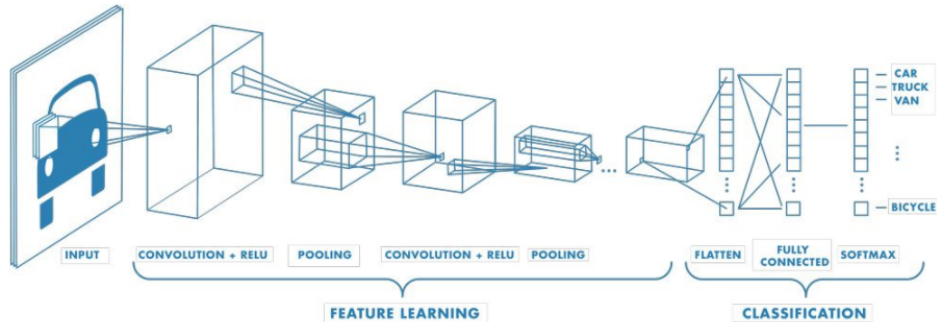


Figura 2.6: Esempio di una rete con numerosi layer convoluzionali. A ciascuna immagine di addestramento vengono applicati dei filtri a diverse risoluzioni e l'output di ciascuna immagine viene utilizzato come input per il layer successivo.

Il *Convolutional Layers* (o Livello convoluzionale) è l'elemento fondamentale di una rete convoluzionale che fa la maggior parte del lavoro pesante di calcolo. Esso prende il nome dall'operazione di convoluzione che viene fatta con le matrici di input e il filtro o kernel da cui si ottiene la *feature map*<sup>2</sup>.

Il *layer* può essere costituito da uno o più filtri disposti in cascata. Il suo obiettivo è quello di individuare schemi che vengono raffigurati in un'immagine con elevata precisione. Maggiore è il loro numero e maggiore è la complessità della caratteristica che riescono ad individuare.

<sup>2</sup>o mappa di caratteristica ovvero i nodi di una rete neurale normale tentano di classificare

Il *Layer non lineare* è un layer di neuroni che applica funzioni di attivazioni di non linearità come la *Rectified Linear Unit*, ovvero l'Unità Lineare Rettificata. Questa è una funzione di attivazione definita come la parte positiva del suo argomento (neurone) (Fig. 2.7).

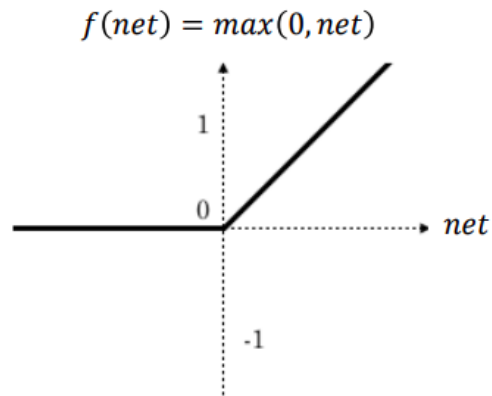


Figura 2.7: Funzione di attivazione

Il *Layer di Pooling* riduce la dimensione spaziale, ha il fine di diminuire la quantità dei parametri della rete. Ciò si ottiene partizionando la matrice di input in un insieme di sotto regioni non sovrapposte e trovando il valore massimo dei pixel in ciascuna regione: si parla di *max-pooling*. Il *max-pooling* è un'operazione che si basa sulla sua valutazione sul valore massimo presente all'interno di una determinata area del tensore<sup>3</sup> definita dalla *sliding windows*<sup>4</sup>.

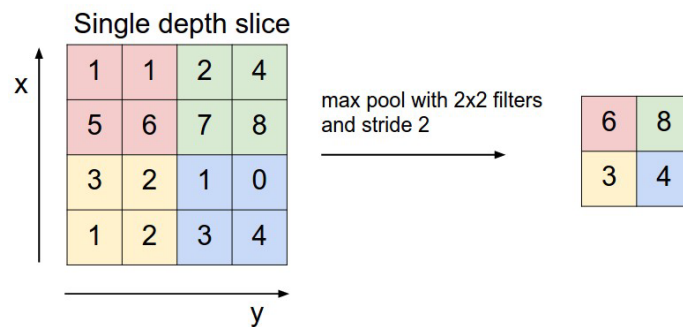


Figura 2.8: visualizzare un Max Pooling.

In figura 2.8 si mostra come l'algoritmo posizioni la finestra all'inizio del tensore ed inserisca il suo valore massimo nel nuovo tensore. La finestra poi viene fatta scorrere e il processo ricomincia.

<sup>3</sup>nozione matematica che viene definito a partire da uno spazio vettoriale e un campo, ai fini dell'analisi delle CNN le introduciamo come strutture dati

<sup>4</sup>Rappresenta la dimensione e la forma della porzione di tensore in cui il percettore viene connesso



# Capitolo 3

## Strumenti Utilizzati

In questo capitolo approfondiamo gli strumenti utilizzati quali *MediaPipe Face Mesh*, un dataset preimpostato per l'addestramento della rete neurale, la *MobileNetV2* e le Librerie.

### 3.1 MediaPipe Face Mesh

*MediaPipe Face Mesh* [19] è una soluzione per la geometria del viso in cui troviamo una stima di 468 punti di riferimento del viso, chiamati landmarks, real-time (Fig. 3.1).

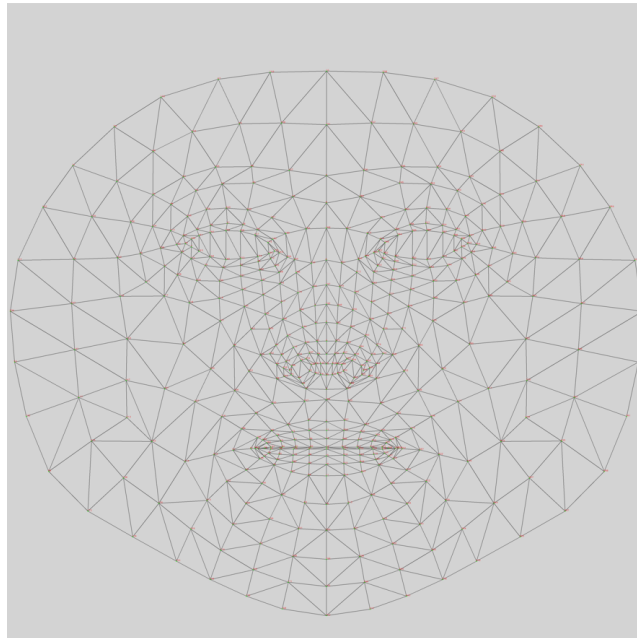


Figura 3.1: Maschera facciale di Face Mesh.

Questa soluzione offre prestazioni in tempo reale fondamentali per le esperienze online.

*MediaPipe* permette di eseguire diverse azioni e tra queste ci sono:

- Rilevamento facciale e mesh facciale;
- Posa e rilevamento olistico;
- Rilevamento e tracciamento di oggetti;
- Stima della Posa;

## 3.2 Dataset

Con il termine *Dataset* andiamo ad indicare l'insieme dei dati con cui il modello verrà addestrato. In questo elaborato abbiamo utilizzato più di un *Dataset* per l'implementazione della *Neural Network*. Inizialmente, per il rilevamento degli occhi e delle parti del viso, per la stima dello sguardo e la frequenza di ammiccamento degli occhi (*eye blinking*), che sono tutte mansioni fondamentali della visione artificiale, abbiamo preso in considerazione un *Dataset* già esistente: *MRL Eye Dataset* (Fig. 3.2)[7]. Questo *Dataset* è un set di dati su larga scala delle immagini dell'occhio umano. La particolarità di questo set di dati è che le immagini sono suddivise in diverse categorie, il che le rende più adatte per l'addestramento.

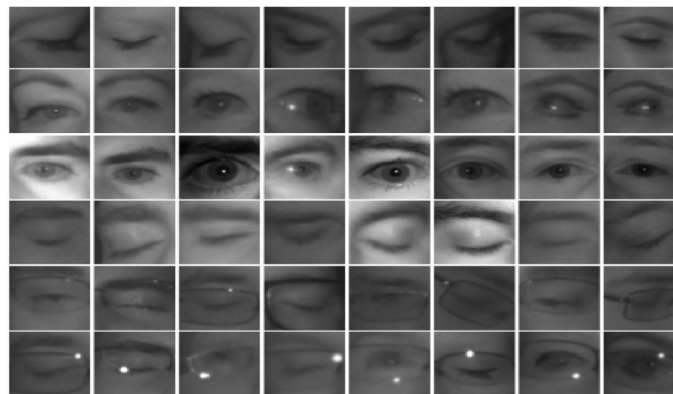
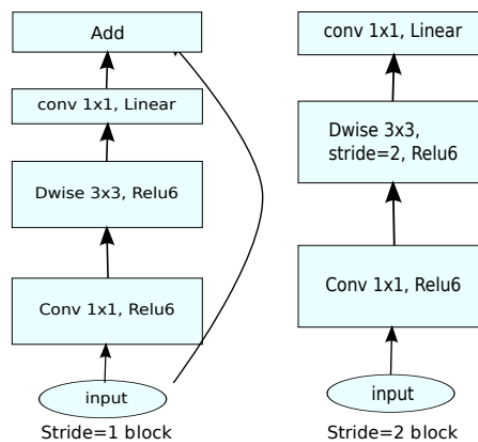


Figura 3.2: Esempio del Dataset MRL.

Per completare l'addestramento della rete neurale abbiamo utilizzato un dataset contenente immagini ricavate da frame di video da noi registrati.

## 3.3 MobileNetV2

*MobileNetV2* è un'architettura di rete neurale convoluzionale.<sup>1</sup> Si basa su una struttura residua invertita in cui le connessioni residue sono tra gli strati del collo di bottiglia. Lo strato di espansione intermedio utilizza leggere circonvoluzioni in profondità per filtrare le caratteristiche come fonte di non linearità. Nel complesso, l'architettura di *MobileNetV2* contiene il livello iniziale di convoluzione completa con 32 filtri, seguito da 19 livelli residui di collo di bottiglia. [20] (Fig. 3.3)



(d) Mobilenet V2

Figura 3.3: Struttura della MobileNetV2.

## 3.4 Librerie

### 3.4.1 OpenCV

*OpenSource Computer Vision Library* o *OpenCV* è una libreria *software open-source* che opera nell'ambito della *Computer Vision* e del *Machine Learning*. Per lo sviluppo del progetto di cui ci siamo occupati, vengono utilizzate principalmente librerie per la visualizzazione delle immagini.

<sup>1</sup>servono per estrarre le caratteristiche delle immagini o video, utilizzando come operatore principale le convoluzioni

### 3.4.2 Keras

*Keras* è una libreria *opensource* che serve per l'apprendimento automatico e per l'addestramento delle Reti Neurali, usando il linguaggio *Python*.

Lo scopo di tale libreria è quello di permettere la configurazione di reti neurali; *Keras* non funge da *Framework*, ma da interfaccia semplice (API) per l'accesso e programmazione a diversi *Framework* di apprendimento automatico.

Tra questi *Framework* supporta le librerie *TensorFlow*, *Microsoft Cognitive Toolkit* (in precedenza CNTK) e *Theano*.<sup>2</sup>

Questa libreria mette a disposizione dei componenti fondamentali su cui si va poi a sviluppare modelli complessi di apprendimento automatico.

In questo elaborato si fa uso della libreria *Keras* [21] per andare ad addestrare la *Neural Network*.

## 3.5 Linguaggi utilizzati

Il principale linguaggio di programmazione utilizzato in questo lavoro è stato *Python*.

*Python* [22] è un linguaggio dinamico orientato agli oggetti e il suo utilizzo è diffuso in molte aree dell'informatica, dalla creazione di applicazioni distribuite, computazione numerica etc.

Nasce agli inizi degli anni novanta dal suo creatore Guido Van Rossum ed è un linguaggio multi-paradigma che come caratteristiche fondamentali ha la flessibilità, la dinamicità e semplicità. Due peculiarità di *Python* sono l'importanza dell'indentazione e le variabili non tipizzate, tanto che il controllo dei tipi viene fatto a *runtime*<sup>3</sup>.

---

<sup>2</sup>Libreria per la computazione numerica per sviluppare codice *Python*. La sintassi del *Theano* è molto simile a *Numpy*

<sup>3</sup>Il codice *Python* non ha bisogno di essere compilato in quanto è un linguaggio di programmazione interpretato il cui codice viene interpretato a runtime

## 3.6 Editor di Sviluppo

Per sviluppare il nostro progetto abbiamo utilizzato l'editor chiamato *VisualStudio code* con cui siamo andati a sviluppare un codice in *Python*. Dopodichè per l'addestramento della *Neural Network* si è usato *Google Colab*.

## 3.7 VisualStudio Code

Per lo sviluppo del progetto si è utilizzato l'editor di *VisualStudio Code* (Fig. 3.4). [23] Si tratta di un *IDE cross-platform*<sup>4</sup> compatibile con *windows*, *linux*, *MacOS* e permette di utilizzare qualsiasi linguaggio ed avere un supporto per il *debugging*, controllo *Git* e *IntelliSense*<sup>5</sup>. VisualStudio Code è gratuito ed *opensource*.

Il funzionamento di questo editor si appoggia su *Electron* che è un noto *framework* con cui è possibile realizzare applicazioni *Node.js*

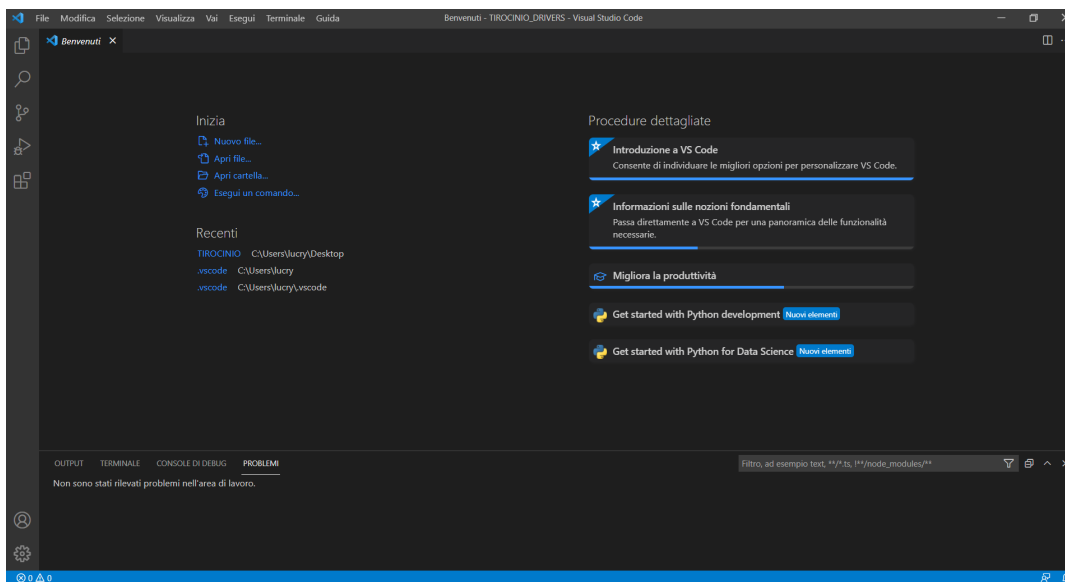


Figura 3.4: Interfaccia Iniziale VisualStudio Code.

<sup>4</sup>software o hardware che può essere utilizzato con più piattaforme

<sup>5</sup>completamente automatico delle istruzioni

## 3.8 Google Colab o Colaboratory

*Google Colab* è una piattaforma che, dopo aver creato un account Google, ci permette di eseguire codice direttamente sul Cloud.

Per scrivere un codice con *Google Colab*, si utilizza il *Jupyter Notebook* [24], ovvero documenti interattivi in cui possiamo andar a scrivere un codice; più precisamente questi documenti permettono di suddividere il codice scritto in celle per eseguire e spiegare il codice stesso. [25] Il Linguaggio più utilizzato su Colab è Python.

Analizziamo in dettaglio cosa ci possiamo fare con questa piattaforma:

- scrivere ed eseguire codice in Python,
- documenta il tuo codice che supporta equazioni matematiche,
- Integra *PyTorch*, *TensorFlow*, *Keras*, *Opencv*,
- uso della GPU.

In conclusione possiamo dire che l'introduzione di *Colaboratory* (Fig. 3.5) ha semplificato l'apprendimento e lo sviluppo di molte applicazioni di *Machine Learning*.

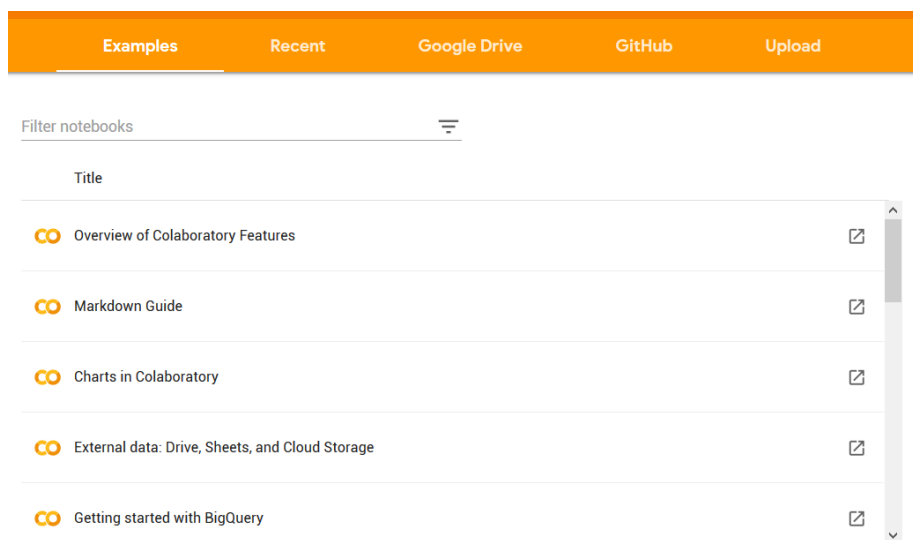


Figura 3.5: Interfaccia Iniziale Google Colab.

### 3.8.1 TensorFlow

*TensorFlow* è una libreria software *opensource end-to-end* per l'Intelligenza Artificiale che aiuta a sviluppare e addestrare modelli ML. Fa uso di API intuitive come *Keras* fondamentale per lo sviluppo di Reti Neurali e sono ad esecuzione rapida e rende l'iterazione del modello immediata e con un facile debug. *TensorFlow* è stato sviluppato da Google nel progetto *Google Brain* (AI). [26] Ad oggi questa libreria viene utilizzata in molti ambiti della *Deep learning*. "Cosa rappresenta *TensorFlow*"? Il nome è composto da due termini:

- *Tensor* (o Tensore in algebra lineare) é un array multidimensionale ossia una matrice di tre o più dimensioni.
- *Flow* è un flusso di operazioni.





# Capitolo 4

## Sviluppo del Progetto

L'obiettivo del progetto, ricapitolando, è quello di addestrare una rete neurale per renderla in grado di percepire, in base alle sue espressioni facciali, la stanchezza e/o la poca concentrazione del conducente durante la guida del veicolo.

In questo elaborato ci soffermeremo sulla prima parte del progetto, l'analisi di un video e la raccolta di dati da esso, ricavandone poi alcuni grafici che risulteranno utili all'addestramento della rete neurale (che è stata approfondita, come già detto, dalla mia collega, Lucrezia).

### 4.1 Estrapolare dati da un video

Il primo passo da compiere all'interno del progetto è quello di essere in grado di trasformare un video nel quale è presente, tra le altre cose, il volto di un conducente di un veicolo in dati che la macchina può utilizzare e analizzare per dichiarare il livello di attenzione del soggetto. Partendo dal video salvato in locale, per poter ottenere i dati da cui derivano poi i grafici finali sono stati necessari diversi passaggi.

In primo luogo, è stata applicata la maschera facciale di Face Mesh al volto del soggetto nel video, in questo modo è stato possibile individuare i punti di interesse del viso e la loro posizione attraverso le coordinate dei singoli landmarks.

Nella maschera facciale di Face Mesh, come è già stato detto, sono presenti 468 landmarks, che sono decisamente troppi per raggiungere il nostro fine. Il passo successivo, quindi, è stato quello di selezionare i landmark necessari ad identificare le nostre parti d'interesse: l'occhio destro, l'occhio sinistro e le labbra. Da ogni landmark è possibile ricavare la posizione sull'asse X e la posizione sull'asse Y all'interno del video.

Le coordinate X e Y dei landmarks rappresentano i dati che la macchina riceve durante l'esecuzione del video da parte del programma da cui origineranno i grafici che dovrà realizzare. Per modellare i dati, però, è necessario poterli

salvare e recuperare. Per questo motivo, all'avvio del video, viene creato un file di tipo JSON nel quale vengono inseriti i dati necessari.

A questo punto avremo i nostri dati di input da cui scaturiranno i grafici desiderati e utili all'addestramento della rete.

## 4.2 Approfondimento delle Classi

In questa sezione analizzeremo le singole classi nello specifico chiarendo quali sono i loro output e l'obiettivo di ognuna.

Le classi, che sono state programmate in linguaggio *Python* attraverso l'editor *Visual Studio Code*, sono le seguenti:

- `myLandmarks.py`, contiene i landmarks da tenere in considerazione per identificare le parti d'interesse del viso
- `facemesh.py`, riproduce il video con i bounding boxes degli occhi e delle labbra e, infine, restituisce un grafico che indica i momenti di chiusura degli occhi;
- `evolution.py`, riporta i grafici delle posizioni degli occhi e delle labbra
- `cnt.py`, riporta i grafici delle posizioni, della velocità, dell'eye-blinking e dell'aspect ratio;

### 4.2.1 `myLandmarks.py`

In questa classe vengono selezionati i landmarks che andranno a delineare l'occhio destro, l'occhio sinistro e le labbra del soggetto. Per i due occhi abbiamo scelto 16 landmarks per ognuno, mentre per le labbra sono 87 landmarks.

```

1  #lips
2  lipsUpperOuter = [61, 185, 40, 39, 37, 0, 267, 269, 270, 409, 291]
3  lipsLowerOuter = [146, 91, 181, 84, 17, 314, 405, 321, 375, 291]
4  lipsUpperInner = [78, 191, 80, 81, 82, 13, 312, 311, 310, 415, 308]
5  lipsLowerInner = [78, 95, 88, 178, 87, 14, 317, 402, 318, 324, 308]
6  lipsup1 = [76, 184, 74, 73, 72, 11, 302, 303, 304, 408, 306]
7  lipslow1 = [76, 77, 90, 180, 85, 16, 315, 404, 320, 307, 306]
8  lipsup2 = [62, 183, 42, 41, 38, 12, 268, 271, 272, 407, 292]
9  lipslow2 = [62, 96, 89, 179, 86, 15, 316, 403, 319, 325, 292]
10 LIPS = lipsUpperOuter + lipsLowerOuter + lipsUpperInner + lipsLowerInner +
    lipsup1 + lipslow1 + lipsup2 + lipslow2
11
12 #right eye
13 rightEyeUpper0 = [246, 161, 160, 159, 158, 157, 173]
14 rightEyeLower0 = [33, 7, 163, 144, 145, 153, 154, 155, 133]
15 R_EYE = rightEyeUpper0 + rightEyeLower0
16
17 #left eye
18 leftEyeUpper0 = [466, 388, 387, 386, 385, 384, 398]
19 leftEyeLower0 = [263, 249, 390, 373, 374, 380, 381, 382, 362]
20 L_EYE = leftEyeUpper0 + leftEyeLower0

```

Inoltre, all'interno di questo file, vengono definite le connessioni tra i vari landmarks in modo da definire i contorni delle parti interessate allo studio.

```

1 LIPS_CONNECTIONS = frozenset([
2 (61, 146), (146, 91), (91, 181), (181, 84), (84, 17), (17, 314), (314, 405),
3 (405, 321), (321, 375), (375, 291), (61, 185), (185, 40),
4 (40, 39), (39, 37), (37, 0), (0, 267), (267, 269), (269, 270), (270, 409),
5 (409, 291), (78, 95), (95, 88), (88, 178), (178, 87),
6 (87, 14), (14, 317), (317, 402), (402, 318), (318, 324), (324, 308), (78,
7 191), (191, 80), (80, 81), (81, 82), (82, 13), (13, 312),
8 (312, 311), (311, 310), (310, 415), (415, 308) ])
9
10 LEFTEYE_CONNECTIONS = frozenset([# Left eye.
11 (263, 249), (249, 390), (390, 373), (373, 374), (374, 380), (380, 381),
12 (381, 382), (382, 362), (263, 466), (466, 388), (388, 387),
13 (387, 386), (386, 385), (385, 384), (384, 398), (398, 362),
14 # Left eyebrow.
15 (276, 283), (283, 282), (282, 295), (295, 285), (300, 293), (293, 334),
16 (334, 296), (296, 336) ])
17
18 RIGHTEYE_CONNECTIONS = frozenset([# Right eye.
19 (33, 7), (7, 163), (163, 144), (144, 145), (145, 153), (153, 154), (154,
20 155), (155, 133), (33, 246), (246, 161), (161, 160), (160, 159),
21 (159, 158), (158, 157), (157, 173), (173, 133),
22 # Right eyebrow.
23 (46, 53), (53, 52), (52, 65), (65, 55), (70, 63), (63, 105), (105, 66), (66,
24 107)])

```

### 4.2.2 facemesh.py

Eseguito questa classe avvieremo la riproduzione del video in bianco e nero che conterrà i bounding boxes degli occhi e della bocca, uno interno ed uno esterno, il contatore del numero di frame, l'aspect ratio e lo stato della bocca e degli occhi (aperti o chiusi). Durante lo scorrimento del video, il contatore cnt aumenterà di



Figura 4.1: Screenshot del video riprodotto con facemesh.py

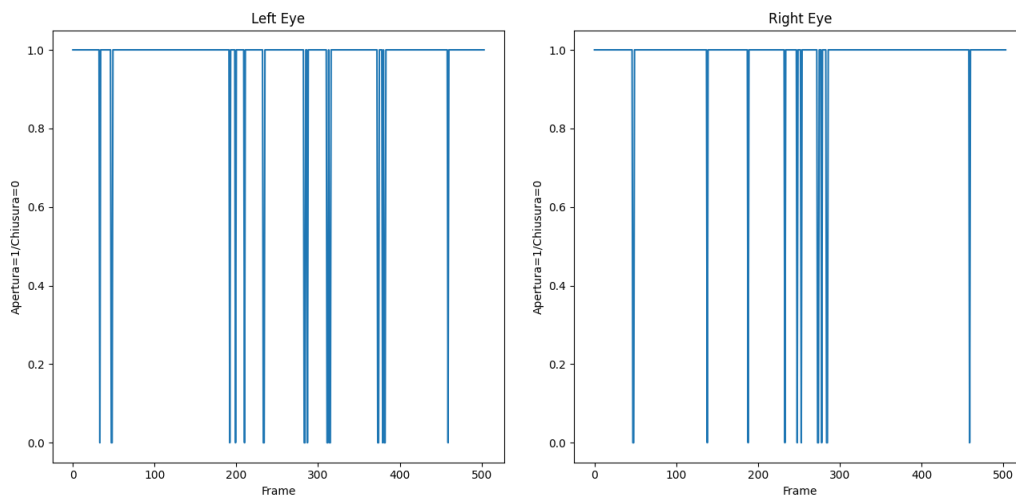


Figura 4.2: Grafici apertura e chiusura occhi

uno ad ogni frame.

Il termine *ar*, invece, indica l'*aspect ratio* dell'occhio sinistro. Tale valore è indice di quanto l'occhio è aperto e viene calcolato attraverso la formula:

```
1 round((y1_left - y2_left) / (x2_left - x1_left) , 2)
```

dove *y1\_left* indica la posizione del landmark dell'occhio sinistro più in alto, *y2\_left* di quello più in basso, *x1\_left* di quello più a sinistra e *x2\_left* di quello più a destra.

I termini *left*, *right* e *open\_lips* indicano rispettivamente se l'occhio sinistro, l'occhio destro e le labbra vengono rilevati come aperti o chiusi dal programma; se hanno valore 1, allora vengono considerati aperti, mentre un valore pari a 0 indica che sono chiusi. Per indicare se l'occhio o la bocca è aperto o chiuso inizialmente si è utilizzato l'*aspect ratio*: sotto un certo valore di *aspect ratio* la parte del viso veniva considerata chiusa, altrimenti era valutata aperta. Di seguito, dopo l'addestramento della rete neurale, anziché l'*aspect ratio*, si utilizza, appunto, la rete per valutare l'apertura o la chiusura degli elementi del volto.

Una volta terminata la riproduzione del video, compariranno i grafici relativi all'apertura e chiusura degli occhi. In questi grafici si riesce ad identificare con precisione i frame nei quali il sistema percepisce una chiusura degli occhi.

### 4.2.3 evoultion.py

La classe *evoultion.py* fornisce in output i grafici relativi alla posizione di ogni landmark degli elementi d'interesse del viso, l'occhio sinistro, l'occhio destro e le labbra. Per ogni elemento vengono stampati due grafici, uno relativo alla posizione nell'asse X e l'altro relativo alla posizione nell'asse Y. Da questi grafici

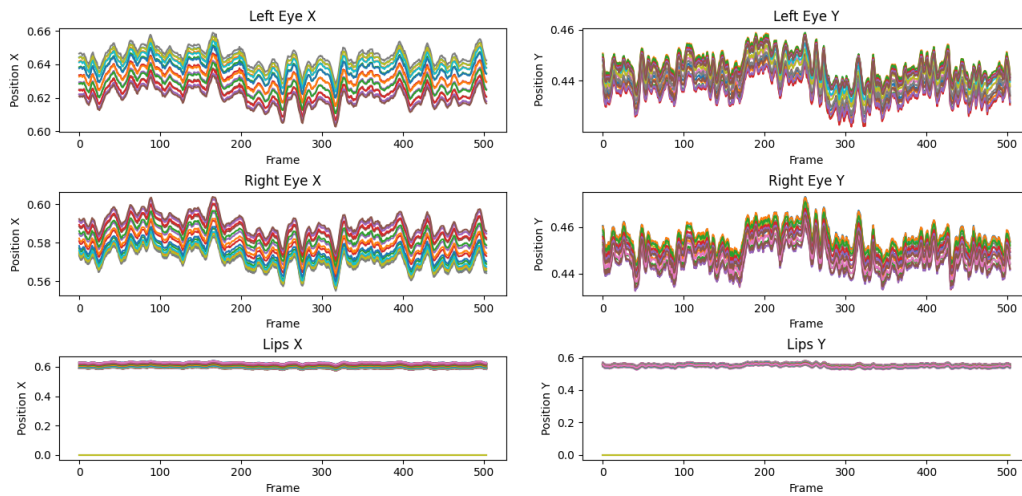


Figura 4.3: Grafici di evolution.py

si potrebbe dedurre i momenti (frame) in cui si ha una chiusura degli occhi o un'apertura della bocca. Ad esempio, per quanto riguarda gli occhi, i momenti in cui il grafico si restringe sta ad indicare che i landmarks si avvicinano tra loro, ovvero che l'occhio si sta chiudendo. Dei metodi migliori per valutare la loro apertura o chiusura sono attraverso l'*eye-blinking* e l'*aspect ratio*.

Per ottenere questi grafici, si è fatto leggere al programma i dati che sono stati salvati all'interno del file JSON. I dati presenti in questo file hanno fornito al programma i valori delle posizioni X e Y in ogni frame e sono stati inseriti in un array.

```

1 #LEGGO IL JSON
2 with open('list.json','r') as outline:
3     data = json.load(outline)
4
5 #LEGGO I VALORI X,Y DI LEFT EYE DAL JSON E INIZIALIZZO DUE ARRAY
6 x_array = np.zeros((len(data), LEFT_EYE_DIM))
7 y_array = np.zeros((len(data), LEFT_EYE_DIM))
8 for i in data:
9     frame = i['frame']
10    for t,y in enumerate(i['leftEye']):
11        x_array[frame-1,t] = y['x']
12        y_array[frame-1,t] = y['y']
13
14 #STESSO DISCORSO PER RIGHT EYE
15 x_array2 = np.zeros((len(data), RIGHT_EYE_DIM))
16 y_array2 = np.zeros((len(data), RIGHT_EYE_DIM))
17 for i in data:
18    frame = i['frame']
19    for t,y in enumerate(i['rightEye']):
20        x_array2[frame-1,t] = y['x']
21        y_array2[frame-1,t] = y['y']
22
23 #UGUALE PER LIPS
24 x_array3 = np.zeros((len(data), LIPS_DIM))
25 y_array3 = np.zeros((len(data), LIPS_DIM))
26 for i in data:

```

```

27     frame = i['frame']
28     for t,y in enumerate(i['lips']):
29         x_array3[frame-1,t] = y['x']
30         y_array3[frame-1,t] = y['y']

```

A questo punto, utilizzo gli array per stampare i grafici finali. I grafici ottenuti saranno 6, uno per l'asse X e uno per l'asse Y dei tre elementi di interesse.

```

1  #GRAFICI DEGLI OCCHI E LABBRA
2  fig, ax = plt.subplots(3,2, figsize = (30,18))
3  ax[0,0].set_title('Left Eye X')
4  ax[0,0].plot(x_array[0:x_array.shape[0], 0:x_array.shape[1]]) #plotto da 0 fino
   alla dimensione finale
5  ax[0,0].set_xlabel('Frame')
6  ax[0,0].set_ylabel('Position X')
7
8  ax[0,1].set_title('Left Eye Y')
9  ax[0,1].plot(y_array[0:x_array.shape[0], 0:x_array.shape[1]])
10 ax[0,1].set_ylabel('Position Y')
11 ax[0,1].set_xlabel('Frame')
12 #-----
13
14 ax[1,0].set_title('Right Eye X')
15 ax[1,0].plot(x_array2[0:x_array2.shape[0], 0:x_array2.shape[1]]) #plotto da 0
   fino alla dimensione finale
16 ax[1,0].set_xlabel('Frame')
17 ax[1,0].set_ylabel('Position X')
18
19 ax[1,1].set_title('Right Eye Y')
20 ax[1,1].plot(y_array2[0:x_array2.shape[0], 0:x_array2.shape[1]])
21 ax[1,1].set_ylabel('Position Y')
22 ax[1,1].set_xlabel('Frame')
23 #-----
24
25 ax[2,0].set_title('Lips X')
26 ax[2,0].plot(x_array3[0:x_array3.shape[0], 0:x_array3.shape[1]]) #plotto da 0
   fino alla dimensione finale
27 ax[2,0].set_xlabel('Frame')
28 ax[2,0].set_ylabel('Position X')
29
30 ax[2,1].set_title('Lips Y')
31 ax[2,1].plot(y_array3[0:x_array3.shape[0], 0:x_array3.shape[1]])
32 ax[2,1].set_ylabel('Position Y')
33 ax[2,1].set_xlabel('Frame')
34
35 fig.tight_layout(pad=10.0)
36 plt.show()

```

*evolution.py* ci permette, quindi di vedere l'evoluzione delle posizioni sull'asse X e Y degli elementi del viso durante lo scorrimento del video.

#### 4.2.4 cnt.py

L'esecuzione della classe *cnt.py* permettere di stampare a video altri grafici relativi a capire lo stato di apertura e chiusura degli elementi facciali.

I primi quattro grafici sono quelli relativi alle posizioni X e Y degli occhi, come quelli stampati in *evolution.py*, ma con la differenza che vengono mostrati solo gli intervalli di frame in cui c'è una chiusura degli occhi del soggetto. Si sono presi, infatti i frame in cui il soggetto nel video chiude gli occhi e si aggiungono 10 frame

prima e dopo. Questi frame sono inseriti in un file csv, creato dal programma che prende in input i frame di inizio e fine chiusura occhi, da cui vengono estratti per andare a stampare i grafici desiderati. Nello stesso modo si realizzano i successivi quattro grafici, quelli della velocità lungo gli assi X e Y degli occhi. In questi grafici si descrive, appunto, la velocità con la quale si muove uno dei landmark di *rightEyeUpper* per l'occhio destro e di *leftEyeUpper* per l'occhio sinistro.

Inizialmente, quindi, vengono definiti tali frame:

```

1 #INTERVALLI IN CUI GLI OCCHI INIZIANO A CHIUDERSI, ISTANTE IN CUI SONO CHIUSI ,
  E ISTANTE IN CUI SONO APERTI (Video Prof)
2 data=[
3     [46,48,52],
4     [136,139,143],
5     [232,234,238],
6     [282,285,290],
7     [458,460,465],
8     [520,522,528],
9     [544,547,553],
10    [597,599,604],
11    [625,627,632],
12    [700,702,707],
13    [755,758,763],
14    [768,771,775],
15    [818,821,826],
16    [849,851,856],
17    [873,874,877],
18    [915,917,922],
19    [944,947,952],
20    [974,976,979],
21    [1041,1043,1048],
22    [1129,1131,1136],
23    [1231,1233,1237],
24    [1270,1272,1277],
25    [1294,1296,1299],
26    [1384,1387,1391],
27    [1403,1406,1410],
28    [1432,1434,1438],
29    [1458,1460,1464],
30    [1477,1479,1484],
31    [1500,1503,1509],
32    [1609,1611,1618],
33    [1626,1628,1632],
34    [1701,1704,1708],
35    [1773,1776,1780]
36 ]

```

Dopodiché si crea il file csv:

```

1 #creo il csv
2 with open('count.csv',mode='w', newline='') as file:
3     file=csv.writer(file)
4     file.writerows(data)
5
6 #leggo il csv
7 with open('count.csv', mode='r') as file:
8     reader = csv.reader(file)
9     list = list(reader)
10
11
12 frames = np.zeros((len(list),2))
13
14 lenght_array = 0

```

```

15 for i in range(0,len(list)):
16     frames[i,0] = int(list[i][0]) - 10           #salvo dentro la riga del frames
           il valore del csv -10(10 frame prima)
17     frames[i,1] = int(list[i][2]) + 10         #salvo dentro la riga del frames
           il valore del csv +10(10 frame dopo)
18     lenght_array = lenght_array + (int(list[i][2]) - int(list[i][0]) + 21)

```

Si inizializzano gli array:

```

1 #INIZIALIZZO LE VARIBILI
2 x_array = np.zeros((len(data) , LEFT_EYE_DIM))
3 y_array = np.zeros((len(data) , LEFT_EYE_DIM))
4
5 x_array2 = np.zeros((len(data) , RIGHT_EYE_DIM))
6 y_array2 = np.zeros((len(data) , RIGHT_EYE_DIM))
7
8 velocitaY = np.zeros((len(data) , 1))
9 velocitaX = np.zeros((len(data) , 1))
10
11 EAR_array1 = np.zeros((len(data) , 1))
12 EAR_array2 = np.zeros((len(data) , 1))
13 EAR_array3 = np.zeros((len(data) , 1))
14
15 asp_ratio1 = np.zeros((len(data) , 1))
16 asp_ratio2 = np.zeros((len(data) , 1))

```

Si calcola la velocità:

```

1 c = 0
2 for i in data:
3     frame = i['frame']
4     if frame >= int(frames[c,0]) and frame <= int(frames[c,1]):
5         for t,y in enumerate(i['rightEye']):
6             x_array2[frame-1,t] = y['x']
7             y_array2[frame-1,t] = y['y']
8         if frame==frames[c,1]:
9             c = c + 1
10        if t==4 or t==5 :
11            spaceY=z['y'] - space_prec
12            spaceX=z['x'] - space_prec
13            time = i['timeOfFrame'] - time_prec
14            velocitaY[frame-1,0] = spaceY/time
15            velocitaX[frame-1,0] = spaceX/time           #salvo dentro a
un vettore
16            space_precY = y['y']
17            space_precX = y['x']           #spazio
precedente aggiornato
18            if frame==frames[c,1]:
19                c = c + 1
20                time = 0
21        else :           #questo else
serve per tenere il passo della y
22            for t,y in enumerate(i['rightEye']):
23                if t==4 or t==5 :
24                    space_precY = y['y']
25                    space_precX=y['x']
26            time_prec = i['timeOfFrame']

```

E si valorizzano gli array per andare a stampare i grafici:

```

1 for i in range (0,velocitaY.shape[0]):
2     for y in range (0,velocitaY.shape[1]):
3         if velocitaY[i,y] == 0:
4             velocitaY[i,y] = np.nan

```



```

5
6 for i in range (0,velocitaX.shape[0]):
7     for y in range (0,velocitaX.shape[1]):
8         if velocitaX[i,y] == 0:
9             velocitaX[i,y] = np.nan
10 #####
11 for i in range (0,x_array2.shape[0]):
12     for y in range (0,x_array2.shape[1]):
13         if x_array2[i,y] == 0:
14             x_array2[i,y] = np.nan
15
16 for i in range (0,y_array2.shape[0]):
17     for y in range (0,y_array2.shape[1]):
18         if y_array2[i,y] == 0:
19             y_array2[i,y] = np.nan

```

Gli altri grafici stampati all'esecuzione di questa classe sono i grafici relativi al *blink detection* degli occhi (*eye blinking*) e delle labbra e all'*aspect ratio* degli occhi.

I primi ci permettono di notare l'ammicciamento (blinking) degli occhi e della bocca. Per calcolare il blinking si calcola la distanza euclidea utilizzando i due landmarks più in alto, i due più in basso, quello più a destra e quello più a sinistra dell'occhio:

```

1 #EYE BLINKING RIGHT_EYE
2 for i in data:
3     frame = i['frame']
4     for t,z in enumerate(i['rightEye']):
5         if t == 7:
6             xr1 = z['x']
7             yr1 = z['y']
8         if t == 2:
9             xr2 = z['x']
10            yr2 = z['y']
11        if t == 4:
12            xr3 = z['x']
13            yr3 = z['y']
14        if t == 15:
15            xr4 = z['x']
16            yr4 = z['y']
17        if t == 12:
18            xr5 = z['x']
19            yr5 = z['y']
20        if t == 10:
21            xr6 = z['x']
22            yr6 = z['y']
23 #DISTANZA EUCLIDEA
24 vert1 = math.sqrt(pow((xr2-xr6),2)+pow((yr2-yr6),2))
25 vert2 = math.sqrt(pow((xr3-xr5),2)+pow((yr3-yr5),2))
26 oriz = math.sqrt(pow((xr1-xr4),2)+pow((yr1-yr4),2))
27 right_EAR = (vert1+vert2)/(2*oriz)
28 EAR_array1[frame-1,0] = right_EAR
29
30 #####
31 #EYE BLINKING LEFT_EYE
32 for i in data:
33     frame = i['frame']
34     for t,z in enumerate(i['leftEye']):
35         if t == 7:
36             x11 = z['x']
37             y11 = z['y']
38         if t == 2:

```

```

39         x12 = z['x']
40         y12 = z['y']
41     if t == 4:
42
43         x13 = z['x']
44         y13 = z['y']
45     if t == 15:
46         x14 = z['x']
47         y14 = z['y']
48     if t == 12:
49         x15 = z['x']
50         y15 = z['y']
51     if t == 10:
52         x16 = z['x']
53         y16 = z['y']
54 #DISTANZA EUCLIDEA
55     vert1 = math.sqrt(pow((x12-x16),2)+pow((y12-y16),2))
56     vert2 = math.sqrt(pow((x13-x15),2)+pow((y13-y15),2))
57     oriz = math.sqrt(pow((x11-x14),2)+pow((y11-y14),2))
58     left_EAR = (vert1+vert2)/(2*oriz)
59     EAR_array2[frame-1,0] = left_EAR
60
61 #####
62 for i in data:
63     frame = i['frame']
64     for t,z in enumerate(i['lips']):
65         if t == 21:
66             xr1 = z['x']
67             yr1 = z['y']
68         if t == 24:
69             xr2 = z['x']
70             yr2 = z['y']
71         if t == 28:
72             xr3 = z['x']
73             yr3 = z['y']
74         if t == 31:
75             xr4 = z['x']
76             yr4 = z['y']
77         if t == 39:
78             xr5 = z['x']
79             yr5 = z['y']
80         if t == 35:
81             xr6 = z['x']
82             yr6 = z['y']
83 #DISTANZA EUCLIDEA
84     vert1 = math.sqrt(pow((xr2-xr6),2)+pow((yr2-yr6),2))
85     vert2 = math.sqrt(pow((xr3-xr5),2)+pow((yr3-yr5),2))
86     oriz = math.sqrt(pow((xr1-xr4),2)+pow((yr1-yr4),2))
87     lips_EAR = (vert1+vert2)/(2*oriz)
88     EAR_array3[frame-1,0] = lips_EAR

```

Infine, i grafici dell'*aspect ratio* indicano il rapporto di aspetto, cioè il rapporto tra la lunghezza e l'altezza, dei bounding boxes degli occhi. L'*aspect ratio* viene calcolato in questo modo:

```

1 #CALCOLO L'ASPECT RATIO DEI DUE OCCHI
2 for i in data:
3     frame = i['frame']
4     for t,z in enumerate(i['rightEye']):
5         if t == 15:
6             xmin = z['x']
7         if t == 7:
8             xmax = z['x']
9         if t == 11:

```

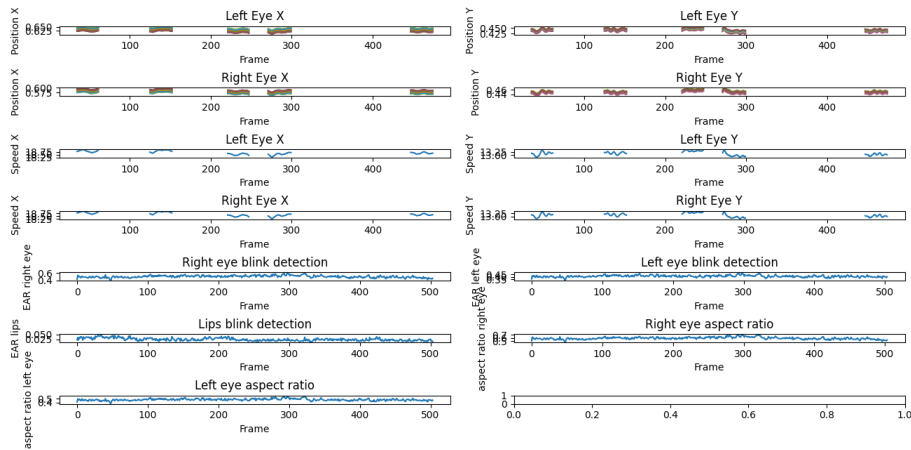


Figura 4.4: Grafici di cnt.py

```

10     ymin = z['y']
11     if t == 3:
12         ymax = z['y']
13
14     ar = (ymax - ymin)/(xmax - xmin)
15     asp_ratio1[frame-1,0] = ar
16
17 for i in data:
18     frame = i['frame']
19     for t,z in enumerate(i['leftEye']):
20         if t == 7:
21             xmin = z['x']
22         if t == 15:
23             xmax = z['x']
24         if t == 11:
25             ymin = z['y']
26         if t == 3:
27             ymax = z['y']
28     ar = (ymax - ymin)/(xmax - xmin)
29     asp_ratio2[frame-1,0] = ar

```

### 4.3 Analisi dei risultati

Usando quanto presentato nei paragrafi precedenti possiamo notare che in certe situazioni il programma può identificare gli occhi come chiusi quando in realtà sono aperti o viceversa. Possono esserci diversi motivi perché ciò accade (come l'angolazione del video o la scarsa qualità delle riprese). Per questo motivo risulta necessario l'addestramento di una rete neurale al fine di ridurre il numero di errori ad una percentuale vicina allo zero. Infatti, nel proseguimento del nostro progetto, come spiegato nella tesi della mia collega, abbiamo addestrato attraverso *Google Colab* una rete neurale in grado di riconoscere lo stato degli occhi presenti in immagini e video garantendo un tasso di successo decisamente superiore.



# Capitolo 5

## Conclusioni

Attraverso lo sviluppo di questo progetto ho imparato un nuovo linguaggio di programmazione che non avevo mai approfondito, il Python, ed ho potuto vedere come diversi tipi di linguaggi possono essere combinati al fine di ottenere un risultato complesso.

### 5.1 Sviluppi Futuri

Quanto svolto nell'ambito prima del tirocinio e poi in questo lavoro di tesi apre la strada a futuri lavori che potrebbero migliorare gli aspetti relativi ai dataset utili a riconoscere lo stato di apertura di occhi e bocca per accrescere il grado di affidabilità dell'intero sistema.



# Bibliografia

- [1] Istat. Incidenti stradali. 2020.
- [2] <https://www.puntosicuro.it/sicurezza-stradale-C-104/guida-distratta-un-fattore-di-rischio-da-non-trascurare-AR-20192/> (articolo).
- [3] <https://www.europarl.europa.eu/news/it/headlines/society/20200827ST085804/che-cos-e-l-intelligenza-artificiale-e-come-viene-usata>.
- [4] <https://www.bigdata-lab.it/2edizione/corsi/artificial-intelligence-machine-learning/>.
- [5] <https://www.brumbrum.it/blog/rilevatore-di-stanchezza/6959/>.
- [6] <https://opencv.org/>.
- [7] <http://mrl.cs.vsb.cz/eyedataset>.
- [8] <https://www.digital4.biz/executive/ai-cos-e-l-intelligenza-artificiale-e-come-puo-aiutare-le-imprese/>.
- [9] <https://www.intelligenzaartificiale.it/>.
- [10] A.M. Turing. Computing machinery and intelligence. *Mind*, 1950.
- [11] <https://www.andreaminini.com/ai/test-di-turing/>.
- [12] <https://www.cyberlaws.it/2018/la-storia-dellintelligenza-artificiale-da-turing-ad-oggi/>.
- [13] W. Pitts W.S. McCulloch. A logical calculus of the ideas immanent in nervous activity. *Bulletin of Mathematical Biophysics*, 5:115–133, 1943.
- [14] S. Papert M.L. Minsky. *Perceptrons: an Introduction to Computational Geometry*. MIT Press, 1988.

- 
- [15] G. Hinton A. Krizhevsky, I. Sutskever. Imagenet classification with deep convolutional neural net- works. *NIPS*, 2012.
- [16] M. Somalvico. *Intelligenza Artificiale*. Hewlett-Packard Italiana Spa, 1987.
- [17] <https://www.ai4business.it/intelligenza-artificiale/machine-learning/machine-learning-cosa-e-applicazioni/>.
- [18] G. Hinton Y. LeCun, Y. Bengio. Deep learning. *Nature*, 2015.
- [19] [https://google.github.io/mediapipe/solutions/face\\_mesh.html](https://google.github.io/mediapipe/solutions/face_mesh.html).
- [20] <https://paperswithcode.com/method/mobilenetv2>.
- [21] <https://www.ionos.it/digitalguide/online-marketing/marketing-sui-motori-di-ricerca/cose-keras/>.
- [22] <https://www.python.it/>.
- [23] [https://www.ilsoftware.it/articoli.asp?tag=Visual-Studio-Code-cos-e-e-come-funziona\\_19189](https://www.ilsoftware.it/articoli.asp?tag=Visual-Studio-Code-cos-e-e-come-funziona_19189).
- [24] <https://jupyter.org/>.
- [25] <https://isolution.pro/it/t/google-colab/what-is-google-colab/google-colab-che-cos-e-google-colab>.
- [26] <https://www.andreaminini.com/ai/tensorflow/>.
- [27] <https://it.mathworks.com/discovery/neural-network.html>.



# Elenco delle figure

2.1	Rappresentazione della prima fase del <i>Test di Turing</i> . . . . .	12
2.2	Rappresentazione della seconda fase del <i>Test di Turing</i> . . . . .	13
2.3	Schema di una Rete Neurale. . . . .	17
2.4	Modello McCulloch-Pitts, Struttura Percettrone. . . . .	18
2.5	Zone di overfitting e underfitting in funzione del costo. . . . .	21
2.6	Esempio di una rete con numerosi layer convoluzionali. A ciascuna immagine di addestramento vengono applicati dei filtri a diverse risoluzioni e l'output di ciascuna immagine viene utilizzato come input per il layer successivo. . . . .	22
2.7	Funzione di attivazione . . . . .	23
2.8	visualizzare un Max Pooling. . . . .	24
3.1	Maschera facciale di Face Mesh. . . . .	25
3.2	Esempio del Dataset MRL. . . . .	26
3.3	Struttura della MobileNetV2. . . . .	27
3.4	Interfaccia Iniziale VisualStudio Code. . . . .	29
3.5	Interfaccia Iniziale Google Colab. . . . .	30
4.1	Screenshot del video riprodotto con facemesh.py . . . . .	35
4.2	Grafici apertura e chiusura occhi . . . . .	36
4.3	Grafici di evolution.py . . . . .	37
4.4	Grafici di cnt.py . . . . .	43



# Ringraziamenti

A conclusione di questo elaborato, vorrei menzionare coloro che mi sono stati accanto in questo percorso di laurea.

Ringrazio il mio relatore Adriano Mancini per la disponibilità e la pazienza durante il tirocinio e la stesura della tesi.

Ringrazio i miei genitori per avermi sostenuto ed essermi stati sempre vicini in questi anni.

Ringrazio anche i miei colleghi, i miei amici e la mia ragazza per avermi incoraggiato e per aver condiviso molti momenti insieme.

Un ringraziamento speciale va a mia nonna, che mi ha sempre supportato e mi è sempre stata accanto, e continua a farlo anche adesso che non c'è più.

