UNIVERSITÀ POLITECNICA DELLE MARCHE

FACOLTÀ DI INGEGNERIA

Corso di Laurea Triennale in
Ingegneria Informatica e dell'Automazione

Progettazione e sviluppo di un algoritmo basato su Deep Learning per la classificazione di immagini RGB nell'ambito della sicurezza stradale

Design and Development of an algorithm based on Deep Learning to classify RGB images in the context of road safety

Relatore:
Dott. Mancini Adriano

Laureando:
Antenucci Lucrezia

Anno Accademico 2020-2021

Indice

Capitolo 1

Introduzione

Il presente lavoro di tesi è stato svolto insieme al collega Matteo Abbruzzetti. Si è deciso di suddividere la tesi in due parti: Matteo in particolare nel suo elaborato presenta aspetti relativi a Facemesh, mentre questo lavoro di tesi tratta l'addestramento delle Reti Neurali al fine di riconoscere l'apertura / chiusura di occhi da immagini RGB.

Nel 2020 si è avuta una diminuzione sia della mobilità che del tasso di incidenti a causa della crisi sanitaria ed economica che ancora oggi viviamo . In modo particolare, i lunghi periodi di *lockdown* imposti dai decreti governativi hanno modificato radicalmente le nostre abitudini determinando il blocco quasi totale della mobilità. Ciò ha influito in maniera determinante sull'incidentalità stradale, che è diminuita del 31.3 % [1]. Nel rapporto Istat 2020, tra le cause principali degli incidenti stradali troviamo al primo posto la distrazione alla guida, seguita dal mancato rispetto della precedenza e dalla velocità troppo elevata.

Ulteriore fattore da tenere in considerazione è la stanchezza al volante che in primo luogo è dovuta alla mancanza di riposo, alla sonnolenza e ai viaggi notturni. Spesso si prosegue il viaggio nonostante il sopraggiungere di chiari sintomi di affaticamento e stanchezza. Questo perchè la maggior parte dei conducenti sottovaluta le limitazioni delle capacità di reazione dovute proprio alla stanchezza.

Oltre a tener conto di queste cause, c'è da aggiungere che il guidatore è soggetto a numerosi stimoli interni ed esterni al veicolo quali l'utilizzo del cellulare, la conversazione con altri passeggeri, ma anche cartelloni pubblicitari e segnaletica inadeguata. Tutto questo compromette in modo notevole il livello di attenzione del conducente.

Per arginare il problema della distrazione alla guida si è pensato di adottare nuovi metodi nell'ambito dell'Intelligenza Artificiale, quale ad esempio la guida autonoma che è da ritenersi quell'innovazione che rappresenta il futuro dei trasporti e le cui svariate applicazioni avranno un notevole impatto sia in ambito automobilistico che nel quotidiano.

Lo scopo principale dell'Intelligenza Artificiale è quello di sviluppare macchine dotate di specifiche capacità di apprendimento tipiche dell'essere umano, quali l'apprendimento, l'adattamento, la percezione visiva, la percezione spazio-temporale, il ragionamento, l'interazione con l'ambiente e la pianificazione, che siano capaci di prefiggersi un obiettivo e di prendere decisioni che di solito sono affidare alle persone. L'Intelligenza Artificiale è applicata a diversi ambiti del nostro quotidiano. Ad esempio, se si visita un sito web successivamente questo fornisce messaggi in base all'interesse manifestato: ciò avviene perchè l'analisi predittiva consente di connettersi e di "parlare" all'utente in modo da fornirgli assistenza personalizzata. Un'altra applicazione dell'Intelligenza Artificiale molto diffusa è il *Chatbot*. Il *Chatbot*, o Assistenza Virtuale, è uno strumento che fornisce assistenza 24 ore al giorno sia ai clienti che ai dipendenti delle aziende rispondendo ad una serie di domande che gli vengono poste. In realtà le applicazioni dell'Intelligenza Artificiale non finiscono qui, perchè può essere applicata a diversi ambiti del nostro quotidiano in quanto riguarda diversi settori come il manifatturiero, finanziario, design, produzione e sanitario [2].

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 Avanzato) è l'algoritmo che permette alle macchine di migliorarsi nel tempo aiutandole ad apprendere e migliorarsi in base all'esperienza. Intelligenza Artificiale e *Machine Learning* sono strettamente connessi tra loro, ma non sono la stessa cosa: il *Machine Learning* è un'applicazione dell'Intelligenza Artificiale. 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* [3].

Recentemente grande interesse è stato rivolto al *Deep Learning* dovuto principalmente alle numerose conquiste in campo informatico, relative soprattutto alla sfera dell'hardware. Inoltre ha fatto passi da gigante in una varietà di problemi di visione artificiale, come il rilevamento di oggetti, il rilevamento del movimento, il riconoscimento di azioni e la stima del posizionamento del corpo umano. Il *Deep Learning* è un'ampia famiglia di metodi che comprende reti neurali, modelli probabilistici gerarchici e una varietà di algoritmi di apprendimento delle funzionalità non supervisionati e supervisionati. Esso 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. Il *Deep Learning* è una rete neurale composta da più livelli.

Questa metodologia ha svariate applicazioni, ma tra le più importanti tro-

viamo 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.

Ma anche, il *Deep Learning* trova applicazione nell'ambito della guida automatica. Grazie all'aiuto di sensori e di telecamere capaci di elaborare le immagini, la guida automatica 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: dunque è un aiuto notevole per il conducente. 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, dunque garantisce una maggiore affidabilità. Non solo, poichè questo permetterebbe anche di ridurre la congestione stradale, di gestire meglio il flusso del traffico e non ultimo 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 in cui si imbatte continuando il suo percorso alla guida. Questi dispositivi, presenti nella maggior parte dei casi nelle nuove autovetture, possono emettere un suono acustico oppure può comparire una scritta su un display in cui si invita l'autista a fare una pausa. Ad oggi possiamo trovare due tipologie di dispositivi [4]:

- 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 azzera quando il motore viene spento.

- Rilevatore di stanchezza reale. In questo caso viene effettivamente misurata la stanchezza del guidatore grazie a 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, di notte, ed anche se il conducente indossa gli 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 assonnato 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 quello della progettazione e dello 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[1] e una *Neural Network*[2] 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.

Per poter fare ciò ci serviamo del pacchetto Mediapipe, utilizzando *Facemesh* e la libreria *OpenCV* per lavorare e manipolare il video [5]. 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] Dall'inglese Red Green Blue, è un modello di colori le cui specifiche sono state descritte nel 1936 dalla CIE (Commission internationale d'éclairage).

[2] modello di calcolo la cui struttura stratificata assomiglia alla struttura della rete di neuroni nel cervello, con strati di nodi connessi.

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 utilizzando il metodo *Facemesh* 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 chiusura degli occhi. Ulteriore step è quello di creare un bounding box per valutare l'eye-blinking anche usando l'aspect ratio. Successivamente si sviluppa la Neural Network inizialmente testata su un Dataset già esistente *MRL Eye Dataset* [6], 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

Introduzione all'Intelligenza Artificiale

L'Intelligenza Artificiale (*Artificial Intelligence*) è un ramo della computer science che permette la programmazione di sistemi hardware e software che permettono di dotare le macchine di caratteristiche che sono tipiche dell'essere umano, come l'apprendimento, il ragionamento, e la creatività. In realtà si va oltre all'intelligenza intesa come capacità di calcolo o di conoscenza di dati astratti, ma si parla di forme di intelligenza che vanno dall'intelligenza spaziale a quella sociale. Dunque l'Intelligenza Artificiale permette ai sistemi di capire l'ambiente circostante e di mettersi in relazione con quello che percepisce in modo che si prefigge un obiettivo e prende decisioni che di solito sono affidate alle persone.

2.1 È davvero possibile simulare la mente umana attraverso una macchina?

Nel 1956 venne organizzata una conferenza estiva presso il *Dartmouth College* di Hannover (USA) durante la quale per la prima volta sono stati messi insieme i termini "Intelligenza" e "Artificiale". L'obiettivo del convegno era quello di riunire un gruppo selezionato di scienziati che per l'intera estate avrebbero lavorato su alcuni dei principali aspetti dell'Intelligenza Artificiale: cercare di capire come le macchine possano usare il linguaggio, formare concetti, risolvere problemi riservati solo agli esseri umani [7].

Ma in realtà è durante 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* descrisse un criterio per determinare se una macchina fosse intelligente o meno: *"The Imitation Ga-*

me". Questo metodo, anche noto con il nome *"Test di Turing"*, si suddivide in due fasi [8].

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

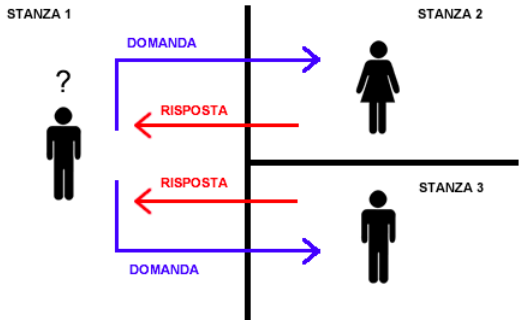

Figura 2.1: Rappresentazione della prima fase del *Test di Turing*

L'esaminatore fa delle domande alle due persone senza vederle, quindi non sa se sta rivolgendo la domanda all'uomo o alla donna. Le domande hanno lo scopo di riuscire a stabilire chi è l'uomo e chi la donna. Una volta ricevute le domande i partecipanti devono rispondere in forma scritta.

È ovvio che i partecipanti possono decidere di comportarsi in modi diversi. Da una parte si può essere sinceri nelle risposte date in modo da agevolare l'identificazione, dall'altra si può mentire mettendo in difficoltà l'intervistatore.

Pertanto l'intervistatore non solo non sa chi è l'uomo e chi la donna, ma non sa neanche se uno 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 uno dei due partecipanti è un computer (Fig. 2.2) e questa volta l'intervistatore deve capire se a rispondere è l'uomo o il computer.

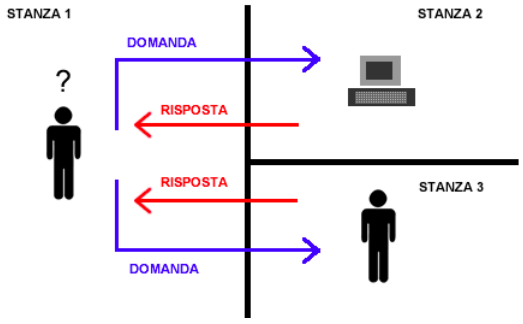

Figura 2.2: Rappresentazione della seconda fase del *Test di Turing*

Anche questa volta l'intervistatore pone delle domande ai partecipanti senza vederli, quindi non sa se sta rivolgendo la domanda all'umano o alla macchina. Il

gioco si ripete N volte e l'intervistatore sbaglia l'identificazione dei partecipanti per Y volte, quindi il tasso di errore è $\frac{Y}{N}$.

Il *Test di Turing* è superato se la percentuale di errore nel gioco in cui partecipa il computer $\frac{Y}{N}$ è simile o inferiore a quella in cui partecipano solo gli umani $\frac{X}{N}$, ovvero se $\frac{X}{N} \simeq \frac{Y}{N}$. Se il test è superato si può affermare che il computer è intelligente in quanto è indistinguibile dall'essere umano [9].

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 [10]. Nel 1943 *McCulloch e Walter Pitts* [11] 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* [12] pubblicarono un saggio in cui dimostrarono che i network neurali fossero in grado di compiere solo alcune operazioni elementari, mentre non c'èra speranza che portassero a termine compiti più complessi. Questo determinò la fine della ricerca sui *network neurali* fin quando nel 2012 [13] 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 neuroscience e la cibernetica.

Nel 1987 Somalvico [14], 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.

Sebbene l'impulso alla ricerca dell'Intelligenza Artificiale abbia come punto di partenza l'intenzione di emulare l'intelligenza umana, nel tempo si sono formati due approcci: l'Intelligenza Artificiale debole e forte.

2.2.1 L'Intelligenza Artificiale Debole

Quando si parla di Intelligenza Artificiale Debole si fa riferimento ai sistemi progettati per risolvere problemi specifici di varia complessità. Il suo scopo non è quello di realizzare macchine che abbiano un'intelligenza umana quindi non si ha la pretesa di comprendere e replicare il funzionamento di tutto ciò che il cervello umano è capace di fare. Bensì lo scopo dell'Intelligenza Artificiale Debole è quello di sviluppare forme di intelligenza capaci di risolvere problemi e di fare ragionamenti anche di portata superiore alle possibilità umane. L'Intelligenza Artificiale Debole opera in questo modo: indaga su casi simili, li confronta, elabora possibili soluzioni e sceglie quella più adeguata. In questo contesto entra in gioco la simulazione del comportamento umano. Pertanto, le applicazioni basate sull'Intelligenza Artificiale Debole risultano essere perfette per suggerire all'uomo quali decisioni prendere perchè gli offrono più informazioni per supportare la propria scelta.

2.2.2 L'Intelligenza Artificiale Forte

Un approccio completamente differente a quello dell'Intelligenza Artificiale Debole è quello dell'Intelligenza Artificiale Forte. Nell'Intelligenza Artificiale Forte 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, che non necessariamente sono quelli umani.

2.3 Tecniche di Intelligenza Artificiale

Prima della nascita dell'Intelligenza Artificiale, le macchine riuscivano ad eseguire delle istruzioni precise date in sequenza. Oggigiorno le macchine sono capaci di svolgere attività non programmate dall'uomo, ma che hanno appreso in modo autonomo, man mano che hanno acquisito dati attraverso il lavoro che esse stesse svolgono.

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. Dunque è un sistema di apprendimento automatico capace di acquisire una quantità immane di dati per poi raggiungere un obiettivo preciso.

La complessità dell'Apprendimento Automatico ha portato ad una suddivisione su tre principali classi di algoritmi: supervisionato, non supervisionato e per rinforzo.

L'apprendimento supervisionato consiste nel fornire alla macchina una serie di istruzioni e di obiettivi da raggiungere. In modo particolare, vengono mostrate le relazioni di input, output e risultato. A questo punto la macchina deve essere in grado di trovare una regola generale con la quale deve scegliere l'output corretto per raggiungere l'obiettivo prefissato [15].

L'apprendimento senza supervisione avviene mediante l'analisi dei risultati, senza una relazione diretta tra input e output. La macchina dovrà fare le sue scelte senza essere stata istruita sulle differenti possibilità di output, dunque in questo caso impara dai propri errori.

Infine troviamo l'apprendimento per rinforzo. Questo metodo è basato sul merito: la macchina viene premiata se nelle sue valutazioni ottiene un risultato in linea con le aspettative. In questo caso la macchina interagisce con un ambiente dinamico e deve muoversi non avendo nessuna indicazione se non essere a conoscenza se è riuscita o meno a raggiungere lo scopo finale.

2.3.2 Il *Deep Learning*

Una sottocategoria del *Machine Learning* è il *Deep Learning*.

Il *Deep Learning* [16] 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 *Machine Learning* utilizza le reti neurali per analizzare grandi quantità di dati, per poter apprendere dalle decisioni sbagliate e correggere i propri errori. Il *Deep Learning* (o Apprendimento Profondo) addestra le reti neurali e le rende capaci di imparare a gestire gli input, a fare previsioni man mano più accurate, e a risolvere problemi con maggiore efficienza [17].

Il *Deep Learning* si occupa di una rete neurale con più di due livelli; possiamo definire il *Deep Learning* come Reti Neurali con un gran numero di parametri e livelli in una delle seguenti quattro architetture di rete:

- Reti preaddestrate non supervisionate;
- Reti Neurali convoluzionali;
- Reti Neurali ricorrenti;
- Reti Neurali ricorsivi.

Il *Deep Learning* è il ramo più avanzato del *Machine Learning*: è una tecnica di apprendimento in cui si espongono Reti Neurali artificiali a grandi quantità di dati, in modo che possano imparare a svolgere compiti.

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 non solo sono un sottoinsieme del *machine learning* ma sono anche l'elemento centrale per sviluppare algoritmi nel *deep learning*. La struttura delle Reti Neurali è ispirata al cervello umano: l'idea è di "imitare" il modo in cui i neuroni inviano segnali.

Lo sviluppo delle tecnologie legate al Calcolo Parallelo e il calcolo mediante GPU ha permesso a questi modelli di vedere un grande sviluppo negli ultimi anni ottenendo così risultati sempre migliori e un costante interesse di ricerca.

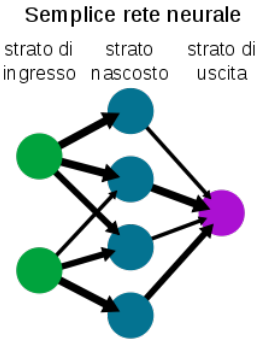

Figura 2.3: Schema Elementare di una Rete Neurale.

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 in cui gli si associano un peso e una soglia. 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 di rete. In caso contrario non si passa alcun dato al livello successivo di rete.

Da un punto di vista matematico alla base delle Reti Neurali possiamo esprimere una funzione $f(x)$ come composizione di altre funzioni $g(x)$ 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.

In generale, le Reti Neurali si affidano ai dati di addestramento per imparare a migliorare la loro esattezza. Una volta ottimizzati questi algoritmi di apprendimento sono dei potenti strumenti nella *computer science* e nell'Intelligenza Artificiale.

Alla base delle Reti Neurali c'è il Percettrone, in completa analogia con il Neurone in una rete neurale biologica. In Figura 2.3 ogni pallino è un nodo che rappresenta un Percettrone. Questi sono 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 [11], 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.

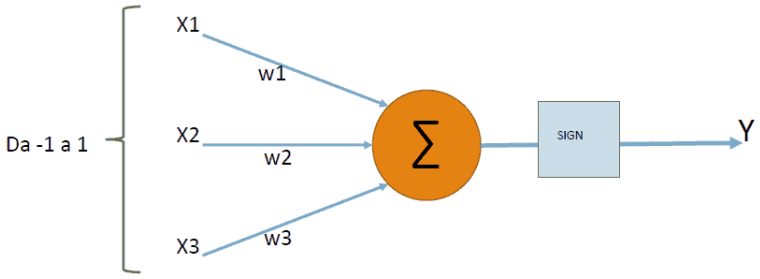

Figura 2.4: Modello McCulloch-Pits, Struttura Percettrone.

La struttura del Percettrone è del tutto analoga a quella di un neurone biologico. Come input abbiamo i dendriti assieme alle sinapi. Il nucleo e gli assoni che 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 $(x_1, x_2, \ldots, x_n)$. Inoltre avremo un ulteriore array dei pesi indicati con $w_1, w_2, \ldots, w_n$. 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:

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

o analogamente :

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

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

Il termine b viene chiamato *bias* del percettone 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 percettrone 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 percettrone. Sia ϕ 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 ottimizzano i risultati ottenuti.

In questo lavoro ci si è occupato 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 fatta la fase di *Model Evaluation* la rete ha a disposizione una serie di valori e predizioni desiderate. In questa fase I/O vengono mantenuti 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. Le dimensioni di batch e il numero di epoch 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*:

  $$\text{Dimensione del lotto} = \text{Dimensione del set di allenamento.}$$

  – *Stochastic Gradient Descent*:

  $$\text{Dimensione del lotto} = 1.$$

– *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 sensibili alla variazione parametriche 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.

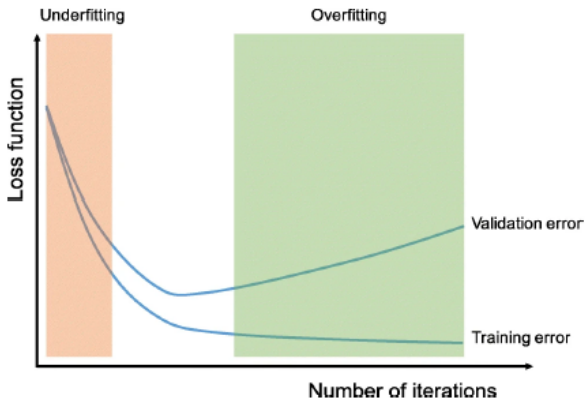

Figura 2.5: Zone di overfitting e underfitting in funzione del costo.

2.7 Reti Neurali Convolutive

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

Le *Convolutional Neural Network* (o Reti Neurali Convolutive) è un tipo di rete *feed-forward*[1]. Le Reti Neurali Convolutive sono una classe di Reti Neurali Artificiale 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 ad esempio livelli di pooling, livelli di convoluzione e l'unità lineare rettificata (ReLU).

[1] ispirata all'organizzazione della corteccia visiva

Possiamo affermare che la *Convolutional Neural Network* è un modello di apprendimento profondo per l'elaborazione dei dati, 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 esprime un'unica funzione: dai pixel dell'immagine grezza su un'estremità ai punteggi delle classi dall'altra.

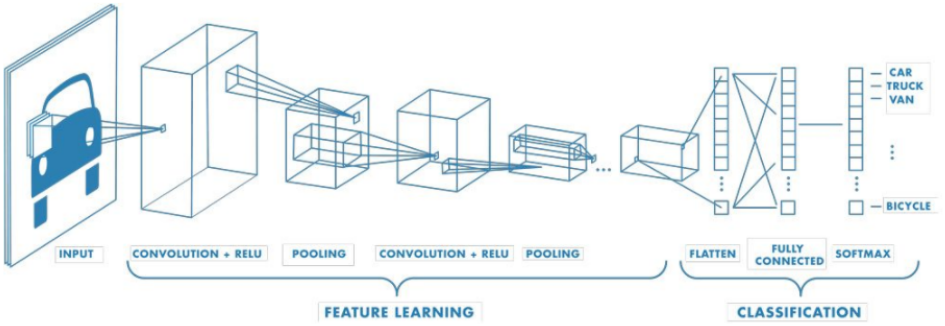

Figura 2.6: Esempio di una rete con numerosi layer convoluzionali. A ciascuna immagine di addestramento vengono applicati dei filtri a diverse risoluzioni e l'output di ciascuna immagine viene utilizzato come input per il layer successivo.

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 matrice di input e il fitro o kernel da cui si ottiene la *feature map*[2].

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.

[2]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).

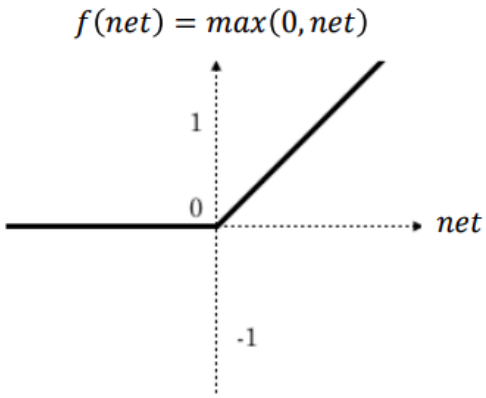

Figura 2.7: Funzione di attivazione

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[3] definita dalla *sliding windows*[4].

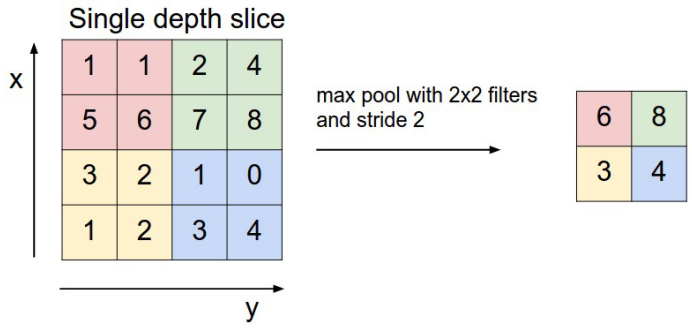

Figura 2.8: visualizzare un Max Pooling.

In questa figura (Fig. 2.8) l'algoritmo posiziona la finestra all'inizio del tensore e inserisce il suo valore massimo nel nuovo tensore. La finestra poi viene fatta scorrere e il processo ricomincia.

[3]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

[4]Rappresenta la dimensione e la forma della porzione di tensore in cui il percettone viene connesso

Capitolo 3

Strumenti Utilizzati

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

3.1 MediaPipe Face Mesh

MediaPipe Face Mesh è una soluzione per la geometria del viso in cui troviamo una stima di 468 punti di riferimento del viso in tempo reale (Fig. 3.1).

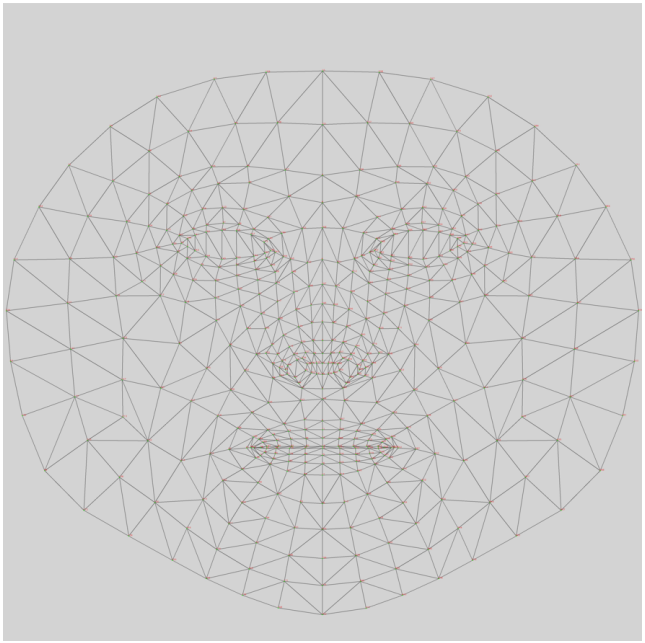

Figura 3.1: Esempio di maschera facciale.

Questa soluzione offre prestazioni in tempo reale fondamentali per le esperienze online.
Cose che possiamo eseguire con l'aiuto di *MediaPipe*:

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

Possiamo trovare diversi modelli di riferimento tra cui il Modello di riferimento facciale. [18]

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 un *Dataset* per l'implementazione della *Neural Network* o rete neurale. Inizialmente, per il rilevamento degli occhi e delle loro parti, per la stima dello sguardo e la frequenza di ammiccamento degli occhi, che sono tutte mansioni fondamentali della visione artificiale, abbiamo preso in considerazione un *Dataset* già esistente: *MRL Eye Dataset* (Fig. 3.2). [6] 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.

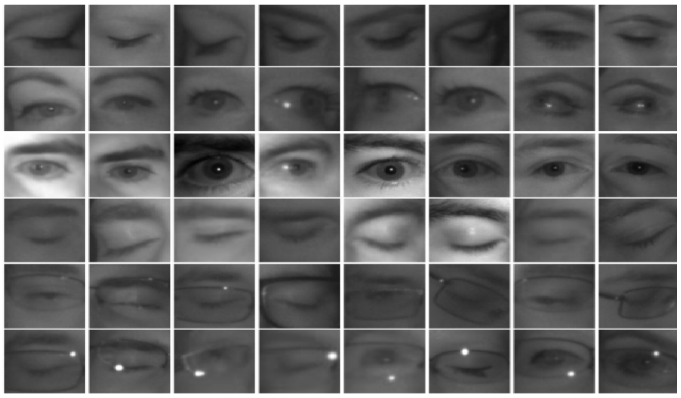

Figura 3.2: Esempio del Dataset MRL.

3.3 MobileNetV2

MobileNetV2 è un'architettura di rete neurale convoluzionale.[1] 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. [19] (Fig. 3.3)

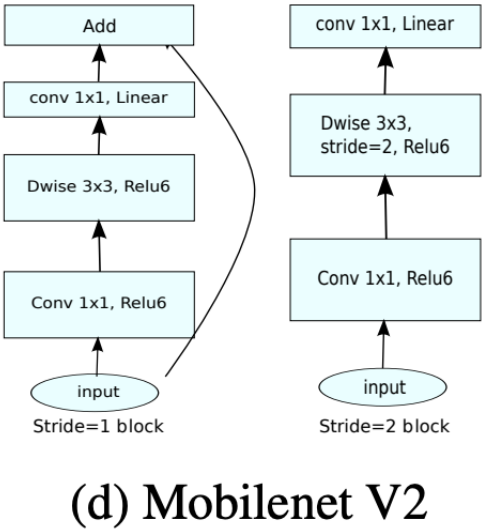

Figura 3.3: Struttura della MobileNetV2.

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.

[1] 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*.[2]

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* [20] per andare ad addestrare la *Neural Network*.

3.5 Linguaggi Utlizzati

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

Python [21] 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*[3].

[2]Libreria per la computazione numerica per sviluppare codice *Python*. La sintassi del *Theano* è molto simile a *Numpy*

[3]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). [22] Si tratta di un *IDE cross-platform*[4] compatibile con *windows, linux, MacOS* e permette di utilizzare qualsiasi linguaggio ed avere un supporto per il *debugging*, controllo *Git e IntelliSense*[5]. 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*

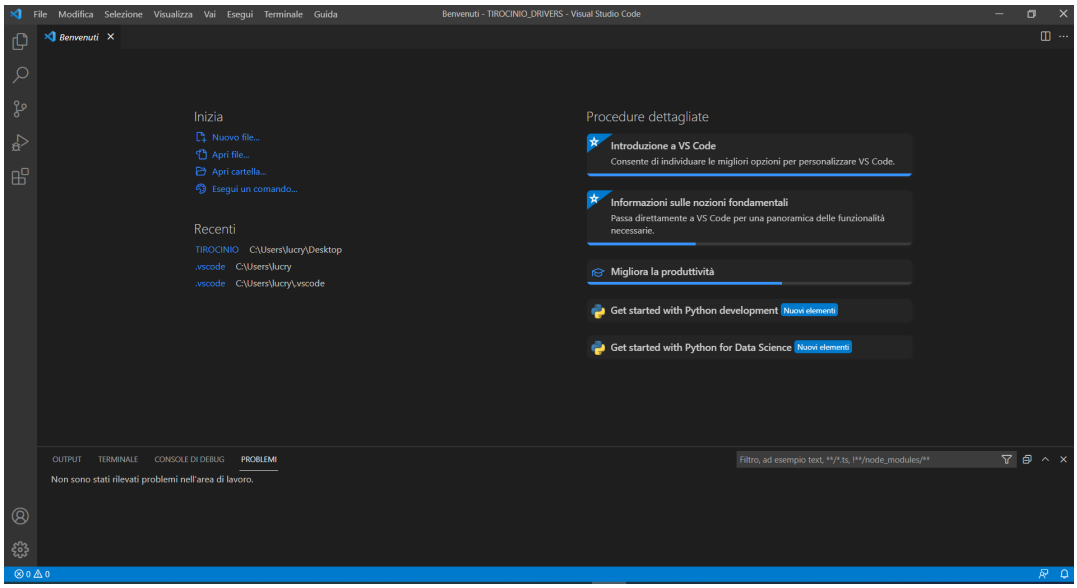

Figura 3.4: Interfaccia Iniziale VisualStudio Code.

[4]software o hardware che può essere utilizzato con più piattaforme
[5]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* [23], 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. [24] 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, OpencCV*,
- uso della GPU.

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

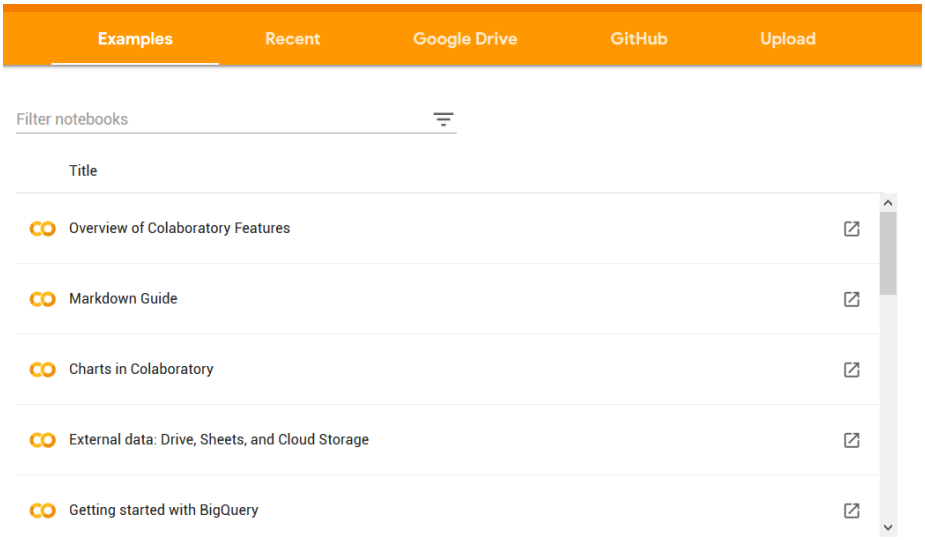

Figura 3.5: Interfaccia Iniziale Google Colab.

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). [25] Ad oggi questa libreria viene utilizzata in molti ambiti della *Deep learning*. *"Cosa significa TensorFlor"*? 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

Grazie agli strumenti illustrati nei precedenti capitoli abbiamo una base solida per poter sviluppare il nostro progetto.

L'obiettivo di questo elaborato è quello di implementare e addestrare una Rete Neurale in grado di rilevare i movimenti del guidatore. In modo particolare focalizziamo la nostra nostra attenzione sull'apertura e la chiusura degli occhi. Per poter ridurre al minimo gli errori commessi, analizziamo anche i falsi positivi e i falsi negativi.

Questo lavoro è stato diviso in due parti: la prima parte relativa a Facemesh, la seconda riguarda l'addestramento della Rete Neurale tramite un set di immagini non molto ampio.

4.1 Panoramica generale sull'Addestramento della Rete neurale

Per addestrare la Rete Neurale ci siamo appoggiati ad un dataset già preimpostato ovvero il Mrl Eye Dataset, il quale è stato implementato facendo uso di Google Colab.

Il presente dataset fa uso del linguaggio Python e viene messo in esecuzione a blocchi, in cui si va a connettere a runtime la GPU[1], per poi allocarlo in un backend a GPU. Mrl Eye Dataset viene scaricato ed è una collezione di circa 37 immagini di persone, che sono registrate in diverse condizioni, ovvero con o senza riflesso etc.

La codifica viene fatta con una serie di attributi (subjet, image, id, gender, classes ed il sensore con il quale è stato fatto). Ora vediamo in generale come viene implementato questo dataset, poi analizzeremo gli script nel nostro caso.

[1]gestione unitaria del programma, in cui si va a gestire e monitorare le attività

Dopo aver importato le librerie, lo script prende il file .zip, lo estrae e va a creare nella directory due cartelle chiamate *open* e *close* in cui va ad inserire le corrispondenti immagini con gli occhi aperti e chiusi suddividendole.

Lo step successivo è il *processing*, ovvero nella cartella MRL Eye vengono copiati i file seguendo la codifica fatta.

Ora una volta fatti questi passaggi iniziamo con l'addestramento: l'80% delle immagini vengono utilizzate per addestrare mentre la restante parte chiamata *Validation dataset* (ovvero il 20 %) valuta il corretto apprendimento. In sostanza possiamo dire che l'apprendimento e la validazione devono essere diversi altrimenti non rispecchia la realtà quindi "splittiamo" le immagini in *train* e *validation*.

Prima di fare l'addestramento, si fa una valutazione con il modello iniziale valutando l'accuratezza tramite una serie di epoche.

L'obiettivo nel nostro lavoro è quello di diminuire la *loss* ed aumentare l'accuracy.

Infine in questa fase di addestramento che viene svolto con *model fit* ha lo scopo di fittare il modello con i dati ed utilizza la GPU poichè richiede tanto tempo e sforzo di occupazione in cui l'*accuracy* cresce già dopo la prima epoca[2].

Una volta raggiunta la parte finale valutiamo i risultati ottenuti mediante il *test dataset* ovvero il test che la macchina non ha mai visto, dopodichè come ultima cosa passiamo le nostre immagini a questo modello per valutare se il dataset è capace di identificare o meno queste immagini sconosciute.

4.2 Addestramento della Rete Neurale

Il modello sviluppato è stato addestrato sulla piattaforma Google Colab sfruttando come già detto in precedenza l'accelerazione GPU. Il dataset è stato diviso in due parti, una che serve all'addestramento vero e proprio e una per la validazione. Questa operazione viene effettuata per migliorare l'apprendimento della rete.

- Il *Test Set* è la parte maggiore del dataset che viene utilizzata durante l'addestramento vero e proprio della rete. Le immagini vengono fatte analizzare, i risultati predetti vengono confrontati con le etichette e i pesi della rete vengono aggiornati.

- Il *Validation Set* è una seconda parte del dataset più piccola che viene utilizzata al termine di ogni epoca. Dopo un primo addestramento del modello questo effettua una predizione ovvero *Validation set* composto da elementi che non facevano parte del processo, quindi oggetti che alla rete risultano sconosciuti, e se ne valuta l'accuratezza e la perdita. Questi valori vengono utilizzati per vedere le prestazioni della rete.

[2]periodo di addestramento

Questo approccio consente di minimizzare il problemi di overfitting riducendo la dipendenza fra gli elementi di addestramento e l'aggiornamento dei pesi.

```
BATCH_SIZE = 32
IMG_SIZE = (96, 96)
from google.colab import drive
drive.mount('/content/drive')
!rm -rf /content/dataset/.ipynb_checkpoints
train_dir = "drive/MyDrive/test"

train_dataset = image_dataset_from_directory(train_dir,
                                             shuffle=True,
                                             batch_size=BATCH_SIZE,
                                             image_size=IMG_SIZE,
                                             validation_split=0.2,
                                             subset="training",
                                             seed = 123
                                             )

validation_dataset = image_dataset_from_directory(train_dir,
                                             shuffle=True,
                                             batch_size=BATCH_SIZE,
                                             image_size=IMG_SIZE,
                                             validation_split=0.2,
                                             subset="validation",
                                             seed = 123
                                             )
```

In questo script andiamo a definire insieme a *Validation* le dimensioni delle immagini con *BATCH_SIZE e IMG_SIZE*.

La dimensione dell'immagine può andare da (224,224) a (32,32) nel nostro caso utilizziamo la dimensione (96,96), mentre il numero di campioni utilizzati sono 32 indicati con *BATCH_SIZE*.

A questo punto si stampano le prime 9 immagini (Fig. 4.1) e le loro etichette del set di allenamento o *train dataset* differenziano nelle diverse directory gli occhi aperti (*open*) e gli occhi chiusi (*close*). La nostra rete deve essere in grado di riconoscere e distinguere l'apertura e la chiusura dell'occhio del conducente.

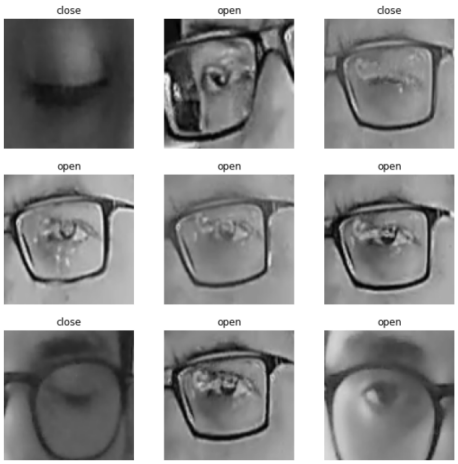

Figura 4.1: Esempi di etichette del dataset.

Queste immagini del dataset sono state estrapolate dai frame di tre video utilizzando fotocamere frontali e laterale. Le immagini utilizzate sono più del numero utilizzato ma è stato sufficiente per l'addestramento delle rete.

4.2.1 Test Set

Il set di dati originali non contiene un *Test Set* quindi andiamo a crearne uno, ma per poterlo fare si deve determinare quanti lotti di dati sono disponibili nel set di convalida usando *tf.data.experimental.cardinality*[3], per poi andare a spostare il 20% di essi sul *Test Set*.

Questo Test corrisponde a una serie di immagini che vengono rimosse dal *Validation* in modo che quando faccio i test l'algoritmo incontra nuove immagini che utilizzano il nostro Dataset, corrispondono a campioni mai visti.

```
val_batches = tf.data.experimental.cardinality(validation_dataset)
test_dataset = validation_dataset.take(val_batches // 5)

test_dir = "drive/MyDrive/test"
!rm -rf /content/test/.ipynb_checkpoints

test_dataset = image_dataset_from_directory(test_dir,
                                            shuffle=True,
                                            batch_size=BATCH_SIZE,
                                            image_size=IMG_SIZE,
                                           )
validation_dataset = validation_dataset.skip(val_batches // 5)

print('Number of validation batches: %d' % tf.data.experimental.cardinality(
    validation_dataset))
print('Number of test batches: %d' % tf.data.experimental.cardinality(
    test_dataset))
```

4.2.2 Data Augmentation

Quando non si dispone di un grande set di immagini, bisogna introdurre artificialmente la diversità del campione applicando delle trasformazioni casuali ma che rispecchiano la realtà. Queste trasformazioni si applicano alle immagini di allenamento, come la rotazione randomica o il *flipping* orizzontale.

Questo ci aiuta a esporre il modello a diversi aspetti di dati di allenamento ed a ridurre l' *overfitting*. In questo caso si va ad utlizzare la libreria *Keras*.

```
data_augmentation = tf.keras.Sequential([
  tf.keras.layers.experimental.preprocessing.RandomFlip('horizontal'),
  tf.keras.layers.experimental.preprocessing.RandomRotation(0.05),
])
```

Questi modelli sono attivi solamente durante l'allenamento e vengono chiamati *model fit*. Quando sono inattivi il modello viene usato in modalità inferenza model.evaluate

[3]restituirà il numero di campioni nel nostro set di dati anche se è sconosciuto quando usiamo la funzione len.

4.2.3 Implementazione MobileNetV2

La *mobilenetV2* è una rete abbastanza semplice da utilizzare e fa parte delle rete convoluzionale in cui si vanno ad estrarre delle caratteristiche delle immagini poi sottoposte a una rete *fullyconnecting*. La *mobilenetV2* utilizza la convoluzione separabile in profonditá come blocchi di costruzioni efficienti, ed introduce anche due funzionalitá nell'architettura: [26]

- colli di bottiglia lineari tra livelli
- collegamenti rapidi tra i colli di bottiglia.

La struttura di base la possiamo vedere in quest'immagine (Fig. 4.2) :

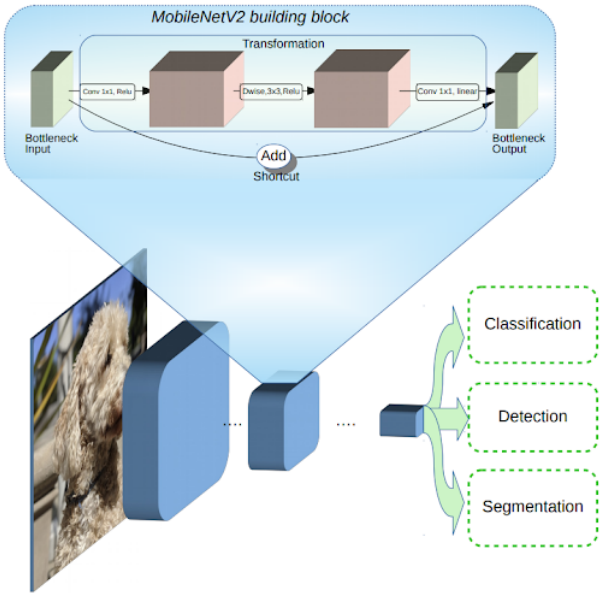

Figura 4.2: Architettura MobileNetV2. I blocchi blu rappresentano blocchi convoluzionali.

Possiamo intuire che i colli di bottiglia hanno la funzione di codificare gli input e gli output intermedi del modello, mentre lo strato interno incapsula la capacità del modello di trasformarsi in concetti tipo i pixel a descrittori di un livello superiore come le categorie di immagini. Possiamo dire che la *MobileNetV2* è un estrattore di funzionalitá molto efficace per il rilevamento e la segmentazione di oggetti.

Utilizzeremo *tf.keras.applications.MobileNetV2* come modello di base. Questo modello si aspetta come valori in pixel [-1, 1] ma nel nostro sono a [0, 255]. Per ridimensionarli usiamo un modello di pre-elaborazione già incluso. È un modello già pre-addestrato sul set di dati ImageNet, che è un grande dataset composto da più di un milione di immagini e 1000 classi.

Lavoriamo con lo strato chiamato "collo di bottiglia". Istanziamo un modello di MobileNetV2 pre-caricato con pesi addestrati su ImageNet.

Specifichiamo *include=False* in modo che utilizziamo una rete dove non vengono caricati gli strati superiori di classificazione. È ideale per l'estrazione delle caratteristiche.

```
# Create the base model from the pre-trained model MobileNet V2
IMG_SHAPE = IMG_SIZE + (3,)
base_model = tf.keras.applications.MobileNetV2(input_shape=IMG_SHAPE,
                                               include_top=False,
                                               weights='imagenet')
```

Questo estrattore converte ogni immagine 160*160*3 in un blocco di caratteristiche 5*5*1280.

4.2.4 Feature Extraction

In questa sezione congeliamo la base convoluzionale creata prima e la useremo come estrattore di caratteristiche. Inoltre aggiungiamo un classificatore di primo livello su cui andremo ad allenare la rete.

```
base_model.trainable = False
```

È importante congelare la base convoluzionale prima di andare avanti con l'addestramento del modello.

Impostando il codice a False impediamo che i pesi in un dato livello possano essere aggiornati durante l'allenamento. Impostando la base del modello a False il Livello di *BatchNormalization* verrà eseguito in modalità inferenza e non aggiornerà le sue statistiche sulla media e sulla varianza. A questo punto tramite il *base.model.summary()* possiamo osservare la struttura del modello.

Nell'implementazione del modello si sono utilizzati in modo particolare due strati: **BatchNormalization** e **Dropout**.

La *BatchNormalization* è un metodo algoritmico che rende più veloce e stabile l'addestramento delle reti neurali. Questo algoritmo usa la media e la varianza del vettore delle uscite di un determinato strato. In questo modo le uscite di un determinato strato seguono una distribuzione normale o chiamata anche Gaussiana. Infine viene calcolata l'uscita del livello come una funzione di due paramenti addestrabili che vanno ad applicare una trasformazione lineare che permette alla rete di scegliere la distribuzione normale ottimale per quello strato.

Il *Dropout* invece è una tecnica che la possibilità di *overfitting* durante la fase di allenamento della rete. Abbiamo anche un'ulteriore strato chiamato *Dense* implementa l'operazione vale a dire dove è passata la funzione di attivazione per elemento come argomento, è una matrice di pesi creata dal livello ed è un vettore di polarizzazione creato dal livello. A questo punto abbiamo 2.5M d parametri della MobileNet congelati ma 1.2K parametri allenabili nello strato Dense che sono divisi tra due oggetti di *tf.Variable* (Fig. 4.3).

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_2 (InputLayer)         [(None, 96, 96, 3)]       0
_________________________________________________________________
sequential (Sequential)      (None, 96, 96, 3)         0
_________________________________________________________________
tf.math.truediv (TFOpLambda) (None, 96, 96, 3)         0
_________________________________________________________________
tf.math.subtract (TFOpLambda (None, 96, 96, 3)         0
_________________________________________________________________
mobilenetv2_1.00_96 (Functio (None, 3, 3, 1280)        2257984
_________________________________________________________________
global_average_pooling2d (Gl (None, 1280)              0
_________________________________________________________________
dropout (Dropout)            (None, 1280)              0
_________________________________________________________________
dense (Dense)                (None, 1)                 1281
=================================================================
Total params: 2,259,265
Trainable params: 1,281
Non-trainable params: 2,257,984
```

Figura 4.3: Riassunto implementazione MobileNetV2.

4.3 Validation

Dopo un allenamento di 10 epoche (Fig. 4.4) dovremmo avere una buona *accuracy* nel set di validazione/convalida. Nell'allenamento valutiamo la *loss* e *l'accuracy*.

```
1 history = model.fit(train_dataset,
2                     epochs=initial_epochs,
3                     validation_data=validation_dataset)
```

Partiamo da una *loss* iniziale di 2.16 e *l'accuracy* di 0.23:

```
Epoch 1/10
6/6 [==============================] - 4s 127ms/step - loss: 1.8006 - accuracy: 0.3439 - val_loss: 1.9394 - val_accuracy: 0.2128
Epoch 2/10
6/6 [==============================] - 1s 55ms/step - loss: 1.6254 - accuracy: 0.3439 - val_loss: 1.7336 - val_accuracy: 0.2128
Epoch 3/10
6/6 [==============================] - 1s 55ms/step - loss: 1.4818 - accuracy: 0.3545 - val_loss: 1.5441 - val_accuracy: 0.2340
Epoch 4/10
6/6 [==============================] - 1s 53ms/step - loss: 1.2269 - accuracy: 0.3757 - val_loss: 1.3731 - val_accuracy: 0.2340
Epoch 5/10
6/6 [==============================] - 1s 55ms/step - loss: 1.1710 - accuracy: 0.3651 - val_loss: 1.2209 - val_accuracy: 0.2128
Epoch 6/10
6/6 [==============================] - 1s 52ms/step - loss: 1.0627 - accuracy: 0.3810 - val_loss: 1.0924 - val_accuracy: 0.2340
Epoch 7/10
6/6 [==============================] - 1s 53ms/step - loss: 0.9313 - accuracy: 0.4127 - val_loss: 0.9859 - val_accuracy: 0.2553
Epoch 8/10
6/6 [==============================] - 1s 55ms/step - loss: 0.9536 - accuracy: 0.3862 - val_loss: 0.8987 - val_accuracy: 0.2979
Epoch 9/10
6/6 [==============================] - 1s 51ms/step - loss: 0.7871 - accuracy: 0.4709 - val_loss: 0.8363 - val_accuracy: 0.3191
Epoch 10/10
6/6 [==============================] - 1s 53ms/step - loss: 0.8167 - accuracy: 0.4868 - val_loss: 0.7863 - val_accuracy: 0.4043
```

Figura 4.4: Primo ciclo di epoche.

Notiamo che man mano alleniamo il nostro modello più diventa alta l'accuratezza, inizialmente prima di fare l'addestramento avevamo una bassa accuratez-

za. Graficamente (Fig. 4.5) rappresentiamo le curve di apprendimento dell'accuratezza/perdita di formazione e convalida quando usiamo il modello di base *MobileNetV2* come estrattore di caratteristiche fisso.

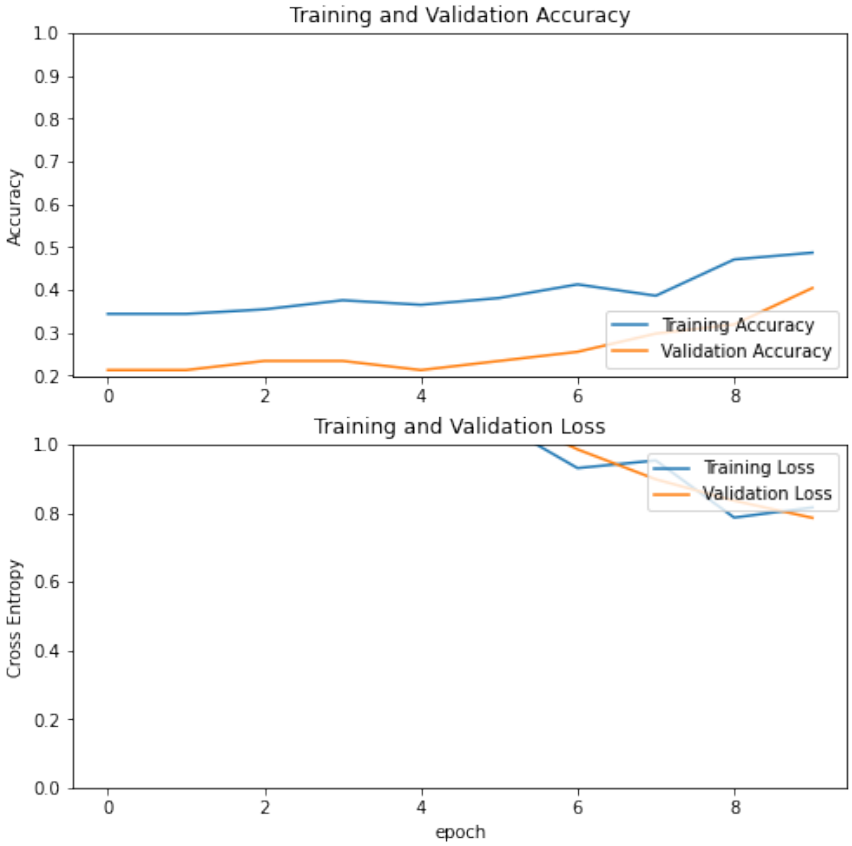

Figura 4.5: Curve di apprendimento dopo le prime dieci epoche.

Le metriche di convalida sono migliori delle metriche di allenamento, questo perchè i livello come il *tf.keras.layers.BatchNormalization e tf.keras.layers.Dropout* influenzano l'accuratezza durante l'allenamento. Un altro motivo è perchè le metriche di allenamento riportano la media di un'epoca mentre le metriche di validazione sono valutate dopo l'epoca e quindi le metriche di validazione vedono che il modello si è allenato un pò più a lungo.

4.3.1 Fine tuning

Come possiamo aumentare le prestazioni di addestramento? Un modo sarebbe quello di addestrare i pesi degli strati superiori del modello pre-addestrato insieme all'addestramento del classificatore aggiunto. Inoltre vanno messi piccoli numeri di strati superiori piuttosto che l'intero modello della *MobileNetV2*.

Nella maggior parte dei casi delle reti convoluzionali più alto è lo strato più è specializzato.

L'obiettivo del *fine tuning* è quello di adattare queste caratteristiche specializzate per lavorare con il nuovo set di dati piuttosto che sovrascrivere l'apprendimento generico.

Quindi ora dobbiamo scongelare semplicemente il modello base e impostare gli strati inferiori come non addestrabili per poi riprendere l'allenamento del modello impostando un tasso di apprendimento più basso altrimenti possiamo andare in *overfit*.

Osserviamo quindi le seconde dieci epoche e la loro accuratezza (Fig. 4.6)

```
Epoch 10/20
6/6 [==============================] - 9s 286ms/step - loss: 0.6649 - accuracy: 0.6032 - val_loss: 0.6316 - val_accuracy: 0.4255
Epoch 11/20
6/6 [==============================] - 1s 69ms/step - loss: 0.6212 - accuracy: 0.6190 - val_loss: 0.4478 - val_accuracy: 0.7872
Epoch 12/20
6/6 [==============================] - 1s 69ms/step - loss: 0.4757 - accuracy: 0.7619 - val_loss: 0.3702 - val_accuracy: 0.8936
Epoch 13/20
6/6 [==============================] - 1s 68ms/step - loss: 0.4979 - accuracy: 0.7778 - val_loss: 0.3505 - val_accuracy: 0.8936
Epoch 14/20
6/6 [==============================] - 1s 73ms/step - loss: 0.4458 - accuracy: 0.7725 - val_loss: 0.3638 - val_accuracy: 0.8511
Epoch 15/20
6/6 [==============================] - 1s 69ms/step - loss: 0.4350 - accuracy: 0.8148 - val_loss: 0.4098 - val_accuracy: 0.7021
Epoch 16/20
6/6 [==============================] - 1s 70ms/step - loss: 0.4308 - accuracy: 0.7725 - val_loss: 0.3212 - val_accuracy: 0.8723
Epoch 17/20
6/6 [==============================] - 1s 69ms/step - loss: 0.3605 - accuracy: 0.8466 - val_loss: 0.2847 - val_accuracy: 0.8723
Epoch 18/20
6/6 [==============================] - 1s 71ms/step - loss: 0.3732 - accuracy: 0.8836 - val_loss: 0.2977 - val_accuracy: 0.8723
Epoch 19/20
6/6 [==============================] - 1s 67ms/step - loss: 0.3806 - accuracy: 0.8148 - val_loss: 0.2603 - val_accuracy: 0.9149
Epoch 20/20
6/6 [==============================] - 1s 70ms/step - loss: 0.3543 - accuracy: 0.8624 - val_loss: 0.2824 - val_accuracy: 0.8936
```

Figura 4.6: Secondo ciclo di epoche.

Ora diamo un'occhiata alla curva di accuratezza/perdita (Fig. 4.7):

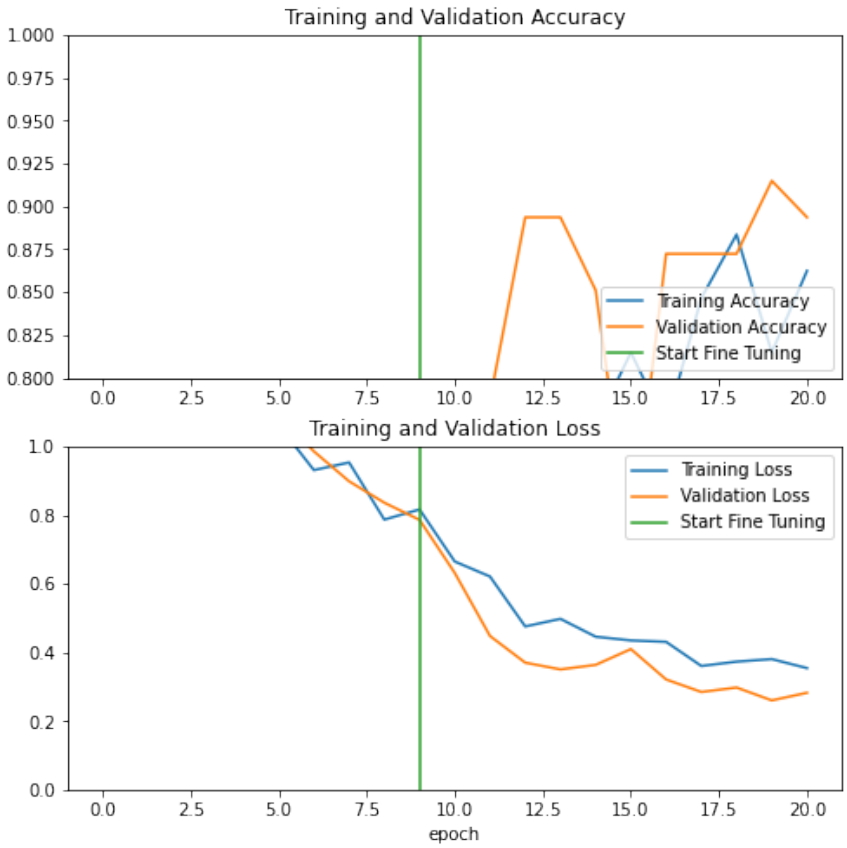

Figura 4.7: Dopo le modifiche, il modello raggiunge il 98% di precisione sul set di validazione.

4.3.2 Evaluation and Prediction

Infine possiamo verificare le prestazioni del modello su nuovi dati utilizzando il *test set*

```
loss, accuracy = model.evaluate(test_dataset)
print('Test accuracy :', accuracy)
```

Il risultato è Test accuracy: 0.90. A questo punto possiamo predirre se l'occhio è aperto o chiuso.

```
#Retrieve a batch of images from the test set
image_batch, label_batch = test_dataset.as_numpy_iterator().next()
predictions = model.predict_on_batch(image_batch).flatten()

# Apply a sigmoid since our model returns logits
predictions = tf.nn.sigmoid(predictions)
print(predictions)
predictions = tf.where(predictions < 0.5, 0, 1)
```

```
9
10 print('Predictions:\n', predictions.numpy())
11 print('Labels:\n', label_batch)
12
13 plt.figure(figsize=(10, 10))
14 for i in range(9):
15   ax = plt.subplot(3, 3, i + 1)
16   plt.imshow(image_batch[i].astype("uint8"))
17   plt.title(class_names[predictions[i]])
18   plt.axis("off")
```

Ultimo passaggio è quello di valutare usando la matrice di confusione.

```
1 confusion_m = tf.math.confusion_matrix(labels = label_batch , predictions =
      predictions)
2 print(confusion_m)
3 accuracy = np.trace(confusion_m) / float(np.sum(confusion_m))
4 misclass = 1 - accuracy
5 print("OA: ",accuracy)
```

Una matrice di confusione o *confusion matrix* nel campo del machine learning è una tabella in cui le previsioni sono rappresentate per colonne e lo stato effettivo è rappresentato nelle righe. Questa matrice serve appunto per predirre , nel nostro caso la usiamo per scoprire se ci sono falsi negativi o falsi positivi derivati dalla predizione.

4.4 Test Locale

Una volta addestrato e salvato il modello possiamo andare a caricare il modello su colab e creare delle immagini di test. Per ogni file vado a fare una lettura, una resized etc.... Si incrementa la dimensione dell'immagine e si va a creare un tensore, ovvero una matrice M*M*K*T con t che equivale all'indice del tensore.

Importiamo in *VisualStudio* il modello caricato per fare il test direttamente nel video in locale. Nel test locale devo predire indicando con:

- 0 occhio chiuso
- 1 occhio aperto

e per ogni *boundingbox* faccio la predizione utilizzando l'immagine convertita a scala di grigi utilizzando la funzione *cvtColor*.

```
1  (image = cv2.cvtColor(image, cv2.COLOR_RGB2GRAY).
```

```
1 Carico il modello importato e creato su Colab facendo:
2 model=keras.models.load.model(#path)
3 una volta caricato stampo le immagini dell'occhio chiuso e dell'occhio aperto,
      riporto come esempio il \textit{left Eye}:
4 cropped_image_left_cb = image[(y2_left-left_offset):y2_left-left_offset+(y1_left
      -y2_left+(left_offset*2)),(x1_left-left_offset):x1_left-left_offset+(x2_left
      -x1_left+(left_offset*2))]
5         resized_left = cv2.resize(cropped_image_left_cb, (96,96), interpolation=
      cv2.INTER_CUBIC)
6
```

```
7         img_l = np.zeros((1,resized_left.shape[0], resized_left.shape[1], 3))
8         for i in range(3):
9           img_l[0,:,:,i] = resized_left
10        predictions = model.predict(img_l)
11
12        # Apply a sigmoid since our model returns logits
13        predictions = tf.nn.sigmoid(predictions)
14        predictions = tf.where(predictions < 0.5, 0, 1)
15        cv2.putText(image,"left:" + str(int(predictions[0][0])), (7, 250), cv2.
      FONT_HERSHEY_SIMPLEX,
16                    3, (0, 255, 0), 3, cv2.LINE_AA)
17        valore_array_sx.append(int(predictions[0][0]))
```

Una volta fatta la predizione abbiamo verificato se la Rete Neurale funzionava correttamente, ovvero se leggeva correttamente lo stato dell'occhio. Per farlo abbiamo utilizzato due metodi; il primo é confrontare il grafico (Fig. 4.8, 4.9) con l'elenco dei frame presenti nel file *csv*. Il secondo metodo é quello di controllare durante l'esecuzione del video se il valore riferito allo stato dell'occhio (0 chiuso, 1 aperto) corrisponde con il suo effettivo stato nel video.

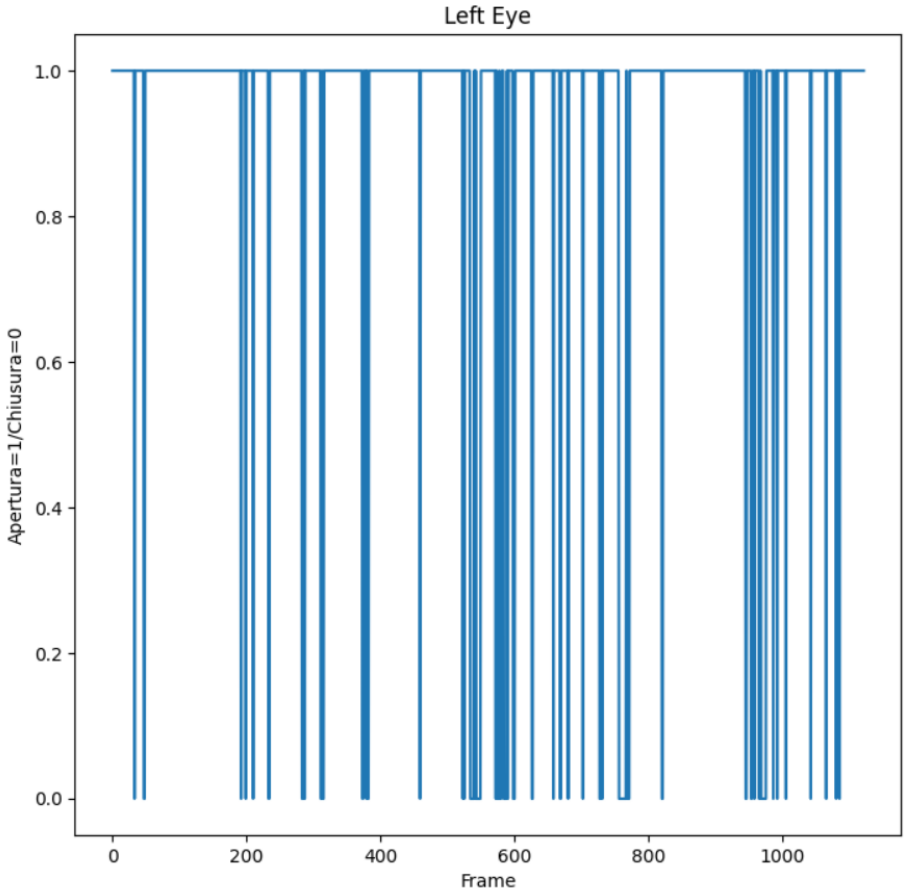

Figura 4.8: Grafico per verificare i falsi positivi riguardante il *Left Eye*

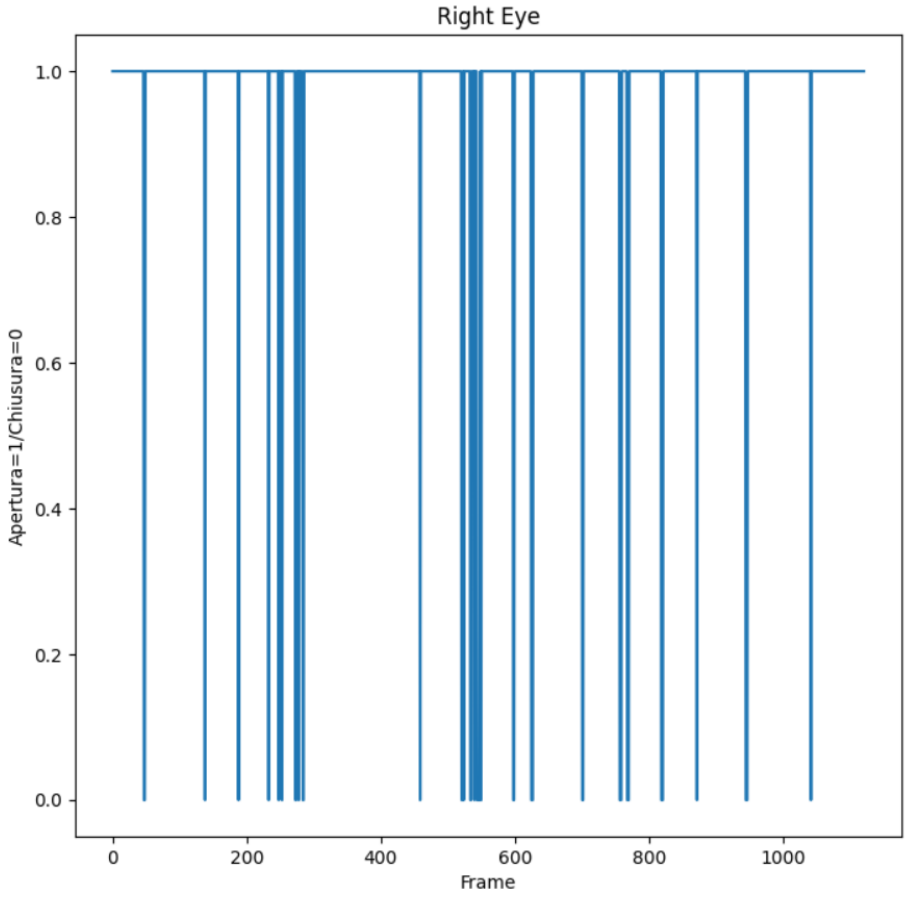

Figura 4.9: Grafico per verificare i falsi positivi riguardante il *Right Eye*

Nei grafici la chiusura dell'occhio é indicata nei momenti in cui il valore assunto dalla variabile y chiamata Apertura/Chiusura é 0. In caso contrario cioé quando il suo valore é 1 l'occhio risulta aperto. Andando poi a verificare la correttezza della rete si é notato che in alcuni casi gli occhi risultavano chiusi alla Rete, quando in realtá nel video erano aperti. Si é quindi constatato che la Rete non é completamente precisa, perció possiamo dire che ci sono dei falsi positivi.

Capitolo 5

Conclusioni

In questo capitolo si esporranno le conclusioni, in considerazione dei risultati ottenuti, ed alcuni dei possibili sviluppi futuri che da questo lavoro di tesi possono essere realizzati. Lo sviluppo di questo progetto mi ha portato ad affrontare un nuovo linguaggio, ovvero Python, ed ha reso possibile vedere come diversi tipi di linguaggi possono essere combinati al fine di ottenere un risultato complesso. Un'altra novità dalla mia parte è stata l'utilizzo di Google Colab, che mi ha permesso di capire come addestrare una Neural Network.

5.1 Sviluppi Futuri

Il tirocinio svolto diventa la base di un altro tirocinio in cui si andranno a prendere attraverso google images una quantità elevata di immagini con facemesh andando ad arricchire quindi il dataset.

Bibliografia

[1] Istat. Incidenti stradali. 2020.

[2] https://blog.osservatori.net/it_it/intelligenza-artificiale-funzionamento-applicazioni.

[3] https://www.bigdata-lab.it/2edizione/corsi/artificial-intelligence-machine-learning/.

[4] https://www.brumbrum.it/blog/rilevatore-di-stanchezza/6959/.

[5] https://it.mathworks.com/discovery/neural-network.html.

[6] http://mrl.cs.vsb.cz/eyedataset.

[7] http://www-formal.stanford.edu/jmc/history/dartmouth/dartmouth.html.

[8] A.M. Turing. Computing machinery and intelligence. *Mind*, 1950.

[9] https://www.andreaminini.com/ai/test-di-turing/.

[10] https://www.cyberlaws.it/2018/la-storia-dellintelligenza-artificiale-da-turing-ad-oggi/.

[11] W. Pitts W.S. McCulloch. A logical calculus of the ideas immanent in nervous activity. *Bulletin of Mathematical Biophysics*, 5:115–133, 1943.

[12] S. Papert M.L. Minsky. *Perceptrons: an Introduction to Computational Geometry*. MIT Press, 1988.

[13] G. Hinton A. Krizhevsky, I. Sutskever. Imagenet classification with deep convolutional neural net- works. *NIPS*, 2012.

[14] M. Somalvico. *Intelligenza Artificiale.* Hewlett-Packard Italiana Spa, 1987.

[15] http://www.umbertosantucci.it/atlante/machine-learning/.

[16] G. Hinton Y. LeCun, Y. Bengio. Deep learning. *Nature*, 2015.

[17] http://www.umbertosantucci.it/atlante/deep-learning/.

[18] https://google.github.io/mediapipe/solutions/face_mesh.html.

[19] https://paperswithcode.com/method/mobilenetv2.

[20] https://www.ionos.it/digitalguide/online-marketing/marketing-sui-motori-di-ricerca/cose-keras/.

[21] https://www.python.it/.

[22] https://www.ilsoftware.it/articoli.asp?tag=Visual-Studio-Code-cos-e-e-come-funziona_19189.

[23] https://jupyter.org/.

[24] https://isolution.pro/it/t/google-colab/what-is-google-colab/google-colab-che-cos-e-google-colab.

[25] https://www.andreaminini.com/ai/tensorflow/.

[26] https://ai.googleblog.com/2018/04/mobilenetv2-next-generation-of-on.html1.

[27] https://www.tensorflow.org/.

Elenco delle figure

Ringraziamenti

Vorrei dedicare qualche riga a tutti coloro che mi sono stati vicini in questo percorso di crescita personale e professionale.

Innanzitutto ringrazio il mio relatore *Adriano Mancini* che mi ha seguita con la Sua infinita disponibilità in questi mesi dandomi suggerimenti e preziose indicazioni per la stesura di questo lavoro di tesi.

Ringrazio i miei *genitori*, mio fratello e tutta la mia famiglia che sono il mio punto di riferimento. Grazie per esser sempre stati al mio fianco in questo percorso. Senza di voi, senza il vostro supporto morale, non avrei potuto raggiungere questo traguardo. *È a voi che dedico la mia tesi.*

Grazie ai miei colleghi di Università e grazie ai miei amici per esserci stati sempre. Grazie per avermi sempre incoraggiato a non mollare mai, e per aver condiviso le cose belle e quelle brutte, in questi anni universitari e non solo.

E infine ringrazio me stessa per la determinazione e l'impegno attuo a raggiungere questo traguardo, la voglia di sfidarmi, per averci provato ed esserci riuscito.