UNIVERSITÀ POLITECNICA DELLE MARCHE

FACOLTÀ DI INGEGNERIA

Corso di Laurea Triennale in
Ingegneria Informatica e dell'Automazione

Progettazione e sviluppo di un algoritmo per la stima della direzione dello sguardo mediante immagini RGB

Design and Development of an Algorithm for Estimating Gaze Direction from RGB Images

Relatore:
Prof. Mancini Adriano

Laureando:
Properzi William

Anno Accademico 2022-2023

Indice

Capitolo 1

Apertura

Nell'era digitale in cui la tecnologia si evolve rapidamente, l'ingegneria informatica e dell'automazione gioca un ruolo cruciale nel plasmare il modo in cui interagiamo con il mondo che ci circonda. La crescente domanda di sistemi intelligenti e interattivi ha portato a un focus sempre maggiore sullo sviluppo di tecnologie avanzate per il riconoscimento e l'interpretazione delle informazioni visive. Il presente lavoro di tesi si colloca in questo ambiente, concentrandosi sul riconoscimento del volto e dei caratteri facciali attraverso l'impiego di algoritmi di ultima generazione.

Questa tesi si propone di affrontare tale sfida, concentrando l'attenzione sulla progettazione e sviluppo di algoritmi avanzati per il riconoscimento del movimento degli occhi, allo scopo di valutare le condizioni di attenzione del conducente durante la guida. Attraverso l'analisi delle caratteristiche facciali e degli indicatori di movimento oculari, si mira a sviluppare un sistema in grado di rilevare segnali di stanchezza, sonnolenza o distrazione, al fine di prevenire potenziali situazioni di pericolo sulla strada.

I dati statistici riguardanti gli incidenti stradali causati da stanchezza o sonnolenza forniscono un quadro chiaro dell'urgenza di questa ricerca. Secondo uno studio condotto dall'Organizzazione Mondiale della Sanità (OMS), circa il 20% degli incidenti stradali è attribuibile alla sonnolenza del conducente. Inoltre, l'Istituto Nazionale di Statistica (ISTAT) riporta che la maggior parte degli incidenti mortali avviene durante le ore notturne, momento in cui la fatica e la sonnolenza possono avere un impatto significativo sulla capacità di guida.

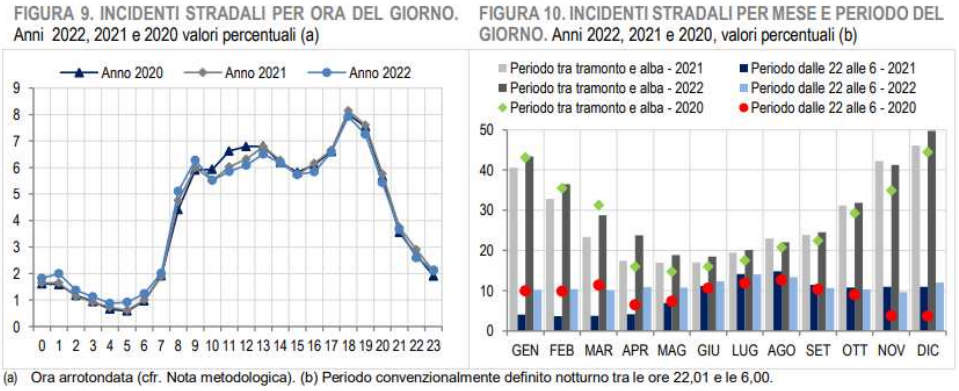

Figura 1.1: Grafico su dati ISTAT con incidenti divisi per ore del giorno[1]

Analizzando questo grafico possiamo già renderci conto dell'impatto che potrebbero avere queste tecnologie nella vita quotidiana:

1. Prevenzione: la rilevazione della stanchezza o delle distrazioni può aiutare a prevenire incidenti stradali, infatti circa il 20% degli incidenti stradali nel mondo sono associati alla stanchezza alla guida.

2. Riduzione dei costi:costi considerevoli in termini di danni materiali, cure mediche e perdita di produttività.

3. Miglioramento della sicurezza: identificando i segni di stanchezza o distrazione, i sistemi basati su telecamere possono avvisare il guidatore o persino attivare misure di sicurezza automatiche

Alcune aziende hanno già provveduto a una prima installazione di alcuni dispositivi di sicurezza e prevenzione come:

1. Mercedes-Benz: offre un sistema chiamato ATTENTION ASSIST, che utilizza sensori per monitorare il comportamento del conducente e rilevare segni di stanchezza e in tal caso il sistema emette un avviso per consigliare al conducente di fare una pausa.

Figura 1.2: Attention Assist [2]

3. BMW: offre un sistema chiamato BMW Driver Attention Warning, che utilizza telecamere per monitorare il movimento degli occhi e la posizione della testa del conducente. Se vengono rilevati segni di stanchezza o distrazione, il sistema emette un avviso per consigliare al conducente di fare una pausa.

Figura 1.3: Rilevamento volto nell'abitacolo [3]

1.1 Obiettivi

Lo scopo principale di questa tesi è lo progettazione e lo sviluppo di un applicativo software che sia in grado di estrapolare in tempo reale alcune maschere degli occhi andando ad analizzare la salute e la capacità di attenzione del conducente.

Il lavoro svolto si può suddividere come segue

1. Generare un dataset formato da video acquisiti attraverso una videocamera con una buona risoluzione;
2. Rilevare il volto nei vari frame e determinarne i punti chiave (keypoints) tramite il modello *MediaPipe Face Mesh*
3. Manipolare i risultati ottenuti e isolare la parte interessata (gli occhi) con la libreria *OpenCV*
4. Utilizzare il modello *SAM(segment anything model)* AI di Meta, che permette di segmentare, mascherare e fornire un'approssimazione quanto più veritiera della posizione oculare
5. Convertire i risultati ottenuti in immagini RGB[1] chiare e nitide che serviranno da modello.

Nota: Il linguaggio di programmazione utilizzato è *Python*, linguaggio di programmazione ad alto livello multi paradigma insieme a Google Colab, piattaforma cloud per diminuire i tempi di esecuzione.

[1]Modello di colori basato su rosso, verde e blu

1.2 Struttura della Tesi

Il presente lavoro di tesi è strutturato nei seguenti capitoli:

- Apertura

1. Introduzione;
2. Tecnologie Utilizzate;
3. Sviluppo del progetto e Realizzazione del software;
4. Conclusione e sviluppi futuri;
5. Appendice.

Il software realizzato è consultabile al seguente link:
`https://colab.research.google.com/drive/11hBRvDuFXvMp3kdVaAUo2krIaFrykf4t?usp=sharing`

Capitolo 2

Introduzione

Intelligenza Artificiale o AI

Per trovare le prime tracce dell'intelligenza artificiale dobbiamo tornare indietro al 1956 quando si tenne la conferenza di Dartmouth.

L'evento viene proposto nel 1955 principalmente da John McCarthy e Marvin Minsky, in un documento informale di 17 pagine noto come 'proposta di Dartmouth'. Il documento introduce per la prima volta il termine di intelligenza artificiale, e motiva la necessità della conferenza con la seguente asserzione: **Lo studio procederà sulla base della congettura per cui, in linea di principio, ogni aspetto dell'apprendimento o una qualsiasi altra caratteristica dell'intelligenza possano essere descritte così precisamente da poter costruire una macchina che le simuli.**

Ciò che si chiedevano informatici e ricercatori era se fosse possibile far sì che i computer svolgessero cose che, quando svolte da esseri umani, richiedono intelligenza. In altre parole, l'obiettivo era quello di sviluppare programmi informatici in grado di simulare processi cognitivi umani, come il ragionamento, l'apprendimento e la risoluzione di problemi.

2.1 Concetti preliminari

2.1.1 Cosa si intende per Intelligenza Artificiale

Quando si legge "intelligenza artificiale" si parla di software di machine learning. Sostanzialmente algoritmi di apprendimento automatico che utilizzano la statistica e la matematica per trovare senso (pattern), a enormi quantità di dati. Per dati intendiamo tutto quanto può essere digitalizzato e quindi memorizzato.

1. Machine Learning: L'apprendimento automatico o machine learning è già presente in molti prodotti o servizi. Questi tool sono in grado con molta più efficienza dell'uomo, sulla base di alcune informazioni, di prevedere servizi e prodotti.
2. Reti Neurali: Le reti neurali sono usate nel machine learning. Attraverso analisi statistiche sulla base dei pesi delle singole connessioni viene calcolata la probabilità che, una data immagine, corrisponda alle informazioni contenuti.
3. Deep Learning: L'apprendimento profondo possiamo definirlo come l'apprendimento automatico potenziato, nel senso che utilizza una tecnica che offre alle macchine una maggiore capacità di individuare pattern attraverso un uso più sofisticato delle reti neurali.

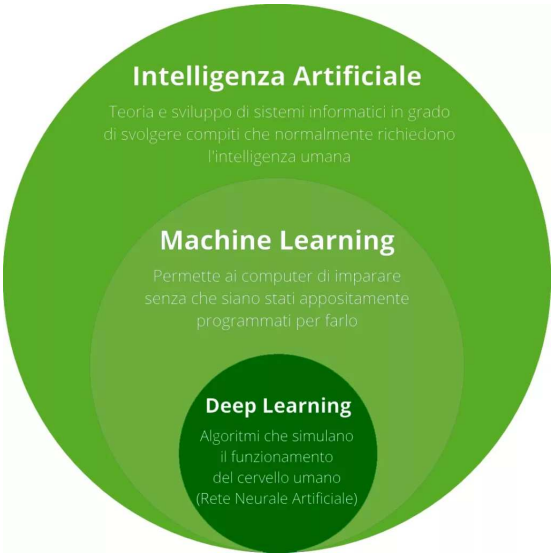

Figura 2.1: Suddivisione intelligenza artificiale[4]

2.1.2 Machine Learning

Abbiamo capito che per funzionare gli algoritmi devono nutrirsi di dati. Più ne mangiano e più crescono. Il machine learning, analizzando le tracce digitale che lasciamo, può per esempio suggerirci contenuti che ci piacciono sulla base delle indicazioni che volontariamente o involontariamente gli abbiamo dato. Possiamo dire che questi sistemi cercano un pattern e lo applicano. Guardano le cose, imparano a identificarle, le riconoscono e cercano delle relazioni. Le macchine imparano in tre modi: apprendimento supervisionato, senza supervisione e rinforzo.

1. Apprendimento supervisionato: i dati sono etichettati in modo da ottenere il loro scopo. Tramite questo approccio si inserisce un set di dati di input e di output opportunamente etichettati, dando la possibilità al sistema di conoscere immediatamente qual è la correlazione tra i due e di mappare a quali input corrispondono tali output.

2. Senza supervisione: nell'apprendimento non supervisionato, i dati non hanno etichette. La macchina cerca solo i modelli che riesce a trovare e il sistema non sarà in grado di collegare immediatamente questi dati con un output definito, ma proverà a relazionare gli input tra di loro per cercare similarità e differenze tra loro, fornendo categorizzazioni e pattern, anche nascosti, tra gli input.

3. Con Rinforzo: Un algoritmo di rinforzo impara per tentativi ed errori per raggiungere un obiettivo chiaro. Prova molte cose diverse e viene "ricompensato o penalizzato" a seconda che i suoi comportamenti aiutino o impediscano il raggiungimento del suo obiettivo.In poche parole possiamo dire che il sistema che apprende è chiamato "agente", mentre l'ambiente con cui interagisce è chiamato "ambiente". L'agente prende decisioni e compie azioni, ricevendo feedback (o "rinforzo") dall'ambiente in base alle sue azioni.

Attraverso ognuna di queste iterazioni di apprendimento, il sistema aumenterà i dati a propria disposizione , originando continuamente output più precisi, completi e migliori andando ad analizzare le nuove informazioni ottenute.

2.1.3 Reti Neurali

Le reti neurali artificiali sono modelli matematici composti da un insieme di unità di elaborazione chiamate neuroni artificiali, organizzate in strati interconnessi. Ogni neurone applica una funzione di attivazione al risultato della somma pesata delle sue connessioni in ingresso, introducendo non-linearità nella rete e consentendo di apprendere relazioni complesse nei dati.

Durante il processo di addestramento, vengono presentati alla rete esempi di input e gli output desiderati. L'obiettivo è regolare i pesi delle connessioni in modo che la rete produca output coerenti con quelli desiderati. Attraverso iterazioni ripetute di questo processo, la rete impara dai dati e diventa sempre più brava nel compiere compiti specifici.

Una rete, alla base, è composta da vari layer:

1. Input Layer: raccoglie ogni tipo di dato di interesse per la rete dall'input.
2. Multiple Hidden Layer: si occupano di identificare pattern tra i dati ed eseguono calcoli per il nodo successivo in base all'addestramento ricevuto.
3. Output Layer: genera le classificazioni e rappresentazioni in base ai dati ottenuti dai layers precedenti.

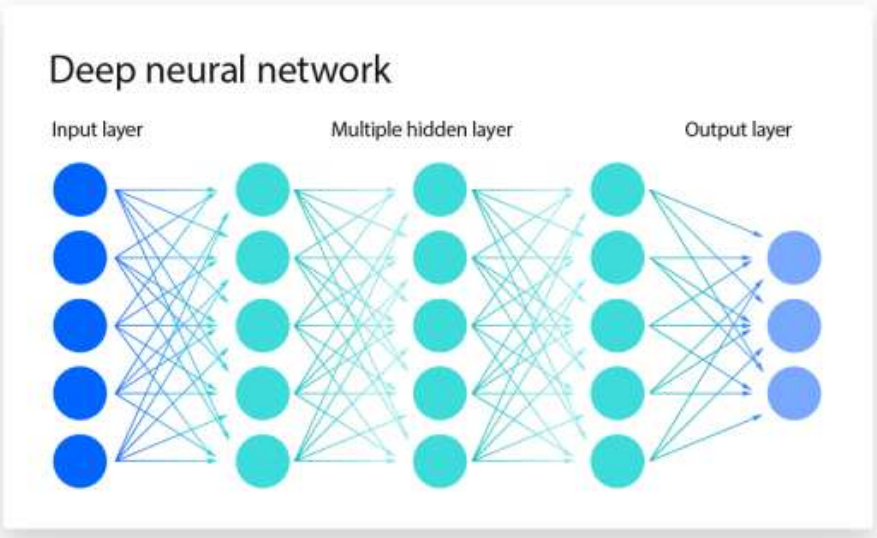

Figura 2.2: Schema di funzionamento reti neurali [5]

2.1.4 Deep Learning

Il deep learning è una sotto-disciplina del machine learning che utilizza reti neurali artificiali profonde per apprendere rappresentazioni complesse dei dati. È caratterizzato dall'utilizzo di reti neurali artificiali con molti strati (da qui il termine "deep"), che consentono di apprendere automaticamente caratteristiche dai dati.

Questo modello va ad operare su reti estremamente complesse, anche con centinaia di hidden layers, che contengono centinaia di miliardi di parametri e avendo un'enorme quantità di dati spesso i tempi computazionali possono essere elevati, allora si tende ad 'addestrare' queste reti tramite GPU[1] e TPU[2]

Se il deep learning è un sottoinsieme del machine learning, in cosa differiscono?

Il deep learning si distingue dal classico machine learning per il tipo di dati con cui lavora e per le modalità con le quali apprende.Infatti questo modello elimina una parte della pre-elaborazione dei dati tipicamente prevista dal machine learning. Questi algoritmi possono acquisire ed elaborare dati non strutturati, come testi e immagini, e automatizzano l'estrazione di componenti eliminando una parte di dipendenza. A grandi linee possiamo schematizzare così le differenze.

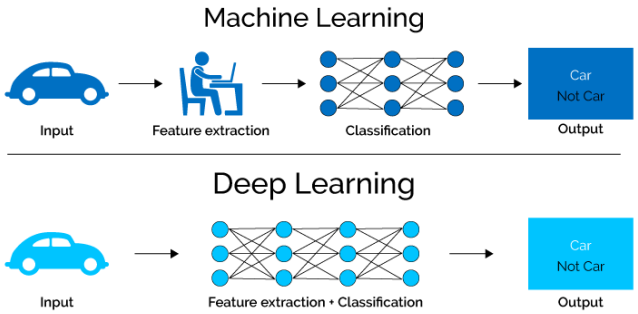

Figura 2.3: Machine Learning vs Deep Learning[6]

[1]Graphics Processing Unit
[2]Tensor Processing Unit

2.2 Python

Come detto in precedenza questo lavoro di tesi si basa su codice scritto in Python e andiamo ad analizzare le principali caratteristiche e il perché è stato convenzionale usare questo linguaggio.

Python è un linguaggio di programmazione interpretato, di alto livello e multi-paradigma, nato dalla mente di Guido van Rossum alla fine degli anni '80. La sua creazione è stata ispirata dalla necessità di un linguaggio che fosse facile da imparare e usare, ma anche potente e flessibile.

Infatti è diventato estremamente popolare grazie alla sua vasta gamma di librerie e framework che coprono praticamente ogni ambito dell'informatica, dalla programmazione web e mobile, all'analisi dei dati, all'intelligenza artificiale e al machine learning.

2.2.1 Caratteristiche fondamentali

Abbiamo detto che questo linguaggio è molto importante, nello specifico:

- Sintassi chiara e leggibile: sintassi semplice e intuitiva, che lo rende facile da imparare e da leggere;
- Ampia libreria standard: Python dispone di una vasta libreria standard che offre moduli e pacchetti per una varietà di compiti, riducendo la necessità di scrivere codice da zero;
- Portabilità: Python è un linguaggio multi-piattaforma che può essere eseguito su una varietà di sistemi operativi, rendendolo adatto per lo sviluppo di software che deve essere distribuito su diverse piattaforme.;

Nonostante i vari punti di forza di questo linguaggio bisogna soffermarsi anche sugli aspetti deboli:

- Gestione della memoria automatizzata: Sebbene la gestione della memoria sia automatizzata in Python (tramite il garbage collection) può anche comportare un aumento dell'overhead e una maggiore variabilità delle prestazioni;
- Prestazioni inferiori: rispetto ad altri linguaggi di programmazione come C++ o Java, Python può essere più lento in termini di esecuzione del

codice, specialmente per applicazioni che richiedono un'elevata efficienza computazionale.

- La natura del linguaggio: essendo di alto livello, ha intrinsecamente un performance penalty molto marcato rispetto ad altri tipi di codice

Nonostante questi difetti, in questo progetto ci siamo serviti interamente del linguaggio python, che implementato opportunamente con moduli e librerie ma soprattutto in un ambiente di sviluppo Cloud ci ha permesso di svolgere operazioni computazionali molto costose ma con un tempo ridotto e con una bassa riduzione delle prestazioni.

Capitolo 3

Strumenti Utilizzati

In questo capitolo andremo a presentare gli strumenti utilizzati per la realizzazione di questo progetto, in particolare:

- *Google Colab*: ambiente di sviluppo software
- *Dataset*: raccolta di immagini e video che analizzeremo
- *MediaPipe Face Mesh*: per il riconoscimento del volto
- *OpenCV*:libreria per la manipolazione delle immagini
- *Supervision*: libreria per la visualizzazione delle maschere
- *SAM*: modello per la segmentazione e per il riconoscimento dell'iride

3.1 Google Colab

Colab, abbreviazione di Google Colaboratory, è una piattaforma di *Cloud computing*[1] gratuita fornita da Google che consente agli utenti di scrivere ed eseguire codice Python attraverso il browser senza dover configurare un ambiente di sviluppo sul proprio computer.

Le funzionalità offerte sono molte infatti è comprensivo di un *IDE, virtual machine, cloud computing e Notebook Jupyter*. Questo formato di file ci permette di scrivere codice python in blocchi separati invece che in un unico script.

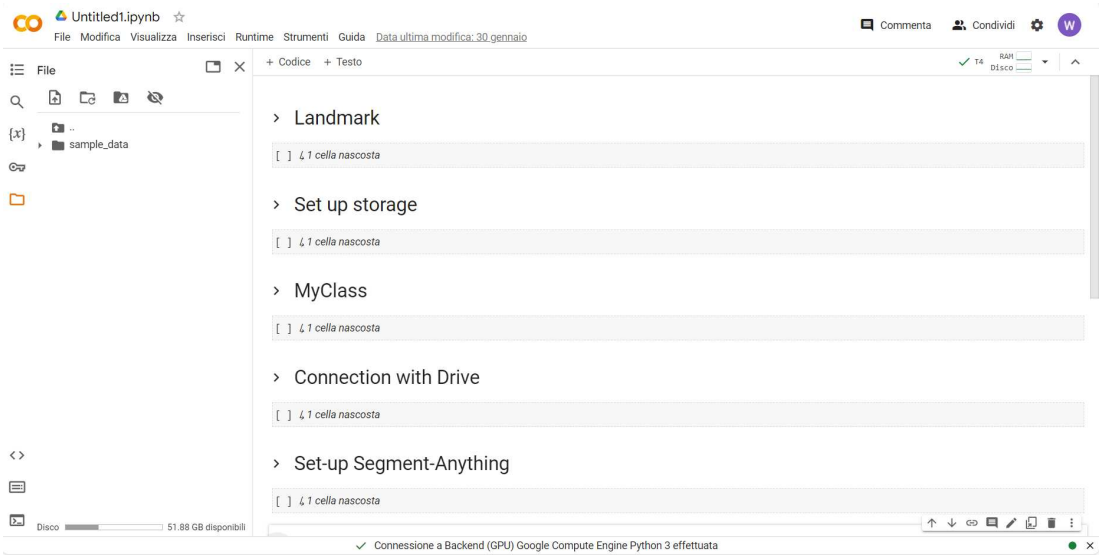

Figura 3.1: Interfaccia Google Colab
[7]

Questo servizio è stato scelto per i numerosi vantaggi che offre:

1. Accessibilità: Essendo una piattaforma basata su cloud, elimina la necessità per gli utenti di installare e configurare il software sui loro computer locali.Inoltre, Colab offre un livello gratuito con accesso a potenti risorse hardware, tra cui GPU[2](Nvidia Tesla T4 con 15 GB di memoria video) e TPU[3](Google Tensor).

[1] *Accesso on demand a risorse di elaborazione*
[2] *Graphics Processing Unit*
[3] *Tensor Processing Unit*

2. Integrazione con i servizi: Colab consente agli utenti di importare ed esportare facilmente dati da varie fonti come Google Drive, Fogli Google e Google Cloud Storage. Inoltre fornisce supporto integrato per librerie di machine learning popolari come TensorFlow e PyTorch, semplificando l'utilizzo di questi framework.

3. Notebook Jupyter: può essere utilizzato come un ambiente di sviluppo integrato (IDE) per Python, sebbene non sia un IDE nel senso tradizionale. Questo tipo di file supporta molti linguaggi di programmazione, tra cui Python, e offre funzionalità che si trovano comunemente negli IDE, come l'esecuzione di codice, la visualizzazione dei risultati, l'inserimento di annotazioni e commenti, e altre caratteristiche utili.

4. Ambiente collaborativo: Gli utenti possono condividere i propri taccuini con altri, consentendo la collaborazione in tempo reale. Inoltre, Colab supporta l'uso di celle Markdown, che consentono agli utenti di includere testo esplicativo, equazioni e visualizzazioni insieme al loro codice.

5. Ricco ecosistema di librerie e risorse preinstallate: Include un'ampia gamma di librerie Python popolari come NumPy, Pandas e Matplotlib, che sono essenziali per la manipolazione, l'analisi e la visualizzazione dei dati.

Purtroppo ci sono anche degli svantaggi come ad esempio i limiti temporali sull'utilizzo del runtime oppure lo storage della Virtual Machine che viene cancellato ad ogni disconnessione. Nonostante questo la possibilità di eseguire codice in celle separate, la velocità computazionale e la versatilità di Google Colab sono stati fondamentali per la progettazione.

3.2 Dataset

Un dataset è un insieme organizzato di dati (in questo caso immagini) strutturati in modo da consentire l'accesso, la gestione e l'aggiornamento efficiente delle informazioni contenute al suo interno.

La creazione del dataset in questione si può suddividere in diverse fasi:

1. Sono state eseguite varie prove con diverse posizioni, altezze, luminosità e con soggetti diversi in modo tale da capire quale sarebbe stata la migliore per l'analisi

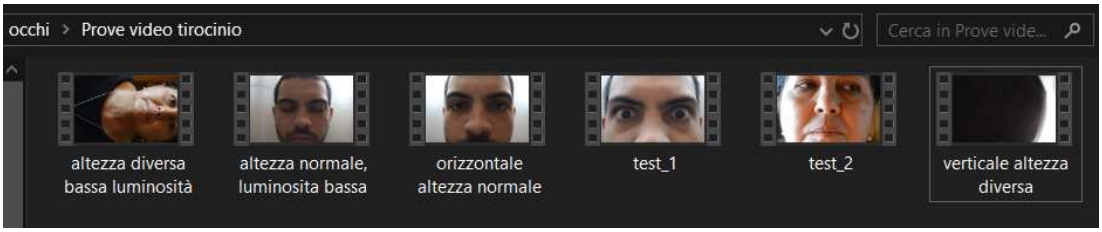

Figura 3.2: Cartella in locale con i video e vari test

2. Una volta analizzati i filmati, *test1, test2*, rappresentanti soggetti diversi sono stati scelti per l'analisi e caricati su Google Drive con un percorso apposito.

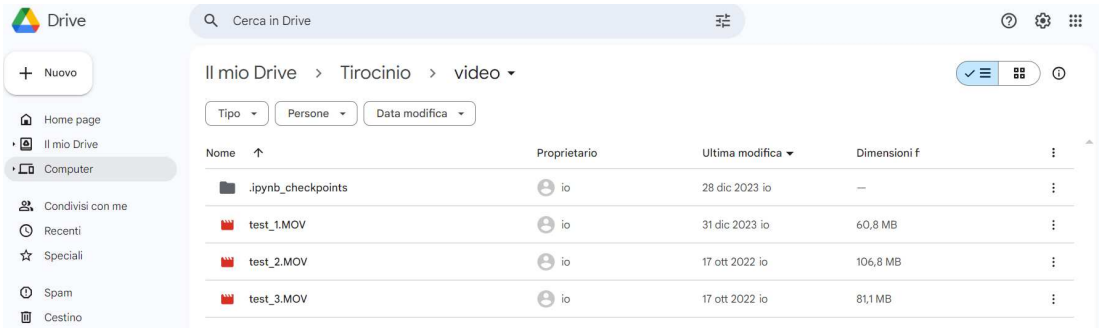

Figura 3.3: Anteprima dataset Google Drive

Abbiamo usato un dataset caricato su Google Drive in Google Colab perché si può accedere direttamente, con un breve passaggio di verifica. In più i dati rimangono caricati su Google Drive anche dopo la disconnessione della sessione e questo permette un accesso persistente.

3.3 MediaPipe Face Mesh

MediaPipe Face Mesh è una tecnologia di geometria facciale che stima 468 landmark facciali in 3D in tempo reale. Utilizza l'apprendimento automatico (ML) per inferire la geometria superficiale in 3D, richiedendo solo un singolo input della fotocamera senza la necessità di un sensore di profondità dedicato. Sfruttando architetture modello leggere insieme all'accelerazione GPU lungo tutto il pipeline, la soluzione offre prestazioni in tempo reale cruciali per esperienze live.

Inoltre, la soluzione è fornita con il modulo Face Geometry che colma il divario tra la stima dei landmark facciali e le utili applicazioni di realtà aumentata (AR) in tempo reale. Stabilisce uno spazio metrico 3D e utilizza le posizioni degli schermi dei landmark facciali per stimare la geometria facciale all'interno di tale spazio.[8]

Figura 3.4: Esempio di maschera con *landmarks*
[8]

E' possibile notare dall'immagine sopra come la maschera dei landmark sia ben definita e come la parte degli occhi sia evidenziata con due colori differenti per distinguerli, anche in caso di progettazione.

Utilizzeremo questo strumento per l'individuazione degli occhi all'interno dei video e tramite le coordinate individuate sarà possibile definire un 'area che racchiuderà l'occhio destro o sinistro, a seconda delle nostre esigenze, per ritagliarlo e lasciare esclusivamente la zona che ci interessa.

3.4 Moduli e librerie

3.4.1 OpenCV

OpenCV, acronimo di "Open Source Computer Vision Library", è una libreria open-source specializzata nell'elaborazione delle immagini e nella visione artificiale. È progettata per fornire un'ampia gamma di funzionalità per l'analisi delle immagini [9]

Le principali funzionalità di OpenCV includono:

- Elaborazione delle immagini: offre una vasta gamma di funzioni per l'elaborazione delle immagini, tra cui filtri, trasformazioni geometriche, ridimensionamento, ritaglio, correzione del colore.
- Tracciamento di oggetti: OpenCV offre strumenti per il tracciamento di oggetti in sequenze di frame video, utilizzando tecniche come il tracciamento ottico del flusso e i filtri di Kalman.

3.4.2 Supervision

E' un sistema di etichettatura e supervisione dei dati offerto da Roboflow. Roboflow è una piattaforma che semplifica il processo di preparazione dei dati per progetti di visione artificiale e machine learning.

Tra le peculiarità di supervision possiamo sicuramente citare la capacità appunto di caricare dataset bilanciati e rappresentativi, garantendo che ci sia una distribuzione adeguata di esempi per tutte le classi di interesse.

Per poter utilizzare questa libreria basta specificarla nel codice attraverso la linea di comando *'import supervision'* Nel nostro caso servirà per la visualizzazione di due tipi di immagine: da una parte l'immagine originale mentre dall'altra l'immagine con le individuazioni.

3.4.3 PyTorch

PyTorch è un framework open source di deep learning sviluppato principalmente da Facebook's AI Research lab (FAIR) ed è diventato molto popolare proprio per la sua flessibilità, facilità di utilizzo e potenza.

PyTorch si basa sul concetto di tensore, una struttura dati simile a un array multidimensionale che può essere utilizzata per rappresentare dati e operazioni matematiche. I tensori in *PyTorch* sono simili ai tensori in *NumPy* ma con l'aggiunta di supporto per *GPU*, che consente un calcolo parallelo veloce.

Una delle caratteristiche distintive di PyTorch però è il suo grafo computazionale dinamico. Questo significa che il grafo computazionale viene creato e modificato al volo durante l'esecuzione del codice, consentendo una maggiore flessibilità nel definire modelli di deep learning rispetto a un grafo statico come quello in *Tensorflow*[4]

3.5 SAM

Segment Anything di Meta è un progetto innovativo che apre nuove frontiere nell'ambito della segmentazione delle immagini. Questo progetto presenta un nuovo compito, un modello avanzato e un dataset senza precedenti dedicati alla segmentazione delle immagini.

Utilizzando un modello efficiente all'interno di un ciclo di raccolta dati, Segment Anything ha creato il più grande dataset di segmentazione mai realizzato, comprendente oltre 1 miliardo di maschere su 11 milioni di immagini, tutte rigorosamente con licenza e rispettose della privacy.

La caratteristica distintiva di Segment Anything risiede nella capacità del modello di adattarsi in modo rapido ed efficiente a nuove distribuzioni di immagini. Questo significa che il modello può essere istruito in modo minimale su nuovi compiti senza la necessità di un addestramento completo, rendendolo estremamente flessibile e versatile.

[4] *libreria software open source per l'apprendimento automatico (machine learning)*

3.5.1 Segment Anything Model

Il Segment Anything Model è stato progettato come modello *promptable* e a differenza degli altri modelli. L'impatto rivoluzionario di SAM risiede nelle sue capacità di inferenza zero-shot. A differenza di altri modelli SAM può segmentare con precisione le immagini senza un precedente addestramento specifico.

Le rivoluzionarie capacità di SAM si basano principalmente sulla sua architettura innovativa, composta da tre componenti principali: l'encoder dell'immagine, l'encoder del prompt e il decoder della maschera.

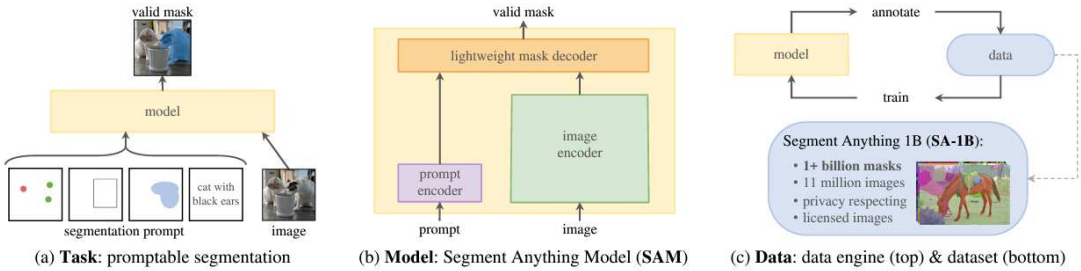

Figura 3.5: Architettura Segment Anyhing Model [10]

1. Encoder dell'immagine: responsabile del trattamento e della trasformazione delle immagini di input in un insieme completo di caratteristiche. Questo encoder comprime l'immagine in una matrice dalla quale il modello estrarrà gli elementi necessari.

2. Encoder del Prompt: interpreta varie forme di prompt di input. Questo encoder traduce questi prompt in un *embedding* che guida il processo di segmentazione. Ciò consente al modello di concentrarsi su aree o oggetti specifici all'interno di un'immagine come indicato dall'input.

3. Decoder della maschera: Sintetizza le informazioni sia dall'encoder dell'immagine che dall'encoder del prompt per produrre maschere di segmentazione accurate. E' responsabile dell'output finale, determinando i contorni precisi e le aree di ciascun segmento all'interno dell'immagine.

L'encoder dell'immagine crea prima una comprensione dettagliata dell'intera immagine, suddividendola in caratteristiche che il motore può analizzare. L'encoder del prompt aggiunge quindi contesto, focalizzando l'attenzione del modello in

base all'input fornito. Infine, il decoder della maschera utilizza queste informazioni combinate per segmentare l'immagine con precisione, garantendo che l'output sia concorde con l'intento dell'input fornito.

Aldilà delle prestazioni e dell'implementazione, il modello può essere utilizzato in due diverse tipologie.

3.5.2 Sam Predictor

Questa tipologia si basa sulla caratteristica *promptable* ovvero l'immagine iniziale viene convertita dall'image encoder e successivamente si forniscono i *prompt* che possono essere di diversi tipi.

Attualmente, i prompt possono essere inviati a SAM in due modi:

- punti di interesse
- bounding box

Figura 3.6: Esempio di predictor con la 'sottrazione' del prompt che non ci interessa e la visualizzazione della maschera precisa
[11]

SAM può ricevere in input un punto di interesse (coordinate x e y) che mira a un pixel dell'immagine rappresentante un oggetto. L'oggetto designato dal punto di interesse permetterà a SAM di generare la maschera associata a quest'ultimo.

Può anche ricevere in input un box che delimita i contorni di un oggetto in un'immagine. Sulla base di questi contorni, SAM genererà una maschera appropriata.

3.5.3 AutomaticMaskGenerator

La seconda tipologia invece non richiede all'utente il caricamento dei vari *prompt*, ma va a massimizzare l'output di maschere individuate generandone il maggior numero possibile.

Richiama la classe *SamAutomaticMaskGenerator* con il quale il modello elabora l'immagine e ne calcola le maschere. Una volta terminata l'elaborazione, vista la possibilità di ottenere le stesse maschere per prompt differenti il modello filtra quelle che reputa uguali e restituisce il risultato.

Nonostante il filtraggio non sono maschere etichettate e c'è la possibilità che queste risultino diverse ma contenenti la stessa cosa e quindi non risulterebbe, in alcuni casi, ottimale.

Figura 3.7: Esempio di AutomaticMaskGenerator con la visualizzazione di tutte le maschere disponibili nell'immagine
[12]

Capitolo 4

Progettazione e Sviluppo del Software

L'obiettivo di questo progetto è quello di realizzare un programma che, attraverso la manipolazione delle immagine RGB, restituisca un'immagine dell'occhio, o meglio dell'iride per capire il livello di stanchezza e sonnolenza del guidatore

Possiamo suddividere il nostro lavoro in varie fasi di sviluppo:

- La prima fase consiste nel raccogliere immagini e video del volto di soggetti diversi e con varie posizioni degli occhi, in modo tale da avere più esempi di occhi in posizioni diverse e non statiche.
- In secondo luogo abbiamo manipolato queste immagine con Opencv e Mediapipe Facemesh cosi da ottenere immagini più specifiche come ad esempio un *bounding box* degli occhi e delle caratteristiche.
- Come terzo step abbiamo preso il box dell'occhio e abbiamo applicato il modello SAM, ottenendo delle maschere degli occhi del video preso in analisi.
- Una volta ottenute le maschere e salvate su *Google Drive*, andiamo a prendere la maschera con la forma più circolare (iride) all'interno di questo elenco e analizziamo la grandezza, forma e larghezza cosi da capire lo stato del soggetto in questione.

4.1 Struttura dell'algoritmo

In questa sezione andremo a vedere l'implementazione alla base del nostro algoritmo.

Nonostante Google Colab sia integrato con python 3 dobbiamo eseguire una prima configurazione e installare moduli, risorse e dipendenze per poter lavorare.

Inizialmente vengono importate le librerie *os* e *sys* per le funzionalità del progetto e successivamente si andrà a specificare il percorso*(root)* di lavoro. Si definisce *"HOME"*, il nostro ambiente di lavoro, e tramite comando *os.getcwd()* si crea la cartella apposita.

Tramite comando *pip* scarichiamo il modello SAM e creiamo tre cartelle nello specifico: input, output e weights.

La cartella weights è indispensabile per installare uno dei checkpoint del modello SAM, in questo caso Vit-H. Al momento esistono 3 tipi diversi di *checkpoint*: Vit-B, Vit-L, Vit-H distinguibili per i diversi parametri che utilizzano e di conseguenza una diversa precisione.

```
import os
import sys
HOME = os.getcwd()
print("HOME:", HOME)
%cd {HOME}

using_colab = True
![sys.executable] -m pip install 'git+https://github.com/facebookresearch/segment-anything.git' #scarico SAM
!pip install wheel
!pip install -q jupyter_bbox_widget roboflow dataclasses-json supervision #dipendenze e altri moduli

# Preparo le directorys e scarico il checkpoint
%cd {HOME}
!mkdir {HOME}/weights
%cd {HOME}/weights
!wget -q https://dl.fbaipublicfiles.com/segment_anything/sam_vit_h_4b8939.pth #checkpoint
CHECKPOINT_PATH = os.path.join(HOME, "weights", "sam_vit_h_4b8939.pth")
print(CHECKPOINT_PATH, "; exist:", os.path.isfile(CHECKPOINT_PATH))


%cd {HOME}
!mkdir {HOME}/input
!mkdir {HOME}/output
%cd {HOME}
```

Figura 4.1: SAM e creazione cartelle con checkpoint

Questi *checkpoint* permettono di evitare il *training* della macchina, in quanto su un database come *SA-1B*[1] richiederebbe risorse computazionali enormi. Il *checkpoint* più leggero è il Vit-Base, molto rapido con circa 91M di parametri. Il *checkpoint* successivo è il Vit-Large, con 308M di parametri, una buona precisione e tempi di esecuzione accetabili. Infine abbiamo il *checkpoint* Vit-Huge che duplica i parametri in considerazione, 636M, aumentando la precisione delle maschere ma anche i tempi di esecuzione.

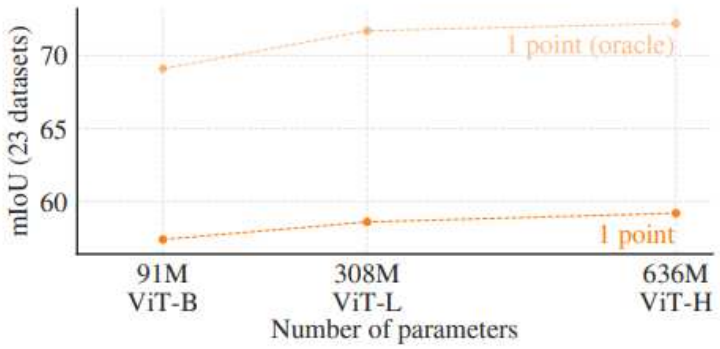

Figura 4.2: Grafico prestazionale dei checkpoint

Avendo a disposizione delle risorse fornite da *colab* come l'ampio storage e la presenza di un acceleratore *Nvidia Tesla* specializzato in calcoli paralleli con CUDA, è possibile scegliere il pesante ViT-H ottenendo tempi di elaborazione accettabili.

Inoltre, vista la memoria volatile della *VirtualMachine* è necessario collegare uno *storage* che ci permetta un accesso più rapido evitando ogni volta le operazioni di upload. Utilizzando *Google Colab* abbiamo a disposizione il servizio *Drive* che si interfaccia in modo semplice e molto flessibile.

[1]Dataset progettato per addestrare modelli di segmentazione di oggetti a uso generale a partire da immagini di tutto il mondo(1.1 miliardi di maschere).

4.2 Manipolazione delle immagini

Il processo di manipolazione delle immagini consiste nel prendere i video di partenza dal nostro dataset, estraendo dei frame e il relativo bounding box dell'occhio (tramite i *landmarks*), si cambiano i formati delle immagini, e poi si generano le maschere con la segmentazione del modello SAM.

4.2.1 Facemesh e landmarks

Questa fase dello sviluppo definisce innanzitutto alcuni colori in formato RGB (verde, rosso e blu) e poi definisce delle liste di coordinate per gli occhi destro e sinistro. Dopodiché, vengono combinate queste coordinate per creare due insiemi di punti rappresentanti gli occhi e i collegamenti tra questi punti vengono definiti come insiemi immutabili di connessioni.

Successivamente, creiamo una lista di numeri rappresentanti tutti i *landmark*(punti di riferimento) disponibili, aggiungendo prima gli indici degli occhi (destro e sinistro) a questa lista e poi determinando gli altri landmark come quelli che non appartengono agli occhi.

```
# Right eye.
rightEyeUpper0 = [246, 161, 160, 159, 158, 157, 173]
rightEyeLower0 = [33, 7, 163, 144, 145, 153, 154, 155, 133]
R_EYE = rightEyeUpper0 + rightEyeLower0
RIGHTEYE_CONNECTIONS = frozenset([
(33, 7), (7, 163), (163, 144), (144, 145), (145, 153), (153, 154), (154, 155),
(155, 133), (33, 246), (246, 161), (161, 160), (160, 159),
(159, 158), (158, 157), (157, 173), (173, 133),])

# Left eye.
leftEyeUpper0 = [466, 388, 387, 386, 385, 384, 398]
leftEyeLower0 = [263, 249, 390, 373, 374, 380, 381, 382, 362]
L_EYE = leftEyeUpper0 + leftEyeLower0
LEFTEYE_CONNECTIONS = frozenset([
(263, 249), (249, 390), (390, 373), (373, 374), (374, 380), (380, 381),
(381, 382), (382, 362), (263, 466), (466, 388), (388, 387),
(387, 386), (386, 385), (385, 384), (384, 398), (398, 362),])
```

Figura 4.3: Codice utilizzato per la descrizione dei landmark degli occhi

Infine utilizzeremo una funzione chiamata *recognizeLandmark* che prende in input un valore e restituisce una stringa che indica se quel valore corrisponde a un punto di riferimento dell'occhio destro, sinistro o ad altro punto di riferimento.

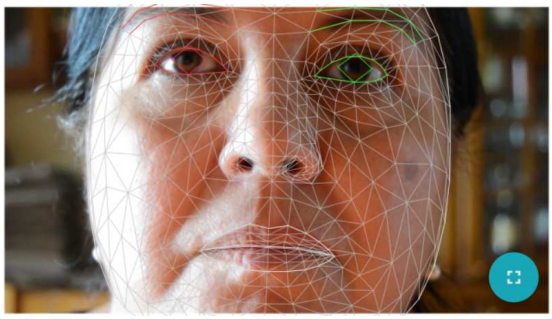

Figura 4.4: Prima fase di rilevamento dei landmarks

A questo punto, come si può notare dall'immagine4.4, si distinguono occhio destro e sinistro per restringere la regione di studio e analizzare solo i punti che ci interessano.

4.2.2 Bounding box

In questa sezione andremo ad estrapolare dei box dal risultato precedente4.4.

Innanzitutto prendiamo i punti che più si avvicinano agli occhi e tracciamo un riquadro intorno ad essi tramite la funzione *cv2.rectangle* andando però ad aggiungere un margine di confidenza con *right e left offset* per avere in esame tutto l'occhio.

```
#Ritaglia e salva rinominando ogni immagine
frame = cv2.rectangle(frame, start_point, end_point, (0,255,0), 2)
cropped_image_left = frame[(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))]
image_left_name=f"{file_name}_{count:06d}_L.png"
cv2.imwrite(os.path.join(path, image_left_name),cropped_image_left)
```

Figura 4.5: Codice realizzato per il tracciamento dei bounding box sinistro

Viene realizzato lo stesso tipo di codice facendo una distinzione per l'occhio destro, in quanto sarà possibile lavorare separatamente su entrambi. Una volta individuato, il box viene visualizzato sull'immagine escludendo tutti gli altri punti e salvandoli sotto forma di immagini statiche con un nome e con un indice nel nostro *storage* con un percorso definito.(Fig.4.7)

In questo momento il risultato che andremo a visualizzare sarà l'immagine di partenza su cui produrremo delle maschere per analizzare lo stato della pupilla.

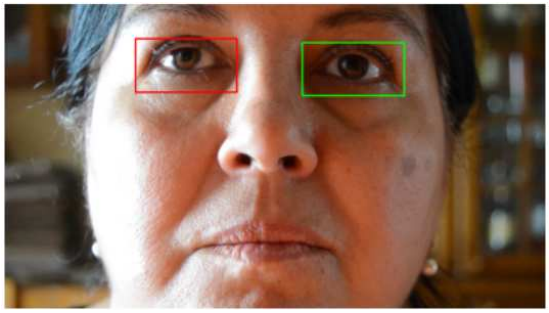

Figura 4.6: Bounding box

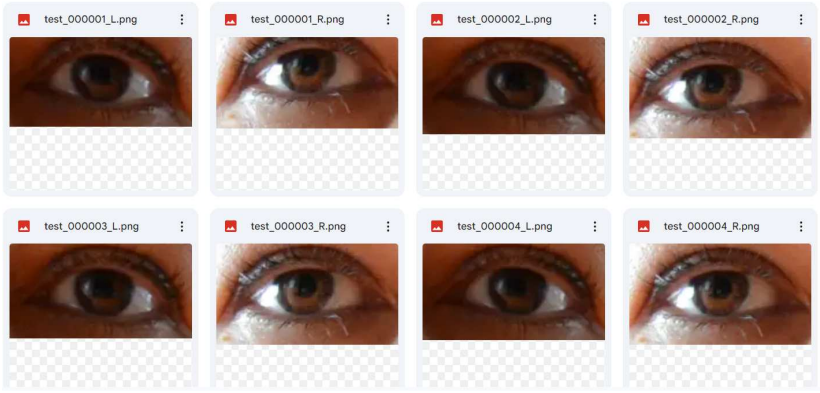

Figura 4.7: Bounding box salvati come immagini nello *storage drive*

4.3 Segmentazione con il modello SAM

In questa fase vedremo come utilizzare il modello SAM con le giuste implementazioni con entrambe le tipologie, ovvero *AutomaticMaskGenerator e SamPredictor*, ma come vedremo nel paragrafo4.3.2 questo tipo di modello non sarà ottimale per il nostro caso e non approfondiremo.

Per iniziare importiamo il modulo *torch* (cap3.4.3), specifichiamo il dispositivo su cui eseguire il modello, CUDA se disponibile altrimenti CPU. Dopodiché specifichiamo il *checkpoint(Vit-H)*. A questo punto inizializziamo il modello con il percorso del *checkpoint* e il dispositivo da utilizzare.

```
import torch

DEVICE = torch.device('cuda:0' if torch.cuda.is_available() else 'cpu') #utilizzo cuda se disponibile
MODEL_TYPE = "vit_h" #specifico il tipo di checkpoint

from segment_anything import sam_model_registry, SamAutomaticMaskGenerator, SamPredictor

sam = sam_model_registry[MODEL_TYPE](checkpoint=CHECKPOINT_PATH).to(device=DEVICE) #fornisco la configurazione del modello
```

Figura 4.8: Set-up Segment Anything

4.3.1 AutomaticMaskGenerator

Utilizziamo questo tipo di implementazione prendendo i risultati 4.8 salvati nello *storage Drive* generando tutte le maschere possibili e grazie al modello SAM abbiamo a disposizione, per ogni individuazione, un set di *keys* come l'area della maschera e l'accuratezza necessari per la valutazione del risultato.

Dato il video di partenza andremo a estrarre il maggior numero di maschere per ogni singolo ritaglio, fornendo per ogni frame analizzato i seguenti dati:

- Il numero di maschere generate
- Il predicted-iou e stability-score
- Le coordinate del centroide

A questo punto, una volta ottenute tutte le maschere cerchiamo di ottenere la migliore andando a selezionare dalla lista quella con la forma circolare più esatta che corrispondere all'iride.

Nell'ultima parte della segmentazione invece andiamo a ottenere i seguenti risultati.

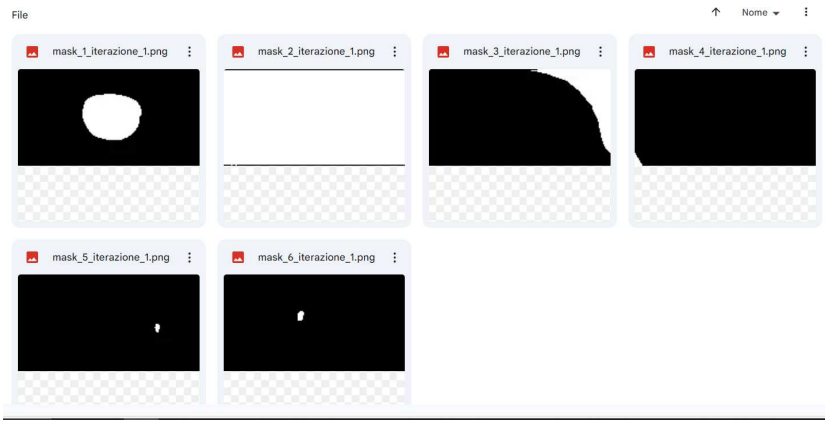

Figura 4.9: Lista di maschere salvate nello *storage* e classificate

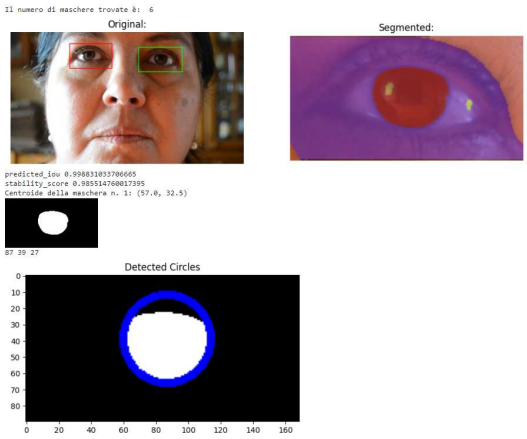

Figura 4.10: Risultato ottenuto dopo la segmentazione e la ricerca della maschera migliore

Vorrei precisare che le maschere prodotte sono molte rispetto a quelle visualizzate, questo perché ci siamo soffermati sull'analisi di 5 frame diversi, infatti andando avanti nell'esecuzione si otterranno altre maschere con l'iride in posizioni differenti rispetto a quella centrale.

Non si può escludere di migliorare ulteriormente la precisione del modello andando ad agire sui parametri di configurazione:

1. analizzare il video dinamicamente e non ogni singolo frame in modo statico
2. aumentare la velocità di esecuzione generando solo la maschera necessaria allo studio;
3. fornire un *dataset* con immagini a risoluzione più alta;

4.3.2 SamPredictor

Utilizzando questa tipologia di modello i risultati che abbiamo ottenuto non erano del tutto concordi con il nostro progetto, le maschere ottenute erano confuse e non precise nell'individuazione dell'iride.

Questo modello, dopo la segmentazione fornisce il bounding box segmentato e 3 maschere di base per ogni singolo frame analizzato ma come si nota dalla figura4.12 non otteniamo un forma esatta dell'iride.

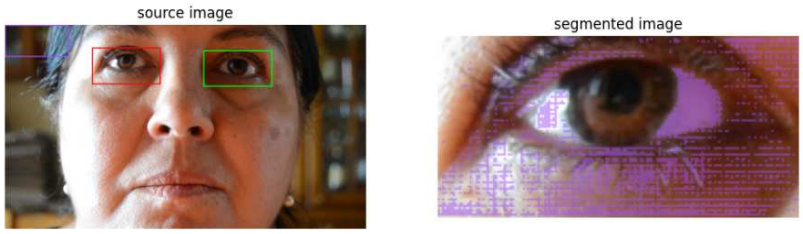

Figura 4.11: Segmentazione

Figura 4.12: Maschere generate dal SamPredictor

4.4 Valutazione delle maschere

Ottenuti i risultati per una decina di frame andiamo ad analizzarli. Come detto in precedenza il modello SAM mette a disposizione dei *keys* che misurano la correttezza delle maschere e la loro precisione.

Abbiamo calcolato questi valori per ogni maschera e abbiamo confrontato i dati per la maschera migliore dei primi 5 frame ottenendo dei range di confidenza. In entrambi i casi più il valore è alto e più significa che il modello è stato preciso.

Frame	Maschere generate	Stability Score	Predicted IoU
1	10	0.962	0.999
2	6	0.985	0.998
3	7	0.977	0.996
4	7	0.976	0.997
5	8	0.981	0.998

Il Predicted IoU è una misura di quanto la maschera prevista si sovrapponga con la maschera di riferimento. Maggiore è il valore maggiore è la precisione della maschera prevista.

Lo Stability Score è una misura della coerenza o della stabilità della maschera prevista nel tempo o nello spazio. Maggiore è il valore, maggiore è la stabilità della maschera.

4.4.1 Osservazioni

Nonostante siano valori presi su pochi frame possiamo subito renderci conto della precisione del modello, infatti in ogni caso otterremo una stabilità al 96% e una precisione delle maschere al 99%

Considerando i nostri strumenti e le tecniche utilizzate abbiamo ottenuto risultati quasi perfetti nella stima e nella precisione delle maschere ottenute.

Capitolo 5

Considerazioni finali

5.1 Sviluppi futuri

I campi dove è possibile sfruttare queste tecnologie e questa implementazione sono molteplici infatti si può estendere il seguente lavoro ad altri tratti del volto, per avere un maggior controllo sul conducente.

Si possono realizzare dei sistemi di guida semi-autonoma con le giuste implementazioni, con conseguente rilevamento dello stato psico-fisico di un individuo monitorando le espressioni facciali.

Il lavoro svolto ha gettato le basi per la creazione di sistemi di *Machine Learning*, rendendosi versatile per eventuali implementazioni, anche in altri ambienti.

5.2 Conclusioni

Terminato il presente lavoro di tesi si può affermare nuovamente l'importanza dello sviluppo di soluzioni avanzate per riconoscere il grado di stanchezza di un guidatore.

Sono rimasto affascinato e colpito dall'importanza che può avere ogni piccolo dettaglio e dato nello sviluppo di un sistema e sono sempre più convinto che l'*Intelligenza Artificiale* sarà fondamentale per la prevenzione di incidenti di ogni tipo e accompagnerà sempre di più il nostro sviluppo.

Bibliografia

[1] https://www.istat.it/it/files/2023/07/REPORT_INCIDENTI_STRADALI_2022_IT.pdf.

[2] https://mbenz.it/mercedes-attention-assist.

[3] https://www.dailymail.co.uk/sciencetech/article-8046733/SEAT-experimenting-technology-studies-driver-detect-theyre-falling-asleep.html.

[4] https://www.froglearning.it/intelligenza-artificiale-ed-e-learning/.

[5] https://www.ibm.com/it-it/topics/neural-networks/.

[6] https://blog.mooncascade.com/machine-learning-vs-deep-learning-when-do-you-need-an-expert/.

[7] https://colab.research.google.com/.

[8] https://mediapipe-studio.webapps.google.com/demo/face_landmarker.

[9] https://opencv.org/.

[10] https://viso.ai/deep-learning/segment-anything-model-sam-explained/.

[11] https://arxiv.org/pdf/2304.02643.pdf.

[12] https://inside-machinelearning.com/en/sam-segment-anything-tutorial/.

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

[14] https://www.elettronicain.it/blog/2019/12/26/le-telecamere-che-salvano-la-vita-a-conducente-e-passeggeri/.

[15] https://www.iotworlds.com/it/
what-is-the-role-of-machine-learning-in-iot/.

[16] https://www.oracle.com/it/artificial-intelligence/
machine-learning/what-is-machine-learning/.

[17] https://www.bigdata4innovation.it/intelligenza-artificiale/
deep-learning-cose-ed-esempi-dellapprendimento-profondo.

[18] https://www.ai4business.it/intelligenza-artificiale/
deep-learning/reti-neurali/.

[19] https://semiengineering.com/deep-learning-spreads/.

[20] https://www.ibm.com/topics/deep-learning/.

[21] https://www.image-net.org/.

[22] https://cloud.google.com/tpu/docs/tpus?hl=it.

[23] https://www.tensorflow.org/lite?hl=it.

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

Elenco delle figure

Ringraziamenti

Vorrei ringraziare la mia famiglia, a loro devo questo percorso e a loro dedico questo traguardo, perchè senza il supporto che mi hanno dato non sarei mai riuscito a raggiungerlo. Non sempre sono stato in grado di dirlo o di dimostrarlo ma vi amo, dal profondo del mio cuore.

Ringrazio tutte le persone che hanno fatto parte indirettamente di questo percorso, ai miei amici e colleghi conosciuti durante gli studi e non, con i quali ho condiviso i momenti piu difficili vissuti finora.

Infine vorrei ringraziare una persona che mi è stata sempre vicina e con il quale ho condiviso più di un percorso di studi ma quasi un percorso di vita. A un amico con la A maiuscola: grazie Diste, ti sarò sempre grato.