UNIVERSITÀ DI PISA

Corso di Laurea Triennale in Informatica

Sviluppo di una Piattaforma di Acquisizione Multi-sensoriale per Determinare la Quality of Experience e lo Stato Emozionale dell'Utente

Tutore Accademico:
Prof. Vincenzo Lomonaco
Tutore Esterno:
Dott. Alberto Gotta

Candidato:
Gabriele Masciotti

Anno Accademico 2020/2021

Indice

Capitolo 1

Introduzione al progetto

Il contenuto di questo elaborato costituisce una relazione sulla realizzazione del progetto di tirocinio da me svolto presso l'Istituto di Scienza e Tecnologie dell'Informazione del CNR di Pisa.

L'obiettivo del tirocinio è di creare un framework che estrae dei dati dalle espressioni del volto di un utente e da un sensore che egli indossa e costruisce dei dataset da utilizzare per determinare in maniera automatica la Quality of Experience (QoE) dell'utente mentre compie un azione specifica (per esempio mentre guarda un video o guida un drone da remoto) ed il suo stato emozionale. Tale framework verrà impiegato nel progetto europeo H2020 Teaching, insieme ad altre componenti realizzate da studenti e ricercatori del CNR e del dipartimento di Informatica[1].

Nella fase iniziale del tirocinio mi sono dedicato ad uno studio approfondito del linguaggio di programmazione python, concentrandomi su strutture dati e librerie principali, per poi passare ad installare e studiare il software OpenFace[2], che ho utilizzato per estrarre le features dal volto impiegando la videocamera del PC.

Quindi, ho implementato lo script che raccoglie i dati estratti da OpenFace e ho fatto un test di predizione della QoE; dopo aver allenato il modello di machine learning sfruttando un dataset già costruito[3], ho avviato l'estrazione live delle features dal mio volto per ottenere in output la mia Quality of Experience.

[1] https://teaching-h2020.eu/

[2] https://github.com/TadasBaltrusaitis/OpenFace [1]

[3] https://net4u.diee.unica.it/?p=1728 [3]

In seguito, dopo aver partecipato ad un meeting del progetto Teaching, ho esteso il progetto allo scopo di integrare il dataset ottenuto con OpenFace con i dati estratti da un sensore indossabile della ShimmerSensing[4], in modo da poter determinare insieme alla QoE anche l'emotional state dell'utente sia da features del volto che da features sensoriali quali ECG (elettrocardiogramma), EMG (elettromiogramma) e GSR (galvanic skin response).

Tutti i dettagli riguardanti le varie fasi di svolgimento del progetto sono forniti nelle sezioni seguenti. Inizierò con una breve introduzione al concetto di Quality of Experience ed Emotion Recognition nonché alla loro importanza in diversi ambiti, come quello di Human In The Loop, per poi passare ad illustrare il software OpenFace. Soltanto dopo questa fase introduttiva descriverò i dettagli implementativi dei due use case del progetto di tirocinio, il test di predizione della QoE e il costruttore di dataset.

Tutte le installazioni, le implementazioni e i test descritti in questo documento sono stati effettuati sul sistema operativo Ubuntu 20.04.3 LTS in esecuzione su PC Lenovo Yoga 910 con memoria RAM 8 GB e processore Intel® Core™ i7-7500U 2.70GHz × 4. Se eseguiti su computer non performanti, i programmi, soprattutto il creatore di dataset, potrebbero subire sensibili rallentamenti e possibili malfunzionamenti.

Questo tirocinio di laurea è stato effettuato interamente in modalità a distanza.

[4]https://shimmersensing.com/products/development-kits/

Capitolo 2

Approccio alla Quality of Experience e al concetto di Human In The Loop

L'idea di base del tirocinio nasce dalla necessità sempre crescente di individuare ed adattare in modo automatico la qualità del servizio, nel senso più ampio del termine, di cui un utente sta usufruendo. Nel mondo sempre più informatizzato e tecnologico in cui ci siamo abituati a vivere negli ultimi anni, il numero e la diversità dei servizi digitali che tutti utilizziamo ogni giorno sono impressionanti. Basti pensare che soltanto in Italia, alla fine del 2020, sono quasi 50 milioni le persone online ogni giorno, con oltre 80 milioni di smartphone per una popolazione residente di 60 milioni[1].

Dallo streaming video a quello musicale, dai software di instant messaging a quelli per le video-conferenze online, dalla navigazione GPS alle piattaforme di realtà virtuale, dalla guida di veicoli a pilotaggio remoto o in presenza di sistemi di guida autonoma, al controllo di robot sempre più avanzati. In tutti questi casi d'uso, l'elemento comune è il concetto di *human in the loop*, essendo l'uomo o l'attore che fa parte del sistema, un soggetto che usufruisce del servizio offerto da un sistema cyber-fisico. Con quest'ultimo termine, oggi molto in voga, si definisce un sistema dotato di capacità di computing e di networking e, non da ultimo, anche eventualmente di intelligenza artificiale, in cui si richiede l'interazione umana. Nei sistemi HITL infatti, un essere umano è sempre al centro dei sistemi stessi e parte integrante di ogni

[1] https://wearesocial.com/blog/2020/01/digital-2020/

simulazione che viene effettuata, e di conseguenza ne influenza decisamente l'esito e l'individuazione di problemi e requisiti che altrimenti non sarebbero facili da captare.

Negli ultimi anni si è parlato molto di machine learning e intelligenza artificiale, che ormai sembra aver raggiunto ogni settore in cui la tecnologia può esprimere al meglio il suo potenziale, sviluppando capacità di calcolo mai viste prima. Tuttavia, il machine learning da solo non può fare magie, richiede sempre la mediazione dell'uomo, che è l'unico in grado di dare ai computer l'accesso alla conoscenza del mondo reale e fare in modo che siano effettivamente capaci di rispondere alle nostre reali esigenze. L'approccio human in the loop coinvolge le persone nel circolo virtuoso – il loop appunto – in cui si addestrano, perfezionano e monitorano i modelli di Machine Learning.

2.1 Progetto europeo H2020 Teaching

Il concetto di HITL trova immediata applicazione nei sistemi cyber-fisici *(CPSoS - Cyber-Physical System of Systems)* definibili come un'integrazione di sistemi di diversa natura, che si originano dalla fusione di oggetti fisici e reali (in quanto "percepibili" dai sensi umani; da qui il termine "Physical"), piattaforme computazionali e reti di telecomunicazioni, il cui scopo principale è il controllo di un processo fisico e, attraverso il feedback, il suo adattamento in tempo reale a nuove condizioni operative. I CPS, dunque, si fondano su oggetti correlati che, attraverso sensori, attuatori e connessioni di rete, generano e acquisiscono dati di varia natura, riducendo così le distanze e le asimmetrie informative tra tutti gli elementi del sistema e definendo un ambiente sfaccettato e dinamico in cui l'autonomia è fondamentale per governare la complessità delle interazioni tra il mondo virtuale e quello fisico con il minimo intervento umano (si pensi per esempio alla guida autonoma di veicoli). Tuttavia, anche quando viene esercitato il grado più avanzato di autonomia del sistema, l'essere umano è una variabile che non può essere esclusa dall'equazione CPSoS, in particolare in scenari critici per la sicurezza come per esempio la guida autonoma.

L'obiettivo del progetto europeo Teaching è infatti proprio quello di abbracciare il concetto di ***Intelligenza Umanistica***, dove le entità cibernetiche e biologiche cooperano in un reciproco potenziamento verso un traguardo condiviso e dove il feedback umano costituisce un fattore cruciale per l'adattività dei CPSoS.

Diventa pertanto di primaria importanza stabilire efficacemente la qualità del servizio (QoE) percepita dall'utente ed interpretarla come importante feedback al fine di progettare un software e un sistema informatico che supportino lo sviluppo di applicazioni CPSoS adattive e affidabili, consentendo di guidare, ottimizzare e personalizzare la fornitura dei servizi offerti. Tutto questo si ottiene grazie alla cooperazione sinergica uomo-CPSoS nello spirito dell'Intelligenza Umanistica, sfruttando le metodologie di intelligenza artificiale e il monitoraggio continuo dello stato fisiologico, emotivo e cognitivo (PEC) umano per consentire applicazioni con livelli di autonomia e flessibilità senza precedenti, pur mantenendo l'affidabilità richiesta da qualsiasi sistema critico per la sicurezza che opera con un essere umano nel circuito.

Il mio compito, con il progetto di tirocinio descritto in queste pagine, è stato fare il primo passo nella direzione che abbiamo delineato nelle righe sopra, cioè raccogliere tutti i dati che possono essere utili nel processo di individuazione della **QoE** e quindi del feedback umano, prendendo in considerazione le features estratte dalle espressioni del volto e alcuni dati sensoriali, allo scopo di costruire dei dataset in modo automatico, che poi verranno utilizzati per determinare lo **stato emozionale** dell'utente e procedere di conseguenza con l'adattamento del funzionamento del sistema e del servizio offerto.

Capitolo 3

OpenFace

Il primo strumento fondamentale che ho utilizzato nello svolgimento del progetto è OpenFace. OpenFace è un software open-source sviluppato da un ricercatore dell'Università di Cambridge in collaborazione con il CMU MultiComp Lab presso il Language Technologies Institute della Carnegie Mellon University, un'università privata di Pittsburgh, in Pennsylvania.

La missione di MultiComp Lab è costruire gli algoritmi e le basi computazionali per comprendere l'interdipendenza tra i comportamenti umani verbali, visivi e vocali espressi durante le interazioni comunicative sociali, integrando competenze di apprendimento automatico, visione artificiale, elaborazione del linguaggio e del linguaggio naturale, informatica affettiva e psicologia sociale.

Proprio in quest'ottica il software OpenFace è stato implementato, ed è ancora attivamente sviluppato, per essere destinato appunto ai ricercatori di visione artificiale e apprendimento automatico, alla comunità dell'informatica affettiva e in generale alle persone interessate alla creazione di applicazioni interattive basate sull'analisi del comportamento facciale.

Il **comportamento facciale** è stato sostanzialmente definito come una composizione di:

- *facial landmark location*, che costituisce il rilevamento e la localizzazione di determinati punti chiave del viso (come mostrato nella figura 3.1), importante step in numerosi ambiti che vanno dal riconoscimento biometrico alla comprensione degli stati mentali;

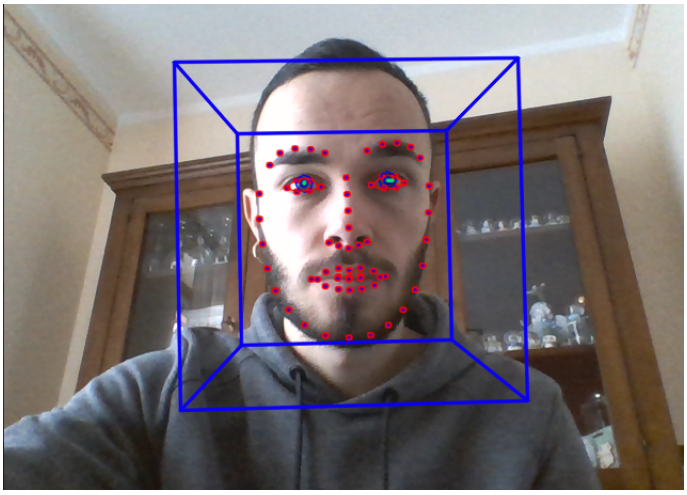

Figura 3.1: Nell'immagine sono evidenziati i landmarks facciali individuati da OpenFace.

- *posa della testa*, che svolge un ruolo importante nella percezione e nell'espressione delle emozioni e dei segnali sociali;
- *direzione dello sguardo*, fondamentale quando si valutano attenzione, abilità sociali e salute mentale, così come l'intensità delle emozioni;

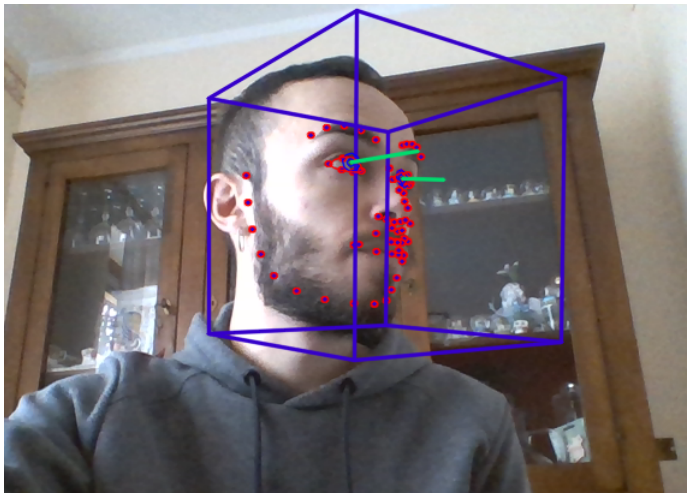

Figura 3.2: Notare il riquadro blu che individua la posizione della testa e i raggi verdi che seguono la direzione dello sguardo.

- *espressioni facciali*, che rivelano intenti ed esprimono emozioni. Queste vengono rilevate dal software grazie all'utilizzo delle Action Units, delle quali tratterò tra poche righe.

Il toolkit OpenFace, mediante l'utilizzo di una rete neurale convoluzionale e di alcuni algoritmi di rilevamento facciale[1], è in grado di determinare le features di cui ai punti precedenti analizzando volti in file video, in immagini statiche o anche fornendo prestazioni in live camera essendo in grado di funzionare con una semplice webcam, senza alcun hardware specializzato. Queste features estratte dai volti vengono poi inserite in un file prodotto in output.

3.1 Formato dei dati prodotti da OpenFace

Il più significativo output prodotto da OpenFace è il file csv contenente una grande tabella di features estratte dal volto analizzato, che ho utilizzato sia nel test di predizione della QoE sia nella costruzione del dataset finale completo (come sarà illustrato nei dettagli in seguito). In questa sezione riporterò, per completezza e per comodità del lettore, la descrizione delle features più importanti e soprattutto di quelle utilizzate in questo progetto, rimandando per maggiori dettagli alla lettura del wiki al link in piè di pagina.

Il file di output è costituito da una serie di record, uno per ogni video frame, il cui riferimento è enumerato dalla prima colonna. Dopo alcuni elementi di base quali, il timestamp (timer del video processato, in microsecondi), l'id del volto processato (nel caso ci fossero più volti questi avrebbero id diversi) e una confidence che evidenzia quanto il tracker sia sicuro in ogni istante della sua stima sulla localizzazione dei landmarks di cui abbiamo parlato poco righe fa, OpenFace inserisce nel file un lungo insieme di dati interessanti.

Direzione dello sguardo

Per quanto riguarda l'analisi dello sguardo, OpenFace utilizza delle coordinate x,y,z che individuano il vettore di direzione dello sguardo di entrambi gli occhi, quello sinistro, chiamato occhio 0, e quello destro, occhio 1. Tale vettore viene riprodotto come un raggio che parte dalla pupilla dell'occhio e

[1]Per maggiori dettagli sulle tecnologie utilizzate è possibile visitare: `https://github.com/TadasBaltrusaitis/OpenFace`

si muove seguendo appunto lo sguardo dell'utente (notare la rappresentazione grafica in figura 3.2). Non meno importanti sono i dati che evidenziano l'angolo di direzione dello sguardo in radianti. Questi vengono utilizzati per rilevare con precisione se l'utente inquadrato muove gli occhi da destra a sinistra e viceversa, determinando una variazione da positiva a negativa della feature *"gaze_angle_x"*, oppure se sta guarda verso l'alto o verso il basso, causando un cambiamento in *"gaze_angle_y"*. Se la persona avesse lo sguardo fisso di fronte a se guardando la videocamera, come ho fatto al momento dello scatto rappresentato nella figura 3.1, i due angoli appena descritti avrebbero valore vicino allo 0.

Posizione della testa

Seguono una serie di elementi di posizionamento della testa nello spazio dell'inquadratura della camera. Anche in questo caso, delle coordinate che riflettono la localizzazione dalla testa da parte del software e la sua rotazione in radianti lungo gli assi x, y e z, consentono al toolkit di determinare con accuratezza il cubo blu, visibile in figura 3.1 e 3.2, che si sposta a seconda dei movimenti effettuati dalla testa inquadrata.

Rilevamento dei landmarks facciali

Come abbiamo visto nella figura 3.1, il programma individua una serie di landmarks nel volto della persona inquadrata. Un'analisi della loro variazione può essere utilizzata in numerosi ambiti, come quello della determinazione dello stato emozionale. Le features $x_0, \ldots, x_{67}$, $y_0, \ldots, y_{67}$ e $X_0, \ldots, X_{67}$, $Y_0, \ldots, Y_{67}$, $Z_0, \ldots, Z_{67}$ rappresentano rispettivamente il posizionamento dei landmarks di OpenFace in pixel 2D ed in millimetri 3D. Nella figura 3.3 è illustrato un indice dei landmarks estratti dal volto da parte di OpenFace.

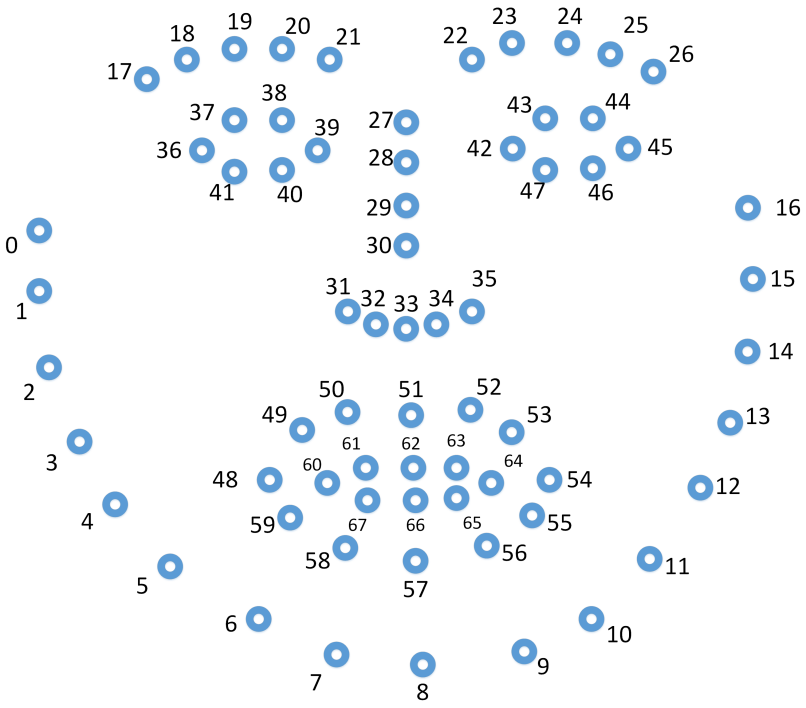

Figura 3.3: Indice dei landmarks individuati dal toolkit OpenFace.

Action Units facciali

Le ultime colonne del file riguardano le action units, le quali costituiscono un modo per descrivere l'espressione facciale umana. Le action units verranno trattate nei dettagli nella prossima sezione. Il toolkit OpenFace è in grado di individuare l'*intensità* di 17 action units diverse, ciascuna delle quali è identificata dalle features AU_r e la *presenza* o assenza di 18 action units, identificata dalle features AU_s.

3.2 Facial Action Coding System (FACS)

Un argomento sul quale vale senz'altro la pena spendere due parole è quello costituito dal FACS e di conseguenza dalle action units. Il Facial Action Coding System (detto in sigla FACS) è un sistema per classificare i movimenti del volto, così come appaiono nelle varie espressioni umane. Pubblicato nel

1978, è il frutto di una lunga ricerca portata avanti da Paul Ekman, professore di Psicologia al Dipartimento di Psichiatria dell'Università della California, e dal suo collega psicologo Fallace Friesen. Questo metodo ha l'intento di andare a studiare in che modo le contrazioni dei muscoli facciali, singolarmente o in combinazione con altri muscoli, modificano le sembianze di un viso. I movimenti dei singoli muscoli facciali sono codificati da FACS attribuendo una combinazione di codici corrispondenti a determinati micromovimenti, chiamati ***Action Unit***, effettuati dalla persona. Successivamente è possibile siglare anche l'intensità del movimento. Utilizzando FACS si possono codificare sistematicamente quasi tutte le espressioni facciali anatomicamente possibili, riducendole alle action units specifiche. Consiglio di visitare il sito al link in piè di pagina, in cui si può trovare una lista delle principali AU e dei corrispondenti micromovimenti muscolari facciali.[2].

Poiché le AU sono indipendenti da qualsiasi interpretazione, possono essere utilizzate per qualsiasi processo decisionale, quali: il riconoscimento delle emozioni di base, i comandi pre-programmati per un ambiente intelligente, contesti giuridici e investigativi (aggiungendo ulteriore materiale di analisi che può essere sapientemente utilizzato per formulare domande di approfondimento e di investigazione) o nella relazione medico-paziente. Esistono infatti correlazioni specifiche e predittive tra l'uso di determinate micro-espressioni e il disturbo presentato (fisico o psicologico che sia).

Figura 3.4: Alcune microespressioni correlate ai codici delle rispettive AU.

[2]Una lista delle principali action units può essere scorsa qui: `https://www.cs.cmu.edu/~face/facs.htm`

Le sette emozioni di base

Particolarmente interessante, anche per gli scopi del progetto descritto in questo elaborato, è il riconoscimento delle sette espressioni di base. In quello che può essere definito, per certi versi, il dizionario delle espressioni facciali di Ekman, il FACS, vengono definite le sette emozioni primarie (o universali) che sono espresse con le medesime espressioni facciali in tutti i soggetti osservati nei molteplici studi coinvolgenti culture di tutto il mondo. Queste 7 emozioni sono: **felicità, rabbia, paura, tristezza, disprezzo, disgusto** e **sorpresa**.

La ragione di questo fenomeno sembra risiedere nelle radici del nostro DNA che tutti noi, in quanto appartenenti alla specie umana, condividiamo a prescindere dall'etnia o nazionalità. Addirittura, secondo alcuni studiosi, senza queste emozioni primarie e questa capacità di esprimerle e riconoscerle universalmente l'uomo, isolato dai suoi simili e fisicamente inferiore agli altri animali, probabilmente oggi sarebbe estinto.

Una classificazione di queste emozioni è data utilizzando il sistema FACS, e quindi le action units corrispondenti ai movimenti del volto che risultano dall'espressione di queste, nella tabella sottostante (in figura 3.5).

Emozione	Action Units
Felicità	6+12
Tristezza	1+4+15
Sorpresa	1+2+5+26
Paura	1+2+4+5+7+20+26
Rabbia	4+5+7+23
Disgusto	9+15+16
Disprezzo	12+14

Figura 3.5: Codifica delle emozioni di base nelle action units corrispondenti.

Con questa codifica delle espressioni facciali umane è chiaramente possibile costruire un modello di intelligenza artificiale e allenarlo a riconoscerle. Come già detto, lo scopo del mio progetto di tirocinio è infatti proprio quello di costruire basi di dati adatte, fra le altre cose, a poter essere utilizzate per

allenare modelli di machine learning, in modo da renderli in grado di effettuare Emotion Recognition, sfruttando soprattutto i dati delle Action Units estratte da OpenFace.

3.3 Le Action Units di OpenFace

Il toolkit OpenFace è in grado di riconoscere un sottoinsieme delle action units, in particolare: 1, 2, 4, 5, 6, 7, 9, 10, 12, 14, 15, 17, 20, 23, 25, 26, 28, e 45. Per ognuna delle precedenti AUs OpenFace fornisce due dati.

- La presenza: intesa come la visibilità o meno nel volto analizzato del movimento muscolare correlato al codice dall'action unit specifica. Il dato relativo alla presenza è inserito nelle colonne che terminano con "_c" e può avere valore 1, per indicare che nel frame a cui il record in questione fa riferimento la action unit si è attivata, o 0 altrimenti.

- L'intensità: che indica quanto intensa su una scala da 1 a 5 (minimo e massimo rispettivamente) è l'AU presente nel frame del video. I valori delle intensità sono inseriti nelle colonne che terminano con "_r" e possono o avere valore pari a 0, quando l'action unit non è presente nel frame del video, oppure variare da 1 (action unit presente ma con la minima intensità) a 5 (presente con massima intensità).

OpenFace può estrarre Action Units da immagini, da sequenze di immagini e da video. Può anche farlo da video in cui sono inquadrate più persone ma non con la stessa accuratezza. Di seguito riporto alcune immagini di prove che ho effettuato personalmente durante l'utilizzo del software. Si noti come si attivano le varie Action Units con diverse intensità a seconda delle espressioni che OpenFace rileva dal mio volto.

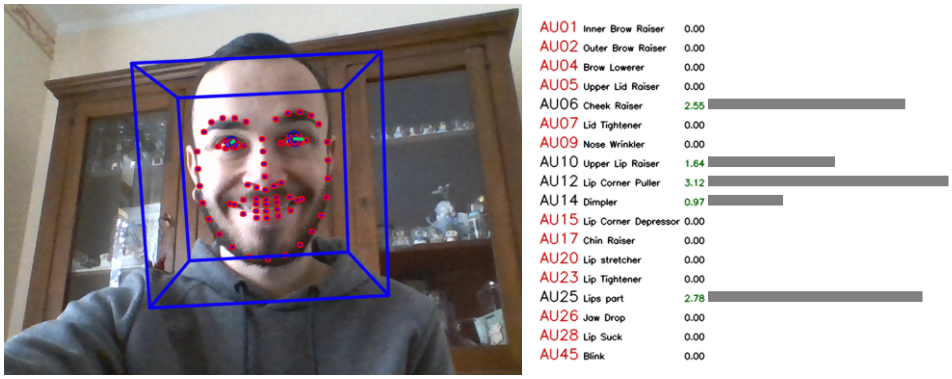

Figura 3.6: Quando OpenFace rileva un viso sorridente si attivano le AU corrispondenti. Si vede infatti che le attivazioni di maggior impatto sono quelle dell'emozione di base "Felicità", ossia quelle elencate nella tabella della figura 3.5.

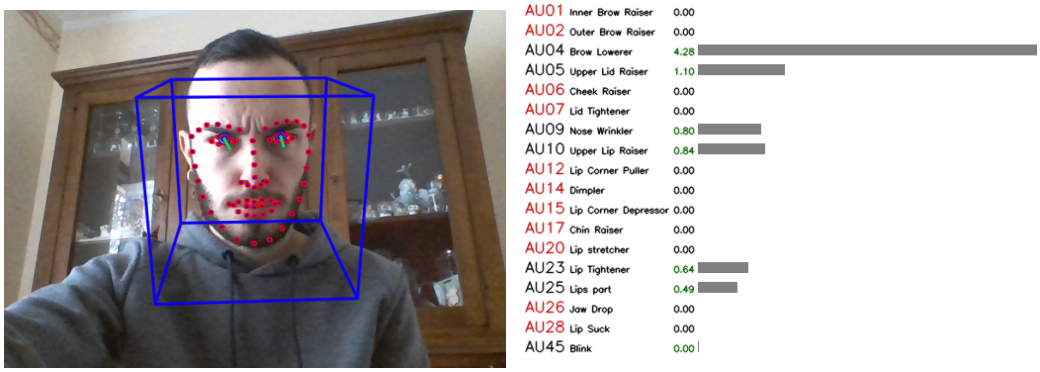

Figura 3.7: Viso che esprime l'emozione di base "Rabbia" e relative AU attivate.

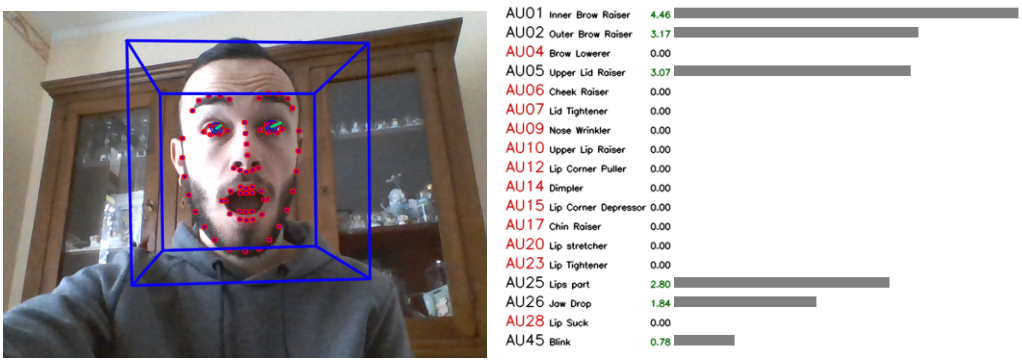

Figura 3.8: Viso che esprime l'emozione di base "Sorpresa" e relative AU attivate.

3.4 Installazione di OpenFace

In questa sezione riporto brevemente la lista dei passi da compiere per installare il software OpenFace su sistema operativo Ubuntu con versione superiore alla 16.04. L'installazione può essere effettuata su qualsiasi sistema ovviamente, e le istruzioni sono disponibili anche sul sito di OpenFace, ma nelle righe che seguono inserirò commenti e suggerimenti che il lettore troverà senz'altro utili in fase di installazione su Ubuntu, che può rivelarsi più complessa di quello che si pensa.

Si comincia con l'installazione del software e delle dependencies di cui OpenFace necessita per funzionare. Si apra quindi un terminale e si digitino i seguenti comandi:

```
sudo apt-get update
sudo apt-get install build-essential
sudo apt-get install g++-8
```

per ottenere il ***compilatore GCC***.

Si procede installando ***cmake***, che sarà necessario per effettuare la compilazione e il build di OpenFace:

```
sudo apt-get install cmake
```

Ora si passa all'installazione delle librerie che il toolkit utilizza per il suo funzionamento. Otteniamo pertanto ***OpenBLAS*** con il seguente comando:

```
sudo apt-get install libopenblas-dev
```

per poi procedere a scaricare e compilare ***OpenCV*** 4.1.0 con questi comandi:

```
sudo apt-get install git libgtk2.0-dev pkg-config
↪  libavcodec-dev libavformat-dev libswscale-dev

sudo apt-get install python-dev python-numpy libtbb2
↪  libtbb-dev libjpeg-dev libpng-dev libtiff-dev
↪  libdc1394-22-dev

wget https://github.com/opencv/opencv/archive/4.1.0.zip

sudo unzip 4.1.0.zip
cd opencv-4.1.0
sudo mkdir build
cd build

sudo cmake -D CMAKE_BUILD_TYPE=RELEASE -D
↪  CMAKE_INSTALL_PREFIX=/usr/local -D BUILD_TIFF=ON -D
↪  WITH_TBB=ON ..
sudo make -j2
sudo make install
```

Per finire abbiamo bisogno di effettuare download e compilazione di ***dlib***:

```
sudo wget http://dlib.net/files/dlib-19.13.tar.bz2
sudo tar xf dlib-19.13.tar.bz2
cd dlib-19.13
sudo mkdir build
cd build
sudo cmake ..
sudo cmake --build . --config Release
sudo make install
sudo ldconfig
cd ../..
```

Il completamento delle operazioni precedenti potrebbe richiedere svariati minuti. Opzionalmente si può scaricare una libreria di boost, che OpenFace utilizza se disponibile, con il comando che segue. Nella realizzazione del progetto descritto in questo documento tale libreria NON è stata utilizzata.

```
sudo apt-get install libboost-all-dev
```

Dopo aver seguito attentamente i passi precedenti si può finalmente procedere con l'installazione di OpenFace. Posizionarsi nella directory in cui si desidera che i file del software risiedano (tipicamente nella home) e digitare il seguente comando:

```
git clone https://github.com/TadasBaltrusaitis/OpenFace.git
```

dopodiché, come suggeriscono le istruzioni sul sito di OpenFace, è bene crearsi una cartella per memorizzare i file compilati:

```
cd OpenFace
mkdir build
cd build
```

A questo punto si compila il codice scaricato per mezzo delle seguenti istruzioni:

```
cmake -D CMAKE_CXX_COMPILER=g++-8 -D CMAKE_C_COMPILER=gcc-8
↪  -D CMAKE_BUILD_TYPE=RELEASE ..
```

```
make
```

Dopo aver atteso la terminazione della compilazione, che può richiedere un po' di tempo, si è conclusa l'installazione del software OpenFace.

A questo punto si può provare ad utilizzare il software. Per farlo basta per esempio digitare il codice riportato qui di seguito per estrarre le feature facciali da un video campione scaricato nel pacchetto di OpenFace.

```
./bin/FaceLandmarkVid -f "../samples/changeLighting.wmv" -f
↪  "../samples/2015-10-15-15-14.avi"
```

Si consiglia di leggere attentamente le righe che seguono allo scopo di risolvere facilmente un problema comune che non è riportato nei siti ufficiali del toolkit, ma che ne impedisce il funzionamento. Allo stato attuale delle impostazioni, il comando precedente dovrebbe terminare con questo errore:

```
ERROR: Could not load the landmark detector
```

Per sopperire al problema è necessario recarsi a questo link: `https://github.com/TadasBaltrusaitis/OpenFace/wiki/Model-download` e scaricare manualmente i quattro modelli CEN patch experts, 0.25, 0.35, 0.50, 1.00. Una volta effettuato il download, posizionarsi nella directory specificata dal path

```
/OpenFace/build/bin/model/patch_experts
```

e copiarci i quattro file .dat appena scaricati.

Arrivati a questo punto OpenFace funziona correttamente. Si può quindi provare di nuovo ad eseguire il comando precedente per visualizzare il tracking del volto delle persone riprese nel video di prova.

Una lista dei possibili comandi e degli argomenti da utilizzare è consultabile a questo indirizzo: `https://github.com/TadasBaltrusaitis/OpenFace/wiki/Command-line-arguments`.

Capitolo 4

Test di predizione della QoE

Il primo obiettivo del tirocinio, come già detto, è effettuare un test di predizione della Quality of Experience percepita da un utente che viene inquadrato da una videocamera mentre compie una determinata azione come per esempio guardare un video, utilizzare servizi digitali di vario genere oppure, in modo più specifico, guidare un drone da remoto. Quest'ultimo caso d'uso è di particolare interesse per un progetto del CNR, i cui ricercatori, al momento in cui scrivo queste righe, sono impegnati nella progettazione di un framework che aggiusta la qualità di streaming del video proveniente da un drone in base alla QoE percepita dall'operatore che lo guida, il cui volto viene inquadrato da una camera.

Pertanto in questo capitolo illustrerò come ho proceduto nella realizzazione del task richiestomi, descrivendo le fasi preparatorie all'esperimento e il funzionamento dello script che esegue le predizioni di QoE utilizzando i soli dati estratti dal software OpenFace, ampiamente descritto nel capitolo 3. Come verrà mostrato in seguito, la determinazione del valore della QoE, a seconda delle features estratte dal volto dell'utente, verrà effettuata da un modello di machine learning allenato utilizzando un dataset già predisposto, il quale sarà adattato alle esigenze mediante il calcolo di alcune metriche. Aprirò la trattazione partendo proprio da questa elaborazione e preparazione iniziale dei dati, per poi procedere all'allenamento del modello di machine learning con il dataset adattato, all'implementazione del programma che estrae i dati dal volto dell'utente utilizzando OpenFace e concluderò mostrando il funzionamento di quest'ultimo e i risultati ottenuti nelle prove di predizione fatte.

4.1 Dataset di allenamento

Il dataset che ho utilizzato in questo programma per allenare il modello di machine learning è quello che mi è stato proposto direttamente dal CNR, ossia quello prodotto da uno studio effettuato al Net4U [1], il laboratorio Networks for Humans, un gruppo di ricerca composto da circa 20 persone tra docenti, ricercatori e dottorandi dell'Università degli Studi di Cagliari ed affiliato al CNIT, Consorzio Universitario Italiano per le Telecomunicazioni. Lo studio [3] che ha avuto come risultato il dataset in questione riguarda proprio l'indagine sulle potenzialità di stimare implicitamente la qualità dell'esperienza (QoE) di un utente che usufruisce di servizi di streaming video, acquisendo un video del suo viso e monitorando la sua espressione facciale e direzione dello sguardo.

Per far questo, è stato condotto un test di crowdsourcing durante il quale ai partecipanti è stato chiesto di guardare e valutare la qualità di 20 video soggetti a diverse "impairments" (che in italiano può essere tradotto come "danneggiamenti", ossia alterazioni dei video che possono essere costituite da un ritardo di caricamento, una sfocatura di alcune scene, eccetera), mentre il loro volto è stato registrato con la webcam del loro PC. Successivamente è stato raccolto un insieme di dati estratti con OpenFace dai volti registrati degli utenti per poi utilizzarlo per studiare il comportamento di vari modelli d'intelligenza artificiale in fase di training e in fase di testing.

Il dataset prodotto, contiene le features estratte da 400 video di test e, per ciascuno di questi anche i punteggi di QoE assegnati dagli utenti che vanno da 1 (pessima qualità) a 5 (ottima qualità). Tale dataset può essere richiesto navigando al seguente link, alla sezione "QoE prediction from face expressions and gaze direction": `https://net4u.diee.unica.it/?p=1728`.

Preparazione del dataset

Dopo aver scaricato il dataset, composto da 10 file .csv contenenti i record di features estratte da OpenFace per ogni frame dei video del volto degli utenti ripresi e la QoE stimata da quest'ultimi, ho avviato la lettura di suddetti file da parte di uno script python che ho sviluppato per convertirli nel formato 'pickle', più agevole e veloce da utilizzare per i programmi che ho implementato.

[1] `https://net4u.diee.unica.it/`

Nel replicare la tecnica di costruzione del dataset di allenamento proposta nel paper di riferimento [3], secondo quanto richiesto dal CNR, non ho utilizzato tutte le features estratte da OpenFace ma soltanto un sottoinsieme selezionato utilizzando un'analisi ANOVA (ANalysis Of VAriance - analisi della varianza), al fine di individuare quelle più significative allo scopo. Nel paper, che invito a consultare qualora interessassero maggiori dettagli qui tralasciati perché lontani dalle finalità di questo documento (con attenzione a non farsi confondere da piccole imprecisioni nella selezione delle features, qui rettificate), si legge che le features più significative al fine di stimare la QoE degli utenti sono quelle elencate nella tabella sottostante.

Eye0-gaze x vector	AU 20 r
Eye0-gaze y vector	AU 23 r
Eye1-gaze x vector	AU 25 r
Eye1-gaze y vector	AU 26 r
Gaze x angle	AU 01 c
Gaze y angle	AU 02 c
AU 01 r	AU 04 c
AU 02 r	AU 05 c
AU 04 r	AU 06 c
AU 07 r	AU 09 c
AU 09 r	AU 10 c
AU 12 r	AU 15 c
AU 14 r	AU 17 c
AU 15 r	AU 23 c
AU 17 r	AU 28 c
	AU 45 c

Figura 4.1: Features di OpenFace selezionate mediante analisi ANOVA, significative per la predizione della QoE.

Sono stati quindi selezionati i dati relativi alla direzione dello sguardo, nello specifico i vettori di direzione x e y dello sguardo da entrambi gli occhi e gli angoli formati dalla direzione dello sguardo in radianti (le features estratte

da OpenFace sono spiegate nella sezione 3.1), e a 25 Action Units. Per quanto riguarda queste ultime, l'analisi ANOVA ha individuato che per la stima della QoE sono più significative le microespressioni seguenti:

- **AU 01**: *sollevamento interno delle sopracciglia*, da considerarsi sia nella presenza che nell'intensità (notare che infatti sono presenti sia AU01_c che AU01_r);
- **AU 02**: *sollevamento esterno delle sopracciglia*, anch'esso sia nella presenza che nell'intensità;
- **AU 04**: *abbassamento delle sopracciglia*, presenza e intensità;

Figura 4.2: Illustrazione dei micromovimenti esprimibili con le sopracciglia, selezionati nei punti precedenti.

- **AU 5**: presenza di *palpebre spalancate*, in segno di incredulità o stupore;
- **AU 6**: presenza di *guance sollevate*, che esprimono gioia, in un sorriso marcato;

Figura 4.3: Microespressione corrispondente alla AU 6.

- **AU 07**: intensità del *restringimento delle palpebre*

Figura 4.4: Microespressione corrispondente alla AU 7. Palpebre socchiuse come in segno di sospetto o scarsa visuale.

- **AU 09**: *arricciamento del naso*, presenza e intensità;

Figura 4.5: Microespressione corrispondente alla AU 9.

- **AU 10**: presenza di *sollevamento del labbro superiore*, ad indicare disgusto o disappunto;

Figura 4.6: Microespressione corrispondente alla AU 10.

- **AU 12**: intensità del *sollevamento dell'angolo delle labbra*, ad annunciare un sorriso;
- **AU 14**: intensità delle *fossette sulle guance*;

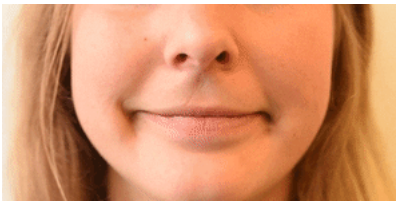

Figura 4.7: Microespressione corrispondente alla AU 14.

- **AU 15**: *abbassamento degli angoli delle labbra*, presenza e intensità;

Figura 4.8: Microespressione corrispondente alla AU 15.

- **AU 17**: *innalzamento del mento*, presenza e intensità;

Figura 4.9: Microespressione corrispondente alla AU 17.

- **AU 20**: *allungamento delle lebbra* nell'intensità;

Figura 4.10: Microespressione corrispondente alla AU 20.

- **AU 23**: *contrazione delle labbra* sia nell'intensità che nella presenza;

Figura 4.11: Microespressione corrispondente alla AU 23.

- **AU 25**: *labbra socchiuse*, tenendo in considerazione l'intensità (come fosse l'intensità dello stupore);
- **AU 26**: *bocca spalancata*, anche di quest'ultima considerando l'intensità;
- **AU 28**: presenza di *assottigliamento delle labbra*;

Figura 4.12: Microespressione corrispondente alla AU 28.

- **AU 45**: presenza di *battito di ciglia*.

Ho quindi proseguito la preparazione del dataset di allenamento con un filtraggio delle features contenute in quello scaricato, al fine di ridurlo a contenere soltanto i dati interessanti ai fini dell'esperimento. Per completezza di trattazione e per agevolare il lettore che desidera riprodurre l'esperimento, riporto qui di seguito il codice python per eseguire questa procedura.

```
import pandas as pd
import numpy as np

needed_features = [' id ',' gaze_0_x',' gaze_0_y','
↪  gaze_1_x',' gaze_1_y',' gaze_angle_x',' gaze_angle_y','
↪  AU01_c',' AU02_c',' AU04_c',' AU05_c',' AU06_c','
↪  AU07_c',' AU09_c',' AU10_c',' AU12_c',' AU14_c','
↪  AU15_c',' AU17_c',' AU20_c',' AU23_c',' AU25_c','
↪  AU26_c',' AU28_c',' AU45_c',' AU01_r',' AU02_r','
↪  AU04_r',' AU07_r',' AU09_r',' AU12_r',' AU14_r','
↪  AU15_r',' AU17_r',' AU20_r',' AU23_r',' AU25_r','
↪  AU26_r','QoE']

pickle1 = pd.read_pickle("dataset/pickle1")
pickle1 = pickle1.filter(needed_features)
pickle2 = pd.read_pickle("dataset/pickle2")
pickle2 = pickle2.filter(needed_features)
pickle3 = pd.read_pickle("dataset/pickle3")
pickle3 = pickle3.filter(needed_features)
pickle4 = pd.read_pickle("dataset/pickle4")
pickle4 = pickle4.filter(needed_features)
pickle5 = pd.read_pickle("dataset/pickle5")
pickle5 = pickle5.filter(needed_features)
```

```
pickle6 = pd.read_pickle("dataset/pickle6")
pickle6 = pickle6.filter(needed_features)
pickle7 = pd.read_pickle("dataset/pickle7")
pickle7 = pickle7.filter(needed_features)
pickle8 = pd.read_pickle("dataset/pickle8")
pickle8 = pickle8.filter(needed_features)
pickle9 = pd.read_pickle("dataset/pickle9")
pickle9 = pickle9.filter(needed_features)
pickle10 = pd.read_pickle("dataset/pickle10")
pickle10 = pickle10.filter(needed_features)

frames = [pickle1, pickle2, pickle3, pickle4, pickle5,
↪  pickle6, pickle7, pickle8, pickle9, pickle10]

complete_dataset = pd.concat(frames)

complete_dataset.to_pickle("complete_dataset")
```

Otteniamo pertanto un file pickle che costituisce il dataset con le features di nostro interesse; in aggiunta a quelle utili alla predizione della QoE elencate prima, ci sono anche tutte le altre AUc perché serviranno per il calcolo delle metriche, come illustrato nella prossima sezione. Tale dataset avrà il seguente formato:

	id	gaze_0_x	gaze_0_y	gaze_1_x	gaze_1_y	gaze_angle_x	gaze_angle_y	AU01_c	AU02_c	AU04_c	...	AU09_r	AU12_r	AU14_r	AU15_r	A
0	1	0.082010	0.062940	-0.124776	0.080031	-0.022	0.072	0	0	0	...	0.00	0.0	0.00	0.00	
1	1	0.078303	0.060465	-0.128854	0.077414	-0.025	0.069	0	0	0	...	0.00	0.0	0.00	0.00	
2	1	0.080084	0.059067	-0.130403	0.076090	-0.025	0.068	0	0	0	...	0.00	0.0	0.00	0.00	
3	1	0.081053	0.058107	-0.129042	0.074312	-0.024	0.067	0	0	0	...	0.00	0.0	0.00	0.00	
4	1	0.082496	0.059614	-0.128192	0.076915	-0.023	0.069	0	0	0	...	0.00	0.0	0.00	0.00	
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
67392	400	0.057415	-0.149977	-0.143566	-0.121270	-0.044	-0.137	0	0	0	...	0.49	0.0	0.39	1.31	
67393	400	0.057313	-0.149406	-0.143716	-0.120715	-0.044	-0.136	0	0	0	...	0.47	0.0	0.44	1.29	
67394	400	0.057270	-0.148547	-0.143747	-0.119794	-0.044	-0.135	0	0	0	...	0.45	0.0	0.48	1.30	
67395	400	0.056576	-0.149564	-0.143961	-0.120783	-0.044	-0.136	0	0	0	...	0.45	0.0	0.51	1.34	
67396	400	0.056668	-0.149601	-0.143986	-0.120848	-0.044	-0.136	0	0	0	...	0.47	0.0	0.55	1.38	

676648 rows × 39 columns

Figura 4.13: Porzione della tabella che rappresenta il dataset con le features selezionate.

Come si vede dalla figura 4.13, per ogni video del volto di un utente, identificato dalla colonna "id", si trovano diversi record, ciascuno dei quali è composto dai dati estratti da OpenFace in un determinato frame del video stesso. L'ultima colonna, che qui non è visibile, è quella della QoE che gli utenti inquadrati in ciascun video hanno stimato da 1 a 5 durante la visione del filmato che gli è stato mostrato.

Successivamente, ho estratto dal dataset appena creato 4 ulteriori file pickle, uno contenente i record con le attivazioni di tutte le action units (AUc), non solo di quelle selezionate con l'ANOVA, uno contenente i record con le intensità delle action units selezionate con l'ANOVA (AUr), uno composto dai valori della direzione dello sguardo (GD) e un ultimo che racchiude i valori delle QoE stimate dagli utenti per ogni video. L'utilità di questi file sarà chiarita nella prossima sezione.

```
complete_dataset = pd.read_pickle("complete_dataset")

AUr_features = [' id ', ' AU01_r', ' AU02_r', ' AU04_r', '
↪  AU07_r',' AU09_r', ' AU12_r', ' AU14_r', ' AU15_r', '
↪  AU17_r',' AU20_r', ' AU23_r', ' AU25_r', ' AU26_r']

AUc_features = [' id ',' AU01_c',' AU02_c',' AU04_c','
↪  AU05_c',' AU06_c',' AU07_c',' AU09_c',' AU10_c','
↪  AU12_c',' AU14_c',' AU15_c',' AU17_c',' AU20_c','
↪  AU23_c',' AU25_c',' AU26_c',' AU28_c',' AU45_c']

#estrazione delle intensità delle AU
AUr = pd.read_pickle("complete_dataset")
AUr = AUr.filter(AUr_features)

#estrazione delle attivazioni delle AU
AUc = pd.read_pickle("complete_dataset")
AUc = AUc.filter(AUc_features)

#creazione dei file pickle
AUr.to_pickle("AUr")
AUc.to_pickle("AUc")
```

```
#estrazione delle features con i dati sulle direzioni
↪  degli sguardi
GD = complete_dataset[[" id "," gaze_0_x"," gaze_0_y","
↪  gaze_1_x"," gaze_1_y"," gaze_angle_x"," gaze_angle_y"]]

GD.to_pickle("GD")

#estrazione delle quality of experience per ogni video
_id = 1
QoE = pd.DataFrame()
dic = {}
tr = complete_dataset.set_index(" id ")
while _id<401:
    q = tr["QoE"][_id][:1][_id]
    dic[" id "] = _id
    dic["QoE"] = q
    QoE = QoE.append(dic,ignore_index=True)
    _id = _id+1

QoE = QoE.set_index(" id ")

QoE.to_pickle("QoE")
```

4.2 Calcolo delle metriche sui dati

Per procedere all'allenamento del modello di machine learning utilizzando i dati preparati nella sezione precedente, è stato necessario calcolare delle metriche. Queste, determinate applicando le formule che seguono, sono state utilizzate al fine di ottenere, per ogni video, un record di dati, il quale è composto da campi che contengono i valori delle metriche stesse, che sono calcolate prendendo in considerazione i valori delle features di interesse estratte da ogni frame di ciascun video. Per rendere più chiaro il discorso, nelle righe che seguono riporto le formule da utilizzare, le righe di codice per effettuare il calcolo e i risultati prodotti.

Le tre metriche da calcolare sono:

1. La **frequenza di attivazione** di ogni AUc considerata in seguito ad ANOVA in rapporto all'attivazione di tutte le AU.

$$F_{AU_c^j} = \frac{\sum_{n=1}^{N} a_n^j}{\sum_{n=1}^{N} \sum_{j=1}^{J} a_n^j}$$

Figura 4.14: Frequenza di attivazione della j-esima AUc. Nella formula: a_n^j assume valore 1 se la j-esima AUc si è attivata nel frame n, altrimenti 0.

2. L'**intensità** di ogni AUr considerata in seguito ad ANOVA (contenuta nel file AUr).

$$I_{AU_r^k} = \sum_{n=1}^{N} AU_{r,n}^k$$

Figura 4.15: Intensità di attivazione della k-esima AUr. Anche qui n identifica il frame del video.

3. La **varianza** di direzione dello sguardo, calcolata come la variazione della posizione delle pupille dello spettatore durante tutta la sequenza di frame del video del suo viso.

$$V_{GD^g} = \frac{\sum_{n=1}^{N} (GD_n^g - \overline{GD^g})}{N}$$

Figura 4.16: Varianza nella direzione dello sguardo. Nelle formula: n identifica il frame del video e g la features riguardante lo sguardo tra le 6 selezionate (quelle contenute nel file GD).

Per effettuare il calcolo delle metriche proposte ho utilizzato lo script seguente.

```
import pandas as pd
import numpy as np
```

```
AUc = pd.read_pickle("AUc")

#somma delle attivazioni di tutte le AUc
total = AUc.drop(' id ',axis=1).sum().sum()

#CALCOLO DELLA PRIMA METRICA (FREQUENZA DI ATTIVAZIONE)
dic = {}
dataset = pd.DataFrame()
feat = AUc.drop(' id ',axis=1).columns
_id=1
denominatore = pd.DataFrame()
den = {}
sumTemp = 0
while _id<401:
    for feature in feat:
        temp = AUc[[' id ',feature]]
        temp = temp[temp[' id ']==_id]
        temp = temp.drop(' id ',axis=1)
        s = temp.sum()
        dic[feature] = s[0]
        sumTemp = sumTemp+s[0]
    _id = _id+1
    dataset = dataset.append(dic,ignore_index=True)
    den["Total"] = sumTemp
    denominatore =
    ↪  denominatore.append(den,ignore_index=True)
    sumTemp = 0

needed_features = [' AU01_c',' AU02_c',' AU04_c','
↪  AU05_c',' AU06_c',' AU09_c',' AU10_c',' AU15_c','
↪  AU17_c',' AU23_c',' AU28_c',' AU45_c']
_id = 1
d = {}
F_AUc = pd.DataFrame()
while _id<401:
    i = denominatore['Total'][_id-1]
    #alcuni frame risultano con denominatore 0 e quindi
    ↪  vengono scartati
```

```
    if i==0:
        i
    else:
        for feat in needed_features:
            d[feat]=(dataset[feat][_id-1])/i
        d[" id "]=_id
        F_AUc = F_AUc.append(d,ignore_index=True)
    _id = _id+1

F_AUc = F_AUc.set_index(' id ')

#produzione pickle prima metrica
F_AUc.to_pickle("F_AUc")

#CALCOLO DELLA SECONDA METRICA (INTENSITÀ DELLE AU)
AUr = pd.read_pickle("AUr")

dic = {}
I_AUr = pd.DataFrame()
feat = AUr.drop(' id ',axis=1).columns
_id=1
while _id<401:
    for feature in feat:
        temp = AUr[[' id ',feature]]
        temp = temp[temp[' id ']==_id]
        temp = temp.drop(' id ',axis=1)
        s = temp.sum()
        dic[feature] = s[0]
    dic[" id "] = _id
    _id = _id+1
    I_AUr = I_AUr.append(dic,ignore_index=True)

I_AUr = I_AUr.set_index(' id ')

#produzione pickle seconda metrica
I_AUr.to_pickle("I_AUr")
```

```
#CALCOLO DELLA TERZA METRICA (VARIANZA DIREZIONE DELLO
↪  SGUARDO)
GD = pd.read_pickle("GD")

#calcolo delle medie dei 6 valori per ogni video

dic = {}
means = pd.DataFrame()
_id=1
while _id<401:
    temp = GD[GD[' id ']==_id]
    temp = temp.drop(' id ',axis=1)
    s = temp.sum().sum()
    n = temp.shape[0]*temp.shape[1]
    media = s/n
    dic["mean"] = media
    _id = _id+1
    means = means.append(dic,ignore_index=True)

#calcolo finale del Vgd

dic = {}
Vgd = pd.DataFrame()
_id = 1
while _id<401:
    temp = GD[GD[' id ']==_id]
    temp = temp.drop(' id ',axis=1)
    temp = temp.subtract(means["mean"][_id-1])
    s = temp.sum().sum()
    variance = s/temp.shape[0]
    dic[" id "] = _id
    dic["Var"] = variance
    _id = _id+1
    Vgd = Vgd.append(dic,ignore_index=True)

Vgd = Vgd.set_index(" id ")

#produzione pickle terza metrica
```

```
Vgd.to_pickle("Vgd")
```

4.3 Creazione dataset di allenamento e Data Augmentation

Avendo calcolato tutte le metriche che servono, ho concluso la preparazione del dataset di training concatenando i file pickle prodotti nella sezione precedente questa. Il risultato è un set di dati composto da un record per ogni video, ciascuno dei quali contiene i campi con i valori delle metriche calcolate e la QoE all'ultima colonna. In totale quindi abbiamo 379 righe corrispondenti a 379 video di volti di utenti e 27 colonne di features, 12 F_AUc, 13 I_AUr, 1 Vgd e una colonna di QoE.

```
Vgd = pd.read_pickle("Vgd")
F_AUc = pd.read_pickle("F_AUc")
I_AUr = pd.read_pickle("I_AUr")
QoE = pd.read_pickle("QoE")

#video da rimuovere perché durante il calcolo della
↪  metrica F_AUc sono risultati avere denominatore pari a
↪  0
video_da_rimuovere = [58, 59, 60, 61, 62, 63, 64, 65, 66,
↪  67, 68, 69, 70, 71, 72, 73, 74, 75, 76, 193, 196]

#rimozione video
for video in video_da_rimuovere:
    Vgd = Vgd.drop(video)
    I_AUr = I_AUr.drop(video)
    QoE = QoE.drop(video)

F_AUc.columns = F_AUc.columns.str.lstrip()
I_AUr.columns = I_AUr.columns.str.lstrip()

training_dataset = pd.concat([Vgd,F_AUc,I_AUr,QoE],axis=1)

#produzione dataset per allenamento
training_dataset.to_pickle("training_dataset")
```

Raggiunto questo punto si è ottenuto il dataset pronto per l'allenamento dell'estimatore di QoE basato su modello di machine learning. Tuttavia, questo dataset presenta un problema di *squilibrio di classi*, ossia il numero dei record per ogni valore di Quality of Experience (da 1 a 5) è molto sbilanciato. Facendo un rapido calcolo si osserva quanto segue: 3.0: 108, 5.0: 94, 4.0: 93, 2.0: 52, 1.0: 32. Come si vede chiaramente, il numero di record di features disponibili per il valore di QoE 3 è sostanzialmente più alto di quello del valore 2 e 1 soprattutto. Per risolvere questo inconveniente, ho utilizzato l'algoritmo "adaptive synthetic" (ADASYN)[2], un metodo di sovra-campionamento per la creazione di campioni sintetici dalla classe di minoranza mediante interpolazione lineare, allo scopo di riequilibrare le varie classi. Dopo una doppia chiamata ad ADASYN, il conteggio delle features per ogni valore di QoE diventa: 3.0: 108, 2.0: 104, 1.0: 102, 5.0: 94, 4.0: 93.

Segue il breve codice da eseguire per ottenere questo sovra-campionamento.

```
import pandas as pd
import numpy as np
from collections import Counter
from imblearn.over_sampling import ADASYN

data = pd.read_pickle("training_dataset")

#from pandas dataframe to numpy arrays
X = data.iloc[:,:-1].values      #tutte le features
y = data.iloc[:,-1].values       #i valori delle QoE

adasyn = ADASYN(sampling_strategy='minority',
↪  random_state=420, n_neighbors=5)
X_res, y_res = adasyn.fit_resample(X,y)
X_res, y_res = adasyn.fit_resample(X_res,y_res)

X = pd.DataFrame(X_res,columns=cols.drop('QoE'))
y = pd.DataFrame(y_res,columns=['QoE'])

oversampled_dataset = pd.concat([X,y],axis=1)
oversampled_dataset =
↪  oversampled_dataset.set_index(np.arange(1,
↪  oversampled_dataset.shape[0]+1))
```

```
#produzione dataset bilanciato
oversampled_dataset.to_pickle("oversampled_dataset")
```

4.4 Stimatore di QoE dalla live camera

Procedo ora ad illustrare il programma che ho realizzato per effettuare le predizioni di Quality of Experience sfruttando il modello di machine learning allenato con il dataset descritto nelle sezioni precedenti ed estraendo dati dal volto dell'utente direttamente dalla webcam del computer. Descriverò il codice python che costituisce lo script suddividendolo in sezioni, in modo da favorire la chiarezza e rendere evidente l'organizzazione delle varie fasi d'esecuzione.

Librerie necessarie

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.svm import SVC
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import
↪  classification_report,confusion_matrix,accuracy_score
from sklearn import tree
import time
import pyautogui
from datetime import datetime
```

Allenamento e scelta del modello di ML

All'inizio dell'esecuzione, il programma allena il modello di machine learning utilizzando "l'oversampled_dataset" creato nella sezione precedente. Il modello che utilizza è il classificatore Random Forest, poiché è quello che ha conseguito la maggior accuratezza in fase di test durante la validazione iniziale e confronto con gli altri modelli che ho testato.

In particolare, ho confrontato le prestazioni del classificatore KNN (K-nearest neighbors), SVM (Support-vector machine) e, appunto, del Random Forest. Per prima cosa, si procede con la suddivisione del dataset in due parti:

una destinata all'addestramento del modello e una al testing (per valutare la capacità di predizione sulle features target).

```
dataset = pd.read_pickle("oversampled_dataset")

x_train, x_test, y_train, y_test =
↪  train_test_split(dataset,dataset['QoE'],test_size=0.35)
```

Dopo l'esecuzione della riga di codice qui sopra, x_train è un array composto dai record delle features da utilizzare per addestrare il modello, correlati con i valori di QoE (feature dipendente) contenuti in y_train. x_test e y_test contengono invece rispettivamente i record con le features per il test delle prestazioni del modello dopo l'allenamento (pari al 35% del dataset, in questo caso) e i valori delle QoE da utilizzare per il confronto con quelli predetti durante il test.

Utilizzando queste porzioni di dataset si può procedere con l'addestramento dei modelli e con la valutazione delle prestazioni.

1. **KNN**

   ```
   knn = KNeighborsClassifier(n_neighbors=3)
   knn = knn.fit(x_train,y_train)      #il modello viene
   ↪  istruito
   prediction = knn.predict(x_test)    #predizione dei
   ↪  valori di QoE
   ```

   Per valutare come è andato il KNN nel predire la Quality of Experience, utilizzo la libreria sklearn.metrics come segue.

   ```
   print("Accuracy:",accuracy_score(y_test, prediction))
   print(classification_report(y_test,prediction))
   print(confusion_matrix(y_test,prediction))
   ```

   I comandi precedenti confrontano l'array y_test, contenente le QoE obiettivo, con l'array prediction, che è costituito dai valori determinati dal modello. Eseguendo si ottiene questo output:

```
Accuracy: 0.48295454545454547
              precision    recall  f1-score   support

         1.0       0.55      0.97      0.70        31
         2.0       0.51      0.71      0.60        35
         3.0       0.34      0.26      0.29        43
         4.0       0.44      0.26      0.33        31
         5.0       0.50      0.31      0.38        36

    accuracy                           0.48       176
   macro avg       0.47      0.50      0.46       176
weighted avg       0.46      0.48      0.45       176

[[30  0  1  0  0]
 [ 3 25  3  4  0]
 [ 9 12 11  4  7]
 [ 8  5  6  8  4]
 [ 5  7 11  2 11]]
```

Figura 4.17: Prestazioni del KNN durante la validazione.

Come si vede dalla figura 4.17, il modello KNN ha conseguito **un'accuratezza** di predizione del 48%. La tabella stampata sotto il valore dell'accuracy è il **classification report**, il quale riporta delle metriche di valutazione importanti:

- *precision*, che esprime quanti valori di ciascuna categoria, di tutti quelli che sono stati predetti, sono stati effettivamente indovinati. Per esempio è evidente che di tutte le QoE stabilite essere di valore 1 dal KNN, il 55% sono state determinate correttamente.
- *recall*, tasso di "true positive", la metrica che informa su quanti valori, di ciascuna categoria su cui è eseguita la classificazione, sono stati correttamente individuati di quelli presenti nel dataset di allenamento. Ad esempio in questo caso abbiamo che di tutte le QoE a 1 presenti nel dataset, ne sono state individuate correttamente il 97%.
- *FBeta-Score*, che corrisponde al calcolo della formula $(1+Beta^2)*(Precision*Recall)/Beta^2*(Precision+Recall)$. Si sceglie il valore di Beta in base alle necessità; in questo caso Beta=1 va bene per considerare allo stesso modo falsi positivi e falsi negativi; nel caso in cui si dovesse avere un problema in cui i falsi positivi sono più importanti da tenere in considerazione, allora si riduce il valore

di Beta (tipicamente 0,5 o un altro valore tra 0 e 1), se invece i falsi negativi sono da considerare maggiormente allora si alza il valore di Beta (per esempio mettendolo a 2).

Chiude le statistiche di valutazione delle prestazioni la "**confusion matrix**", la matrice di confusione. Questa matrice ha come indici di colonne i valori reali di QoE obiettivo di studio (actual values) e come indici di riga i valori predetti. La diagonale presenta il numero dei valori che sono stati correttamente indovinati (TP - True Positive) mentre sopra la diagonale sono presenti i valori falsi positivi (FP) (cioè quelli che sono stati dichiarati di quel valore, identificato dalla colonna, ma che in realtà non lo erano) e, sotto, i falsi negativi (FN) (cioè quelli che sono stati dichiarati di un valore diverso ma che in realtà erano del valore della colonna in considerazione).

2. **SVM**

```
svm = SVC(kernel="rbf")
svm = svm.fit(x_train,y_train)
prediction = svm.predict(x_test)

print("Accuracy:",accuracy_score(y_test, prediction))
print(classification_report(y_test,prediction))
print(confusion_matrix(y_test,prediction))
```

```
Accuracy: 0.42613636363636365
              precision    recall  f1-score   support

         1.0       0.46      0.93      0.62        40
         2.0       0.61      0.34      0.44        32
         3.0       0.53      0.24      0.33        42
         4.0       0.33      0.05      0.09        38
         5.0       0.28      0.62      0.39        24

    accuracy                           0.43       176
   macro avg       0.44      0.44      0.37       176
weighted avg       0.45      0.43      0.37       176

[[37  0  1  0  2]
 [16 11  0  1  4]
 [10  3 10  2 17]
 [12  3  6  2 15]
 [ 5  1  2  1 15]]
```

Figura 4.18: Prestazioni del SVM durante la validazione.

3. **Random Forest**

```
RFC = RandomForestClassifier()
RFC = RFC.fit(x_train,y_train)
prediction = RFC.predict(x_test)

print("Accuracy:",accuracy_score(y_test, prediction))
print(classification_report(y_test,prediction))
print(confusion_matrix(y_test,prediction))
```

```
Accuracy: 0.9545454545454546
              precision    recall  f1-score   support

         1.0       1.00      1.00      1.00        35
         2.0       1.00      0.97      0.99        36
         3.0       0.88      0.95      0.91        38
         4.0       0.91      0.89      0.90        36
         5.0       1.00      0.97      0.98        31

    accuracy                           0.95       176
   macro avg       0.96      0.96      0.96       176
weighted avg       0.96      0.95      0.95       176

[[35  0  0  0  0]
 [ 0 35  1  0  0]
 [ 0  0 36  2  0]
 [ 0  0  4 32  0]
 [ 0  0  0  1 30]]
```

Figura 4.19: Prestazioni del Random Forest Classifier durante la validazione.

È piuttosto evidente che, come succede spesso, è conveniente selezionare il classificatore Random Forest, poiché si comporta in modo nettamente migliore nello stimare i valori delle Quality of Experience rispetto agli altri modelli testati.

Stima della Quality of Experience dai dati estratti con la telecamera

Il corpo principale del programma è costituito da un ciclo while all'interno del quale viene letto il file csv prodotto da OpenFace (descritto nella sezione 3.1), vengono calcolate le metriche sui dati di interesse (come fatto nella sezione 4.2) e poi viene dato in output il valore della QoE stimato.

Ricordo che i dati che OpenFace estrae dal volto dell'utente che viene inquadrato, interessanti nel processo di stima della QoE, sono i seguenti:

```
needed_features = ['frame', 'gaze_0_x', 'gaze_0_y',
↪  'gaze_1_x', 'gaze_1_y', 'gaze_angle_x', 'gaze_angle_y',
↪  'AU01_c', 'AU02_c', 'AU04_c', 'AU05_c', 'AU06_c',
↪  'AU07_c','AU09_c', 'AU10_c', 'AU12_c', 'AU14_c',
↪  'AU15_c', 'AU17_c','AU20_c', 'AU23_c', 'AU25_c',
↪  'AU26_c', 'AU28_c', 'AU45_c','AU01_r', 'AU02_r',
↪  'AU04_r', 'AU07_r','AU09_r', 'AU12_r', 'AU14_r',
↪  'AU15_r', 'AU17_r','AU20_r', 'AU23_r', 'AU25_r',
↪  'AU26_r']

total_AUcs = ['AU01_c', 'AU02_c', 'AU04_c', 'AU05_c',
↪  'AU06_c', 'AU07_c','AU09_c', 'AU10_c', 'AU12_c',
↪  'AU14_c', 'AU15_c', 'AU17_c','AU20_c', 'AU23_c',
↪  'AU25_c', 'AU26_c', 'AU28_c', 'AU45_c']

needed_gaze_feat = ['gaze_0_x', 'gaze_0_y', 'gaze_1_x',
↪  'gaze_1_y', 'gaze_angle_x', 'gaze_angle_y']

needed_AUr_feat = ['AU01_r', 'AU02_r', 'AU04_r',
↪  'AU07_r','AU09_r', 'AU12_r', 'AU14_r', 'AU15_r',
↪  'AU17_r','AU20_r', 'AU23_r', 'AU25_r', 'AU26_r']

needed_AUc_feat = ['AU01_c', 'AU02_c', 'AU04_c', 'AU05_c',
↪  'AU06_c', 'AU09_c', 'AU10_c', 'AU15_c', 'AU17_c',
↪  'AU23_c', 'AU28_c', 'AU45_c']
```

Prima del while è importante inizializzare le due variabili:

```
processed_frames = 0
computed_values_sumup = pd.DataFrame()
```

le quali servono rispettivamente per tenere traccia della porzione di file csv, con i dati di OpenFace, già processata e dei valori di QoE stimati durante l'esecuzione del programma.

A questo punto comincia l'esecuzione del ciclo while(True). Come prima operazione, si **legge il file csv** considerando le features di interesse per l'esperimento. Tale file viene continuamente scritto da OpenFace, che vi inserisce i dati estratti dal volto dell'utente come abbiamo visto, e periodicamente letto dallo script. La lettura da parte dello script avviene sempre fino alla

fine del file e a partire dalla riga individuata dalla variabile *processed_frame*, che viene aggiornata ad ogni ciclo per ricordare i frame che sono stati già processati. Il risultato della lettura produce i dati necessari alla predizione della QoE e determina anche le condizioni di terminazione del programma. La lettura può infatti avere tre esiti che fanno scaturire degli errori.

- Se il file da leggere non esiste, il programma segnala che l'estrazione delle features dalla live camera non è iniziata (l'utente non ha ancora avviato il software OpenFace) e si mette in attesa che questa cominci. Raggiunto questo punto, se non viene avviata nessuna estrazione, il programma deve essere interrotto manualmente.
- Se dalla lettura del file non sono stati prodotti dati (il file è vuoto), allora vuol dire che l'utente ha fermato l'estrazione delle features da parte di OpenFace. Il programma segnala pertanto che il processo di stima della QoE è finito e produce un file con il riassunto delle predizioni, prima di terminare.
- Se il file letto contiene dati che non sono riconosciuti (deve essere stato modificato manualmente), il programma termina immediatamente.

Quanto riassunto nelle righe sopra viene effettuato dal codice:

```
try:
    #lettura delle feature estratte dal volto dell'utente
    ↪  inquadrato della telecamera
    features = pd.read_csv("<path del file csv di
    ↪  OpenFace>", usecols=needed_features, skiprows=[i
    ↪  for i in range(1,processed_frames+1)])
    features = features.dropna()
    if features.empty==True:
        raise pd.errors.EmptyDataError()
    gaze = features[needed_gaze_feat]
    AUc = features[needed_AUc_feat]
    AUr = features[needed_AUr_feat]

except FileNotFoundError:
    print("Live feature extraction not started yet!
    ↪  Features file not found...")
    time.sleep(5)
```

```
except pd.errors.EmptyDataError:
    print('\n')
    print("Live feature extraction ended! Features file is
    ↪  empty...")
    print('\n')
    computed_values_sumup.to_csv("computed_QoE__" +
    ↪  str(datetime.now()))
    pyautogui.alert("Live feature extraction ended. Feature
    ↪  file is empty.\n\nProducing results
    ↪  file...\n\nTerminating QoE estimation
    ↪  program...",title="QoE estimation
    ↪  ended",timeout=6000)
    break
except ValueError:
    print("Error! Value not recognised in features file.")
    break
```

Se nessuna delle situazioni precedenti si verifica (la lettura della porzione del file csv è andata a buon fine) allora il programma **calcola le metriche** in modo del tutto analogo a quanto abbiamo visto nella sezione 4.2 e produce un record di features da dare in input al modello di machine learning per la stima della QoE.

```
else:
    #CALCOLO DELLE METRICHE DA UTILIZZARE PER LA STIMA
    ↪  DELLA QoE

    #calcolo della metrica Vgd
    gaze_directions_sum = gaze.sum().sum()
    gaze_directions_number = gaze.shape[0]*gaze.shape[1]
    gaze_directions_mean =
    ↪  gaze_directions_sum/gaze_directions_number
    gaze = gaze.subtract(gaze_directions_mean)
    final_sum = gaze.sum().sum()
    FEATURE_gaze_directions_variance =
    ↪  final_sum/gaze.shape[0]

    #calcolo della metrica F_AUc
    sum_dic = {}
```

```
total_auc_sum = 0
for auc in total_AUcs:
    temp = features[auc]
    auc_sum = temp.sum()
    sum_dic[auc] = auc_sum
    total_auc_sum = total_auc_sum + auc_sum
freq_dic = {}
for AUC in needed_AUc_feat:
    freq_dic[AUC] = sum_dic[AUC]/total_auc_sum
FEATURE_AUc_frequency = pd.DataFrame()
FEATURE_AUc_frequency =
↪  FEATURE_AUc_frequency.append(freq_dic,
↪  ignore_index=True)

#calcolo della metrica I_AUr
sum_aur_dic = {}
for aur in needed_AUr_feat:
    temp = features[aur]
    aur_sum = temp.sum()
    sum_aur_dic[aur] = aur_sum
FEATURE_AUr_intensity = pd.DataFrame()
FEATURE_AUr_intensity =
↪  FEATURE_AUr_intensity.append(sum_aur_dic,
↪  ignore_index=True)

#produzione del record da dare in input al modello per
↪  la predizione
MODEL_INPUT_FEATURES = pd.DataFrame()
d = {}
d['Var'] = FEATURE_gaze_directions_variance
var = pd.DataFrame()
var = var.append(d,ignore_index=True)
q = {}
q['QoE'] = -1
qual = pd.DataFrame()
qual = qual.append(q,ignore_index=True)
```

```
MODEL_INPUT_FEATURES = pd.concat([var,
↪  FEATURE_AUc_frequency, FEATURE_AUr_intensity,
↪  qual], axis=1)
```

L'ultima operazione eseguita all'interno del while è proprio la **determinazione del valore della QoE** dell'utente. A tale scopo, il record di features (contente le metriche calcolate) appena costruito viene passato in input al modello pre-allenato di machine learning ed il risultato della stima viene segnalato all'utente per poi essere salvato nella struttura dati di riassunto delle stime effettuate durante l'esecuzione, che al momento della terminazione del programma sarà convertito in file nella directory di lavoro.

```
#determinazione della quality of experience dell'utente
computed = {}
computed_QoE = pd.DataFrame()
#stima della QoE da parte del Random Forest
pred = RFC.predict(MODEL_INPUT_FEATURES)

computed['Video frames'] = str(processed_frames+1)+' :
↪  '+str(processed_frames + features.shape[0])
computed['QoE prediction'] = pred[0]
computed_QoE =
↪  computed_QoE.append(computed,ignore_index=True)

aux = pd.concat([computed_QoE, var, FEATURE_AUc_frequency,
↪  FEATURE_AUr_intensity], axis=1)
computed_values_sumup = computed_values_sumup.append(aux)

processed_frames = processed_frames + features.shape[0]

print('\n')
print('Statistiche: ',computed_QoE)
pyautogui.alert("Your estimated Quality of Experience is
↪  now:\n\n"+str(pred[0]),title="QoE estimation
↪  output",timeout=3500)
```

Il programma è impostato per fornire una stima della QoE ogni 13,5 secondi. Per modificare questo intervallo è sufficiente cambiare il numero di secondi nella sleep all'ultima riga dello script ed eventualmente il timeout di comparsa in millisecondi della finestra informativa pyautogui alla penultima.

4.5 Eseguire il programma

Questa sezione riporta i passi che il lettore deve seguire se desidera riprodurre i risultati raggiunti con la realizzazione del programma che stima la QoE dai dati estratti da OpenFace per mezzo della telecamera.

La prima cosa da fare è installare il software OpenFace sul sistema Ubuntu, seguendo attentamente le istruzioni mostrate nella sezione 4 del capitolo 3. Successivamente, aprire un terminale e far partire il toolkit installato utilizzando gli appositi comandi CLI, la cui lista si può trovare qui: `https://github.com/TadasBaltrusaitis/OpenFace/wiki/Command-line-arguments`. Ad ogni modo, consiglio di utilizzare il comando già pronto che riporto qui di seguito, il quale utilizza il modulo 'FeatureExtraction" per estrarre tutte le features di OpenFace tramite la telecamera del PC, identificata dal nome specificato dopo l'argomento '-device'. '-out_dir' precede il path della directory in cui si vogliono posizionare gli output di OpenFace e '-of' serve per specificare il nome che si desidera assegnare ai file che OpenFace produce. '-verbose' è l'argomento che chiede ad OpenFace di mostrare verbosamente tutti i dettagli di esecuzione e di estrazione delle features.

```
<path_di_installazione>/OpenFace/build/bin/FeatureExtraction
↪  -device /dev/video0 -out_dir <path_target_desiderato> -of
↪  extracted_features.csv -verbose
```

Il nome della camera è solitamente quello già pre-scritto nel comando riportato; è comunque possibile che sia diverso e in questo caso occorre cambiarlo. La lista dei nomi delle videocamere disponibili nel sistema può essere ottenuta eseguendo il comando bash:

```
v4l2-ctl --list-devices
```

Ricordarsi di aggiustare il path all'interno dello script che implementa lo stimatore di QoE, utilizzato all'inizio del ciclo while, di cui abbiamo parlato nella sezione precedente, per localizzare il file csv con gli output di OpenFace.

A questo punto è sufficiente eseguire il programma descritto nella sezione 4.4 (che ricordo deve essere posto in una cartella insieme col dataset di allenamento nel file pickle che abbiamo chiamato "oversampled_dataset") usando banalmente il comando:

```
python3 stimatore_QoE.py
```

Capitolo 5

Costruttore di datasets per determinare QoE e stato emozionale

Con gli obiettivi preposti, delineati brevemente nel secondo capitolo di questo documento, l'ultimo step nella realizzazione del mio progetto di tirocinio è stata l'implementazione di una piattaforma di acquisizione di dati di diversa natura, estratti dal volto tramite OpenFace e da un sensore indossabile della Shimmer Sensing, per la costruzione automatica di dataset utili ad allenare modelli di predizione di Quality of Experience e stato emozionale degli utenti monitorati.

In questo capitolo descriverò le fasi di sviluppo del framework in oggetto, partendo dai prerequisiti necessari al funzionamento di quest'ultimo fino all'illustrazione delle varie sezioni di codice python che lo compongono.

5.1 Prerequisiti al funzionamento del sistema

Ricordo in partenza che il programma che effettua la costruzione automatica di dataset è stato implementato per funzionare sul sistema operativo Ubuntu. Il funzionamento su altri sistemi non è stato testato e non è pertanto garantito, anche se probabilmente può essere adattato senza troppi problemi.

Al fine di raccogliere dati estratti dalle espressioni del volto degli utenti è ovviamente necessario installare il software OpenFace sulla propria macchina. Si raccomanda a questo scopo di seguire attentamente le istruzioni

riportate nella sezione 3.4, in modo da non commettere errori che potrebbero compromettere il corretto funzionamento di OpenFace e quindi anche del tool descritto in queste righe.

Per la raccolta di dati sensoriali è stato utilizzato il sensore indossabile Shimmer3 GSR+ Unit, gentilmente prestatomi in seguito ad un meeting presso il Dipartimento di Informatica dell'Università di Pisa con alcuni collaboratori del progetto europeo Teaching, insieme al codice necessario per effettuare la connessione bluetooth tra il sensore e il PC su cui è in esecuzione il programma che ho sviluppato. Per quanto riguarda questo codice di connessione, non inserirò in questo capitolo tutte le righe che lo compongono per ragioni di spazio e per la relativamente poca importanza ai fini del capitolo stesso, limitandomi a commentare le modifiche che ho apportato per adattarlo al mio caso d'uso e confidando che al lettore di questo scritto sia stato fornito integralmente.

Connessione bluetooth con il sensore

Per far sì che il costruttore di dataset possa ricevere i dati prodotti dal sensore che l'utente monitorato indossa, si deve instaurare una connessione bluetooth.

Si procede pertanto ad attivare il bluetooth sul computer in cui è installato il programma e ad accendere il sensore utilizzando l'apposito interruttore. Successivamente, è necessario eseguire l'accoppiamento dei due dispositivi. A tal fine, aprire le impostazioni bluetooth del PC e collegare lo Shimmer digitando la password di default che solitamente è 1234. Raggiunto questo punto i due dispositivi sono associati. Per stabilire la connessione vera e propria, tramite la quale saranno trasferiti i dati sensoriali estratti, è fondamentale creare una porta d'ascolto /dev/rfcomm0 procedendo come segue:

- aprire una shell e digitare il comando *bluetoothctl*;
- digitare

  ```
  paired-devices
  ```

  per ottenete la lista di dispositivi bluetooth associati al PC;
- individuare il sensore Shimmer appena accoppiato e copiarne l'indirizzo MAC;

- premere la combinazione di tasti Ctrl-D per uscire dal pannello di controllo del bluetooth;
- creare la porta eseguendo il comando

```
sudo rfcomm bind 0 <MAC address appena copiato> 1
```

- se la shell dovesse rispondere: *Can't create device: Address already in use*, modificare il numero che precede l'indirizzo MAC (quello che è a 0 nel comando precedente) ed utilizzare quindi conseguentemente la porta di connessione /dev/rfcomm? sostituendo al punto interrogativo il numero scelto.

5.2 Adattamento del codice di connessione con il sensore

Per poter stabilire la connessione bluetooth tra PC e sensore utilizzando la porta /dev/rfcomm0, ho dovuto modificare la procedura "connect" dello script "gsrplus.py" fornitomi. Nel metodo "self._device.connect" si deve specificare direttamente la porta su cui effettuare la connessione, come mostrato di seguito.

```
def connect(self) -> bool:
    if self._device.connect("/dev/rfcomm0"):
        if not
        ↪  self._device.set_sampling_rate(self.sampling_rate):
            return False
        # After the connection we want to enable GSR and PPG
        if not self._device.set_enabled_sensors(su.SENSOR_GSR,
        ↪  su.SENSOR_INT_EXP_ADC_CH13):
            return False
        # Set the GSR measurement unit to Skin Conductance
        ↪  (micro Siemens)
        if not
        ↪  self._device.set_active_gsr_mu(su.GSR_SKIN_CONDUCTANCE):
            return False

        print(f"Shimmer GSR+ connected.")
```

```
        self._device.print_object_properties()

        return True
    else:
        return False
```

5.3 Implementazione del costruttore di dataset

Nel descrivere il funzionamento dello script "dataset_creator.py" che implementa il costruttore di dataset per la stima di QoE ed ER, procederò simulandone l'esecuzione. Il programma è dotato di una semplicissima interfaccia grafica che consente all'operatore che lo utilizza di interagire con estrema facilità.

Per far funzionare il programma è necessario che il codice sorgente sia contenuto nello stesso folder in cui si trovano la cartella denominata "sensing_core", contenente il codice per l'interfacciamento con il sensore Shimmer, e la cartella "images" con le immagini di sistema.

Per iniziare è sufficiente aprire un terminale, recarsi nella directory del programma (con la struttura appena delineata) e digitare il comando:

```
sudo python3 dataset_creator.py
```

In questo caso è importante utilizzare i diritti del super-utente per avere le autorizzazione necessarie ad effettuare la connessione con la porta /dev/rfcomm0.

Software esterno impiegato

```
from tkinter import *
from tkinter.ttk import *
from tkinter.messagebox import showinfo
import os
import sys
import time
from datetime import datetime
import pandas as pd
import multiprocessing
```

```
import _thread
from threading import Event
from sensing_core.sensing.device.gsrplus import
↪  ShimmerGSRPlus
```

Se non si dispone delle librerie esterne citate qui sopra potrebbe essere necessario installarle sul PC in uso. Per comodità riporto i comandi bash che sono sufficienti ad installare tutti i moduli utilizzati dal programma:

```
sudo apt install python3-pip
sudo apt install python3.8-tk
pip install pandas
pip install serial
pip install pyserial
```

All'avvio del programma

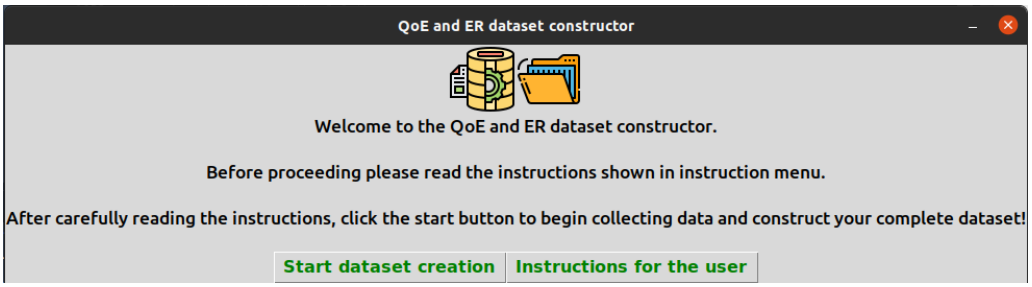

Figura 5.1: Frame di apertura del costruttore di dataset.

Lanciata l'esecuzione del programma, una finestra di benvenuto, illustrata nella figura 5.1, viene mostrata sullo schermo. I due pulsanti consentono di iniziare la costruzione del dataset e di mostrare una lista di pratiche istruzioni da consultare prima di cominciare. Nella scheda delle istruzioni viene riassunto tutto quello che si deve fare prima di dare inizio alla procedura di raccolta dei dati per la costruzione del dataset, inclusa l'installazione di OpenFace, l'aggiustamento dei path del file system utilizzati dallo script, le impostazioni bluetooth per la connessione con il sensore e come indossare lo Shimmer Sensor sulla mano.

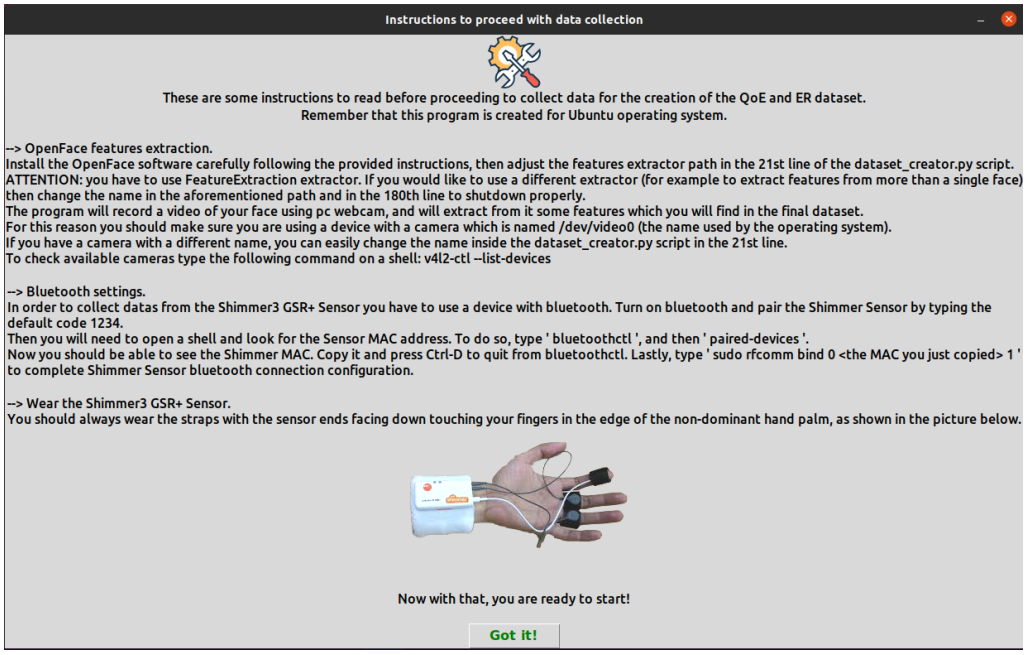

Figura 5.2: Frame che riporta le istruzioni per il corretto utilizzo del programma.

Raccolta dei dati

Alla pressione del pulsante di avvio da parte dell'utente, il programma inizia la raccolta dei dati che poi saranno inseriti nel dataset prodotto come risultato.

Per prima cosa, lo script avvia un processo separato a cui sarà affidata la gestione del software OpenFace. Il processo esegue semplicemente il comando shell di OpenFace per far partire l'estrazione dei dati dal volto dell'utente tramite la telecamera.

```
def OpenFace_processFunction():
    os.system('/OpenFace/build/bin/FeatureExtraction -device
    ↪  /dev/video0 -out_dir ./OpenFace_extracted_features -of
    ↪  extracted_features.csv -verbose')
OpenFace_process =
↪  multiprocessing.Process(target=OpenFace_processFunction)
OpenFace_process.start()
```

Successivamente viene fatto partire un thread che invece si occuperà di raccogliere i dati prodotti dal sensore Shimmer indossato dall'utente. Il thread deve innanzitutto stabilire la connessione con il sensore, chiamando il metodo "connect" di cui abbiamo discusso poche righe fa, per poi iniziare a ricevere dati e a memorizzarli in una struttura dati, infine convertita in un file csv su file system.

```
def ShimmerSensor_processFunction():
        dt = pd.DataFrame()
        device = ShimmerGSRPlus()
        device.connect()
        for n, reads in device.stream():
                if stopShimmerSensor.is_set():
                        break
                if n > 0:
                        dt = dt.append(reads,ignore_index=True)
        print('Shimmer3GSR+Unit data extraction terminated.
        ↪  Producing csv output file...')
        dt.to_csv("Shimmer3GSR+Unit_extractedDatas")

_thread.start_new_thread(ShimmerSensor_processFunction, ())
```

In questa fase, all'utente è mostrata la finestra della figura 5.3, tramite la quale egli può esprimere la volontà di interrompere la raccolta dei dati per procedere con la costruzione del dataset completo.

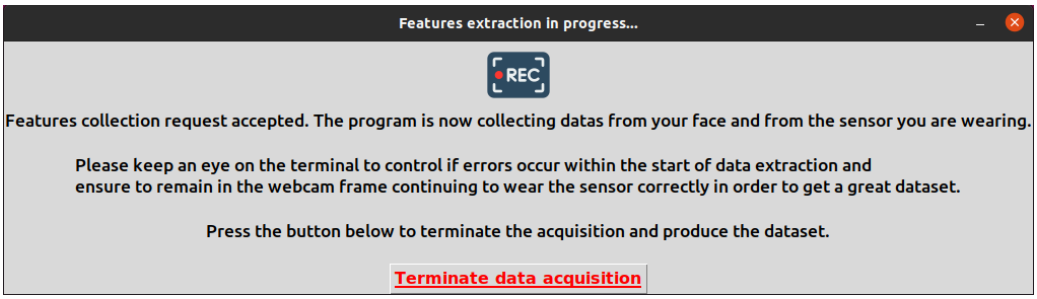

Figura 5.3: Frame mostrato a video durante la raccolta dei dati.

Costruzione del dataset

Dopo aver terminato il processo in cui è in esecuzione OpenFace e il thread che sta raccogliendo dati dal sensore, il programma avvia finalmente la costruzione del dataset da produrre in output.

Per iniziare, prepara il file csv contenente le features estratte da OpenFace (nominato diverse volte in questa relazione) aggiornando i timestamp di ogni record. Questa operazione è necessaria per sincronizzare i dati del volto con quelli sensoriali. Il timestamp inserito nel file da parte di OpenFace, infatti, è il timer dei frame nel video che viene catturato dalla videocamera e processato per estrarre i dati, mentre quello riportato dal thread di lettura dei dati dal sensore è la epoch al momento della lettura. Per sincronizzare questi due input, il programma somma al timestamp di ogni record prodotto da OpenFace la epoch memorizzata in una variabile all'inizio del collezionamento dei dati.

```
features =
↪  pd.read_csv("./OpenFace_extracted_features/extracted_features.csv")
features = features.dropna()

#updating OpenFace timestamps
for i in range(features.shape[0]):
        features.at[i,'timestamp'] = features['timestamp'][i] +
        ↪  globalTimeStamp

features = features.set_index('frame')
features.to_csv('extracted_features_time.csv')
```

A questo punto viene avviato un ultimo thread che genererà il dataset (eseguendo la funzione "dataset_construction_function()"), il quale potrà essere interrotto in ogni momento dall'utente, cliccando il pulsante nella schermata mostrata in figura 5.4, ottenendo un dataset parziale. Il thread costruttore inizia la sua esecuzione leggendo e ripulendo da stringhe inutili il file con i dati del sensore. Dopodiché memorizza il primo timestamp contenuto in tale file (quello che corrisponde alla prima lettura dei dati del sensore) e scarta tutti i record di features di OpenFace generati prima.

```
features = pd.read_csv("extracted_features_time.csv")
features['timestamp'] = features['timestamp'].astype(str)
```

```
shimmer_datas = pd.read_csv("Shimmer3GSR+Unit_extractedDatas")

#shimmer features file cleanup
shimmer_datas['timestamp'] =
↪  shimmer_datas.timestamp.str.replace('[' , '',regex=True)
shimmer_datas['timestamp'] =
↪  shimmer_datas.timestamp.str.replace(']' , '',regex=True)

shimmer_datas['PPG'] = shimmer_datas.PPG.str.replace('[' ,
↪  '',regex=True)
shimmer_datas['PPG'] = shimmer_datas.PPG.str.replace(']' ,
↪  '',regex=True)

shimmer_datas['EDA'] = shimmer_datas.EDA.str.replace('[' ,
↪  '',regex=True)
shimmer_datas['EDA'] = shimmer_datas.EDA.str.replace(']' ,
↪  '',regex=True)

first_time_shimmer = shimmer_datas['timestamp'][0]

#openface datas preparation for dataset creation
epured_open =
↪  features[features['timestamp']>first_time_shimmer]
epured_open = epured_open.reset_index()
epured_open = epured_open.drop("index",axis=1)
```

Soltanto ora si può procedere con la sincronizzazione dei due input. Il thread deve tenere conto del fatto che l'estrazione dei dati da parte di OpenFace dalla live camera avviene a circa 30 frame per secondo, mentre dal sensore si ricevono circa 64 letture al secondo; questo significa che alla fine della raccolta dei dati ci troveremo ad avere circa il doppio dei record nel file dello Shimmer Sensor rispetto al csv di OpenFace.

Per sincronizzare al meglio entrambe le sorgenti e provvedere a costruire un dataset composto da una serie di record contenenti le features del volto e i dati sensoriali estratti più o meno allo stesso momento, il thread effettua questa procedura. Per ogni record di features di OpenFace ricerca nel file di dati prodotti dal sensore quello con il timestamp più vicino e glielo associa. Il lettore accorto che riproduce i risultati descritti in questo elaborato noterà che

potrebbe accadere che in un record dello Shimmer siano contenuti i risultati di più di una lettura. In tal caso il thread considera tutti i timestamp della tupla, andando a scegliere quello più vicino a quello contenuto nel record di OpenFace. Si precisa che l'accoppiamento dei dati estratti da OpenFace e dallo Shimmer Sensor avviene ad una precisione nell'ordine dei millisecondi, che considerando le reazioni del corpo umano è più che accettabile per dire che i dati sono, alla fine, correttamente sincronizzati.

```
aborted = False
shimmer_index = -1
open_face_index = -1
selected = -1
output_dataset = pd.DataFrame()

for open_face_timestamp in epured_open['timestamp']:
        if stopConstruction.is_set():
                aborted = True
                break
        open_face_index = open_face_index + 1
        diff = 1000000
        diff_prec = 1000000000
        element_of_tuple = -1
        element_pos = -1
        tupla = False
        while diff<diff_prec:
                selected = shimmer_index
                if element_of_tuple != -1:
                        tupla = True
                        element_of_tuple = -1
                else:
                    tupla = False
                shimmer_index = shimmer_index + 1
                diff_prec = diff
                first_time_shimmer =
                ↪  shimmer_datas['timestamp'][shimmer_index]
                try:
                    first_time_shimmer =
                    ↪  float(first_time_shimmer)
```

```
                diff = abs(float(open_face_timestamp) -
                ↪  first_time_shimmer)
            except ValueError:
                element_diff_prec = 1000000000
                for i in
                ↪  range((first_time_shimmer.count(','))+1):
                        element_of_tuple =
                        ↪  first_time_shimmer.rsplit(',')[i]
                        element_of_tuple =
                        ↪  float(element_of_tuple)
                        element_diff =
                        ↪  abs(float(open_face_timestamp)
                        ↪  - element_of_tuple)
                        if element_diff<element_diff_prec:
                            element_pos = i
                            diff = element_diff
                            element_diff_prec =
                            ↪  element_diff
    if not tupla:
            output_dataset =
            ↪  output_dataset.append(epured_open.loc[open_face_index,
            ↪  :].append(shimmer_datas.loc[selected,
            ↪  ["PPG", "EDA"]]), ignore_index=True)
    else:
            d1 = pd.DataFrame()
            d1 =
            ↪  d1.append(epured_open.loc[open_face_index,:],
            ↪  ignore_index=True)
            dic = {}
            dic['PPG'] =
            ↪  float(shimmer_datas['PPG'][selected].rsplit(',')
            ↪  [element_pos])
            dic['EDA'] =
            ↪  float(shimmer_datas['EDA'][selected].rsplit(',')
            ↪  [element_pos])
            d2 = pd.DataFrame()
            d2 = d2.append(dic,ignore_index=True)
            d1 = pd.concat([d1,d2],axis=1)
```

```
                output_dataset =
                ↪  output_dataset.append(d1,ignore_index=True)

output_dataset.to_csv('constructed_dataset_'+str(datetime.now()))
```

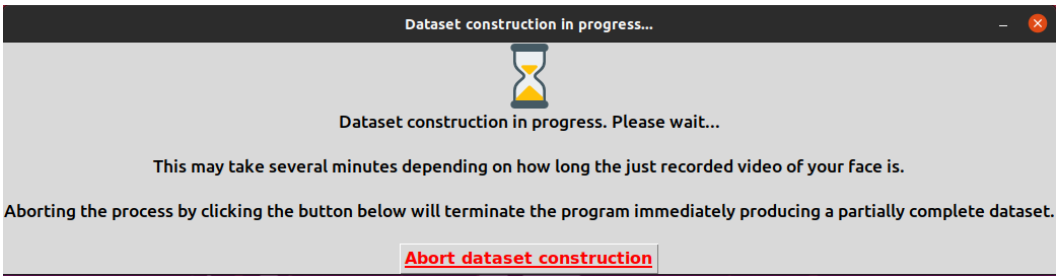

Figura 5.4: Frame mostrato a video durante la costruzione del dataset.

Dataset prodotto in output

Al termine dell'esecuzione l'utente ottiene un file di output che costituisce il dataset completo costruito dal programma, sotto il nome di "constructed_dataset". Questo, è composto da una serie di record ciascuno dei quali contiene tutte le informazioni relative alle features estratte dalle espressioni dell'utente monitorato, utilizzando OpenFace (tutte le features), e due dati sensoriali ricavati dal sensore Shimmer. I dati sensoriali presi in considerazione sono:

- **EDA** (Electrodermal Response, Skin Conductance Activity) anche noto come Galvanic Skin Response, utile a monitorare la conduttività cutanea tramite i due elettrodi collegati alle due dita di una mano. In particolare, l'EDA fornisce informazioni riguardo l'attività eccrina (sudorazione cutanea), il cui cambiamento può aiutare a determinare lo stato emotivo della persona. Il nostro livello di eccitazione emotiva, infatti, cambia in risposta all'ambiente in cui ci troviamo: se qualcosa è spaventoso, minaccioso, gioioso o comunque emotivamente rilevante, allora il successivo cambiamento nella risposta emotiva che sperimentiamo aumenta anche l'attività delle ghiandole sudoripare eccrine.
- **PPG** (Photo Plethysmogram) ossia il fotopletismogramma, eseguito sfruttando la sonda a impulso ottico collegata all'interfaccia jack 3,5mm

che illumina la pelle e misura i cambiamenti nell'assorbimento della luce, allo scopo di rilevare le variazioni del volume del sangue, della frequenza cardiaca e della vasodilatazione/costrizione.

Il dataset è pensato per essere utilizzato nell'addestramento di modelli di intelligenza artificiale di vario tipo; è facilmente adattabile a qualsiasi esigenza essendo memorizzato nel file system sotto forma di semplice file csv (come fatto in questa relazione nel capitolo relativo al test di predizione della QoE), e può essere esteso da altri programmi semplicemente aggiungendo dati alla tabella che lo compone. Esaminando le variazioni di questi dati, opportunamente organizzati, i modelli saranno in grado di determinare ogni valore delle più disparate features target ottenibili con dati estratti dalle espressioni del volto e dal sensore indossabile, primi fra tutti i qui nominati Quality of Experience e riconoscimento delle emozioni.

Per concludere, fornisco un semplice esempio di come le features contenute nel dataset possano essere utilizzate per effettuare Human Emotion Recognition. Nei grafici che seguono sono rappresentati i dati estratti da un frammento di un video di prova del mio volto in cui stavo ridendo. Si nota evidentemente che le due Action Units relative all'espressione della risata, 6 e 12 (vedere tabella in figura 3.5), si sono attivate con alta intensità, mentre, in corrispondenza del picco di queste ultime, il valore del PPG si è abbassato sensibilmente in valore medio identificando una correlazione negativa tra i due processi temporali. Questo fenomeno fornisce uno spunto per approfondire le relazioni tra le features della famiglia delle Action Units e quelle ricavate dallo Shimmer o da altre sorgenti addizionali.

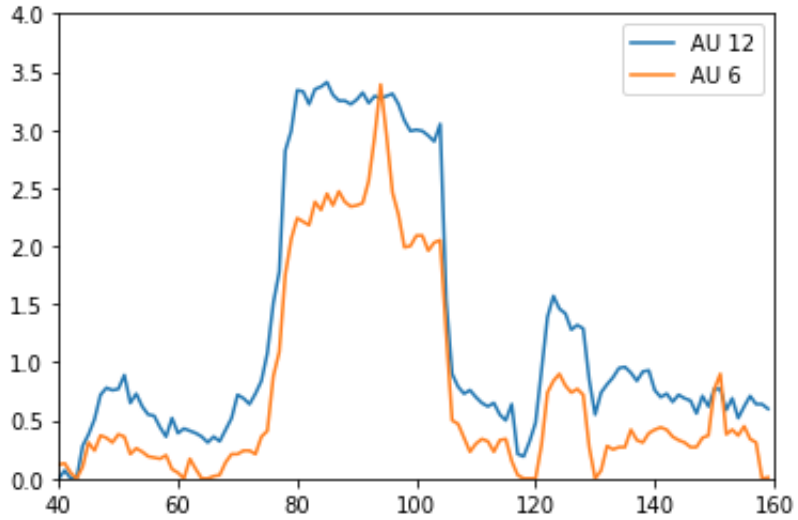

Figura 5.5: Grafico che mostra il picco delle AU 6 e 12 durante la risata.

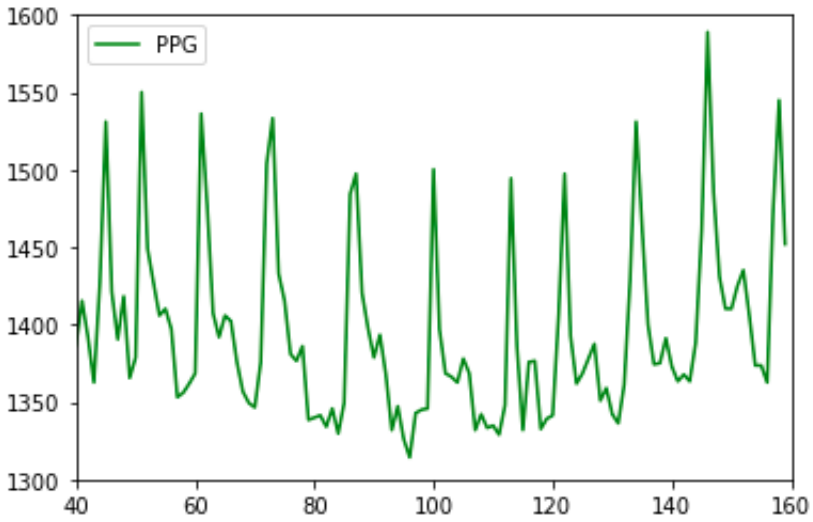

Figura 5.6: Grafico che mostra il calo del PPG durante la risata.

Capitolo 6

Conclusioni e possibili sviluppi futuri

Come si evince dai capitoli precedenti, l'obiettivo del progetto di tirocinio descritto in questo documento è stato implementare un framework per la costruzione di dataset di allenamento per modelli di intelligenza artificiale, allo scopo di addestrarli a poter assolvere a task quali il riconoscimento delle emozioni umane e la determinazione della qualità dell'esperienza di un utente monitorato mentre usufruisce di un servizio. Questo principale obiettivo è stato raggiunto con l'implementazione del programma illustrato nel capitolo 5, il quale raccoglie i dati provenienti da un sensore indossabile e quelli estratti dalle espressioni del volto, per costruire un dataset completo e sincronizzato.

Il set di dati prodotto dal software può essere semplicemente esteso aggiungendo dati di qualsiasi tipologia ed origine; essendo memorizzato in un file col semplice formato csv, consente all'operatore che lo utilizza di poterlo adattare ad ogni esigenza, di poter selezionare soltanto alcune colonne di features (in funzione di un algoritmo di feature selection quale il semplice ANOVA) o solo una determinata sezione di righe, di poter aggiungere o rimuovere informazioni in qualsiasi momento. Infine, consente di poterlo facilmente convertire in altri formati o inserirlo in strutture dati di ogni linguaggio di programmazione.

L'operatore che desidera aggiungere altre sorgenti di dati da utilizzare per comporre il dataset, per esempio altri sensori indossabili, altre videocamere, strumenti professionali per l'estrazione di dati dal volto, telecamere multispettrali (ad esempio infrarossi) o camere 3d, visori professionali per la gaze directions, o qualsiasi altro tipo di strumento, può senz'altro facilmente

modificare lo script presentato nella sezione 5.3 al fine di collezionare le nuove tipologie di dati direttamente built-in nel costruttore. L'utilizzatore più esperto può anche modificare il file del dataset cimentandosi nell'implementazione di un suo script di modifica e sincronizzazione dei dati, replicando i risultati mostrati in questa relazione.

Una delle possibili estensioni future del presente lavoro che posso proporre, sul quale non ho potuto approfondire per ragioni di tempo, è lo sviluppo di un meccanismo di pubblicazione dei dati, provenienti da diverse sorgenti, sfruttando protocolli di rete (vedere ad esempio MQTT [1]). L'idea alla base di questo sviluppo è avere tanti strumenti di produzione di dati utili ad essere inseriti in un dataset, i quali pubblicano i propri risultati in un database remoto sincronizzato utilizzando il timestamp di estrazione dei dati. Ci si avvicina così ad un aspetto molto al centro dell'attenzione in questi ultimi anni che è l'*Artificial Intelligence of Things* (AIoT), cioè la combinazione di tecnologie di intelligenza artificiale con l'infrastruttura dell'Internet delle cose per ottenere operazioni IoT più efficienti, migliorare le interazioni uomo-macchina, migliorare la gestione e l'analisi dei dati e in questo caso permettere uno *smart human state monitoring*.

[1] `https://mqtt.org/`

Bibliografia

[1] Tadas Baltrusaitis, Amir Zadeh, Yao Chong Lim, and Louis-Philippe Morency. Openface 2.0: Facial behavior analysis toolkit. In *2018 13th IEEE international conference on automatic face & gesture recognition (FG 2018)*, pages 59–66. IEEE, 2018.

[2] Haibo He, Yang Bai, Edwardo A Garcia, and Shutao Li. Adasyn: Adaptive synthetic sampling approach for imbalanced learning. In *2008 IEEE international joint conference on neural networks (IEEE world congress on computational intelligence)*, pages 1322–1328. IEEE, 2008.

[3] Simone Porcu, Alessandro Floris, Jan-Niklas Voigt-Antons, Luigi Atzori, and Sebastian Möller. Estimation of the quality of experience during video streaming from facial expression and gaze direction. *IEEE Transactions on Network and Service Management*, 17(4):2702–2716, 2020.