Università degli studi di Torino

SCUOLA DI SCIENZE DELLA NATURA

Corso di Informatica

Tesi di Laurea Triennale

Estensione di dataset per il riconoscimento dell'attività di un guidatore utilizzando ChromaKey, Despill e altre tecniche cinematografiche

Relatore:
Prof.
Elvio Gilberto Amparore

Candidato:
Roberto Craparotta

Anno Accademico 2021/2022

Abstract

L'apprendimento di reti neurali profonde per la classificazione di immagini necessita di dataset molto grandi e con grande variazione tra le immagini. Generare tali dataset può essere molto oneroso, in quanto ottenere molta variabilità è difficile. Una tecnica per mitigare il problema consiste nell'aumentare il dataset (dataset augmentation), cioè generare multiple immagini varie partendo da un insieme più piccolo, attraverso tecniche di ampliamento. Nell'ambito del progetto europeo NextPerception (https://www.nextperception.eu/) la tesi si propone di sviluppare e validare un sistema di dataset augmentation per dataset di Driver Activity Recognition (DAR). Lo scopo è acquisire immagini di guidatori in un sistema simulato, con un green screen in sfondo. È stata quindi creata una pipeline di trasformazione che elimina lo sfondo verde, e lo sostituisce con l'immagine di un veicolo, ottenendo così un dataset più realistico. La pipeline creata è composta da 6 macro passaggi. Il primo passaggio è l'acquisizione del video con lo sfondo verde, a cui segue la calibrazione e la creazione di una palette di colori, ovvero la creazione di una tavolozza di colori estratta dallo sfondo dell'immagine, così da massimizzare, nella chromakey, la rimozione dello sfondo e minimizzare la rimozione del soggetto. In seguito, avviene l'applicazione del Chromakey, un algoritmo che servendosi della Palette sopracitata crea una maschera, la quale applicata all'immagine, rende trasparente la parte occupata dallo sfondo verde. A ciò segue l'applicazione del Despill, ovvero un algoritmo che rimuove i residui di verde dal soggetto. Dopo l'applicazione dei due precedenti algoritmi, si passa alla ricalibrazione dei colori dell'immagine in riferimento al nuovo sfondo, così da rendere l'immagine composta più realistica. L'ultimo passaggio consiste nella ricomposizione del video, ossia la sovrapposizione di ogni frame, su cui viene applicata la maschera con il nuovo sfondo. Confrontando i risultati ottenuti con lo stato dell'arte attuale, si evince il raggiungimento di un buon risultato, sia da un punto di vista numerico, sia, sopratutto, da un punto di vista visivo.

"Dichiaro di essere responsabile del contenuto dell'elaborato che presento al fine del conseguimento del titolo, di non avere plagiato in tutto o in parte il lavoro prodotto da altri e di aver citato le fonti originali in modo congruente alle normative vigenti in materia di plagio e di diritto d'autore. Sono inoltre consapevole che nel caso la mia dichiarazione risultasse mendace, potrei incorrere nelle sanzioni previste dalla legge e la mia ammissione alla prova finale potrebbe essere negata."

Indice

Abstract **i**

1 Introduzione **1**
1.1 Descrizione del problema 1
1.2 Stato dell'arte 1
1.2.1 DaVinci Resolve 1
1.2.2 L'articolo "Interactive High-Quality Green-Screen Keying via Color Unmixing" 5
1.3 Spazi colore utilizzati 6

2 Tecnologie usate **7**
2.1 OpenCV 7
2.2 Numpy 7

3 Metodologie di costruzione **9**
3.1 Calibrazione e Palette 11
3.1.1 Palette 11
3.1.2 Calibrazione 17
3.2 Chromakey 19
3.3 Despill 23
3.4 Ricalibrazione dei colori 25
3.5 Ricostruzione del video 29

4 Risultati **32**
4.1 Benchmark 32
4.1.1 Analisi quantitativa 33
4.1.2 Analisi qualitativa 37
4.2 Score 38
4.2.1 Analisi quantitativa 38
4.2.2 Analisi qualitativa 43
4.3 Tool finale 48
4.3.1 Organizzazione 48
4.3.2 Utilizzo 50

5 Conclusioni **51**

Bibliography **52**

Capitolo 1

Introduzione

1.1 Descrizione del problema

Nell'ambito dell'apprendimento di reti neurali per la classificazioni di immagini è molto importante avere dei dataset molto grandi. Avere dei dataset contenenti una grande quantità di immagini permette di avere molta varietà e di conseguenza un apprendimento migliore. Generare dei dataset così importanti risulta molto costoso; si possono però utilizzare delle tecniche di dataset augumentation. Queste tecniche si basano sul generare immagini multiple, diverse tra di loro, partendo da un dataset piccolo. Nell'ambito del progetto europeo NextPerception l'obiettivo della tesi è quello di sviluppare un sistema di dataset augumentation che utilizza la tecnica del chromakey per sostituire lo sfondo a delle immagini di un guidatore in un simulatore.

1.2 Stato dell'arte

Lo stato dell'arte della tecnica del chromakey è molto ampio. In commercio si possono trovare molti applicativi sia a pagamento sia gratuiti, che permettono di applicare questa tecnica su filmati o singole immagini. Ogni applicativo utilizza un suo algoritmo con cui applica la tecnica. Come metro di paragone per valutare il risultato ottenuto dal tool creato sono stati scelti DaVinci Resolve [1] e l'articolo "Interactive High-Quality Green-Screen Keying via Color Unmixing" [2].

1.2.1 DaVinci Resolve

DaVinci Resolve [1] è un potente software di editing video che può essere utilizzato per la creazione di contenuti video sia da un esperto del settore che da un principiante che si avvicina a questo mondo. Gli utenti con una minima esperienza di montaggio video possono imparare a utilizzare le funzioni di editing di base, ma possono anche sperimentare i processi di editing di livello superiore offerti dal software gratuito. Il programma ha sette schermate di lavoro (Workflow): Media, Cut, Edit, Fusion, Color, Fairlight e Deliver. In ognuna di esse è possibile modificare gli elementi del progetto

corrente che viene organizzato in delle "timeline". Queste linee temporali, possono essere parti del video finale, versioni diverse o anche sottoprodotti che utilizzano lo stesso materiale. DaVinci Resolve è molto utilizzato nella produzione di contenuti del grande e del piccolo schermo; come nei film: Avatar, Bohemian Rhapsody e nei programmi televisivi come il vincitore dell'Emmy Awards il Trono di spade.
DaVinci permette, attraverso alcuni potenti strumenti, di applicare la tecnica del chromakey. Il chromakey è una tecnica usata per creare effetti speciali, si usa per ambientare soggetti e oggetti su sfondi virtuali, aggiunti separatamente e successivamente. Tale tecnica permette di miscelare due (o più) sorgenti video, sfruttando un particolare colore di sfondo (solitamente il verde o il blu). Tale colore viene eliminato ottenendo un'immagine scontornata, componibile con altri sfondi o immagini.
Dopo aver deciso quali filmati utilizzare, averli importati, ed aver configurato la timeline in modo tale che l'area con il green screen venga sostituita con il filmato desiderato, la schermata di lavoro che ci permette di applicare la tecnica del chromakey è la workflow Color.

Esempio di applicazione del Chromakey con Green Screen

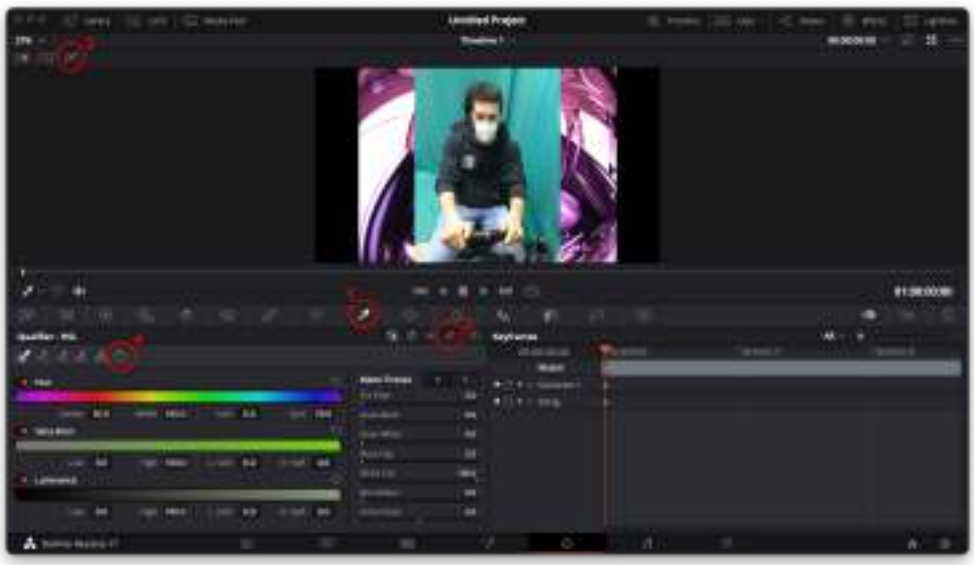

Figura 1.1: Schermata workflow Color

Il primo passaggio che bisogna eseguire è la selezione dell'area di filmato contenente il green screen. Lo strumento che permette di selezionare l'area da rimuovere è il "Qualifier 3D" (1)(2). Questo strumento, tramite il mouse, permette di selezionare il colore che si desidera isolare, rendendo trasparenti tutti gli altri colori. Si seleziona quindi la parte di filmato che si vuole rimuovere (Figura 1.2). Attivando il comando "Highlight"(3) verrà mostrata in grigio la parte che si è resa trasparente (Figura 1.3), che sarà l'opposto di quello desiderato, quindi con il comando "Invert"(4) inverto l'area grigia in modo che solo il green screen sia reso trasparente (Figura 1.4).

Figura 1.2: Qualifier 3D

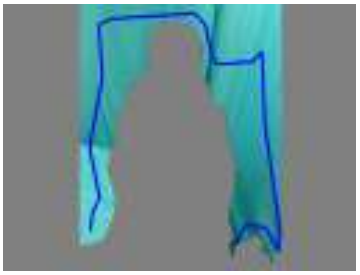

Figura 1.3: Highlight

Figura 1.4: Invert

Dopo aver reso trasparente la porzione di immagine contente il green screen si può procede a miscelare il filmato con il suo nuovo sfondo. Questa operazione è permessa dalla workflow "Nodes" che permette di aggiungere una "alpha output" che, se collegata al nodo del nostro filmato, consente di fondere l'area occupata dal green screen (trasparente) con il contenuto presente nella traccia video sottostante.

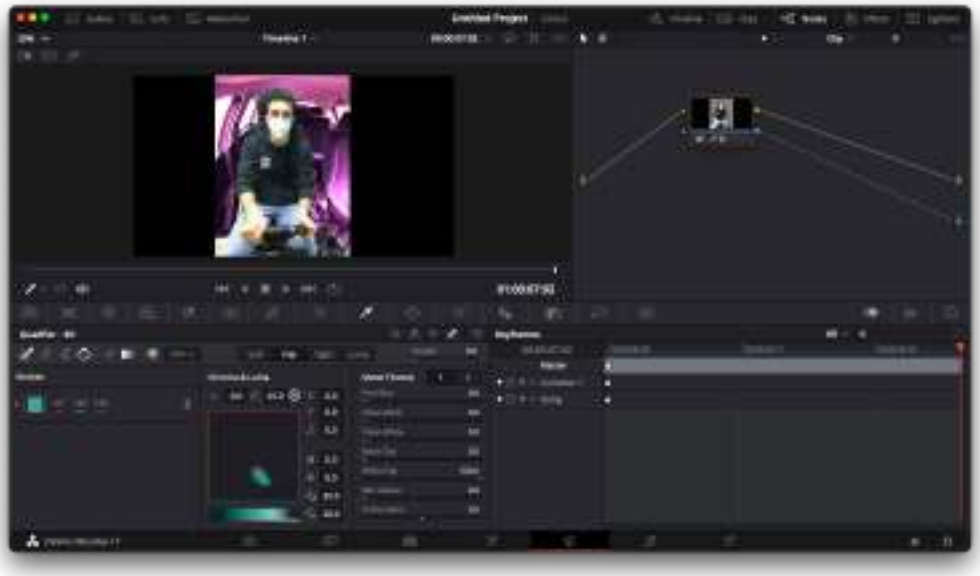

Figura 1.5: Schermata workflow Nodes

In alcuni filmati può essere necessario ritagliare alcuni oggetti indesiderati che sono entrati nell'inquadratura. Questo può essere fatto con le "Power Windows", anche dette maschere. Si tratta di una tecnica utile se si desidera ridimensionare o riposizionare il layer del green screen. In pratica, tutto ciò che si trova all'interno della power Window sarà visibile, al contrario, tutto ciò che nello stesso layer si trova al di fuori viene reso invisibile. Così facendo lo spazio al di fuori della Power Window appare come qualsiasi cosa sia presente nella traccia sottostante.

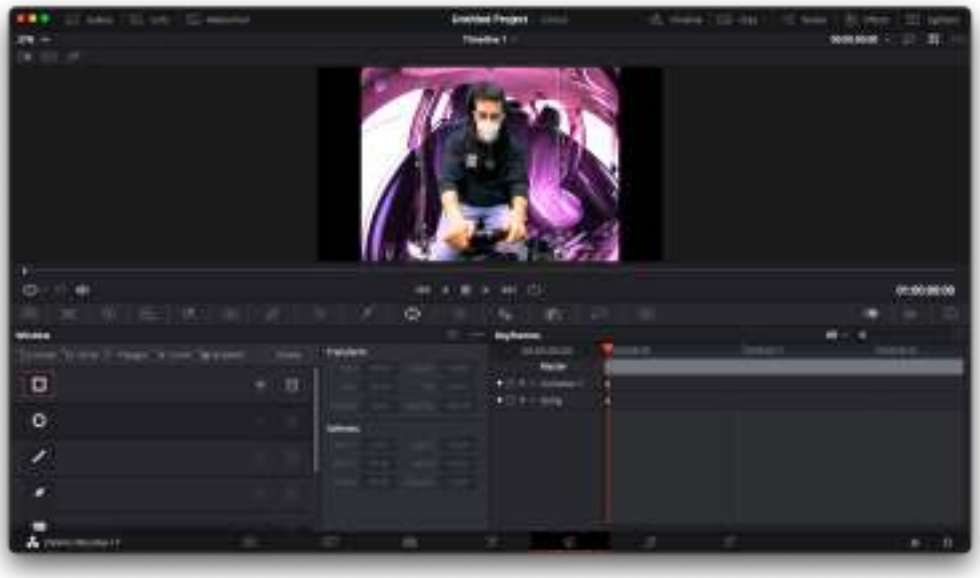

Figura 1.6: Schermata workflow Nodes

Per effettuare delle migliorie nell'eliminazione dello sfondo verde, come ad esempio, la definizione del contorno del soggetto. DaVinci Resolve nella workflow "Qualifier" mette a disposizione alcuni strumenti che permettono di avere una maggior qualità nella rimozione dello sfondo, come:

- Il "Clean Black value" che, dopo aver mostrato l'immagine in bianco e nero, permette di eliminare qualsiasi disturbo del nero (background) nelle aree bianche (foreground).
- Il "Clean White value" che opera in modo opposto rispetto al precedente.
- Il comando "In/Out Ratio" per ridurre o espandere i bordi dell'area trasparente.
- Il "Blur Radius" per rendere più morbidi i bordi dell'area trasparente.

Una scarsa illuminazione del set può comportare degli ostacoli nell'applicazione della tecnica del chromakey.
Principalmente si può notare che:

- A causa di una scarsa illuminazione del green screen si creano delle ombre e delle luci diverse, con una conseguente cattura dell'area occupata dal green screen meno precisa
- A causa di una scarsa illuminazione del soggetto, si possono notare dei riflessi verdi su di esso rendendo meno realistico il risultato finale.

Per risolvere questo tipo di problema DaVinci offre un strumento chiamato "Despill". Questo strumento va ad agire sul immagine correggendo pixel per pixel i valori "RGB", così da rimuovere i residui di verde.

Infine per rendere il filmato ancora più naturale DaVinci Resolve offre una grande mole di strumenti per la manipolazione del colore.

Figura 1.7: Pipeline elaborazione immagine DaVinci Resolve

Figura 1.8: Immagine iniziale

Figura 1.9: Nuovo background

Figura 1.10: Risultato finale

1.2.2 L'articolo "Interactive High-Quality Green-Screen Keying via Color Unmixing"

Come detto in precedenza, lo stato dell'arte della tecnica del chromakey è molto ampio. Come metro di paragone è stato scelto il seguente articolo che attraverso l'uso di questa tecnica riesce ad ottenere dei risultati eccellenti. L'articolo in questione è "Interactive High-Quality Green-Screen Keying via Color Unmixing" [2]. Esso, propone una soluzione all'ottimizzazione della tecnica del chromakey nell'ambito professionale e non, andando a ridurre i tempi di applicazione della tecnica. Nella soluzione proposta viene utilizzato un metodo di unmixing del colore. Dall'immagine viene creato un modello di colori globale, una raccolta di colori presenti nell'immagine. Questo metodo è di tipo parametrico, cioè, i parametri per la creazione del modello vengono ottenuti tramite l'interazione con l'utente. Le informazioni di cui si ha bisogno vengono quindi estratte, dall'utente, in modo intuitivo, a cui viene chiesto di fare dei piccoli scarabocchi sui colori che intente isolare. Il modello di colore globale viene quindi perfezionato in modelli di colore locali. Questi modelli vengono, successivamente, utilizzati nell'estrazione di più livelli colore tramite la separazione del colore. Viene quindi combinato un

sottoinsieme di essi per ottenere il risultato finale. Alcuni vantaggi di questa soluzione sono:

- Non è imposto nessun vincolo sulla creazione delle maschere.
- Non si basa sulla sola soppressione del green screen, ma può essere applicato con qualsiasi colore.
- Il metodo calcola più livelli RGBA come output, il che consente ulteriori altre applicazioni.

I risultati ottenuti dal metodo proposto dicono che: con un frame a 1080p, l'elaborazione richiede circa 10 secondi per la stima del colore locale, un altro secondo circa per la propagazione del modello del colore locale al frame successivo e di circa 3 secondi per la rimozione del colore. Pertanto, l'elaborazione totale di un immagine, è di circa 13 secondi.

1.3 Spazi colore utilizzati

Uno spazio colore è la combinazione di un modello colore, un modello matematico che permette di rappresentare i colori in forma numerica, con la corrispettiva funzione di mappatura. Gli spazi colore che sono stati utilizzati nell'implementazione sono:

- RGB: composto da "R" con cui si indica il valore del rosso, da "G" con cui si indica il valore del verde e "B" con cui si il valore del blu.
- YCbCr: composto da "Y" che è la componente di luma, mentre "Cb" e "Cr" indicano i componenti di crominanza (cioè la differenza dal verde nel campo del blu e del rosso, rispettivamente).
- LAB: composto da "L" che indica il valore della luminosità ed "A" e "B" che sono i componenti cromatici che descrivono rispettivamente il valore del rosso/verde e del blu/giallo.

Esempio del colore nero e del colore bianco e del colore verde nei tre spazi colore:

- RGB: bianco (255,255,255), nero (0,0,0), verde (0,255,42).
- YCbCr: bianco (255,128,128), nero (0,128,128), verde (162,42,37).
- LAB: bianco (255,128,128), nero (0,128,128), verde (255,44,211).

Capitolo 2

Tecnologie usate

2.1 OpenCV

OpenCV [3] (Open Source Computer Vision Library) è una libreria software open source per la computer vision, il machine learning e l'elaborazione delle immagini. La libreria dispone di oltre 2500 algoritmi ottimizzati, che possono essere utilizzati per rilevare e riconoscere volti, identificare oggetti, unire immagini, trovare immagini simili da un dataset d'immagini, ecc. Gli usi implementativi di OpenCV spaziano dall'unione di immagini di Google Street view, al rilevamento di intrusioni nei video di sorveglianza in Israele, al rilevamento d'incidenti di annegamento in piscine in Europa. Il linguaggio di programmazione principalmente utilizzato per lo sviluppo di OpenCV è C++, per sfruttare l'elaborazione multi-core, ma è anche possibile interfacciarsi anche con C , Java e Python. OpenCV può anche essere integrato con varie librerie, come ad esempio NumPy, che ci fornisce strutture dati che vengono utilizzate per l'implementazione di OpenCV su Python. Questo permette di identificare il pattern dell'immagine e le sue varie caratteristiche.

2.2 Numpy

Numpy [4], una libreria open source, è il pacchetto fondamentale per la computazione scientifica in Python. La libreria fornisce un oggetto array multidimensionale e vari oggetti da esso derivati, come array e matrici. Numpy offre un grande assortimento di operazioni su array, tra cui matematica, logica, selezione, I/O e molto altro. Al centro del pacchetto NumPy, c'è `ndarray`. Esso incapsula array n-dimensionali di tipi di dati omogenei ed ha a disposizione molte operazioni, eseguite in codice compilato, che permettono di avere grandi prestazioni. Le principali differenze tra gli array di NumPy e le liste standard di Python sono le seguenti:

- Gli array NumPy hanno una dimensione fissa al momento della creazione, a differenza delle liste di Python che sono dinamiche.
- Gli elementi di un array NumPy devono essere tutti dello stesso tipo di dati e quindi avranno le stesse dimensioni in memoria.
- Gli array NumPy facilitano operazioni matematiche avanzate e di altro tipo su un gran numero di dati, a differenza delle liste di Python sono più efficienti e rendono più leggibile il codice.

La libreria offre grandi prestazioni grazie alla vettorializzazione e alla sua implementazione in codice C ottimizzato e precompilato. La vettorializzazione fornisce molti vantaggi, tra cui:

- Il codice vettorializzato è più conciso e più facile da leggere.
- Meno righe di codice generalmente significa meno bug.
- Il codice assomiglia più da vicino alla notazione matematica standard.

Capitolo 3

Metodologie di costruzione

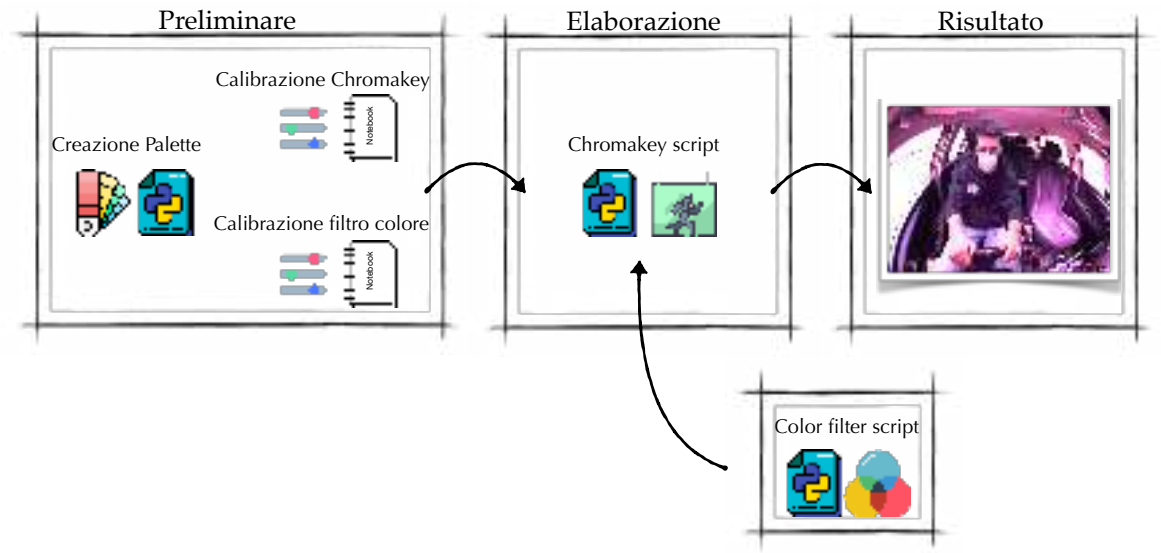

Figura 3.1: Blocchi delle metodologie di costruzione

Come illustrato nella Figura 3.1 la metodologia di costruzione é stata divisa in due macro blocchi: il blocco preliminare e il blocco di elaborazione.
Il blocco preliminare comprende la creazione della palette, la calibrazione del chromakey e la calibrazione del filtro colore.
Il blocco di elaborazione invece comprende l'applicazione del chromakey, l'applicazione del despill, l'applicazione del filtro colore e infine la ricostruzione del video finale.

Più nel dettaglio:

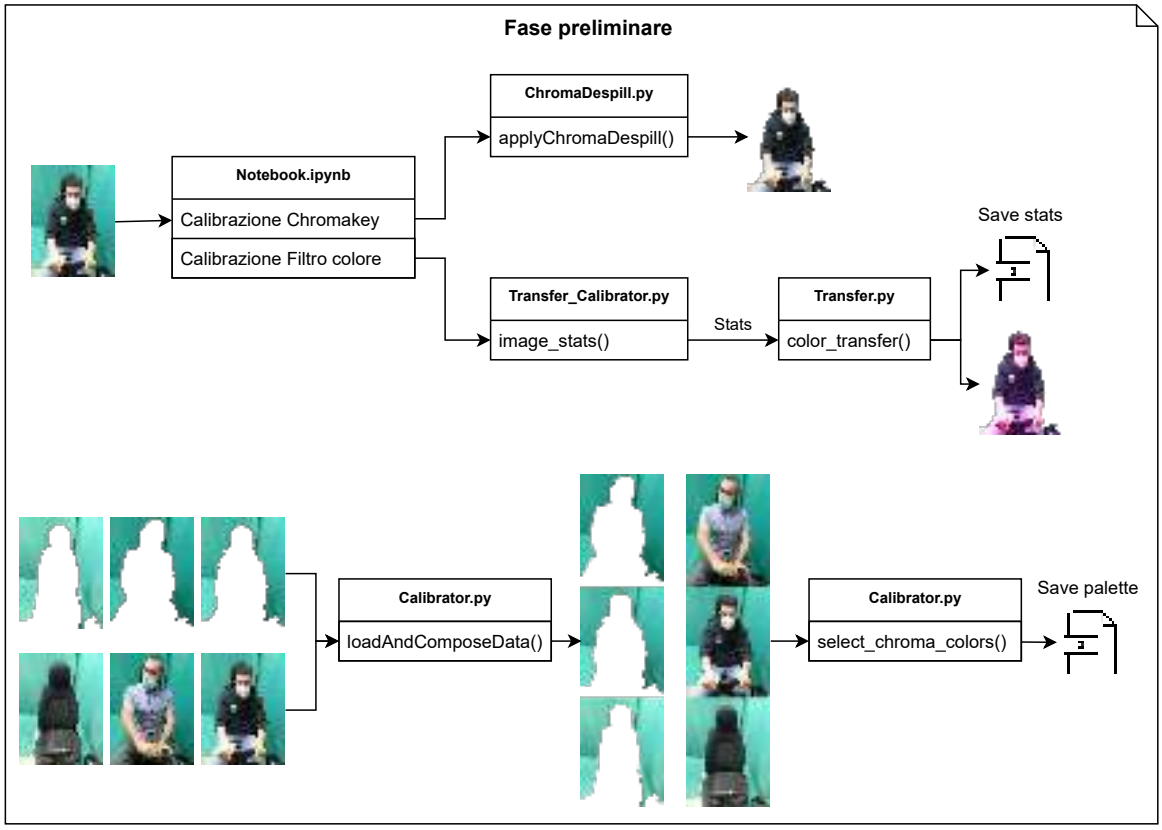

Figura 3.2: Fase preliminare

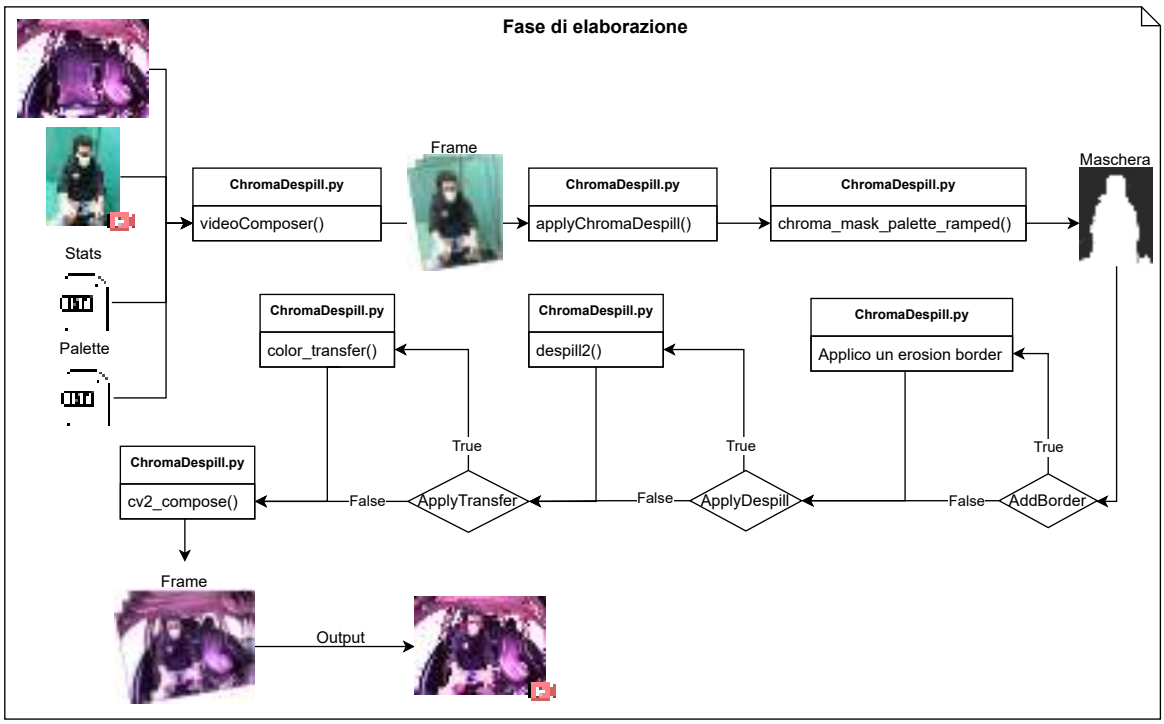

Figura 3.3: Fase di elaborazione

3.1 Calibrazione e Palette

Per sfruttare al pieno le potenzialità del tool è stata prevista una parte preliminare. Questa prima parte prevede: una creazione di una palette di colori, appartenenti al green screen della immagine a cui si vuole applicare la tecnica del Chromakey, e la calibrazione, attraverso un notebook interattivo, dei parametri della funzione del Chromakey e del filtro colore da applicare al soggetto.

3.1.1 Palette

La prima parte della fase preliminare è la creazione della palette di colori estratta dal green screen. La palette, che viene poi utilizzata nell'algoritmo di Chromakey che permette di rendere la porzione di immagine contenente il green screen trasparente, di massimizzare la rimozione del green screen e al contempo di minimizzare la rimozione del resto dell'immagine. Per la creazione della palette è stato implementato lo script "calibrator.py". Questo script contiene la funzione `"select_chroma_colors()"` che, prese in input due immagini: Un'immagine di cui si vuole eliminare il green screen (Figura 3.4) e una stessa immagine a cui è stato rimosso il soggetto (Figura 3.5), restituisce la palette.

Figura 3.4: Immagine di input

Figura 3.5: Immagine di input con soggetto rimosso

Passando alla funzione `"select_chroma_colors()"` come parametri la (Figura 3.4) e la (Figura 3.5) verrà creata una palette specifica per il setup di quell'immagine con quel soggetto. Questo è un limite, perché, se cambia il soggetto, l'algoritmo del Chromakey non riuscirà più a distinguere in modo corretto il background (la porzione da rimuovere) con il foreground (la porzione da mantenere). Questo è dovuto dal fatto che ogni soggetto crea delle luci e delle ombre diverse sul green screen. Per poter risolvere questo problema è stata implementata la funzione `"loadAndComposeData()"`. Questa funzione prende in input una serie di immagini come la (Figura 3.4) e la (Figura 3.5), ma con diversi soggetti, e li compone in un unica immagine (Figura 3.6) e (Figura 3.7) .

Figura 3.6: Immagini di input composte

Figura 3.7: Immagini di input senza il soggetto composte

Calcolando, quindi, la palette utilizzando queste immagini composte si riesce ad avere un range di colori più ampio, di conseguenza, dato un setup è necessario eseguire la calibrazione soltanto una volta.

Figura 3.8: Confronto dell'applicazione del Chromakey utilizzando una palette singola e una composta

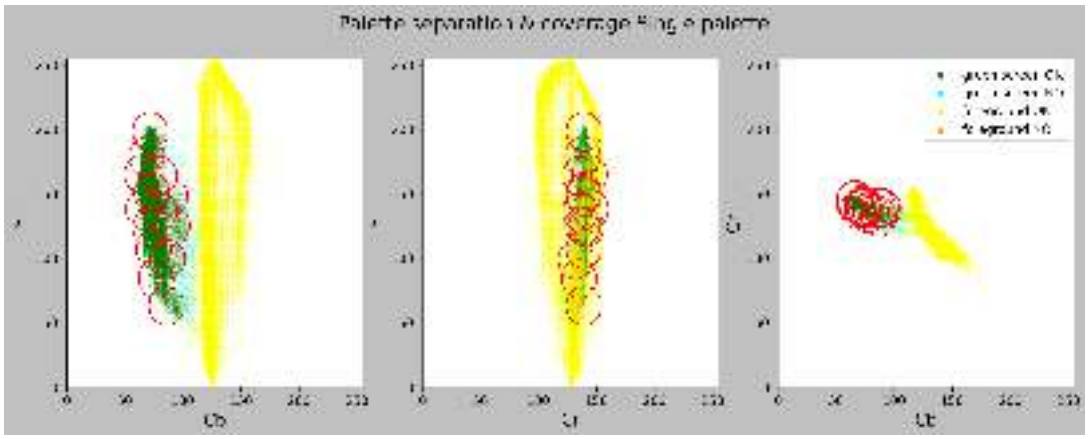

Figura 3.9: Palette separation and coverage della singola palette

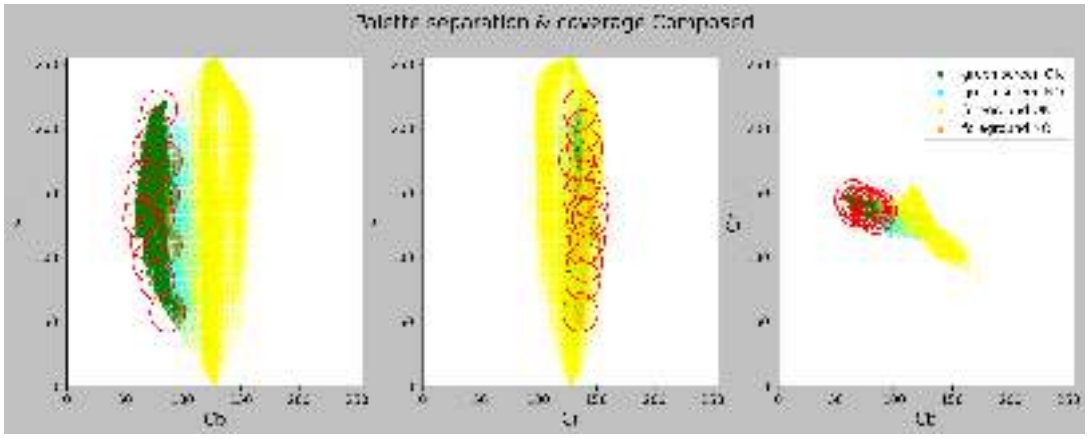

Figura 3.10: Palette separation and coverage della palette composta

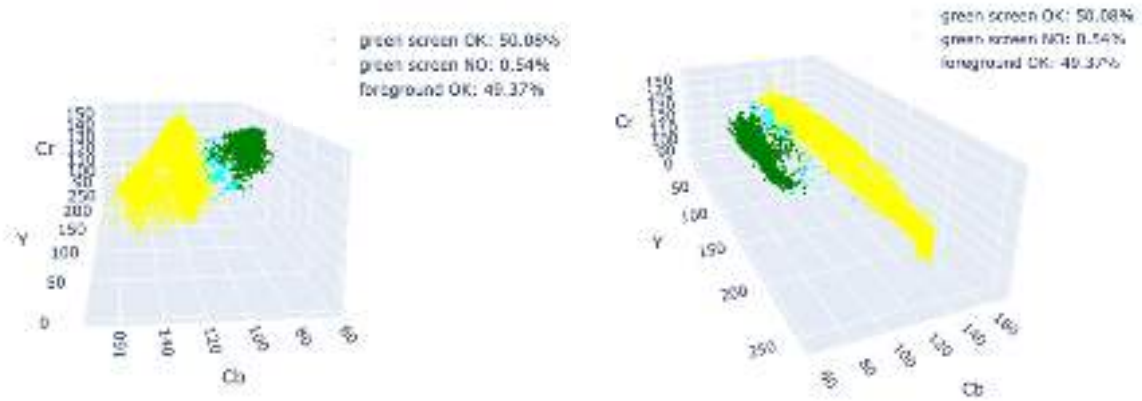

Figura 3.11: Visualizzazione 3D del grafico Palette and Coverage

La (Figura 3.9) e la (Figura 3.10) rappresentano il grafico "Palette separation and coverage" che mostra i pixel dell'immagine classificati. I pixel, attraverso l'utilizzo della palette, sono stati classificati in 4 categorie:

- Green screen OK (verde): che contiene tutti quei pixel del background che rientrano nel range della palette.
- Green screen NO (ciano): che contiene tutti quei pixel del background che non rientrano nel range della palette.
- Foreground OK (giallo): che contiene tutti quei pixel del foreground che non rientrano nel range della palette.
- Foreground NO (arancione): che contiene tutti quei pixel del foreground che rientrano nel range della palette.

Osservando la Figura 3.9 e la Figura 3.10 si possono notare due principali differenze: nel grafico Y/Cb della palette composta, rispetto al grafico Y/Cb della palette singola,

la zona occupata dai pixel verdi è molto più ampia ed è ben distaccata dalla zona occupata dai pixel gialli, e che nel grafico Cr/Cb i pixel verdi tendono a concentrarsi su un punto distaccato dal resto dei pixel. Questo a prova che attraverso la palette composta si ha un maggior range di colori e, al contempo, si riesce a distinguere in modo ottimale il background dal foreground.

Il codice

La funzione che ci permette di calcolare la palette è `select_chroma_colors()` (Codice 3.1) che come parametri, riceve, come spiegato in precedenza, l'immagine per intero e l'immagine a cui è stato rimosso il soggetto. Il modello colore delle immagini è YCbCr con "Y" che è il componente di luma, mentre Cb e Cr indicano i componenti di crominanza (cioè la differenza dal verde nel campo del blu e del rosso, rispettivamente). Attraverso la funzione di OpenCv `im.read()` le immagini vengono caricate e convertite in una matrice di triplette [Y,Cb,Cr], ogni tripletta rappresenta un pixel. Per prima cosa viene linearizzata la matrice attraverso la funzione `linearize_pixels()` che ridimensiona la matrice in una array lineare senza modificarne i dati. Dopo di che si vanno ad eliminare dall'array dell'immagine di background tutti i pixel neri. Questi pixel sono caratterizzati da una luminosità molto bassa, quindi, con un valore del parametro "Y" piccolo. Attraverso dei test è stato prefissato il valore `"BACKGROUND_MIN_Y_VALUE"`, tutti quei pixel con il parametro "Y" inferiore ad esso saranno quindi eliminati. I pixel rimossi corrispondo alla porzione di immagine in cui è stato rimosso il soggetto, invece i pixel rimanenti, che corrispondo al green screen, saranno utili per la creazione della palette .

```
1 CHROMA_RADIUS = 15
2 BACKGROUND_MIN_Y_VALUE = 25
3 # Data un immagine con il green screen e separatamente il suo sfondo,
    calcola una palette di colori della porzione di immagine da
    rimuovere
4 def select_chroma_colors(backgroundYCbCr, foregroundYCbCr):
5     # Si ridimensionano le immagini in un array lineare
6     linB = linearize_pixels(backgroundYCbCr)
7     linF = linearize_pixels(foregroundYCbCr)
8     # Dal background vengono rimossi tutti i pixel neri
9     linB = linB[ linB[:,0] > BACKGROUND_MIN_Y_VALUE ].astype(float)
```

Codice 3.1: Preparazione all'estrazione della palette

Dopo la prima fase di preparazione si passa alla creazione vera e propria della palette (Codice 3.2). Come primo passo, viene estratto, in modo casuale, un colore (`clr`) dall'array contente tutti i pixel del green screen (`linB`), dopo di che, attraverso la funzione `np.linalg.norm()`, vengono calcolate le normali dell'array differenza tra `linB` e `clr` (`distB`), nello stesso modo viene calcolato `distF` ma utilizzando l'array differenza tra `linF` (l'array contenente l'immagine per intero) e `clr`. Successivamente vengono creati due array booleani (`inClusterB` e `inClusterF`) della stessa dimensione di `distB` e di `distF` , che conterranno valori `'True'` o `'False'` se il corrispettivo valore in `distB` o in `distF` è minore o meno della costante `'CHROMA_RADIUS'`. La costante `'CHROMA_RADIUS'` ha un valore che è frutto di un'analisi empirica e indica il limite

massimo per la selezione dei valori, rappresentato nella Figura 3.9 e nella Figura 3.10 come un cerchio, con raggio `'CHROMA_RADIUS'`, rosso. Dato che in Python `'True'` ha valore numerico '1' e `'False'` ha valore numerico '0', sfruttiamo la funzione `np.sum()`, per calcolare la sommatoria degli elementi di `inClusterB` e di `inClusterF`, così facendo si ottiene il numero di volte cui il colore `clr` è presente nello sfondo (`countB`) e il numero di volte in cui è presente nel soggetto (`countF`). Dati quindi `countB` e `countF`, è possibile calcolare il punteggio del colore `clr` estratto (`ratioBF`). Il punteggio sarà inversamente proporzionale al numero di volte in cui `clr` è presente sul soggetto, questo perché per avere un ottima rimozione del green screen, la palette dovrà essere composta da dei colori che sono presenti sullo sfondo ma non sul soggetto. Se il punteggio del colore è superiore a 10, il colore (`clr`) con il suo punteggio (`ratioBF`) viene inserito nella lista dei candidati per far parte della palette. Questa serie di istruzioni sono incapsulate all'interno di un ciclo `for` in cui vengono iterate `'NUM_TEST_SAMPLES'` volte, alla fine del ciclo `for` la lista di canditati conterrà al più `'NUM_TEST_SAMPLES'` colori. Quindi la lista di candidati viene ordinata e viene estratto il colore con il punteggio maggiore. Il colore estratto viene rimosso da `linB` e viene aggiunto alla lista `colors` che conterrà tutti i colori della palette. Il codice appena descritto è contenuto in un ciclo `while` che viene iterato fino a che: la lista `colors` contiene al massimo 30 colori o nell'array `linB` non c'è più nessun colore da estrarre.

```
1  while len(linB) > 0 and len(colors) < 30 and iters < 1000:
2          iters += 1
3          # Si prendono NUM_TEST_SAMPLES colori,
4          # viene selezionato quello con il cluster piu‘ grande
5          NUM_TEST_SAMPLES = 16
6          candidates = []
7          for ii in range(NUM_TEST_SAMPLES):
8              # Si prende un verde dal background in modo casuale
9              clr = linB[random.randint(0, len(linB)-1)]
10             # Viene calcolata la normale per ogni valore,
11             # dell'array a cui viene sottratto il verde preso
12             distB = np.linalg.norm(linB - clr, axis=1, ord=2)
13             distF = np.linalg.norm(linF - clr, axis=1, ord=2)
14             # Si verifica se ogni valore sia minore della costante
15             inClusterB = (distB < CHROMA_RADIUS)
16             inClusterF = (distF < CHROMA_RADIUS)
17             # Si fa la sommatoria dei valori dell'array
18             # (true = 1, false = 0)
19             countB, countF = np.sum(inClusterB), np.sum(inClusterF)
20             # Viene calcolato un punteggio per il colore scelto
21             ratioBF = countB / (countF + 1)
22             # Se il punteggio e‘ superiore a 10, il colore viene
23             # aggiunto alla lista dei candidati per la palette
24             if ratioBF > 10:
25                 candidates.append( (ratioBF, clr) )
26         if len(candidates) == 0:
27             break
28         # I candidati sono ordinati per punteggio
29         candidates = sorted(candidates, key=lambda x:x[0])
30         # Viene selezionato il colore con il punteggio maggiore
31         clr = candidates[-1][1]
```

```
        # Vengono rimossi i colori vicini a clr
        distB = np.linalg.norm(linB - clr, axis=1, ord=2)
        inClusterB = (distB < CHROMA_RADIUS)
        linB = linB[ ~inClusterB ]
        # Il colore viene aggiunto alla palette
        colors.append(list(clr))
```

Codice 3.2: Estrazione della palette

3.1.2 Calibrazione

La seconda parte della fase preliminare avviene tramite l'utilizzo di un notebook e comprende la calibrazione del chromakey e del filtro colore. La calibrazione avviene in modo interattivo tramite gli strumenti offerti dalla libreria *matplotlib* [5] che permette in tempo reale di osservare come il cambiamento delle variabili impatta sul risultato.

Figura 3.12: Calibrazione interattiva chromakey

Calibrazione del chromakey

Come mostrato nella Figura 3.12, nel notebook vengono mostrate a video quattro immagini, corrispettivamente, la maschera, la maschera con applicato lo spessore sui bordi, la maschera su cui sono mostrate le parti su cui agisce il despill e infine il risultato finale. La calibrazione interattiva avviene tramite l'utilizzo di due slider, che permettono di variare il valore della `bordeSize` e della `sensitivity`, e di tre bottoni, che permettono di applicare o meno il Despill, l'`addBorder` e la `doRamp`. Il bottone `addBorder` permette di applicare un'erosione al bordo intorno al soggetto così da avere un cambiamento meno netto tra il background e il foreground, lo slider `borderSize`

invece permette di impostare lo spessore del bordo. Il bottone `applyDespill` permette di applicare la tecnica del despill che rimuove i residui di verde dal soggetto, lo slider `sentitivity` invece permette di regolare l'influenza della tecnica sull'immagine. Infine il bottone `doRamp` permette di impostare l'euristica di applicazione del despill.

Figura 3.13: Calibrazione interattiva chromakey

Calibrazioni del filtro colori

Come mostrato nella Figura 3.13, nel notebook vengono mostrate tre immagini, corrispettivamente, la sorgente da cui viene preso il filtro, il risultato finale e l'immagine a cui applicare il filtro. La calibrazione interattiva avviene tramite l'utilizzo di sei slider che permettono di variare per ogni parametro il moltiplicatore `MeanM` e `StdM` e di due pulsanti: il pulsante `preserve_paper` e il pulsante `save`. Gli slider permettono di impostare un moltiplicatore con cui verrà calcolata la media o la deviazione standard del parametro. Il pulsante `preserve_paper` è un opzione che permette di impostare il modo in cui avviene l'applicazione della ricalibrazione dei colori. Infine il pulsante `save` permette di salvare in un file di testo i parametri calcolati con l'utilizzo dei moltiplicatori.

3.2 Chromakey

La tecnica del chromakey è un processo che permette di rimuovere un colore da un immagine o da un video e sostituirlo con un altra immagine o un altro video. I colori di sfondo maggiormente utilizzati per applicare questa tecnica sono il verde e il blu, dato che, sono dei colori non sono dominanti nella pelle umana o nei vestiti. L'algoritmo è stato progettato per acquisire in input una palette di colori estratta nella fase preliminare, ed applicare la tecnica del chromakey su un video a cui si vuole sostituire lo sfondo con un'immagine. L'algoritmo si divide in due fasi, la creazione di una maschera e la composizione dell'immagine con il nuovo sfondo.

Creazione della maschera

La maschera permette di nascondere delle porzioni di immagine. Nel caso riportato la porzione di immagine nascosta è lo sfondo verde, di conseguenza permette di rivelare, se presente, l'immagine sottostante. Creare una buona maschera è un passo fondamentale per ottenere un buon risultato finale. La qualità della maschera dipende molto da come riesce a definire il contorno del soggetto, che deve essere ben definito ma al contempo non troppo netto, perché, renderebbe l'immagine composta poco realistica. Inizialmente era stata implementata una funzione che creava una maschera binaria, cioè una maschera composta da soli 1 (visibile) e 0 (non visibile). Si è però notato che utilizzando una maschera binaria nella composizione delle immagini il bordo di giunzione tra le due immagini risultava troppo netto e frastagliato, quindi molto poco realistico. Si è quindi pensato ad un'implementazione in cui la maschera viene creata in modo non binario, in altre parole, la maschera possiede dei valori che vanno dal 1 (visibile) allo 0 (non visibile), tutti i valori compresi tra questo range saranno più o meno trasparenti. Utilizzando una maschera non binaria il risultato finale riesce a essere molto più realistico, il quale, avendo un bordo molto più morbido, riesce meglio a confondersi con il nuovo sfondo.

Il codice

La funzione che permette di creare la maschera è `chroma_mask_palette_ramped()`. La funzione riceve in input due parametri: l'immagine dalla quale si creerà la maschera (`imageYCbCr`) e il percorso della posizione della palette. Come primo passo viene caricata la palette utilizzando la funzione `loadtxt()` della libreria NumPy. Successivamente utilizzando la funzione di OpenCv `shape()` si ottengono le dimensioni dell'immagine `imageYCbCr`. Utilizzando la funzione di NumPy `zeros()`, viene creata la maschera iniziale (`mask`) avente le stesse dimensioni dell'immagine `imageYCbCr` e contenendo solamente valori uguale a zero. Inizializzata la maschera, ogni colore della palette contribuirà alla creazione della maschera. Utilizzando la funzione di NumPy `linalg.norm()` vengono calcolate, elemento per elemento, le normali dell'array differenza tra `imageYCbCr` e `clr`. L'array risultante (`m2`), conterrà quindi per ogni pixel quanto è cromaticamente "distante" dal colore `clr`. Dopo di che viene chiamata la funzione `linear_ramp()` la quale richiede tre parametri: un array e due costanti. La chiamata alla funzione `linear_ramp()` avviane passando come array `m2` e come costan-

ti, corrispettivamente, la costante `CHROMA_RADIUS` amplificata (`start`) e la costante `CHROMA_RADIUS` ridotta (`delta`). La funzione `linear_ramp()` è quello che permette di creare una maschera non binaria. Attraverso la funzione di NumPy `clip()` ogni valore dell'array passato viene poi classificato nel seguente modo:

- Se il risultato dell'operazione "`(mask - start) / delta`" è un numero minore o uguale a 0, gli viene assegnato il valore 0. Questo sta ad indicare che il valore preso in considerazione, rispetto al colore della palette `clr`, ha una "distanza" cromatica molto piccola, si presume quindi che faccia parte del green screen, dovrà essere reso invisibile.

- Se il risultato dell'operazione "`(mask - start) / delta`" è un numero maggiore o uguale a 1, gli viene assegnato il valore 1. Questo sta ad indicare che il valore preso in considerazione, rispetto al colore della palette `clr`, ha una "distanza" cromatica molto grande, si presume quindi che non faccia parte del green screen, dovrà restare visibile.

- Se il risultato dell'operazione "`(mask - start) / delta`" è un numero compreso tra 0 e 1, il suo valore resta invariato. Questo sta ad indicare che il valore preso in considerazione, rispetto al colore della palette `clr`, ha una "distanza" cromatica intermedia, si presume quindi che sia il bordo tra il green screen e il soggetto, dovrà essere reso trasparente.

Tramite la funzione di NumPy `maximum()`, viene poi preso, elemento per elemento, il massimo tra il complementare dell'array restituito della funzione `linear_ramp` e `mask`. Infine, dopo aver ripetuto queste operazioni per ogni colore appartenente alla palette, la funzione restituisce l'inverso della maschera.

Figura 3.14: Creazione maschera

```
# Viene creata una rampa che va da 1 (il colore piu‘ "vicino" a clr)
# a 0 (il colore piu‘ "lontano")
def linear_ramp(mask, start, delta):
    # I valori vengo limitati al range [0-1]
    return np.clip((mask - start) / delta, 0.0, 1.0)

# Data un'immagine e una palette di colori
# calcola la maschera di composizione
def chroma_mask_palette_ramped(imageYCbCr,palettePath):
    # Vengono salvate le dimensioni dell'immagine
    h,w,c = imageYCbCr.shape
```

```
    # Viene caricata la palette
    palette = np.loadtxt(palettePath)
    # Viene creata una maschera, cui i valori hanno valore 0,
    # delle stesse dimensioni dell'immagine
    mask = np.zeros([h,w], float)
    # Per ogni colore della palette
    for ii, clr in enumerate(palette):
        # Viene calcolata la normale per ogni valore,
        # dell'array a cui viene sottratto il colore
        m2 = np.linalg.norm(imageYCbCr - clr, axis=2, ord=2)
        # Viene restituito il complementare
        m3 = 1.0 - linear_ramp(m2, CHROMA_RADIUS * 1.25,
                               CHROMA_RADIUS * 0.50)
        # Per ogni valore dei due array viene preso il piu' grande
        mask = np.maximum(mask, m3)
    return 1-mask
```

Codice 3.3: Estrazione maschera

Composizione con il nuovo sfondo

La composizione con il nuovo sfondo è la fase finale della tecnica del Chromakey, in cui viene restituito il risultato finale. La composizione permette di unire l'immagine con il suo nuovo sfondo grazie all'applicazione della maschera creata in precedenza.

Figura 3.15: Composizione immagine: sfondo a cui è stato rimosso il soggetto, immagine a cui è stato rimosso lo sfondo, immagine composta

Come si può notare dalla Figura 3.15, per un limite del setup, l'immagine da incollare sul nuovo sfondo ha una tipologia di ripresa differente. Questo perché lo sfondo è stato catturato tramite l'utilizzo di un ottica fisheye, al contrario l'immagine da "incollare" è stata catturata con un ottica standard. Questo limite si potrebbe colmare sostituendo l'ottica standard con un'ottica fisheye.

Il codice

La funzione che permette di comporre l'immagine a cui è stato rimosso il green screen con il nuovo sfondo è `cv2_compose()`. La funzione oltre a comporre una sull'altra le due immagini permette anche di decidere su quale zona "incollare" l'immagine sullo sfondo e con quale dimensione. `cv2_compose()` riceve in input 4 parametri: l'immagine a cui deve essere tolto il green screen (`foreground`), l'immagine da utilizzare

come sfondo (`background`), la maschera (`alphamask`), dove deve essere posizionato il `foreground` sull'asse x rispetto all'origine del `background` (`posx`), dove deve essere posizionata il `foreground` sull'asse y rispetto all'origine del `background` (`posy`) e di quanto deve essere scalata in percentuale l'immagine (`scale`). Inizialmente, la maschera `alphamask`, con l'utilizzo della funzione di NumPy `stack()`, viene trasformata in un array di tre canali tutti uguali in modo che possa essere utilizzata nelle operazioni con il `foreground` e il `background`. Dopo di che, utilizzando la funzione di NumPy `multiply()` viene moltiplicata la maschera con il `foreground`, facendo questa operazione si rimuove lo sfondo verde dal `foreground` come è possibile vedere nella Figura 3.15. Si ottiene questo risultato perché, dato che i valori che compongo l'`alphamask` sono compresi tra 0 e 1, i valori del `foreground` che vengono moltiplicati con 0 si annullano. Successivamente utilizzando la funzione `resizeImg()`, allo stesso modo, vengono ridimensionati sia il `foreground` sia l'`alphamask`. Alla riga 9, utilizzando la funzione `multiply()`, viene moltiplicata l'inverso della maschera con il `background` a partire dalle coordinate (`posx`, `posy`), così facendo si rimuove la figura del soggetto dal `background` come è possibile vedere nella Figura 3.15. Infine sfruttando la funzione di NumPy `add()` vengono sommate tra di loro il `background` e il `foreground` così da ottenere l'immagine composta.

```
1  # Compone due immagini utilizzando la maschera di composizione
2  def cv2_compose(foreground,background,alphamask,posx,posy,scale):
3      # Viene ricordato il vecchio datatype
4      old_type = background.dtype
5      # alphamask viene composta da 3 canali tutti uguali
6      alphamask = np.stack([alphamask]*3, axis=2)
7      # Viene moltiplicata la maschera all'immagine
8      # convertita da uint8 a float
9      foreground = cv2.multiply(alphamask, foreground.astype(float))
10     # La maschera e l'immagine vengono ridimensionate allo stesso modo
11     foreground = resizeImg(foreground,scale)
12     alphamask = resizeImg(alphamask,scale)
13     # Vengono salvate le dimensioni dell'immagine ridimensionata
14     fh,fw,_ = foreground.shape
15     # Viene convertito da uint8 a float
16     background = background.astype(float)
17     # La maschera viene moltiplicata all'immagine di background
18     # nella posizione desiderata
19     background[posy:(posy+fh),posx:(posx+fw)] = cv2.multiply(1.0 -
               alphamask, background[posy:(posy+fh),posx:(posx+fw)])
20     # Viene aggiunto il foreground sul background
21     background[posy:(posy+fh),posx:(posx+fw)] =
        cv2.add(background[posy:(posy+fh),posx:(posx+fw)],foreground)
22     # L'immagine viene riportata al vecchio datatype
23     outImage = background.astype(old_type)
24     return outImage
```

Codice 3.4: Funzione della composizione immagine

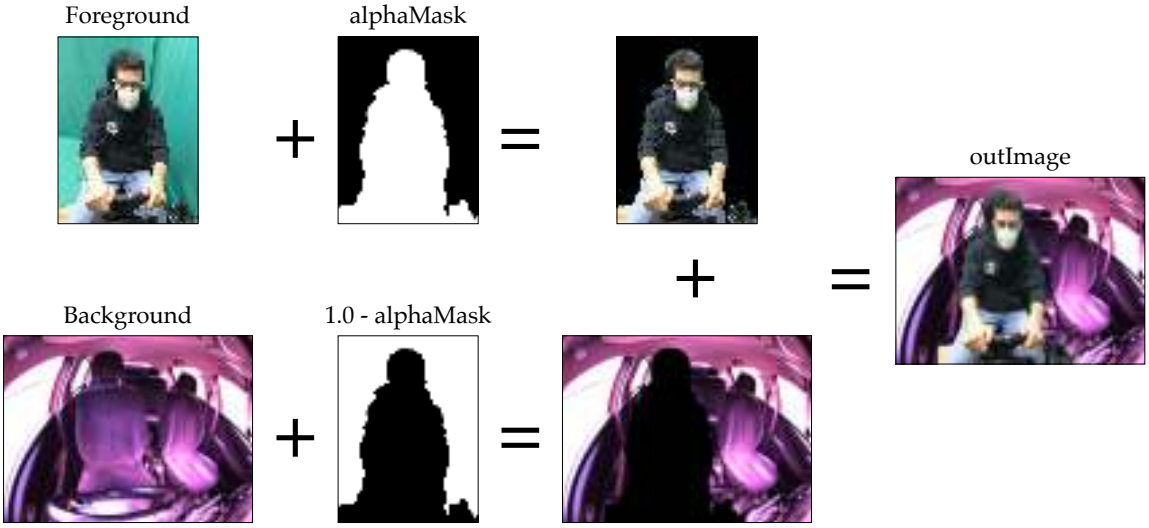

Figura 3.16: Composizione delle immagini

3.3 Despill

Il despill è una tecnica che permette di effettuare una correzione cromatica dell'immagine. L'implementazione di questa tecnica è un passaggio importante nell'applicazione della tecnica del chromakey. Un buon despill consente, in modo automatico, di nascondere le imperfezioni che posso essere dovute dalla creazione di una maschera non perfetta o da una scarsa illuminazione del soggetto. Dato che il soggetto è immerso nel verde, una scarsa illuminazione di può portare a dei riflessi verdi su esso. Il despill, quindi, lavorando pixel per pixel, consente di rimuovere gli aloni verdi che potrebbero rimanere sull'immagine dopo l'applicazione del chromakey. Questa tecnica permettere quindi di ottenere una migliore qualità del risultato finale. Nella Figura 3.17 è possibile notare l'effetto della tecnica sul volto del soggetto. Invece nella Figura 3.18 in cui è rappresentato ogni pixel dell'immagine nello spazio colore, si possono notare in rosso i pixel su cui è stato applicato il despill. Esiste più di un'euristica sul metodo di applicazione del despill. In questa implementazione la condizione in cui viene applicato, su un determinato pixel, è regolata dal valore del canale verde rispetto alla media del canale rosso col canale blu. Queste euristiche sono stabilite tramite delle prove empiriche. Si è scelto di utilizzare questa euristica principalmente per due motivi:

- Dalle prove effettuate si sono riscontrati i risultati più soddisfacenti.
- Nel soggetto, il valore del canale verde dovrebbe essere alto solamente nelle porzioni di immagine bianche, dove quindi anche il rosso e il blu hanno un valore alto. Quindi, idealmente, nelle zone in cui solamente il verde ha un valore alto, si pensa che quella zona può essere stata influenzata dal riflesso del green screen.

Figura 3.17: Immagine pre e post l'applicazione delle tecnica del despill

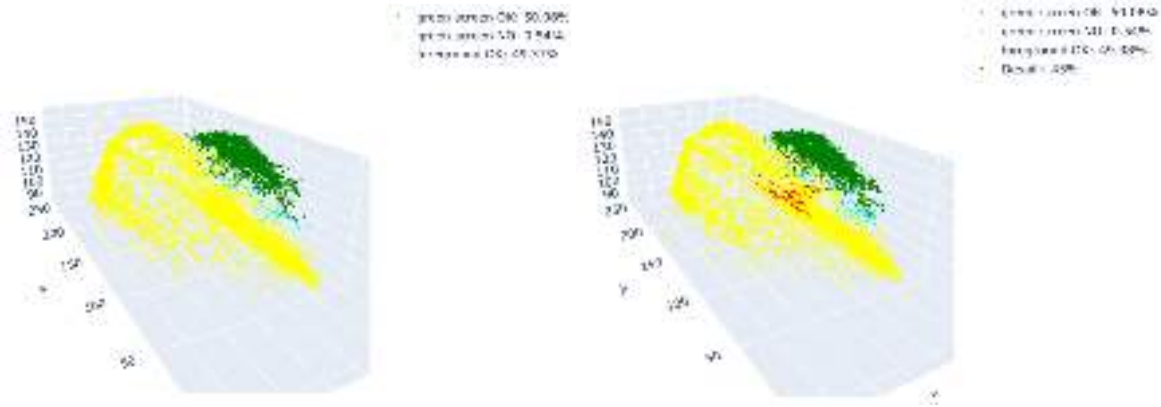

Figura 3.18: Visualizzazione 3D della color space pre e post despill

Il codice

La funzione che implementa il despill è `despill2()`. L'implementazione é stata pensata in modo da permettere all'utente di poter scegliere, tramite il parametro booleano `doRamp`, se applicare il despill in modo binario o meno. Questa possibilità è stata aggiunta perché, come già visto in precedenza nella creazione della maschera, una creazione non binaria permette di ottenere un risultato che si integra in modo poco invasivo con il resto dell'immagine. `despill2()` riceve in input 4 parametri: l'immagine su cui applicare il despill (`imageBGR`), un moltiplicatore (`sensitivity`), un parametro booleano (`doRamp`) e una costante (`rampDelta`) ed opera come segue. Inizialmente viene creata una copia di `imageBGR` (`np_image`) e vengono valorizzate le variabili `r,g,b` corrispettivamente con i canali dell'immagine, rosso, verde e blu. Dopo di che viene valorizzata `replace_val` che conterrà, per ogni pixel, la media tra il canale rosso e quello blu. Viene poi creato l'array `weighted_threshold` il quale conterrà tutti i valori dell'array `replace_val` moltiplicati con il parametro `sensitivity` e , successivamente, attraverso l'utilizzo della funzione di NumPy `clip()`, racchiusi nell'intervallo tra 0 e 255. Viene poi creato l'array `cond` che conterrà, per ogni pixel, la condizione che decide, se su quel pixel deve essere applicato il despill. `cond` come in precedenza è stato detto, può essere calcolato in modo binario o non binario:

- Metodo binario: la condizione di applicazione avrà valore positivo quando il valore di `g` sarà maggiore del corrispettivo valore di `weighted_threshold`.
- Metodo non binario: la condizione di applicazione in questo caso può avere un valore compreso tra 0 e 1 calcolato con la funzione `linear_ramp()`, già utilizzata nella creazione della maschera binaria, a cui viene passato in input l'array `g` e i due parametri `weighted_threshold` e `rampDelta`.

In seguito viene valorizzato `replace_g` utilizzando la funzione di NumPy `minimum()` tra `replace_val` e `g`, funzione che restituisce un array contenente il minimo valore per ogni coppia di elementi di due array della stessa dimensione. Infine viene modificato il canale del colore verde di `np_image` sostituendolo con il corrispettivo valore di `replace_val` se il valore di `cond` è positivo, con il corrispettivo valore di `g` altrimenti.

```
1  # Data un immagine corregge gli aloni di verde
2  def despill2(imageBGR, sensitivity, doRamp, rampDelta=24):
3      # Viene creata una copia dell'immagine
4      np_image = np.array(imageBGR)
5      # Dall'immagine vengo estratti i due canali
6      r = np_image[:,:,2]
7      g = np_image[:,:,1]
8      b = np_image[:,:,0]
9
10     # Viene calcolata la media tra il canale rosso e il canale blu
11     replace_val = b/2 + r/2
12     # Viene definita la soglia pesata
13     weighted_threshold = np.clip(replace_val * sensitivity, 0, 255)
14     # Viene valorizzato l'array delle condizioni
15     cond = linear_ramp(g.astype(float), weighted_threshold, rampDelta)
       if doRamp else (g > weighted_threshold).astype(float)
16     # Viene valorizzato l'array di valori
17     # con i valori del verde corretti
18     replace_g = np.minimum(replace_val, g)
19     # Se la condizione e' positiva il valore del verde,
20     # in un pixel, viene sostituito con quello corretto
21     np_image[:,:,1] = cond * replace_g + (1-cond) * g
22     return cond, np_image
```

Codice 3.5: Funzione del despill

3.4 Ricalibrazione dei colori

È stata implementato uno strumento che consente di applicare una ricalibrazione del colore all'immagine a cui è stato rimosso lo sfondo. Questo permette di rendere più immersiva la composizione del soggetto sul suo nuovo sfondo, così da rendere il risultato finale il più realistico possibile. La ricalibrazione del colore avviene eseguendo un transfer da un'immagine sorgente (il nuovo sfondo) ad un immagine target (l'immagine da "incollare" sullo sfondo). Il codice che ci permette di ottenere la ricalibrazione del colore è stato ispirato dal GitHub di jrosebr1 [6].

Figura 3.19: Ricalibrazione colore

Il codice

Il codice per la ricalibrazione dei colori è diviso in due parti, una parte di calibrazione utilizzata dal notebook interattivo e una parte che è applica il filtro colore sull'immagine.

La calibrazione

`get_Profile()` consente di calcolare i dati in seguito utilizzati per l'applicazione del transfer. La funzione riceve come parametri: l'immagine alla quale si vogliono calcolare le statitiche (`image`), un parametro booleano che consente di applicare o meno la maschera sull'`image` (`applyMask`) e un moltiplicatore per ogni parametro. La funzione viene utilizzata in due casi differenti:

- Per il calcolo delle statistiche dell'immagine sorgente. In questo caso il parametro `applyMask` viene passato con valore `False`, quindi le statistiche vengono calcolate sull'intera immagine.
- Per il calcolo delle statistiche dell'immagine a cui si è applicata la rimozione dello sfondo. In questo caso il parametro `applyMask` viene passato con valore `True`, così facendo il calcolo delle statiche viene limitato solamente sulla porzione di immagine contenente il soggetto.

È stata implementa questa distinzione perché, per l'immagine a cui è stata applicata la rimozione dello sfondo, comprendere la porzione di immagine rimossa portava ad un calcolo errato delle statistiche. Questo perché, altrimenti, tra tutti i pixel presi in considerazione per il calcolo delle statistiche verrebbe inclusi anche i pixel appartenenti allo sfondo; parte nascosta dell'immagine e quindi non di interesse. Fatte questa distinzione, `Image` viene convertita, utilizzando la funzione di OpenCv `cvtColor()`, nello spazio colore L*a*b. Successivamente, con l'utilizzo della funzione di OpenCv `split()` l'immagine viene suddivisa nei sui rispettivi canali. Ottenuti i canali viene effettuato il calcolo della deviazione standard e della media per ciascuno di essi ottenendo il profilo di trasferimento dell'immagine. Quindi, i risultati vengono inseriti in un NumPy array e in seguito salvati in un file di testo.

```
# Data un immagine calcola
# la media e la deviazione standard dei canali
def get_Profile(image, applyMask=False, lMeanM=1, lStdM=1, aMeanM=1,
    aStdM=1, bMeanM=1, bStdM=1):
    if applyMask == True:
        # Viene estratta la maschera dall'alpha dell'immagine
        mask = linearize_mask(image[:,:,3])
        # Viene convertita l'immagine da RGB a L*a*b
        image= cv2.cvtColor(image,cv2.COLOR_BGR2LAB).astype("float32")
        # Dall'immagine vengo estratti i singoli canali
        (l, a, b) = cv2.split(image)

        # I canali vengono convertiti in un array lineare
        l = linearize_mask(l)
        a = linearize_mask(a)
        b = linearize_mask(b)

        # Viene presa solamente la porzione di immagine
        # in cui e‘ presente il soggetto
        l=l[mask!=0]
        a=a[mask!=0]
        b=b[mask!=0]
    else:
        # Viene convertita l'immagine da RGB a L*a*b
        image= cv2.cvtColor(image,cv2.COLOR_BGR2LAB).astype("float32")

        # Dall'immagine vengo estratti i singoli canali
        (l, a, b) = cv2.split(image)

    # Vengono calcolate: la devizione standard e la media
    (lMean, lStd) = (l.mean(), l.std())
    (aMean, aStd) = (a.mean(), a.std())
    (bMean, bStd) = (b.mean(), b.std())

    # le statiche vengono inserite nell'array,
    # che sara‘ poi salvato su un file di testo
    stats = []
    stats.append(lMean * lMeanM)
    stats.append(lStd * lStdM)
    stats.append(aMean * aMeanM)
    stats.append(aStd * aStdM)
    stats.append(bMean * bMeanM)
    stats.append(bStd * bStdM)
    stats = np.array(stats)

    return stats
```

Codice 3.6: Calcolo profilo immagine

Il trasferimento dei colori ad un'immagine target

`color_transfer()` è la funzione che utilizzando le statistiche calcolate dalla funzione `get_Profile()` effettua la ricalibrazione del colore dell'immagine target. La funzione riceve come parametri: l'immagine a cui si applica la ricalibrazione del colore

(`transfer`), le statistiche calcolate sull'immagine sorgente (`sourceStats`), le statistiche calcolate sull'immagine target (`targetStats`) e il parametro booleano `preserve_paper` che definisce il modo in cui viene calcolata la ricalibrazione. Il parametro `preserve_paper` era stato previsto dal codice [6] da cui si è preso spunto per la creazione di questa funzione. Nell'implementazione si è deciso di utilizzare questo parametro sempre a `False`. Perché, si è ritenuto che il modo con cui avveniva la ricalibrazione con il parametro `preserve_paper` a `True` fosse errato. Nonostante ciò è stata mantenuta la possibilità di cambiare valore al parametro, per una maggiore libertà. La funzione, come prima cosa, inizializza le variabili delle statistiche con il corrispondente valore estratto dal file di testo creato precedentemente. Inizializzate le variabili, `target` viene convertito, utilizzando la funzione di OpenCv `cvtColor()`, nello spazio colore L*a*b, e successivamente, con l'utilizzo della funzione di OpenCv `split()` l'immagine viene suddivisa nei rispettivi canali. Per ogni canale viene poi sottratta la propria media. Successivamente, in base all'euristica che impone il parametro `preserve_paper`, i canali vengono ridimensionati in due modi differenti:

- In base al rapporto tra la deviazione standard del target, diviso per la deviazione standard dell'immagine sorgente, se il valore di `preserve_paper` è `True`.
- In base al rapporto tra la deviazione standard dell'immagine sorgente, diviso per la deviazione standard del target, se il valore di `preserve_paper` è `False`.

Viene infine aggiunta, per ogni canale, la media del canale corrispondente dell'immagine sorgente. Nel caso qualche valore, nelle operazioni precedenti, ha preso un valore fuori dal range [0-255], utilizzando la funzione di NumPy `clip()`, viene mantenuto il range: ai valori minori di 0 viene assegnato il valore 0, ai valori maggiori di 255 viene assegnato il valore 255. Come ultimo passaggio i canali vengo nuovamente uniti, con l'utilizzo della funzione di OpenCv `merge()`; infine, l'immagine viene convertita utilizzando la funzione di OpenCv `cvtColor()`, in RGB.

```
1  # Data un'immagine fa una ricalibrazione del colore
2  # utilizzando delle statistiche precedentemente calcolate
3  def color_transfer(target, sourceStats, targetStats,
4                     preserve_paper=False):
5    # Vengono lette le statistiche e valorizzate le variabili
6    lMeanSrc = sourceStats[0]
7    lStdSrc = sourceStats[1]
8    aMeanSrc = sourceStats[2]
9    aStdSrc = sourceStats[3]
10   bMeanSrc = sourceStats[4]
11   bStdSrc = sourceStats[5]
12
13   lMeanTar = targetStats[0]
14   lStdTar = targetStats[1]
15   aMeanTar = targetStats[2]
16   aStdTar = targetStats[3]
17   bMeanTar = targetStats[4]
18   bStdTar = targetStats[5]
19
20   # Viene convertita l'immagine da RGB a L*a*b
21   target = cv2.cvtColor(target, cv2.COLOR_BGR2LAB).astype("float32")
```

```
22  # Dall'immagine vengo estratti i singoli canali
23  # subtract the means from the target image
24  (l, a, b) = cv2.split(target)
25
26  # Ad ogni canate e‘ sottratta la propria media
27  l -= lMeanTar
28  a -= aMeanTar
29  b -= bMeanTar
30
31  # Vengono ridimensionati i canali
32  if preserve_paper:
33    l = (lStdTar / lStdSrc) * l
34    a = (aStdTar / aStdSrc) * a
35    b = (bStdTar / bStdSrc) * b
36  else:
37    l = (lStdSrc / lStdTar) * l
38    a = (aStdSrc / aStdTar) * a
39    b = (bStdSrc / bStdTar) * b
40
41  # Ad ogni canate e‘ aggiunta la corrispettiva media della sorgente
42  l += lMeanSrc
43  a += aMeanSrc
44  b += bMeanSrc
45
46  # I valori vengo limitati al range [0-255]
47  l = np.clip(l, 0, 255)
48  a = np.clip(a, 0, 255)
49  b = np.clip(b, 0, 255)
50
51  # I canali vengono nuovamente uniti e l'immagine viene poi
     convertita in RGB
52  transfer = cv2.merge([l, a, b])
53  transfer = cv2.cvtColor(transfer.astype("uint8"), cv2.COLOR_LAB2BGR)
54
55  return transfer
```

Codice 3.7: Transfer

3.5 Ricostruzione del video

Infine è stata implementata una parte che permette di comporre, da ogni frame elaborato, il video finale. La funzione che fa ciò è `videoComposer()` che riceve come parametro `args`, una lista di argomenti opportunamente parsificati. Dopo aver inizializzato le variabili con i corrispettivi argomenti di `args`, utilizzando la funzione `VideoWrite` di OpenCv, viene inizializzato il video di output avente le caratteristiche del video di input e le dimensioni del nuovo sfondo. Successivamente, per ogni frame del video di input, viene chiamata la funzione `applyChromaDespillTransfer()` (Codice 3.9), una funzione che concatena le funzioni precedentemente descritte e restituisce l'immagine elaborata. Quindi ogni frame elaborato viene inserito nel video di output, fino all'esaurimento dei frame.

```
def videoComposer(args):
    # Vengono letti tutti i parametri passati come argomenti:
    # Coordinate e scala con cui deve essere "incollato il soggetto"
    posx = args.posx
    posy = args.posy
    scale = args.scale
    # Percorso della palette
    palettePath = args.palettePath
    # Caricate le statistiche per la ricalibrazione del colore
    sourceStats = np.loadtxt(args.sourceStats).astype("float32")
    targetStats = np.loadtxt(args.targetStats).astype("float32")
    # Video da elaborare
    cap = cv2.VideoCapture(args.video)
    # Il nuovo sfondo
    newBackground = cv2.imread(args.background)
    # il percorso in cui salvare l'output
    output_filename = args.outpath
    # Vengono salvate le dimensioni del nuovo sfondo
    h , w , _ = newBackground.shape
    frame_width = w
    frame_height = h
    # Vengono salvate le informazioni del video dato in input
    fps = int(cap.get(cv2.CAP_PROP_FPS))
    frame_count = int(cap.get(cv2.CAP_PROP_FRAME_COUNT))
    # Viene inizializzato il video di output
    out = cv2.VideoWriter(output_filename,
                cv2.VideoWriter_fourcc('M','J','P','G'),
                fps , (frame_width,frame_height))
    print(f"{output_filename}  {frame_height}x{frame_width}@{fps}
             frames:{frame_count}")
    # Fino all'ultimo frame del video:
    while cap.isOpened():
        done = False
        t = threading.Thread(target=animate)
        t.start()
        # Viene estratto un frame
        ret, frame = cap.read()
        # Se non ci sono piu‘ frame il programma viene arrestato
        if not ret:
            break
        # Ad ogni frame avviene l'elaborazione
        _,_,_,procFrame = mask, borderMask, despMask, image =
            applyChromaDespillTransfer(frame, newBackground,
    palettePath,
                posx, posy,scale, sourceStats, targetStats,
                BORDER_SIZE, SENSITIVITY,applyDespill=True,
                addBorder=True, doRamp=True,applyTransfer =True)
        # Il frame elaborato viene inserito sul video di output
        out.write(procFrame)
    done = True
    cap.release()
    out.release()
    del cap, out
```

Codice 3.8: Composizione video

```
def applyChromaDespillTransfer(imageBGR, backgroundBGR, palettePath, posx,
        posy,scale, sourceStats, targetStats, borderSz, sensitivity,
        applyDespill, addBorder, doRamp, applyTransfer):
    # Viene convertita l'immagine da BGR a Y*Cb*Cr
    imageYCbCr = cv2.cvtColor(imageBGR, cv2.COLOR_BGR2YCR_CB)
    # Viene creata la maschera
    chromaMask = chroma_mask_palette_ramped(imageYCbCr,palettePath)
    if addBorder:
        # Viene applicata l'erosione del bordo
        erosion_kernel = np.array([ [0,1,0], [1,1,1], [0,1,0] ],
                                   np.uint8)
        chromaMask = cv2.erode(chromaMask, erosion_kernel)
        borderMask = np.clip(gaussian_filter(chromaMask,
                            sigma=borderSz), 0.0, 1.0)
        alphaMask = np.minimum(chromaMask, borderMask)
    else:
        alphaMask = chromaMask
        borderMask = np.array([[1.0]])
    alphaMask = chromaMask
    # Viene applicato il despill
    despMask, despilled = despill2(imageBGR, sensitivity, doRamp)
    despMask = np.minimum(despMask, chromaMask)
    srcCompose = despilled if applyDespill else imageBGR
    # Viene effettuata la ricalibrazione colore
    if applyTransfer:
        srcCompose = color_transfer(srcCompose, sourceStats,
                                   targetStats)
    result = cv2_compose(srcCompose, backgroundBGR, alphaMask, posx,
                        posy, scale )
    return alphaMask, borderMask, despMask, result
```

Codice 3.9: Metodo applyChromaDespillTransfer

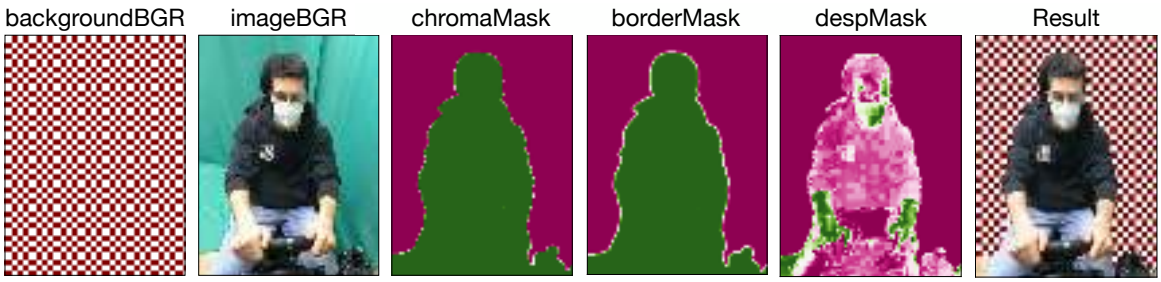

Figura 3.20: Illustrazione variabili Codice 3.9

Il codice implementato, per elaborare un video di 7 secondi con una risoluzione di 1542x2056 a 24 Fps, impiega circa 113 secondi, con una velocità di elaborazione di circa 0.7 secondi per frame.

Capitolo 4

Risultati

4.1 Benchmark

Allo scopo di confrontare l'algoritmo presentato con lo stato dell'arte attuale, sono stati svolti diversi benchmark. Per effettuare tale confronto sono stati presi in considerazione gli algoritmi che sono stati presentati inizialmente: l'applicativo DaVinci Resolve e l'algoritmo presentato nell'articolo "Interactive High-Quality Green-Screen Keying via Color Unmixing" [2]. I benchmark sono stati divisi in due tipi di analisi: un'analisi quantitativa ed un'analisi qualitativa. L'analisi quantitativa è stata un tentativo di costruire un sistema di valutazione numerico. Questo tipo di valutazione non è stata soddisfacente perché, le valutazioni numerica non ha rispecchiato a pieno la valutazione visiva. Dato che per un immagine è importante la parte visiva si darà quindi più peso alle analisi qualitative.

Per effettuare i benchmark sono state utilizzati due diversi setup di immagini: il setup del simulatore (Figura 4.1) e il setup delle immagini dell'articolo [7] (Figura 4.2)

Figura 4.1: Immagini utilizzate (Simulatore)

Figura 4.2: Benchmark proveniente dall'articolo [7]

4.1.1 Analisi quantitativa

Classificazione e confronto con matrici di confusione

Per valutare la bontà dell'algoritmo presentato, si è pensato di confrontare le immagini da esso elaborate con una base di conoscenza, assunta come perfetta, che comprende: Photoshop per il "setup Simulatore" (Figura 4.1) e le immagini elaborate fornite dagli autori dell'articolo per il "setup Articolo" (Figura 4.2). Come primo passo sono state elaborate le immagini in tre modi diversi:

- Utilizzando l'algoritmo proposto con tutte le sue funzionalità applicate. Una funzionalità importante nell'ambito di questo benchmark è l'erosione del bordo del soggetto. Essa permette di inglobare meglio il soggetto nel suo nuovo sfondo. L'erosione viene applicata nella maschera lungo tutto il bordo del soggetto andando così ad ampliare il bordo, i valori compresi tra 0 e 1, rendendolo quindi più morbido.
- Utilizzando l'algoritmo proposto senza l'erosione del bordo.
- Utilizzando l'applicativo DaVinci Resolve.

Successivamente, per ogni immagine elaborata si è ottenuto il canale alpha. Questo canale descrive il grado di trasparenza/opacità per ogni determinato pixel dell'immagine. Si è poi proceduto a classificare il canale in tre differenti classi:

- Classe 0: contenente tutti i valori compresi tra 0 e deltaSx. Rappresenta la porzione di immagine invisibile (il green screen).
- Classe 1: contenente tutti i valori compresi tra 1 e deltaDx. Rappresenta la porzione di immagine visibile (il soggetto).
- Classe 2: contenente tutti i valori compresi tra deltaSx e deltaDx. Rappresenta la porzione di immagine trasparente (il bordo).

DeltaSx (limite sinistro) e DeltaDx (limite destro) sono dei limiti che sono stati aggiunti per ammorbidire un po‘ la classificazione. È stato fatto ciò perché i valori molto vicini a 0 ed a 1 possono essere considerati corrispettivamente come invisibili e visibili. Dopo aver effettuato la classificazione, tramite l'utilizzo delle matrici di confusione, si sono confrontate le immagini elaborate con il loro corrispettivo ground truth. Il ground truth è un informazione che è nota per essere reale o vera, viene fornita dall'osservazione diretta e dalla misurazione (cioè l'evidenza empirica). Una matrice di confusione è uno strumento che consente di analizzare una classificazione ed è così composta:

- Sulle righe si trovano i valori reali (il ground truth).
- Sulle colonne si trovano i valori predetti (le immagini elaborate).

Le matrici di confusione permettono anche il calcolo di alcune metriche di valutazione come:

- L'accuratezza
- La precisione
- La recall
- Il punteggio F1

Il codice

Per la classificazione del canale alpha si è implementato il metodo `classifier()` (Codice 4.1). Il metodo riceve in input 3 parametri: il canale da classificare (`alpha`), il limite sinistro (`deltaSx`) e il limite destro (`deltaDx`). Inizialmente, tramite l'utilizzo della funzione `linearize_mask()`, il canale viene ridimensionato in una array lineare senza modificarne i dati. Successivamente, utilizzando la funzione di NumPy `where()`, ai valori del canale vengono assegnati i valori 0, 1, 2 se:

- 0: se il valore del canale è compreso tra `0` e `deltaDx`
- 1: se il valore del canale è compreso tra `deltaDx` e `1`
- 2: se il valore del canale è compreso tra `deltaSx` e `deltaDx`

Infine viene restituito il canale classificato.

```
def classifier(alpha, deltaSx=0.01,deltaDx=0.99):
    # Il canale alpha viene linearizzato
    alpha = linearize_mask(alpha)
    # Se il valore e‘ compreso tra "deltaSx" e "deltaDx" gli viene
   assegnato valore 2
    alpha = np.where((alpha > deltaSx) & (alpha < deltaDx) , 2, alpha)
    # Se il valore e‘ compreso tra "0" e "deltaDx" gli viene assegnato
    valore 0
    alpha = np.where((alpha >= 0) & (alpha <= deltaSx), 0, alpha)
    # Se il valore e‘ compreso tra "deltaSx" e "1" gli viene assegnato
    valore 1
    alpha = np.where((alpha >= deltaDx) & (alpha <= 1), 1, alpha)
    return alpha
```

Codice 4.1: Classificatore

La creazione della matrice di confusione e il calcolo delle metriche di valutazione avviene nella seguente porzione di codice (Codice 4.2). Come prima cosa viene aperto il file di testo su cui verranno salvati i dati. Successivamente per ogni immagine elaborata `f` viene controllato se faccia parte del "setup Simulatore" o del "setup Articolo":

- Se l'immagine fa parte del "setup Simulatore" sarà impostato come ground truth l'immagine elaborata da Photoshop
- Se l'immagine fa parte del "setup Articolo" sarà impostato come ground truth l'immagine elaborata offerta dagli autori dell'articolo [7].

Dopo di che, l'immagine di ground truth viene caricata nella variabile (`true`) e i suoi valori vengono convertiti in tipo float. Successivamente viene isolato il canale alpha dell'immagine, con i valori normalizzati nel range [0 - 1]. Viene infine classificato il canale alpha dalla funzione `classifier()`. Fatto ciò, per ogni immagine da confrontare, vengono ripetuti gli stessi passaggi, a differenza della classificazione che può avvenire con un `deltaSx` differente:

- Uguale 0.18 se `pred` è l'immagine elaborata da DaVinci Resolve
- Uguale 0.01 se `pred` è l'immagine elabora dal tool presentato.

È stata predisposta questa differenziazione perché DaVinci Resolve non rende mai delle porzioni di immagine completamente invisibili ma, le rende molto trasparenti. Quindi se si fosse mantenuto il `deltaSx` uguale per tutti i confronti, si sarebbero potute avere delle valutazioni errate. Dopo di che, grazie alla libreria `sklearn` [8], vengono calcolate le metriche di valutazione e la matrice di confusione con le seguenti funzioni:

- Con `precision_score()` viene calcolata la precisione.
- Con `recall_score()` viene calcolata la recall.
- Con `accuracy_score()` viene calcolata l'accuratezza.
- Con `f1_score()` viene calcolato il punteggio F.

- Con `confusion_matrix()` viene creata la matrice di confusione, con `true` nelle righe e `pred` nelle colonne.

Quindi per ogni confronto viene salvata la matrice di confusione e vengono scritte le valutazioni metriche nel file di testo.

```
import sklearn.metrics as sk
Frame = ['Paper 1','Paper 2','Paper 3','Paper 4','Setup 1','Setup 2','Setup 3']
# Viene aperto il file di testo su cui salvare i dati
file = open(pathFile, "w")
for f in Frame:
    file.write('%s\n' % f)
    # Se l'immagine fa parte del "setup Simulatore" come ground truth viene utilizzato "Photoshop" altrimenti viene utilizzata l'immagine elaborata fornita dal paper
    if f == 'Setup 1' or f == 'Setup 2' or f == 'Setup 3':
        trueName = 'Photoshop'
        prog = ['Chroma','ChromaNoBorder','AlphaDaVinci']
    else:
        trueName = 'Paper'
        prog = ['Chroma','ChromaNoBorder','AlphaDaVinci']
    # Viene letta l'immagine come valore vero
    true = cv2.imread(pathImgTrue +'/%s/%s.png' % (f,trueName), cv2.IMREAD_UNCHANGED)
    # I valori vengono convertiti in float
    true = true.astype(float)
    # Viene estratto il canale alpha e viene ristretto nel range [0-1]
    true = true[:,:,3]/255
    # Il canale viene classificato
    true = classifier(true)

    for p in prog:
        # Viene letta l'immagine predetta
        pred = cv2.imread(pathImgPred + '/%s/%s.png' % (f,p), cv2.IMREAD_UNCHANGED)
        # I valori vengono convertiti in float
        pred = pred.astype(float)
        # Viene estratto il canale alpha e viene ristretto nel range [0-1]
        pred = pred[:,:,3]/255
        # DeltaSx prende valore 0.18 se l'immagine predetta e‘ stata elaborata da DaVinci Resolve, 0.01 altrimenti
        if p == 'AlphaDaVinci':
            deltaSx = 0.18
        else:
            deltaSx = 0.01
        # Il canale alpha viene classificato
        pred = classifier(pred,deltaSx)
        # Viene calcolata la precisione
        precision =  sk.precision_score(true,pred,average='macro')
        # Viene calcolata la recall
        recall = sk.recall_score(true,pred,average='macro',zero_division='warn')
        # Viene calcolata l'accuratezza
```

```
42          accuracy = sk.accuracy_score(true,pred)
43          # Viene calcolata il punteggio F1
44          f1_score = sk.f1_score(true,pred,average='macro')
45          # Viene creata la matrice di confusione
46          cm = sk.confusion_matrix(true, pred)
47          # Vengono settati i parametri per la stampa della matrice di
    confusione
48          disp = sk.ConfusionMatrixDisplay(confusion_matrix=cm,
    display_labels=['B', 'F', 'E'])
49          # Viene stampata la matrice di confusione
50          disp.plot()
51          plt.tight_layout()
52          plt.title("Confusion matrix %s\n %s - %s" % (f,trueName,p))
53          plt.savefig(pathPlot'/%s_%s.pdf' % (f,trueName,p), bbox_inches
    ='tight', pad_inches=0.1)
54          # Vengono scritte le statistiche sul file
55          file.write("Confusion matrix di %s tra alpha%s e alpha%s\
    nprecision = %s\nrecall = %s\naccuracy = %s\nF1_score = %s \n\n" %
     (f,trueName,p,precision,recall,accuracy,f1_score))
56      file.write("\n")
57 # Il file viene chiuso
58 file.close()
```

Codice 4.2: Calcolo matrice di confusione e calcolo delle metriche di valutazione

4.1.2 Analisi qualitativa

Confronto visivo

Un altro metro di paragone utilizzato per la valutazione del tool presentato è il confronto visivo. Questo confronto consiste nel comporre l'immagine finale con un sfondo a scacchiera (bianco e rosso) , come quello della Figura 4.3, in modo da creare un forte contrasto con il soggetto, rendendo quindi le imperfezioni più visibili.

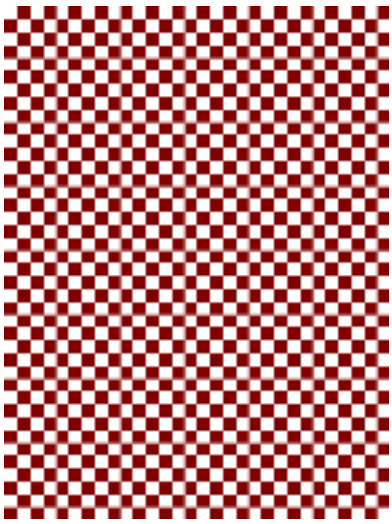

Figura 4.3: Sfondo a scacchiera

4.2 Score

Dai benchmark effettuati si sono potute determinare dei punteggi. Da cui si è potuta dare una valutazione al tool presentato.

4.2.1 Analisi quantitativa

Classificazione e confronto con matrici di confusione

Dal confronto fatto tra le immagini elaborate con la base di conoscenza è stato possibile ottenere alcuni risultati. Per il confronto e la rappresentazione dei dati sono state utilizzate due tipi di rappresentazione:

- Tramite una rappresentazione grafica dei risultati.
- Tramite l'utilizzo delle matrici di confusione.

La rappresentazione dei risultati mostra, per ogni immagine elaborata, utilizzando tre colori: il grigio, il bianco e il rosso, corrispettivamente le tre classi (B,F,E). Viene anche utilizzato un istogramma per indicare la densità di pixel per ogni valore che va da 0 a 1. Nella seguente rappresentazione vengono mostrate le varie elaborazioni dell'immagine "SimIo", con i corrispettivi istogrammi dei singoli canali alpha (Figura 4.4). Come è possibile notare, ogni metodo, adottato per l'elaborazione delle immagini, ha una strategia diversa per la divisione delle tre classi. In particolare si può notare, come detto in precedenza, che DaVinci Resolve, rispetto agli altri, non ha nessun valore vicino lo zero. Invece il risultato dell'elaborazione del metodo implementato non riesce a definire bene come il ground truth il contorno del soggetto (nell'immagine, in rosso).

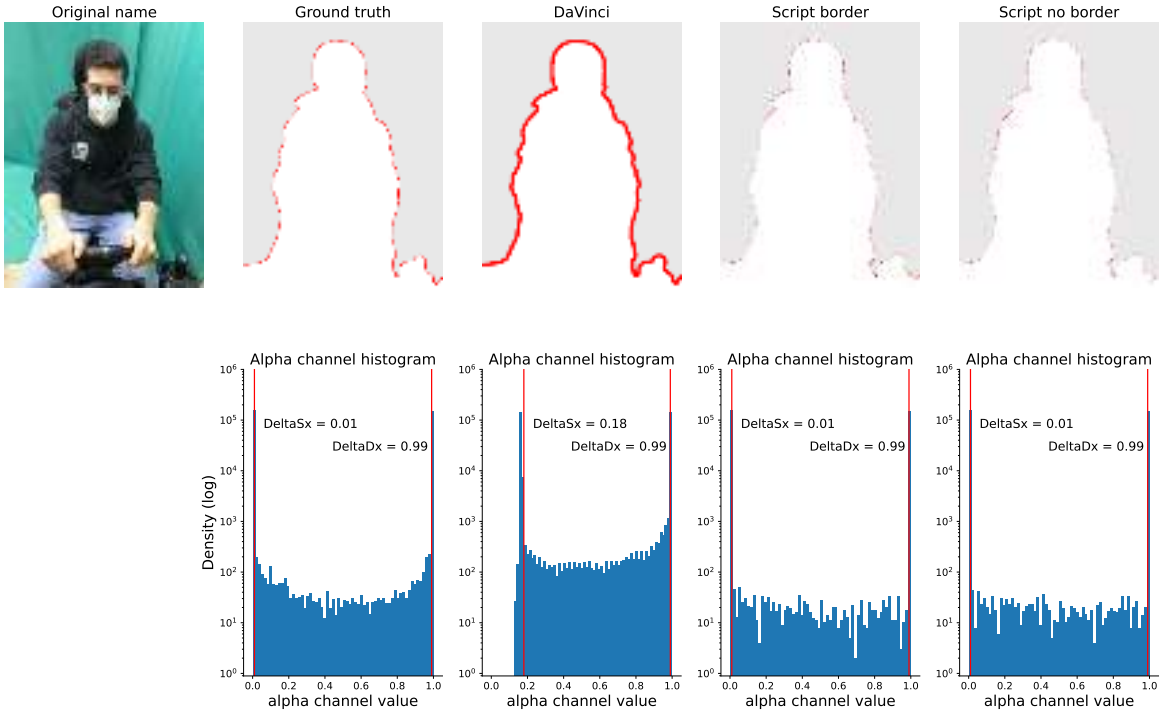

Figura 4.4: Rappresentazione grafica

La matrice di confusione, con sulle righe il ground truth (i valori reali) e sulle colonne l'immagine elaborata (i valori predetti), permette di rappresentare il numero di istanze che la classe predetta predice in modo corretto o meno rispetto alla classe reale. In questo modo sulla diagonale della matrice vanno a posizionarsi tutte le istanze predette correttamente. Successivamente vengono mostrate le matrici di confusione (Figura 4.5, Figura 4.6 e Figura 4.7) ottenute dal confronto tra le immagini elaborate del SimIo e l'immagine elaborata da Photoshop. Esse possiedono tre label, corrispettivamente:

- B (Background): equivale alla classe 0 spiegata precedentemente.
- F (Foreground): equivale alla classe 1 spiegata precedentemente.
- E (Edge): equivale alla classe 2 spiegata precedentemente.

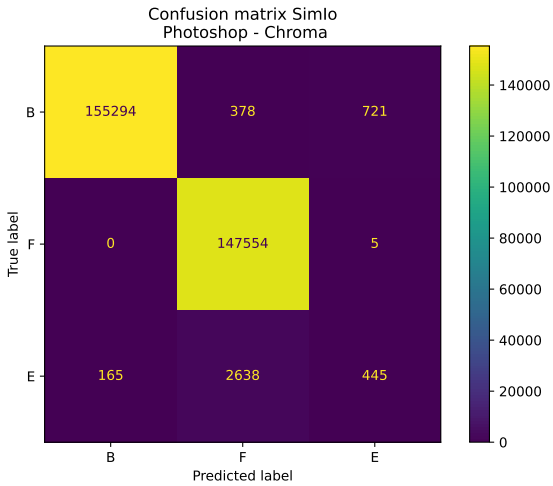

Figura 4.5: Confronto Photoshop - Chroma

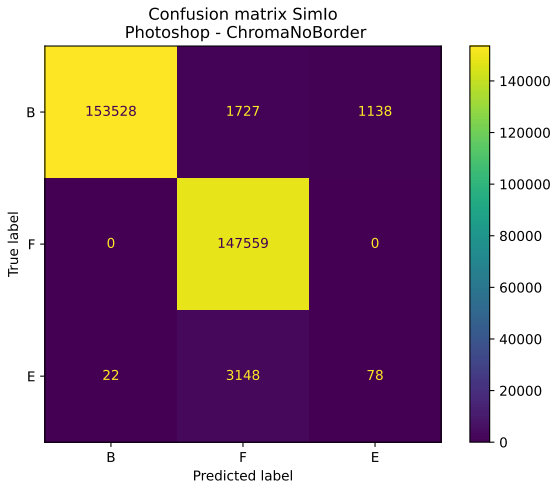

Figura 4.6: Confronto Photoshop - Chroma senza l'erosione dei bordi

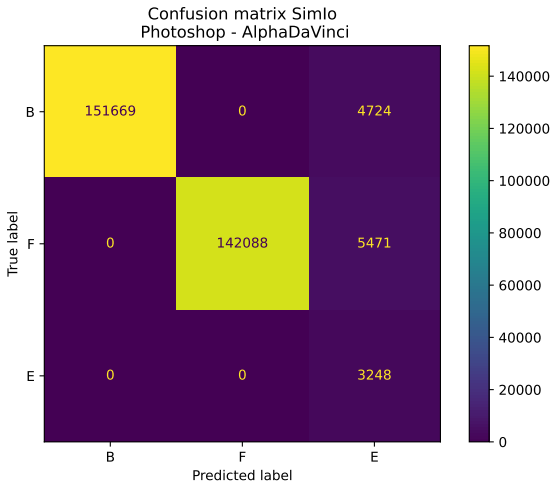

Figura 4.7: Confronto Photoshop - DaVinciResolve

Come detto in precedenza le matrici di confusione permettono di calcolare numerose metriche di valutazione. Le metriche di valutazione che sono state ritenute valide per questa valutazione sono:

- L'accuratezza: un valore compreso tra 0 e 1. È ottenuto dalla somma delle classificazioni corrette diviso il numero totale dei casi.
- La precisione: un valore compreso tra 0 e 1 che quantifica la bontà del classificatore nelle attribuzioni positive. È ottenuto dividendo le previsioni positive corrette sul totale delle previsioni positive date dai valori predetti.
- La recall: un valore compreso tra 0 e 1 che indica la capacità del classificatore di trovare tutte le istanze positive. È ottenuto divindendo le previsioni positive corrette sul totale delle istanze positive.
- Il punteggio F1: un valore compreso tra 0 e 1 che quantifica il test di precisione. È ottenuto facendo la media armonica tra la precisione e il recall.

		Classi predette: Si	Classi predette: No
Classi reali	Si	True Positive (TP)	False Negative (TN)
	No	False Positive (FP)	True Negative (TN)

$$Accuratezza = \frac{TP + TN}{TP + TN + FP + FN}$$

$$Precisione = \frac{TP}{TP + FP}$$

$$Recall = \frac{TP}{TP + FN}$$

$$F1 = \frac{2 \times Recall \times Precision}{Recall + Precision}$$

Figura 4.8: Formule metriche di valutazione con matrice 2x2

Di seguito vengono riportati i grafici rappresentanti le metriche di valutazione per ogni immagine elaborata.

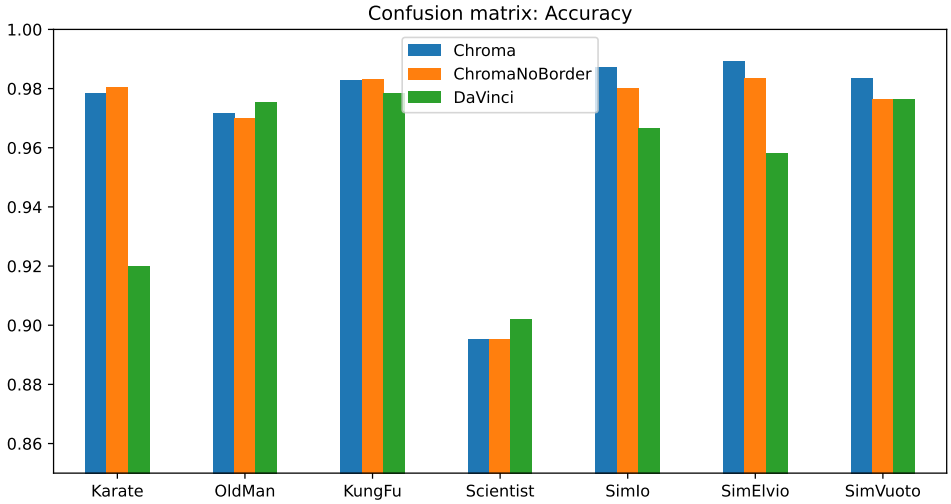

Figura 4.9: Accuratezza

Dal grafico dell'accuratezza (Figura 4.9) si può notare che in media si ha un'elaborazione accurata per tutti e tre i metodi di elaborazione usati.

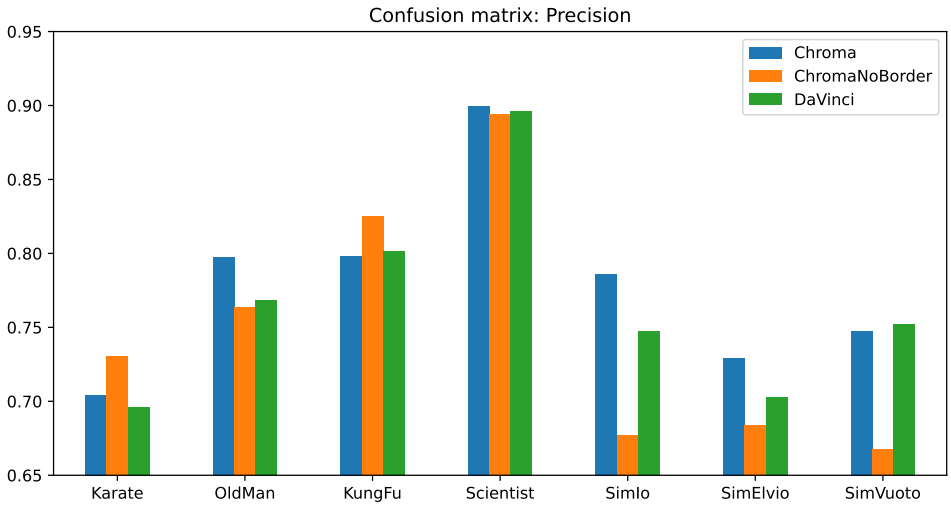

Figura 4.10: Precisione

Dal grafico della precisione (Figura 4.10) si può notare che: nell'elaborazione con il tool senza l'applicazione dell'erosione si ha una carenza di precisione nel "setup Simulatore" diversamente, si riesce ad avere una buona precisione nel "setup Articolo" [7].

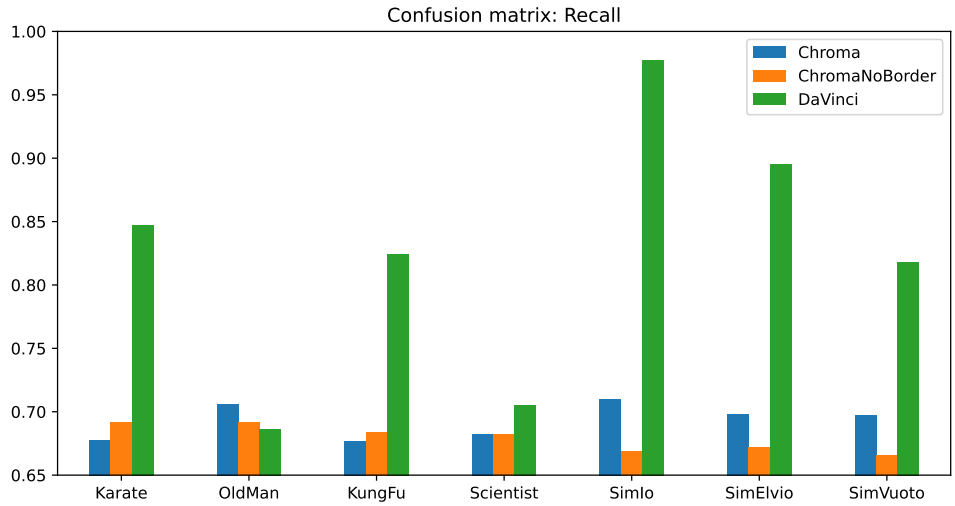

Figura 4.11: Recall

Dal grafico della recall (Figura 4.11) è possibile notare che solamente le immagini elaborate da DaVinci Resolve hanno un valore di recall molto alto. Questo sta ad indicare che è stato in grado di predire correttamente quasi la totalità delle istanze positive.

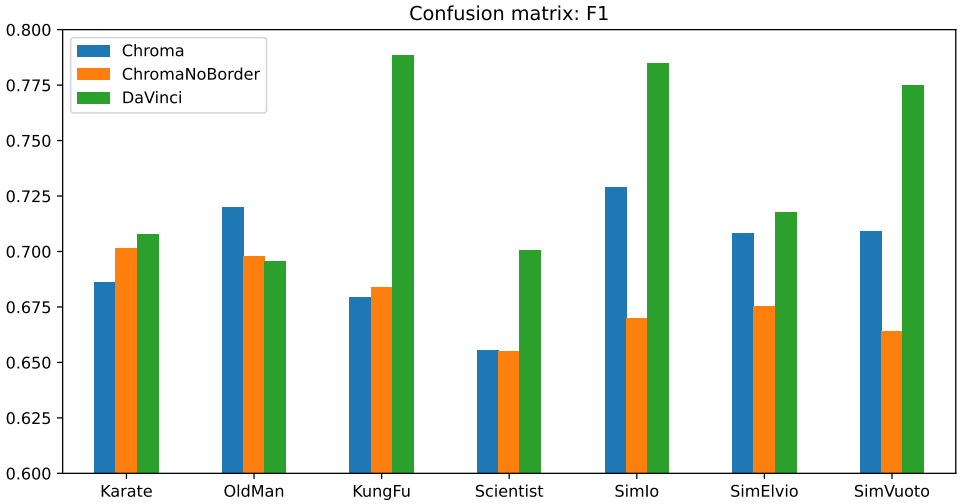

Figura 4.12: Punteggio F1

In conclusione, osservando la Figura 4.12, che con un valore alto indica una precisione e una recall perfetti, si può dedurre che quantitativamente le immagini elaborate da DaVinci Resolve sono quelle che si avvicinano di più alla base di conoscenza che è stata assunta come perfetta.

4.2.2 Analisi qualitativa

Confronto visivo

Il confronto è stato effettuato sia con le immagini del "setup Simulatore" sia con le immagini del "setup Articolo". Anche se le immagini del "setup Articolo" sono in un contesto differente da quello per cui è stato ideato il tool, sono state anch'esse utilizzate per l'analisi visiva, cosi da consentire di valutare la generalità del tool. Il confronto visivo ha permesso di notare le piccole imperfezioni commesse da ciascun metodo di elaborazione. Questo tipo di analisi e molto soggettiva, questo perché la valutazione avviene attraverso l'osservazione delle diverse immagini. A seguire due gruppi di immagini composti con lo sfondo a scacchiera.
Dalle quattro immagini del "SimIo", mostrate nella Figura 4.13, nella figura 4.14 e nella Figura 4.15, si può notare che:

- In tutte le immagini non c'è nessuna perdita di pixel all'interno del soggetto.
- Entrambe le immagini elaborate dal tool, rispetto all'immagine elaborata da DaVinci Resolve presentano un bordo leggermente più frastagliato, questo perché DaVinci permette di effettuare un erosione più accentuata, ma con una leggera perdita di dettaglio, sul bordo.
- L'immagine del tool in cui è stata applicata l'erosione del bordo, rispetto all'immagine del tool in cui non è stata applicata, riesce ad apparire leggermente più realistica. Questo perché grazie all'erosione, si riescono a "nascondere" le piccole parti dello sfondo che non sono state rimosse perfettamente. Per esempio la mano destra del soggetto nella Figura 4.14 rispetto alla Figura 4.13 (Figura 4.17), presenta un leggero alone verde, che è riconducibile ad una non perfetta rimozione del green screen in quella porzione di immagine.

Figura 4.13: Tool

Figura 4.14: Tool senza erosione

Figura 4.15: DaVinci Resolve

Figura 4.16: Ground Truth

Figura 4.17: Particolare mano

Passando ad un contesto diverso da quello per cui è stato sviluppato il tool, dalle tre immagini del "Scientist" [7], mostrate nella Figura 4.18, nella figura 4.19 e nella Figura 4.20, si può notare che:

- Entrambe le immagini del tool, rispetto all'immagine elaborata da DaVinci Resolve, presentano un contorno meno preciso. Per esempio, si può notare che, i capelli del soggetto, nella Figura 4.20 rispetto alla Figura 4.18 e alla Figura 4.19, vengono gestiti in modo migliore dall'algoritmo di DaVinci Resolve, che riesce ad isolarli ottimamente, ma al contempo senza lasciare nessun alone di verde (Figura 4.22).

- Un altra differenza che si può notare è la gestione della porzione di immagine in cui è contenuta la lente scheggiata degli occhiali. L'algoritmo del tool rende la lente poco trasparente facendo cosi notare la condizione della lente. L'algoritmo di DaVinci Resolve invece, una porzione, la rende completamente trasparente (Figura 4.22).

- In tutte le immagini non c'è nessuna perdita di pixel all'interno del soggetto.

Figura 4.18: Tool

Figura 4.19: Tool senza erosione

Figura 4.20: DaVinci Resolve

Figura 4.21: Ground truth

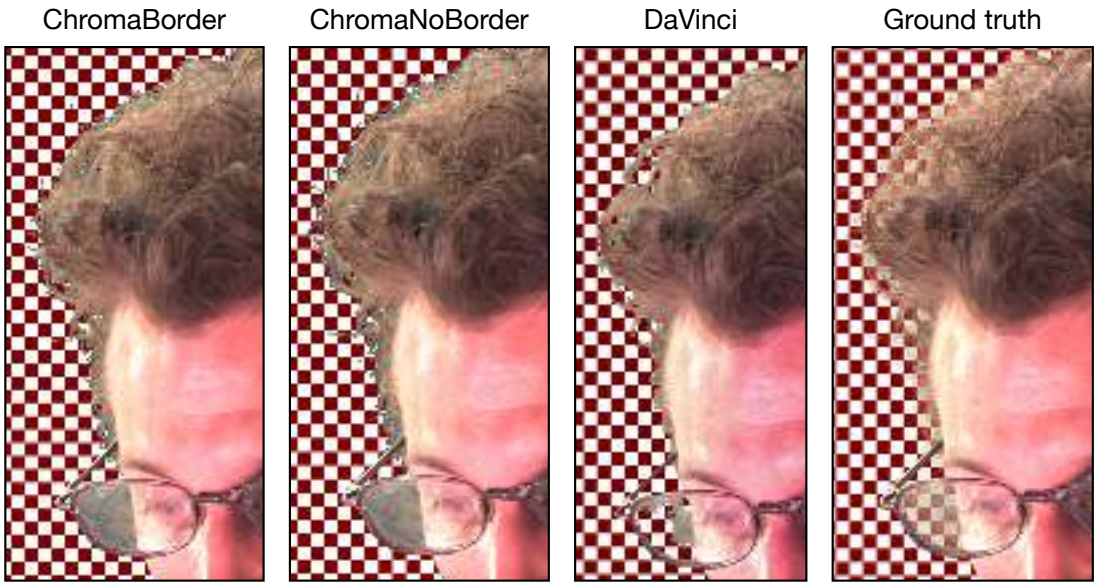

Figura 4.22: Particolare occhiali e capelli

Come si può notare nella Figura 4.4 l'immagine elaborata con DaVinci presenta un bordo molto spesso. Si è quindi voluto fare un confronto aggiuntivo tra l'immagine elaborata da DaVinci e l'immagine elaborata dal tool con un'erosione del bordo molto alta, così da simulare il lavoro fatto da DaVinci.

Figura 4.23: DaVinci Resolve

Figura 4.24: Tool con erosione alta

Figura 4.25: DaVinci Resolve

Figura 4.26: Tool con erosione alta

Sebbene le piccole differenze sopra riportate, dall'analisi qualitativa si può evincere che, le immagini dei due gruppi sono tra loro molto simili e in un contesto video si

possono considerare indistinguibili. Ciò evidenzia la bontà del tool sviluppato, che riesce a produrre risultati paragonabili a quelli prodotti da un software affermato come DaVinci Resolve anche su immagini appartenenti ad un contesto diverso.

4.3 Tool finale

Il tool sviluppato riesce ha soddisfare i requisiti che si erano imposti. Permette di sostituire lo sfondo verde da delle immagini di un guidatore in un simulatore. Tutto ciò permette quindi di attuare una dataset augumentation in modo semi-automatico rendendo il processo molto più efficiente. Il tool non si è limitato alla sola applicazione della tecnica del chromakey, ad esso sono state anche aggiunte due tecniche complementari: Il despill e la ricalibrazione del colore. L'aggiunta di queste due tecniche permettono di avere un risultato finale il più realistico possibile e di una migliore qualità.

4.3.1 Organizzazione

Il tool è stato organizzato in questo modo:

```
Programma
├── Calibrator
│   ├── Input
│   │   ├── Background
│   │   └── Foreground
│   └── OutPut
├── TransferCalibrator
│   ├── Input
│   └── Output
├── Transfer
│   └── OutPut
├── ChromaDespill
│   ├── Input
│   │   ├── Background
│   │   └── Video
│   └── OutPut
├── calibrator.py
├── transfer_calibrator.py
├── transfer.py
├── chromadespill.py
├── notebook.ipynb
└── ReadMe.txt
```

La cartella `Calibrator` che include:

- La cartella `Input`, da cui lo script `calibrator.py` acquisisce i file per l'estrazione della palette, comprende due cartelle:
  - La cartella `Background`, che conterrà le immagini del solo background dell'immagine su cui applicare il tool.
  - La cartella `Image`, che conterrà le immagini a cui si deve applicare il tool.
- La cartella `Output`, in cui verrà salvata la palette calcolata dallo script.

La cartella `TransferCalibrator` che include:

- La cartella `Input`, da cui lo script `transfer_calibrator.py` acquisisce i file da cui calcolare le statistiche, che conterrà due immagini, l'immagine sorgente e l'immagine target a cui è stato rimosso lo sfondo.
- La cartella `Output`, in cui verranno salvati i due file di testo contenenti le statistiche dell'immagine sorgente e dell'immagine target.

La cartella `Transfer` che include:

- La cartella `Output`, in cui verrà salvata l'immagine con applicata la ricalibrazione del colore.

La cartella `ChromaDespill` che include:

- La cartella `Input`, da cui lo script `chromadespill.py` acquisisce i file da elaborare, comprende due cartelle:
  - La cartella `Background`, che conterrà l'immagine da utilizzare come nuovo sfondo.
  - La cartella `Video`, che conterrà il video a cui si deve sostituire il green screen con il nuovo sfondo.
- La cartella `Output`, in cui verrà salvato il video elaborato.

Il file `calibrator.py`: lo script che permette di ottenere la palette di colori.

Il file `transfer_calibrator.py`: lo script che permette di calcolare le statistiche per l'applicazione della ricalibrazione del colore

Il file `transfer.py`: lo script che consente di applicare, grazie alle statistiche calcolate dallo script `transfer_calibrator.py`, la ricalibrazione del colore.

Il file `chromadespill.py`: lo script che consente di applicare, sul video preso in input, la pipeline presentata in questa tesi.

Il file `notebook.ipynb`: il quale permette, in modo interattivo, di calibrare alcuni parametri del chromakey e della ricalibrazione colore, oltre ad illustrare alcuni grafici utili.

Il file `ReadMe.txt` che spiega come utilizzare il tool.

4.3.2 Utilizzo

L'utilizzo del tool avviene nel seguente modo: Il primo passo è la popolazione delle cartelle di input con i file richiesti. Fatto questo si può passare quindi alla calibrazione della palette lanciando lo script `calibrator.py`. Finita l'esecuzione dello script, e ottenuta quindi la palette, si procede con la calibrazione dei parametri del chromakey e della ricalibrazione del colore sul notebook `notebook.ipynb`, nel seguente modo:

- La calibrazione dei parametri del chromakey avviene come mostrato nella Figura 3.12
- La calibrazione dei parametri della ricalibrazione del colore avviene come mostrato nella Figura 3.13

Infine, avvenuta la calibrazione, si procede con il lanciare lo script `chromadespill.py` che, preso in input il video e il nuovo sfondo, applica la pipeline che è stata implementa e restituisce il video elaborato. Il tool è disponibile al seguente GitHub "https://github.com/Lello10/ChromaDespill".

Capitolo 5

Conclusioni

In conclusione il tool presentato è riuscito raggiungere pienamente gli obiettivi richiesti. Riuscendo a effettuare una sostituire dello sfondo verde da delle immagini di un guidatore in un simulatore. Consentendo quindi di effettuare in modo molto efficiente il processo di dataset augumetation in modo semi-automatico, senza perdita di qualità. Il tool grazie anche alle tecniche complementari di cui è stato fornito: l'erosione dei bordi, la tecnica del despill e infine la ricalibrazione del colore, è riuscito a spingersi anche un po' oltre le capacità richieste. Come si è potuto evincere dalle precedenti analisi, il tool, anche con delle immagini appartenenti a dei setup differenti, riesce a produrre un risultato finale di buona qualità, comparabile con dei software affermati come DaVinci Resolve. Il tool potrebbe inoltre, grazie a dei possibili lavori futuri, aumentare ancora di più la qualità delle immagini elaborate, la facilità di utilizzo da parte di un utente e la velocità di elaborazione. Queste migliorie potrebbero essere attuate attraverso l'inserimento di un'interfaccia utente, dando la possibilità all'utente di selezionare a schermo i colori che devono far parte della palette di rimozione dello sfondo, l'inserimento di una palette di colori da non rimuovere, l'ottimizzazione degli algoritmi in modo da renderli più veloci e tante altre funzionalità che permetterebbero al tool di fare un salto di qualità.

Bibliografia

[1] "Davinci resolve." `https://www.blackmagicdesign.com/it/products/davinciresolve`. Accessed: 2022-06-26.

[2] Y. Aksoy, T. O. Aydın, M. Pollefeys, and A. Smolić, "Interactive High-Quality Green-Screen Keying via Color Unmixing," *ACM Trans. Graph.*, vol. 35, no. 5, pp. 152:1–152:12, 2016.

[3] G. Bradski, "The OpenCV Library," *Dr. Dobb's Journal of Software Tools*, 2000.

[4] C. R. Harris, K. J. Millman, S. J. van der Walt, R. Gommers, P. Virtanen, D. Cournapeau, E. Wieser, J. Taylor, S. Berg, N. J. Smith, R. Kern, M. Picus, S. Hoyer, M. H. van Kerkwijk, M. Brett, A. Haldane, J. F. del Río, M. Wiebe, P. Peterson, P. Gérard-Marchant, K. Sheppard, T. Reddy, W. Weckesser, H. Abbasi, C. Gohlke, and T. E. Oliphant, "Array programming with NumPy," *Nature*, vol. 585, pp. 357–362, Sept. 2020.

[5] J. D. Hunter, "Matplotlib: A 2d graphics environment," *Computing in Science & Engineering*, vol. 9, no. 3, pp. 90–95, 2007.

[6] A. Rosebrock, "Jrosebr1/color_transfer: Performs color transfer between two images. loosely based on the "color transfer between images" paper by reinhard et al., 2001.."

[7] "Dataset dell'articolo *Interactive High-Quality Green-Screen Keying via Color Unmixing*." `http://yaksoy.github.io/keying/`. Accessed: 2022-06-27.

[8] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

Un grazie speciale al mio relatore Amparore Elvio, per la sua disponibilità e per avermi guidato, con immensa pazienza, negli ultimi mesi del mio percorso universitario.

Un ringraziamento speciale va a i miei genitori, che durante il mio percorso di studi mi hanno sempre sostenuto e supportato. Permettendomi di raggiungere questo importante traguardo.

Un grazie di cuore va ai miei fratelli, che in questi anni, lontano da casa, sono sempre stati disponibili e non mi hanno mai fatto mancare il loro appoggio.

Un ringraziamento particolare va alla mia ragazza, che mi è sempre stata vicina, dimostrandomi sempre il suo sostegno e mostrandomi il suo supporto nei momenti tristi e nei momenti felici, nonostante la distanza che ci ha diviso.

Per ultimi, ma non per importanza, un grazie speciale va ai miei colleghi, coinquilini e soprattutto amici, Alessandro e Filippo, con cui ho affrontato e condiviso questi tre anni. Senza loro non sarebbe stato lo stesso.