Alma Mater Studiorum – Università di Bologna

SCUOLA DI INGEGNERIA E ARCHITETTURA
DIPARTIMENTO DI INFORMATICA – SCIENZA E INGEGNERIA
CORSO DI LAUREA IN INGEGNERIA INFORMATICA

TESI DI LAUREA
In
Calcolatori Elettronici – T

Oscuramento volti in sequenze video con deep-learning.

CANDIDATO:

Francesco Luzzi

Relatore:

Prof. Stefano Mattoccia

Co-relatori:

Alessio Mingozzi
Filippo Aleotti
Matteo Poggi

Sessione
Anno Accademico 2020/21

Sommario

1 Introduzione

Oggi giorno, tecnologie come navigatori e guida autonoma richiedono una grande affluenza di dati in particolare di immagini stradali e di luoghi pubblici in cui sono presenti molte persone.
È fondamentale nascondere i volti dei soggetti immortalati nelle immagini in quanto si può incorrere in sanzioni legali ed economiche sostanziose per violazione della privacy *(Google Street View ha violato la privacy. Scatta la multa da un milione di euro - Rai News)*.

Alla base di questo progetto, quindi, vi è la necessità di mascherare automaticamente l'identità delle persone presenti all'interno di un qualsiasi inquadratura.
Dopo l'approfondimento del problema, ho sviluppato un'applicazione che sfrutta una rete neurale per determinare la posa di ogni soggetto identificato nel fotogramma.

A seguito del riconoscimento delle diverse posture, il programma si focalizza sulla posa del volto con lo scopo di nasconderlo.
La scelta di sfruttare l'intelligenza artificiale è legata al mio interesse personale per questo nuovo ambito informatico e alla grande probabilità che questa sia la direzione futura della tecnologia.

2 Strumenti utilizzati

Il primo passo verso la risoluzione del problema è stato cercare gli strumenti giusti per il lavoro da intraprendere, le scelte sono ricadute su:

2.1 Anaconda

Anaconda è un package manager, un manager di ambienti virtuali, supporta multiple versioni di Python/R ed è improntata alla data-science, grazie ad una grande collezione di pacchetti open-source.

Anaconda è open-source ed è stata una scelta quasi ovvia nel mio caso: la gestione delle versioni e dipendenze dei diversi pacchetti, rispetto alla versione di python utilizzata, è molto comoda sia dal punto di vista dell'automazione sia perché permette di non intaccare il resto del sistema operativo.

2.2 CMake

CMake è un sistema open-source estendibile che gestisce il processo di compilazione in un Sistema Operativo, in modo indipendente da altri compilatori.
Differentemente da altri sistemi cross-platform, CMake viene utilizzato in combinazione all'ambiente di compilazione nativo per: compilare codice, creare librerie, wrappers, eseguibili, potendo anche sfruttare librerie dinamiche e statiche.
CMake è progettato per supportare gerarchie di directory complesse e applicazioni dipendenti da diverse librerie *(Overview | CMake)*.

2.3 OpenPose

OpenPose è un sistema open-source e real-time per la stima delle pose di multiple persone in un frame sia in 2D che in 3D;
questo processo si basa su diverse reti neurali, generate grazie al deep learning, che si differenziano tra loro per formato di output o per l'elaborazione dell'input.

Queste reti neurali accettano come formato di input il "blob" (Binary Large Object, definizione tratta da *What exactly is a Blob in OpenCV ? - OpenCV Q&A Forum*), infatti, queste sono state generate grazie a "Caffe", un framework specifico per i deep learning (per approfondire *Caffe | Deep Learning Framework*).

Per spiegare l'output di tali reti neurali, è necessario introdurre tre concetti fondamentali:

- Mappe di probabilità, ossia matrici bidimensionali che codificano ad ogni pixel la probabilità di rappresentare ciò che viene ricercato (keypoints o PAF);
- Keypoints, ovvero le diverse parti del corpo che vengono identificate dalla rete neurale;
- PAF (Part Affinities) sono mappe di probabilità che codificano il grado di relazione tra diversi keypoints.

L'output di queste reti neurali è un array di matrici che rappresentano le diverse mappe di probabilità dei diversi "keypoints" rilevati e divise per tipologia, ed i PAF che saranno poi utili successivamente.

Fra le possibili reti neurali prodotte da OpenPose, la scelta è ricaduta su quella più leggera dal punto di vista computazionale e dall'output più limitato, basata su un formato di output dei keypoints MPI con 15 keypoints e le relative PAF (immagine sotto).
La motivazione della scelta è dovuta all'ambito di utilizzo di questa applicazione, principalmente ideata per sistemi mobili o di limitata potenza computazionale.

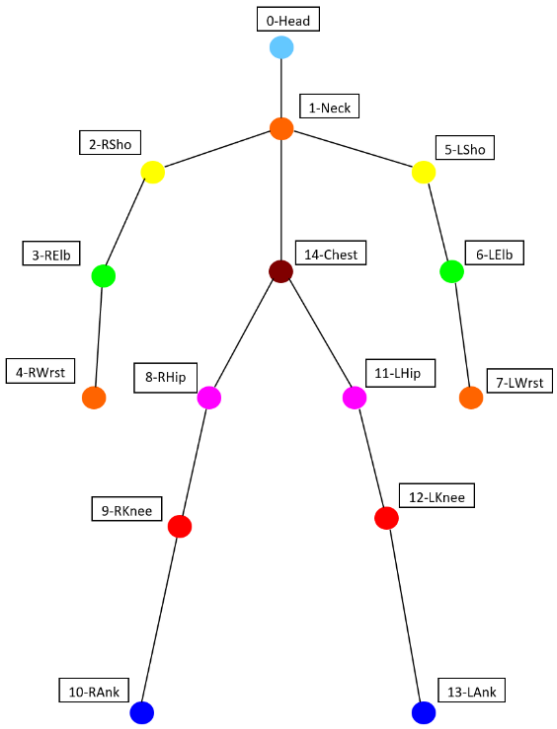


Immagine: mpiKeypoints.jpg

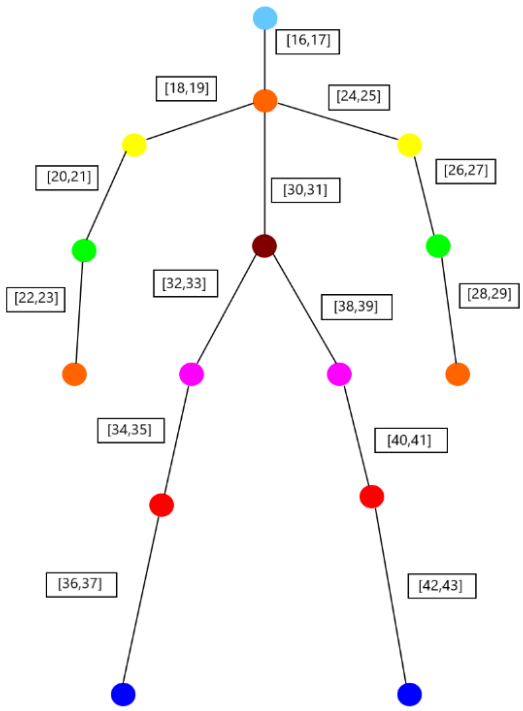


Immagine: mpiPAFS.jpg

2.4 OpenCv

Libreria open-source molto presente nel panorama della computer vision, sviluppata in C++ e resa disponibile tramite un wrapper anche su python.
L'implementazione da me utilizzata è leggermente differente: è stata compilata con CMake utilizzando il source code "originale" di opencv ed il codice della versione "contrib" *(OpenCV (github.com))*.
La modificazione serve per abilitare/sfruttare l'accelerazione GPU data dai drivers CUDA di NVIDIA™; difatti le GPU sono molto più efficienti per questo tipo di workload algebrico/matriciale, rispetto alle CPU.

2.5 NumPy

Libreria OpenSource indispensabile per fare operazioni algebriche in python con overhead e durata dell'esecuzione minori.

In pratica, è una libreria in C con un wrapper per python, molto veloce e performante in quanto il codice utilizzato è precedentemente compilato e non interpretato.

3 Sviluppo

Si elencano le variabili più importanti dell'applicazione.

```
        self.threshold = 0.10
        self.paf_score_th = 0.12
        self.conf_th = 0.8
        self.paf_interp_samples = 10
```

Queste variabili determinano i valori di threshold per differenti scopi:

- threshold, ossia la soglia minima di rilevazione dei keypoints;
- paf_score_th, ovvero la soglia minima di accettazione del paf_score relativo ad un punto;
- conf_th, cioè la percentuale minima di punti con un paf_score maggiore di paf_score_th all'interno di una connessione.
- paf_interp_samples, che rappresenta il numero di punti estrapolati dalla connessione tra due keypoints.

```
        self.paf_idx = [
            [16, 17],
            [18, 19],
            [20, 21],
            [22, 23],
            [24, 25],
            [26, 27],
            [28, 29],
            [30, 31],
            [32, 33],
            [34, 35],
            [36, 37],
            [38, 39],
            [40, 41],
            [42, 43],
```

```
        ]
        self.POSE_PAIRS = [
            [0, 1],
            [1, 2],
            [2, 3],
            [3, 4],
            [1, 5],
            [5, 6],
            [6, 7],
            [1, 14],
            [14, 8],
            [8, 9],
            [9, 10],
            [14, 11],
            [11, 12],
            [12, 13],
        ]
```

Mappatura delle diverse coppie di keypoints e dei relativi PAF presenti nell'output della rete neurale ed esposti in precedenza con le due immagini.

3.1 Input

Immagini utilizzate per il testing e la calibrazione dell'applicazione.

Immagine: Group.jpg

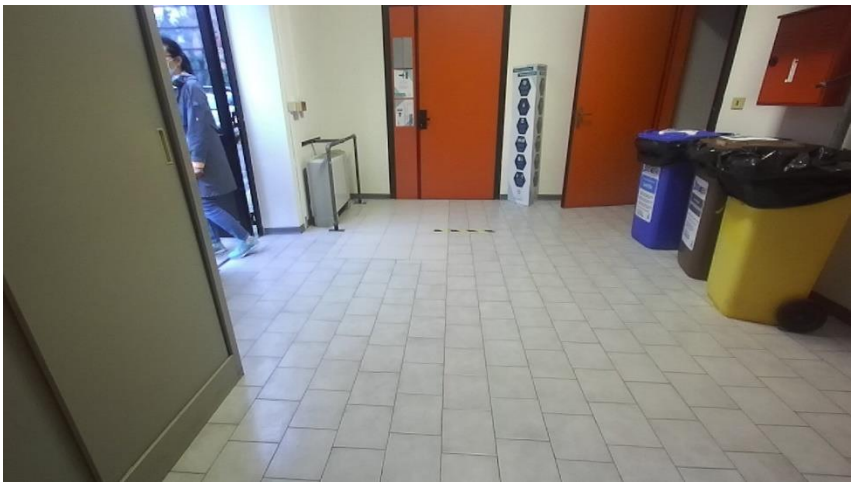


Immagine: 000586.jpg

Immagine: 000443.jpg

Immagine: 000352.jpg

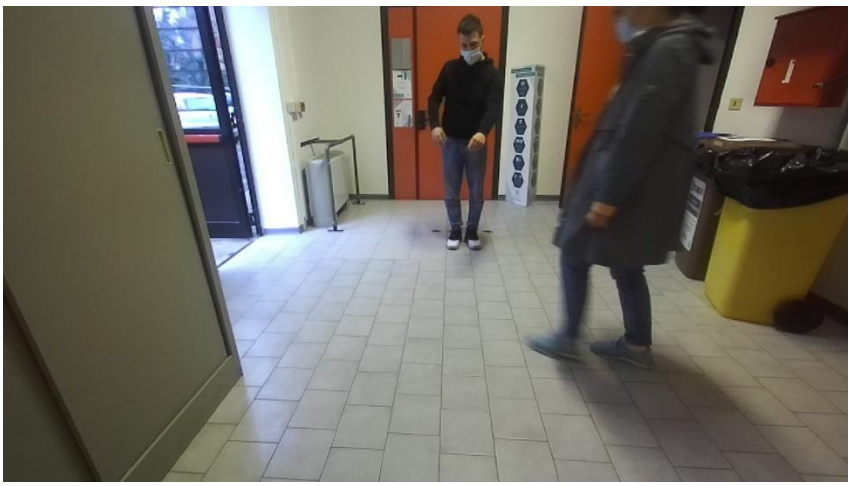


Immagine: 000286.jpg

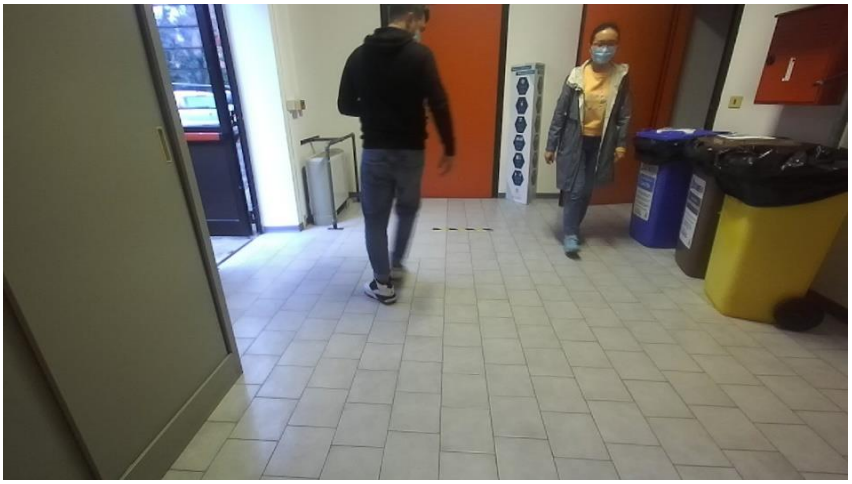


Immagine: 000265.jpg

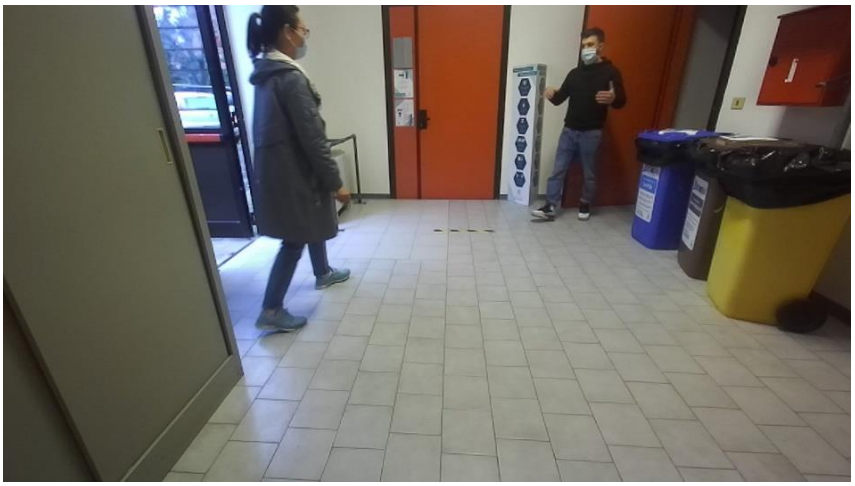


Immagine: 000244.jpg

Immagine: 000173.jpg

Immagine: 000501.jpg

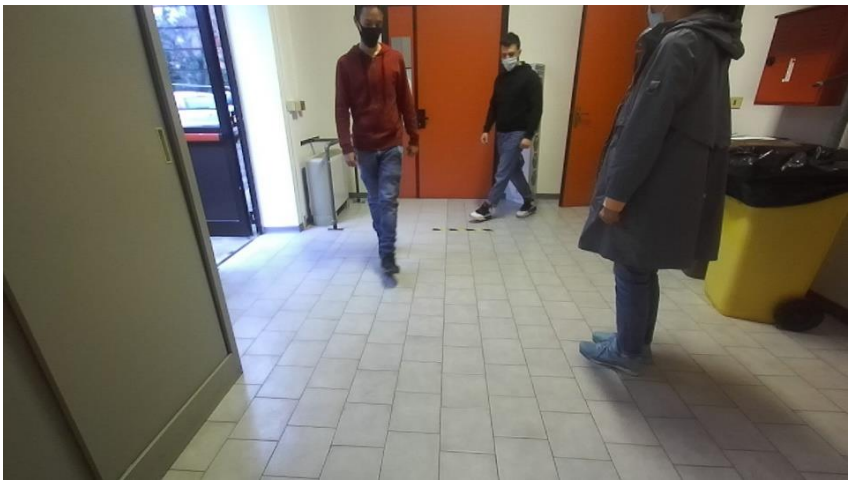


Immagine: 000555.jpg

3.2 Logica

Qual è il concetto principale dietro questa rete neurale? La probabilità.
Questa rete non restituisce in output un valore certo in cui trovare uno specifico keypoint, ma una mappa di probabilità relativo ad esso.

Per spiegare meglio questo concetto viene mostrato nell'immagine il grafico di probabilità dei keypoints della testa in relazione all'input: l'asse Z rappresenta la probabilità ∈ [0,1] del specifico punto di essere il keypoint cercato; dove 1 è la certezza mentre 0 è la completa impossibilità.

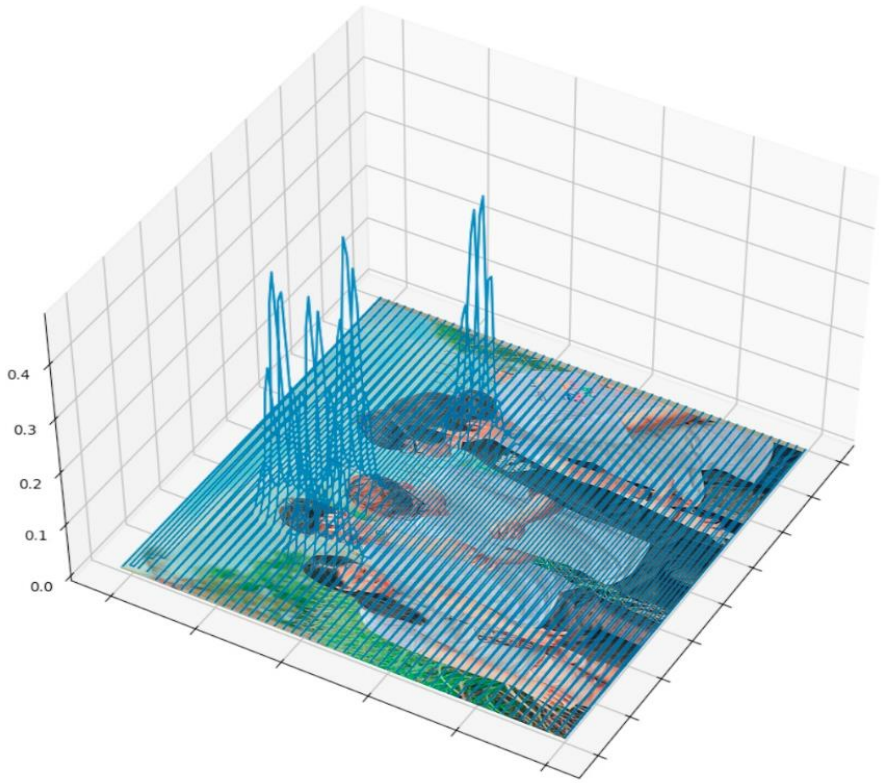


Immagine: probMapHeadGroup.jpg

```
def get_keypoints(self, prob_map):
        map_smooth = cv.GaussianBlur(prob_map, (3, 3), sigmaX=0, sigmaY=0)
        map_mask = np.uint8(map_smooth > self.threshold)
        keypoints = []
        contours, _ = cv.findContours(map_mask, cv.RETR_TREE,
cv.CHAIN_APPROX_SIMPLE)
        for cnt in contours:
            blob_mask = np.zeros(map_mask.shape)
            blob_mask = cv.fillConvexPoly(blob_mask, cnt, 1)
            masked_prob_map = map_smooth * blob_mask
            _, _, _, max_loc = cv.minMaxLoc(masked_prob_map)
            keypoints.append(max_loc + (prob_map[max_loc[1], max_loc[0]],))
        return keypoints
```

Questa parte di codice è adibita a trovare i diversi keypoints nei frame;
utilizzando le mappe di probabilità, si seleziona il valore massimo di ogni raggruppamento di valori con probabilità al di sopra della variabile di threshold, questi verranno poi salvati come keypoints.

Determinati i keypoints, il passo successivo è collegarli tra di loro, ma come?

La prima possibilità è quella di utilizzare un algoritmo che, analizzando la distanza tra due keypoints consecutivi, metta in relazione la coppia di keypoints con la distanza minore tra loro.

Prendiamo come esempio l'immagine successiva:

Immagine: shortesDistance.jpg

Analizzando questa immagine, si può notare chiaramente il limite dell'approccio presentato:
se due persone sono abbastanza vicine o sovrapposte tra loro, si rischia di collegare il keypoint di una persona all'individuo sbagliato vicino ad esso.

Per superare questo limite, entrano in gioco le mappe PAF introdotte precedentemente.
Codificando il grado di relazione tra diversi keypoints, i PAF, sono lo strumento studiato da OpenPose per collegare sempre in modo corretto una coppia di keypoints.

Qui si mostra una heatmap che dimostra graficamente il comportamento dei PAF;
in rosso valori di probabilità minori, in giallo invece valori più alti.

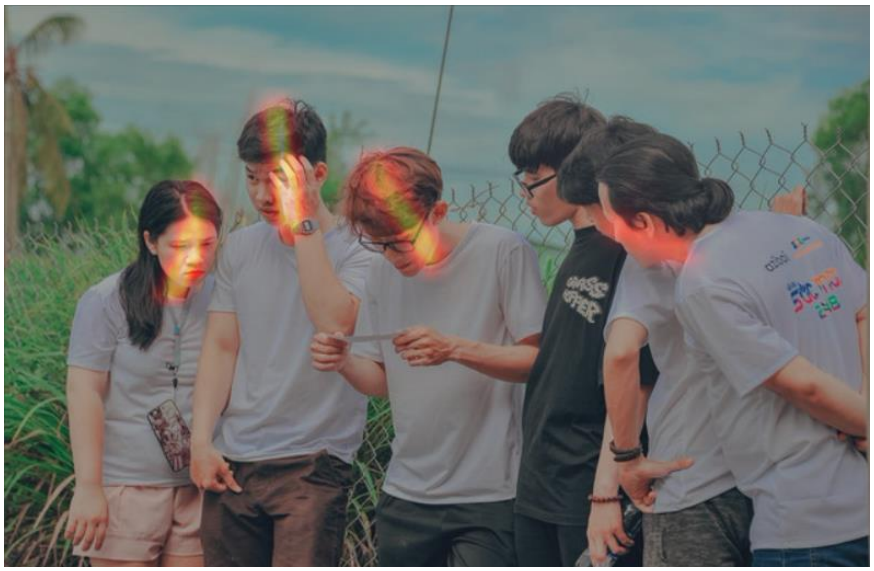


Immagine: explanationPAF.jpg

PAF è stata studiata per essere utilizzata in combinazione con il vettore unitario che indica la direzione tra i due keypoints:

```
def get_valid_pairs(self, output):
        valid_pairs = []
        invalid_pairs = []
        for k in range(self.n_points - 1):
            paf_a = output[0, self.paf_idx[k][0], :, :]
            paf_b = output[0, self.paf_idx[k][1], :, :]
            paf_a = cv.resize(paf_a, (self.frame_width, self.frame_height))
            paf_b = cv.resize(paf_b, (self.frame_width, self.frame_height))
            cand_a = self.detected_keypoints[self.POSE_PAIRS[k][0]]
            cand_b = self.detected_keypoints[self.POSE_PAIRS[k][1]]
            n_a = len(cand_a)
            n_b = len(cand_b)
            if n_a != 0 and n_b != 0:
                valid_pair = np.zeros((0, 3))
                for i in range(n_a):
                    max_j = -1
                    max_score = -1
                    found = 0
                    for j in range(n_b):
                        d_ij = np.subtract(cand_b[j][:2], cand_a[i][:2])
                        norm = np.linalg.norm(d_ij)
                        if norm:
                            d_ij = d_ij / norm
                        else:
                            continue
                        interp_coord = list(
                            zip(
                                np.linspace(
                                    cand_a[i][0],
                                    cand_b[j][0],
                                    num=self.n_interp_samples,
                                ),
                                np.linspace(
                                    cand_a[i][1],
                                    cand_b[j][1],
                                    num=self.n_interp_samples,
                                ),
                            )
                        )
                        paf_interp = []
                        for k in range(self.n_interp_samples):
                            paf_interp.append(
                                [
                                    paf_a[
                                        int(round(interp_coord[k][1])),
                                        int(round(interp_coord[k][0])),
                                    ],
                                    paf_b[
                                        int(round(interp_coord[k][1])),
```

```
                                    int(round(interp_coord[k][0])),
                                ],
                            ]
                        )
                    paf_scores = np.dot(paf_interp, d_ij)
                    # Find avg_PAF_score
                    avg_paf_score = sum(paf_scores) / len(paf_scores)
                        len(np.where(paf_scores > self.paf_score_th)[0])
                        / self.n_interp_samples
                    ) > self.conf_th and avg_paf_score > max_score:
                        max_j = j
                        max_score = avg_paf_score
                        found = 1
                    valid_pair = np.append(
                        valid_pair,
                        [[cand_a[i][3], cand_b[max_j][3], max_score]],
                        axis=0,
                    )
            valid_pairs.append(valid_pair)
        else:
            invalid_pairs.append(k)
            valid_pairs.append([])
    return valid_pairs, invalid_pairs
```

Iterando su tutte le coppie di paf_idx, definite in precedenza, si prendono i PAF di 2 keypoints, denominati paf_a e paf_b, che sono estremi di una connessione, chiamata ramo.

Per ogni coppia bisogna trovare il vettore unitario (d_ij) che rappresenta la direzione del vettore che collega i due keypoints del ramo; successivamente, estrapolare dal vettore che collega i due keypoints n_interp_samples punti equidistanti tra loro. Per ogni punto, usando le relative coordinate nel frame, trovare la coppia di valori nelle mappe paf_a e paf_b e salvarla in paf_inter.

Per trovare il PAF_score è necessario applicare il prodotto scalare tra d_ij ed ogni coppia elementi in paf_inter.

Se il rapporto tra punti del ramo con paf_score>paf_score_th ed il numero totale di punti analizzati è maggiore di conf_th, allora il collegamento tra i due estremi del ramo è accettato.
È fondamentale salvare il ramo accettato con paf_score medio più alto.

Se per uno dei due estremi del ramo non è stato trovato nessun keypoint, è importante determinare per quale index delle coppie in POSE_PAIRS si è riscontrato questo problema.

Successivamente, si collegano le coppie con keypoints in comune tra loro per creare array di keypoints, che rappresentano la posa di una persona.

```
for n in range(len(personwise_keypoints)):
                index = personwise_keypoints[n][np.array(self.POSE_PAIRS[0])]
                if -1 in index:
                    continue
                B = np.int32(self.keypoints_list[index.astype(int), 0])
                A = np.int32(self.keypoints_list[index.astype(int), 1])
                median_x = int(np.absolute(B[0] - B[1]) / 2 + min([B[0], B[1]]))
                median_y = int(np.absolute(A[0] - A[1]) / 2 + min([A[0], A[1]]))
                radius = int(
                 np.sqrt(np.power(B[0] - B[1], 2) + np.power(A[0] - A[1], 2)) * 0.6
                )
                cv.circle(
                    frame,
                    (median_x, median_y),
                    radius,
                    (0, 0, 0),
                    thickness=-1,
                    lineType=cv.FILLED,
                )
```

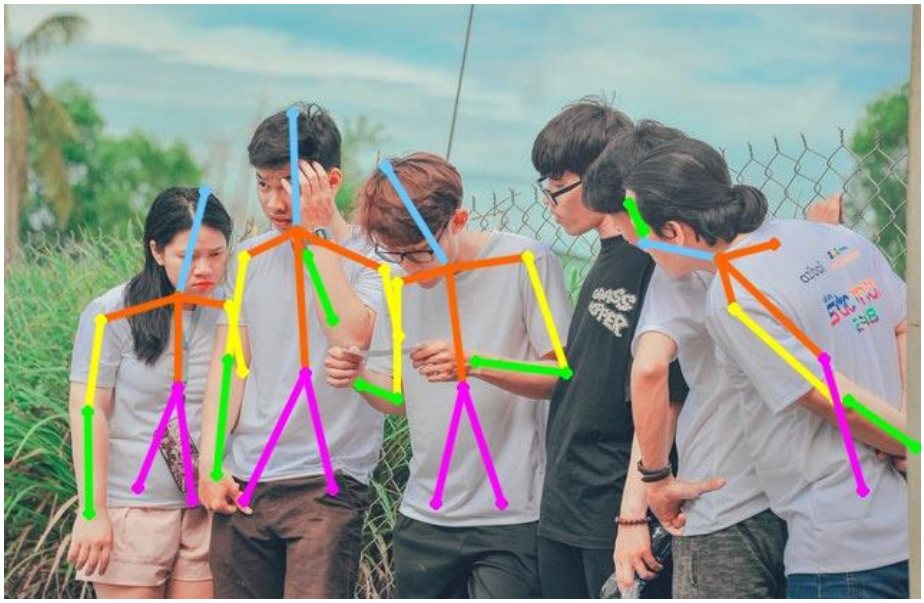
Il ruolo di questa porzione di codice è di prendere i keypoints di testa e collo di ogni persona rilevata, di trovare il punto medio tra di loro, che è approssimativamente il centro del viso, ed infine coprire il volto con un cerchio nero.

3.3 Dati Sperimentali

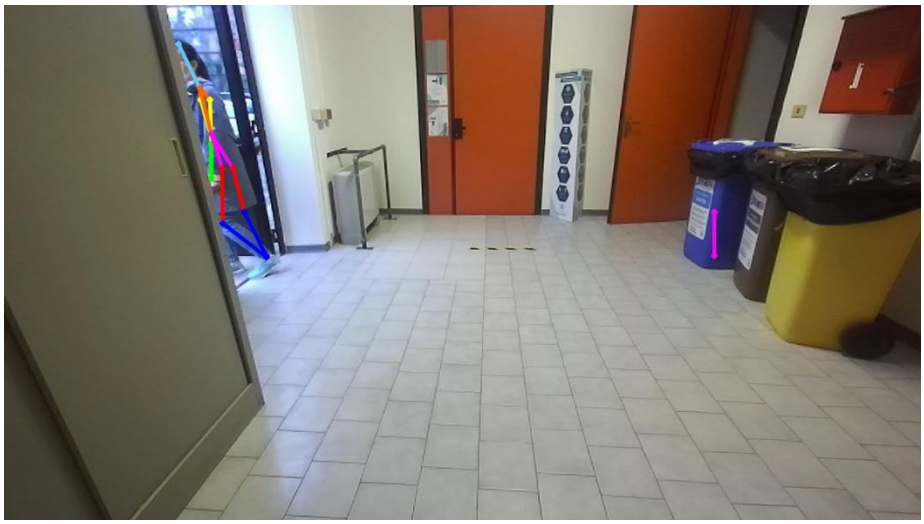
I primi output dell'applicazione presentano dei difetti ben identificabili:

1) Nell'immagine *GroupOut1.jpg* non viene identificato nessun keypoint della quarta persona da sinistra, ciò indica che bisognerebbe abbassare il valore di threshold per poter rilevare più keypoints;

2) Nell'immagine 000586*Out1.jpg* vengono identificati dei keypoints legati ad un cestino (rispettivamente petto e anca-sx), suggerendo invece di aumentare il threshold per evitare di identificare falsi positivi.



1) Immagine: GroupOut1.jpg



2) Immagine: 000586Out1.jpg

A seguito dell'analisi di tali output, bisogna approfondire l'impatto del valore di threshold legato all'identificazione dei keypoints.

3.4 Modifiche dei threshold

Successivamente l'analisi della relazione tra identificazione dei keypoints, valore threshold e prestazioni, si è deciso di aumentare il valore di threshold, difatti, i problemi rilevati in *GroupOut1.jpg* sono legati alla rete neurale stessa, non al threshold di rilevazione.

Immagine: groupOut2.jpg

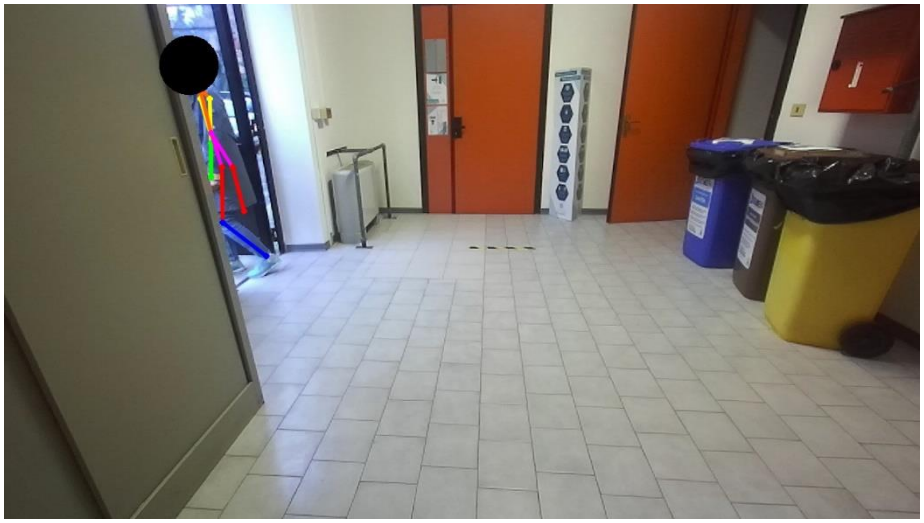


Immagine: 000586Out2.jpg

Eseguendo successivi test, il primo risultato è un valore di threshold=3.01042 (immagini sopra), risolvendo il problema dei keypoints "fantasma" del cestino della spazzatura nell'immagine *000586Out2.jpg*.

Dopo la modifica, si è verificato un problema di individuazione dei keypoints della testa nella seguente immagine:

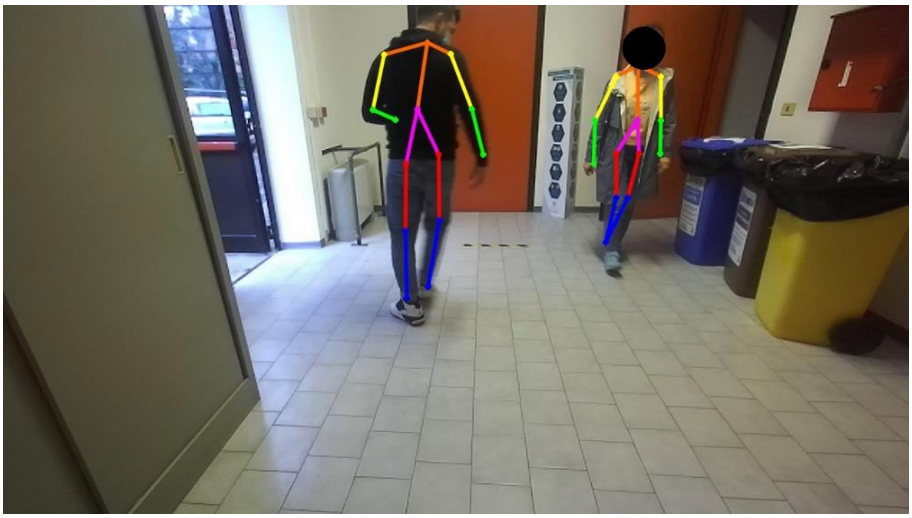


Immagine: 000265Out2.jpg

Ciò ha reso necessario una seconda calibrazione del valore di threshold, in quanto la perdita di precisione nella rilevazione di un keypoint così importante come quello della testa è inaccettabile.

Il valore finale di threshold è stato individuato in:

```
        self.threshold = 0.186737
```

Questo nuovo valore di threshold porta ad un limitato alleggerimento del programma, dato il numero minore di keypoints rilevati e quindi elaborati, con una conseguente piccola perdita in precisione nella rilevazione dei keypoints.
La rete neurale, tendenzialmente, identifica con più sicurezza la coppia di keypoints testa-collo, se il soggetto ben inquadrato nell'immagine.
La perdita di precisione è quindi contemplabile.

I valori di soglia utilizzati per i collegamenti tra keypoints (PAF) non cambiano in modo sostanziale l'output del programma;
ad eccezione di valori troppo alti che impediscono l'accettazione di qualsiasi coppia di keypoints.
Questi valori sono conseguentemente rimasti intoccati.

3.5 Nuovi output

Le seguenti immagini sono gli output risultanti dopo lo studio e la modifica del valore di threshold.

Immagine: groupOutFinal.jpg1

Immagine: 000586OutFinal.jpg

Immagine: 000555OutFinal.jpg

Immagine: 000501OutFinal.jpg

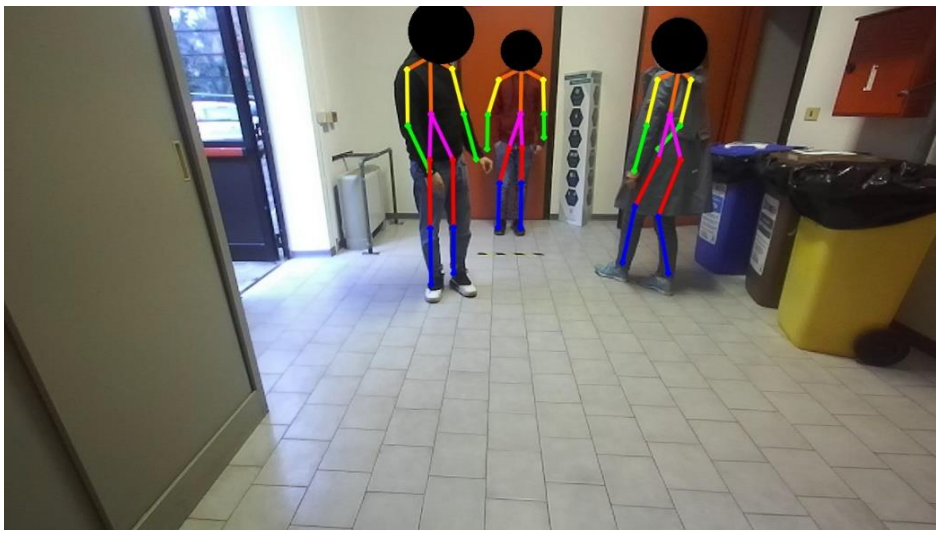


Immagine: 000443OutFinal.jpg

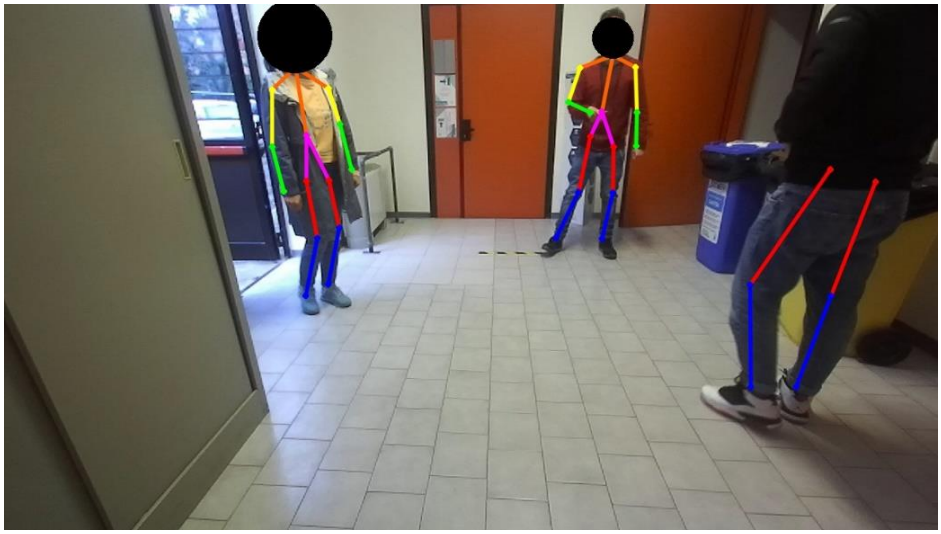


Immagine: 000352OutFinal.jpg

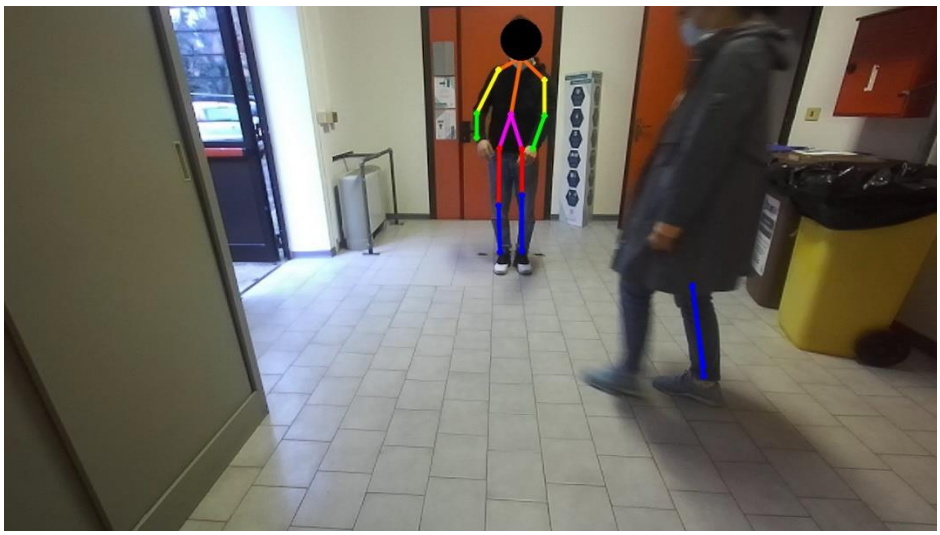


Immagine: 000286OutFinal.jpg

Immagine: 000265OutFinal.jpg

Immagine: 000244OutFinal.jpg

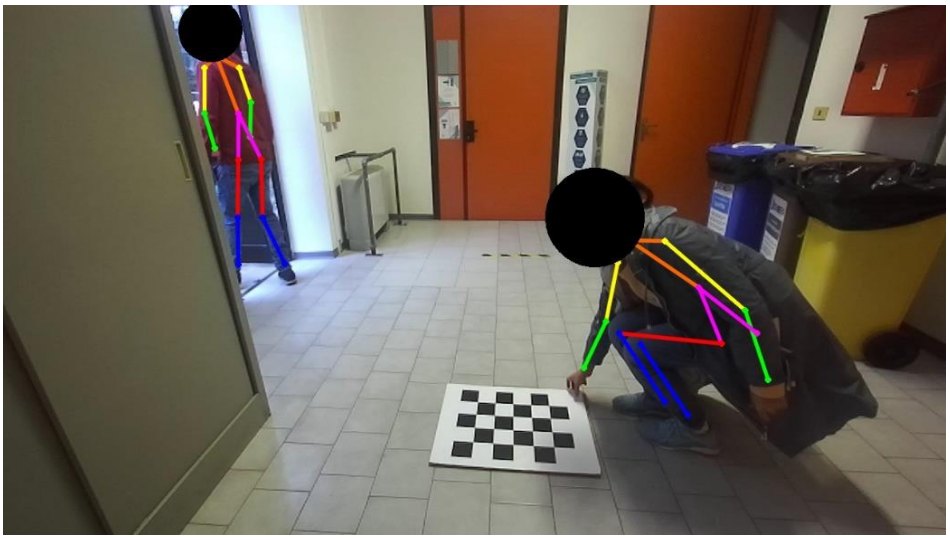


Immagine: 000173OutFinal.jpg

3.6 Limiti dell'approccio

La rete neurale utilizzata è alquanto pesante, quindi non facilmente integrabile con sistemi IOT con potenza computazionale ridotta.
Basando scelta sulla rete con formato di keypoints MPI, che è la più leggera tra quelle prodotte da OpenPose, si ha una ricerca maggiormente imprecisa dei diversi keypoints.

Un ulteriore limite di questa rete neurale è la grande sensibilità all'inquadratura dell'immagine: nell'insieme di immagini presentate, l'inquadratura è bassa e per tanto la rete non approssima una possibile posizione dei punti mancanti, causando errori di rilevazione (esempio nell'immagine *000265Out2.jpg*).

4 Conclusione

L'applicazione ha possibili migliorie attuabili a seconda dell'ambito di utilizzo:

- Con accesso ad una potenza computazionale più elevata, si può sostituire la rete neurale MPI con un'altra di quelle prodotte da OpenPose per aumentare la precisione di rilevazione e il numero dei keypoints;
- Con una minore potenza computazionale, si può limitare lo studio e l'identificazione dei keypoints a quelli di testa e collo, essenziali per la copertura del volto, oppure creare e studiare una rete apposita per questo utilizzo;
- Generalmente utilizzare un'inquadratura più alta è di aiuto alla rete nell'identificare i keypoints del viso dei soggetti.

Quest'applicazione è utilizzabile come base in ambiti medici o sportivi, ad esempio:
per studiare problemi di postura nei pazienti o per correggere i movimenti di sportivi che provocano loro danni e/o dolore, analizzando l'angolo tra diversi arti, rami.

Questa applicazione sarebbe stata impossibile senza l'utilizzo dell'intelligenza artificiale, che credo sarà una grande risorsa per risolvere problematiche anche in ambiti più complessi ed ampi, come l'agricoltura (gestione autonoma ed ottimizzata delle piantagioni), la finanza (per analizzare la borsa) e la medicina (per facilitare la diagnosi medica).

5 Fonti bibliografiche e sitografia

1)https://github.com/CMU-Perceptual-Computing-Lab/openpose

2)https://learnopencv.com/multi-person-pose-estimation-in-opencv-using-openpose/

3)https://learnopencv.com/deep-learning-based-human-pose-estimation-using-opencv-cpp-python/

4)Build and Install OpenCV With CUDA GPU Support on Windows 10 | OpenCV 4.5.1 | 2021 - YouTube

6 Ringraziamenti

Ringrazio il gatto Teo, mamma Nadia, il grande Mike, La Zia, Bafo, la mia fidanzata Catia, Fabbri il designer, LuPro, Gbully e Sforzini.

Ringrazio anche il mio relatore Stefano Mattoccia per la sua grande disponibilità.

Addio, Francesco Luzzi.