**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: Relatore:

Francesco Luzzi Prof. Stefano Mattoccia

Co-relatori:

Alessio Mingozzi

Filippo Aleotti

Matteo Poggi

Sessione

Anno Accademico 2020/21

Sommario

[1 Introduzione 3](#_Toc87093849)

[2 Strumenti utilizzati 3](#_Toc87093850)

[2.1 Anaconda 3](#_Toc87093851)

[2.2 CMake 3](#_Toc87093852)

[2.3 OpenPose 4](#_Toc87093853)

[2.4 OpenCv 4](#_Toc87093854)

[2.5 NumPy 4](#_Toc87093855)

[3 Sviluppo 5](#_Toc87093856)

[3.1 Input 6](#_Toc87093857)

[3.2 Logica 7](#_Toc87093858)

[3.3 Dati Sperimentali 10](#_Toc87093859)

[3.4 Modifiche dei threshold 11](#_Toc87093860)

[3.5 Nuovi output 12](#_Toc87093861)

[3.6 Limiti dell’approccio 13](#_Toc87093862)

[4 Conclusione 13](#_Toc87093863)

[5 Fonti bibliografiche e sitografia 13](#_Toc87093864)

# Introduzione

Oggi giorno tecnologie come navigatori e guida autonoma richiedono una grande affluenza di dati, in particolare di immagini di strade e luoghi pubblici, in cui sono presenti molte persone di cui si vuole nascondere i volti per motivi di privacy, in quanto si possono incorrere sanzioni legali ed economiche non indifferenti.  
Alla base di questo progetto, quindi, vi è la necessità di nascondere automaticamente l’identità delle persone presenti all’interno di un qualsiasi fotogramma.

L’ approfondimento di questo problema mi ha spinto a creare questa applicazione che sfrutta una rete neurale per identificare le diverse parti del corpo concentrandosi ad oscurare il volto delle persone identificate.  
In questa applicazione non bisogna riconosce le persone, ma “semplicemente” la posa di ognuna di esse concentrandosi poi sulla posizione del volto  
Per questo motivo è stato deciso di cercare di sfruttare l’intelligenza artificiale, oltre ad un interesse personale legato a questo nuovo ambito che, credo, sia la direzione che prenderà la tecnologia in futuro.

# Strumenti utilizzati

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

## Anaconda

Anaconda è un package manager, un manager di ambienti virtuali, una distribuzione di Python/R improntata alla data-science, con 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 ma anche perché questo permette di non “sporcare” il resto del sistema operativo.

## 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 ed 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*](https://cmake.org/overview)*)*

## 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*](https://answers.opencv.org/question/50025/what-exactly-is-a-blob-in-opencv/)), infatti, queste sono state generate grazie a “Caffe”, un framework specifico per i deep learning (per approfondire [*Caffe | Deep Learning Framework (berkeleyvision.org)*](https://caffe.berkeleyvision.org/)).  
  
Per spiegare l’output di tali reti neurali, bisogna prima introdurre due concetti molto importanti:

* Keypoints, diverse parti del corpo che vengono identificate dalla rete neurale e di cui abbiamo inizialmente diverse mappe di probabilità;
* PAF (Part Affinities) sono mappe di probabilità che codificano il grado di relazione tra diversi keypoints.

Detto ciò, l’output 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 in un secondo momento.

Delle diverse reti neurali prodotte da OpenPose la scelta è ricaduta su quella più leggera e dall’output più limitato, questa è basata su un formato di output dei keypoints chiamata MPI con 15 keypoints e le relative PAF (immagine sotto).  
La motivazione di questa scelta è legata all’ambito di utilizzo inteso per questa applicazione, principalmente ideata per sistemi mobili o comunque di limitata potenza computazionale.

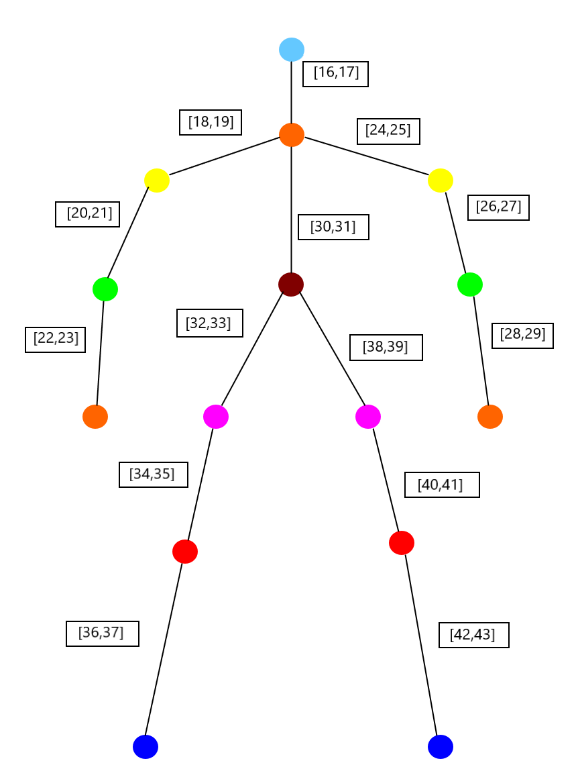
****

Immagine: mpiPAFS.jpg

Immagine: mpiKeypoints.jpg

## 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 modificata:   
è stata compilata con CMake utilizzando il source code “originale” di opencv ed il codice della versione “contrib” *(*[*OpenCV (github.com)*](https://github.com/opencv)*)*, tutto ciò 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.

## NumPy

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

Di fatto 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.

# Sviluppo

Di seguito verranno presentate le variabili più importanti dell’applicazione.

        self.threshold = 0.10

        self.paf\_score\_th = 0.12

        self.conf\_th = 0.8

La definizione dei diversi valori di threshold per differenti scopi:

* threshold, soglia minima di rilevazione dei keypoints;
* paf\_score\_th, soglia minima di accettazione del valore paf relativa ad un punto;
* conf\_th, percentuale minima di punti con un paf score >paf\_score\_th all’interno di una connessione.

        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.

## Input

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



Immagine: Group.jpg



Immagine: 000586.jpg

Immagine che contiene pavimento, piastrellato

Descrizione generata automaticamente

Immagine: 000443.jpg

Immagine che contiene pavimento, interni, piastrellato

Descrizione generata automaticamente

Immagine: 000352.jpg

Immagine che contiene testo, pavimento, interni, piastrellato

Descrizione generata automaticamente

Immagine: 000286.jpg

Immagine che contiene pavimento, interni, piastrellato

Descrizione generata automaticamente

Immagine: 000265.jpg

Immagine che contiene pavimento, interni, piastrellato

Descrizione generata automaticamente

Immagine: 000244.jpg

Immagine che contiene pavimento, interni

Descrizione generata automaticamente

Immagine: 000173.jpg

Immagine che contiene pavimento, interni

Descrizione generata automaticamente

Immagine: 000501.jpg

Immagine che contiene pavimento, interni

Descrizione generata automaticamente

Immagine: 000555.jpg

## Logica

Qual è il concetto principale dietro questa rete neurale? La probabilità.  
Questa non restituisce in output un valore certo in cui trovare uno specifico keypoint, ma una mappa di probabilità in cui ogni pixel ha una determinata probabilità di rappresentare il keypoint richiesto.

Per spiegare meglio questo concetto viene mostrato il grafico di probabilità dei keypoints della testa in relazione all’immagine di 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.

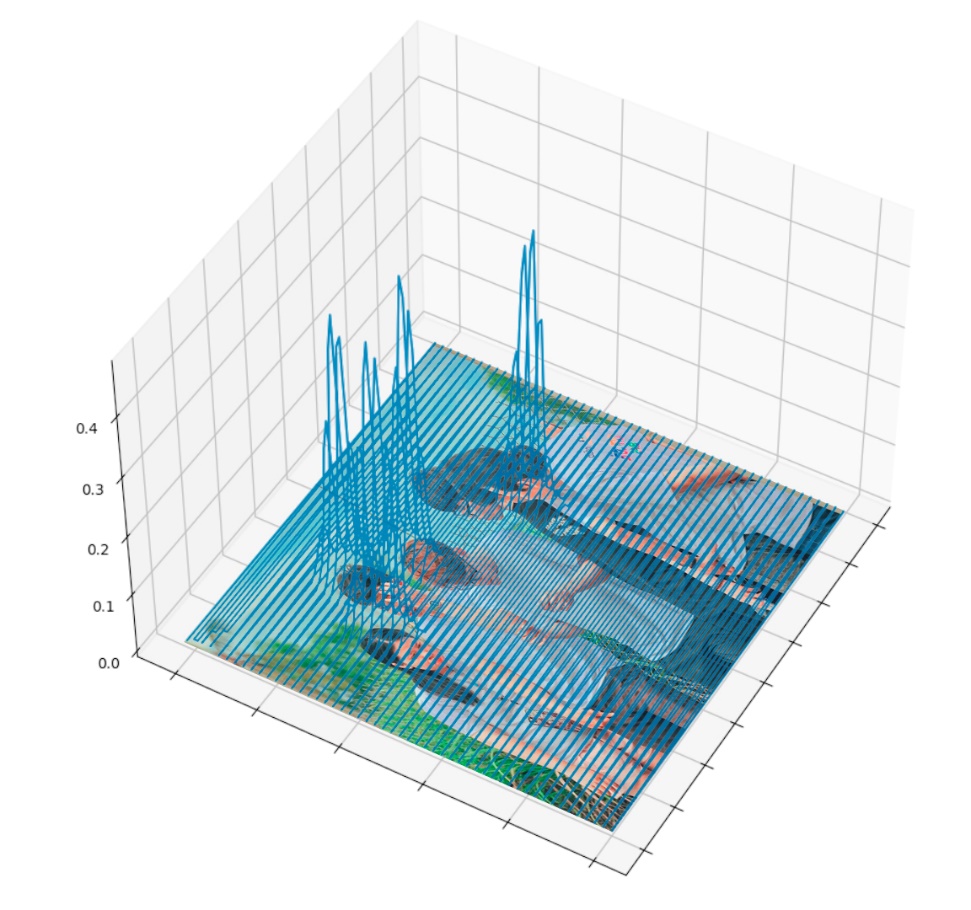


Immagine: probMapHeadGroup.jpg

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 diversi frame;  
utilizzando le mappe di probabilità, si prende il valore massimo di ogni “agglomerato” di valori di probabilità al di sopra del valore di threshold, questi verranno poi salvati come keypoints.

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

Si può pensare 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, questo può bastare?

Prendiamo come caso particolare l’immagine presentata di seguito:

Immagine che contiene persona

Descrizione generata automaticamente

Immagine: shortesDistance.jpg

In questo caso, si può notare chiaramente il limite dell’approccio presentato:  
se due persone sono abbastanza vicine o semplicemente sovrapposte tra loro, si rischia di collegare un keypoint relativo ad una persona alla persona sbagliata.

Per questo motivo 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, rosso valori di probabilità minori, giallo invece valori più alti.

Immagine che contiene persona, esterni, gruppo, persone

Descrizione generata automaticamente

Immagine: explanationPAF.jpg

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

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, che chiameremo paf\_a e paf\_b, estremi di una connessione, che chiameremo 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 (=10) punti equidistanti tra loro. Per ogni punto, usando le relative coordinate nel frame, trovare i valori nelle mappe paf\_a e paf\_b e salvarli in paf\_inter.

Per trovare il PAF\_score bisogna fare 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 > di conf\_th allora, il collegamento tra i due estremi del ramo è accettato.  
Bisogna quindi salvare il ramo accettato con paf\_score medio più alto.

Se per uno dei due estremi del ramo non è stato trovato nessun keypoint, bisogna salvare per quale coppia di keypoints si è riscontrato questo problema.

Successivamente è solo una questione di collegare 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, quindi in modo approssimato il centro del viso, ed infine coprire il volto con un cerchio colorato di nero.

## Dati Sperimentali

I seguenti 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.

Immagine che contiene persona, esterni, colorato, carica

Descrizione generata automaticamente

1. Immagine: GroupOut1.jpg

Immagine che contiene pavimento, interni, edificio, piastrellato

Descrizione generata automaticamente

1. Immagine: 000586Out1.jpg

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

## 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 che contiene persona

Descrizione generata automaticamente

Immagine: groupOut2.jpg

Immagine che contiene pavimento, interni, edificio, piastrellato

Descrizione generata automaticamente

Immagine: 000586Out2.jpg

Eseguendo successivi test si è arrivati, inizialmente ad un valore di threshold=3.01042, risolvendo il problema dei keypoints “fantasma” del cestino della spazzatura nell’immagine *000586Out2.jpg*.

Si è presentato però un problema di individuazione dei keypoints della testa nella seguente immagine:



Immagine: 000265Out2.jpg

Rendendo necessaria una seconda calibrazione del valore di threshold, in quanto la perdita di precisione nella rilevazione di un keypoint così importante come quello della testa, in questa applicazione, è del tutto inaccettabile.

Il valore finale di threshold è stato infine individuato in:

        self.threshold = 0.186737

Questo porta ad un leggero alleggerimento del programma, dato il numero minore di keypoints rilevati e quindi elaborati, con una piccola perdita in precisione nella rilevazione dei keypoints.  
La rete, 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 collegamenti tra keypoints (PAF), si è notato che non cambiano in modo sostanziale l’output del programma;  
esclusi valori troppo alti che impediscono l’accettazione di qualsiasi coppia di keypoints. Questi valori sono conseguentemente rimasti intoccati.

## Nuovi output

Come spiegato in precedenza, l’aumento del threshold ha portato all non identificazione dei falsi positivi, non risolvendo però il problemi di rilevazione in *groupOut2\_Private.jpg.*

Immagine che contiene persona, colorato

Descrizione generata automaticamente

Immagine: groupOutFinal.jpg

Immagine che contiene pavimento, interni, edificio, piastrellato

Descrizione generata automaticamente

Immagine: 000586OutFinal.jpg

Immagine che contiene pavimento, interni

Descrizione generata automaticamente

Immagine: 000555OutFinal.jpg

Immagine che contiene pavimento, interni

Descrizione generata automaticamente

Immagine: 000501OutFinal.jpg

Immagine che contiene pavimento, interni, piastrellato

Descrizione generata automaticamente

Immagine: 000443OutFinal.jpg

Immagine che contiene pavimento, interni, parete

Descrizione generata automaticamente

Immagine: 000352OutFinal.jpg

Immagine che contiene pavimento, interni

Descrizione generata automaticamente

Immagine: 000286OutFinal.jpg

Immagine che contiene pavimento, interni

Descrizione generata automaticamente

Immagine: 000265OutFinal.jpg

Immagine che contiene pavimento, interni, piastrellato

Descrizione generata automaticamente

Immagine: 000244OutFinal.jpg

Immagine che contiene pavimento, interni

Descrizione generata automaticamente

Immagine: 000173OutFinal.jpg

## Limiti dell’approccio

La rete neurale è molto pesante, quindi non facilmente integrabile con sistemi IOT di potenza computazionale ridotta, inoltre, la rete scelta basata sul formato di keypoints MPI, essendo la più leggera tra quelle prodotte da OpenPose, è anche la più imprecisa nella ricerca dei diversi keypoints.

Un ulteriore limite di questa rete neurale è la grande sensibilità rispetto all’inquadratura dell’immagine;  
quando punti importanti del corpo di una persona non vengono inquadrati, come in questo caso le parti finali delle teste a causa dell’inquadratura bassa, questa rete non approssima una possibile posizione di questo punto mancante, portando ad errori di rilevazione.

# Conclusione

Possibili ambiti di utilizzo, utilità di questa applicazione e possibili migliorie quali, per esempio, ridurre il peso di esecuzione limitandosi ad analizzare la posizione soltanto dei keypoints del viso per avere un framerate migliore oppure utilizzare una rete neurale specificatamente studiata per trovare la posizione e la dimensione dei volti in un frame.  
Un problema riscontrato con le immagini usate per calibrare l’applicazione è stato l’angolo e l’altezza dell’inquadratura: una vista poco chiara o incompleta della parte finale della testa manda in confusione la rete neurale.  
Si potrebbe ovviare a questo problema usando un algoritmo per la stima della posizione ( credo però che questo obblighi ad identificare le diverse persone scontrandosi con l’idea iniziale di riconoscere SOLO la posa delle stesse ).

# Fonti bibliografiche e sitografia

*1)*[*https://github.com/CMU-Perceptual-Computing-Lab/openpose*](https://github.com/CMU-Perceptual-Computing-Lab/openpose) *2)*[*https://learnopencv.com/multi-person-pose-estimation-in-opencv-using-openpose/*](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/*](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*](https://www.youtube.com/watch?v=YsmhKar8oOc)

Luzzi Francesco, ringraziamenti ed addio.