**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

Creazione applicazione per il tracking

di persone e oscuramento volti.

CANDIDATO: Relatore:

Francesco Luzzi Prof. Stefano Mattoccia

Correlatore:

Alessio Mingozzi

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

Alla base di questo progetto vi è la necessità di nascondere automaticamente l’identità delle persone presenti all’interno di un fotogramma, coprendone il volto.   
In questo tipo di applicazione non bisogna riconosce le persone, ma “semplicemente” la posa di ognuna di esse concentrandosi, sulla posizione del volto.

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.

# Strumenti utilizzati

Durante lo sviluppo di questa applicazione gli strumenti utilizzati sono stati:

## Anaconda

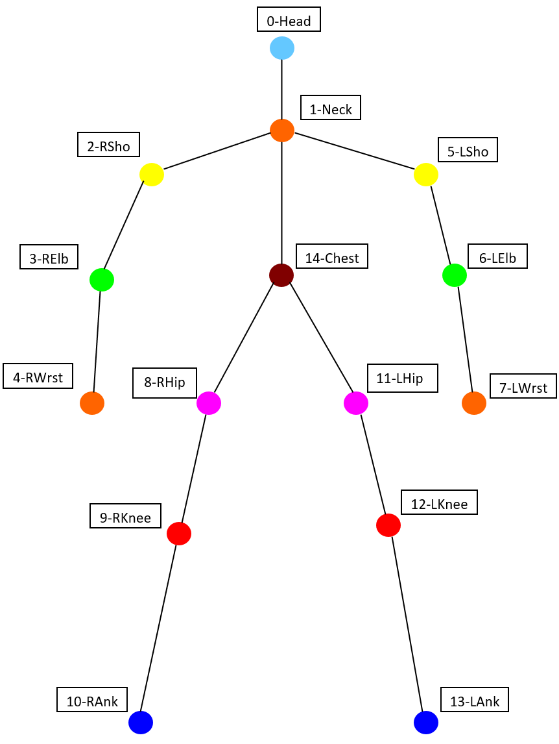
Anaconda è un package manager, un manager di ambienti virtuali, una distribuzione di Python/R improntata alla data-science, con una grande collezione di pacchetti open-source.

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

  
Sfruttando il machine learning OpenPose ha creati diverse reti neurali che, passato come input un’immagine restituiscono una mappa di probabilità delle diverse parti del corpo delle persone presenti nell’immagine.

La rete neurale sfruttata, nello specifico, utilizza un formato di output dei “keypoints” chiamato MPI.

I keypoint sono le parti del corpo che la rete neurale riconosce nel frame analizzato.

Questa rete restituisce anche mappe di probabilità per le diverse “connessioni” tra i diversi keypoints (PAF), molto utili per riconoscere la relazione tra essi (sono della stessa persona o meno).

## 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 sono presenti le variabili più importanti dell’applicazione.

        # threshold to detect the keypoint

        self.threshold = 0.10

        # number of points on PAF

        self.n\_interp\_samples = 10

        # threshold for paf score

        self.paf\_score\_th = 0.12

        # threshold for pair to get accepted

        self.conf\_th = 0.8

Valori di threshold per la rilevazione dei keypoints e per l’accettazione del collegamento tra due kypoints.

        # number of points detected 15 for MPI

        self.n\_points = 15

        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.

## Input

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



Immagine: Group.jpg



Immagine: ghostBin.jpg

## Logica

Immagini del codice con immagini esplicatorie della logica dietro la probMap (da fare) utilizzata per trovare tutti i keypoints.

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 = []

        # find the contours where the keypoints migth be

        contours, \_ = cv.findContours(map\_mask, cv.RETR\_TREE, cv.CHAIN\_APPROX\_SIMPLE)

        # for each contour find the maxima of probability

# to find the keypoint

        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)

            # this is a tuple formed by point\_x, point\_y, probability of that point

            # to contain a valid keypoint

            keypoints.append(max\_loc + (prob\_map[max\_loc[1], max\_loc[0]],))

        return keypoints

Parte di codice adibita a trovare i diversi keypoints nel frame analizzato.

(Spiegazione per quale motivo è sbagliato usare un algoritmo che lega diversi keypoint a seconda della distanza tra loro piuttosto che basarsi sui PAF)

Immagine che contiene persona

Descrizione generata automaticamente

Immagine con display delle mappe di affidabilità PAF, questo ci aiuta a capire quale dovrà essere l’approccio.



Spiegare l’utilizzo delle mappe di probabilità PAF (Part Affinities) che codificano il grado di associazione tra i diversi keypoints, utili per rivendicare diversi keypoints alle persone corrette.

def get\_valid\_pairs(self, output):

        valid\_pairs = []

        invalid\_pairs = []

        # loop for every POSE\_PAIR

        for k in range(self.n\_points - 1):

            # A->B constitute a limb this the axis inverted  
 #to get a point paf value paf[point\_y,point\_x]

            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))

            # Find the keypoints for the first and second limb

            # all keypoints for the POSE\_PAIR[k][0] keypoint (0=Head,...)

            cand\_a = self.detected\_keypoints[self.POSE\_PAIRS[k][0]]

            # all keypoints for the POSE\_PAIR[k][1] keypoint (1=Neck,...)

            cand\_b = self.detected\_keypoints[self.POSE\_PAIRS[k][1]]

            n\_a = len(cand\_a)

            n\_b = len(cand\_b)

            # If keypoints for the joint-pair is detected

            # check every joint in cand\_a with every joint in cand\_b

            # Calculate the unitary direction vector between the two joints

            # Find the PAF values at a set of interpolated points between the joints

            # Dot product between the direction vector and the PAF values to find the value of certainty of that connection

            if n\_a != 0 and n\_b != 0:

                valid\_pair = np.zeros((0, 3))

                # iterate all keypoints for both joints cand\_a and cand\_b

                for i in range(n\_a):

                    max\_j = -1

                    max\_score = -1

                    found = 0

                    for j in range(n\_b):

                        # Find d\_ij

                        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

                        # Create an array of n\_interp\_samples interpolated points on the line joining the two keypoints.

                        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,

                                ),

                            )

                        )

                        # Create an array of n\_interp\_samples probability values of the PAFs in the position of the points that we got beefore.

                        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])),

                                    ],

                                ]

                            )

                        # Check the PAF score of each pair of points, doing the dot product between the PAF and the unit vector giving the direction of the connection

                        paf\_scores = np.dot(paf\_interp, d\_ij)

                        # Find avg\_PAF\_score

                        avg\_paf\_score = sum(paf\_scores) / len(paf\_scores)

                        # Check if the connection is valid

                        # If the fraction of interpolated vectors that has at least paf\_score\_th, is higher then threshold -> Valid Pair

                        if (

                            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

                    # Append the connection to the list [[x,y],score]

                    if found:

                        valid\_pair = np.append(

                            valid\_pair,

                            [[cand\_a[i][3], cand\_b[max\_j][3], max\_score]],

                            axis=0,

                        )

                # Append the detected connections to the global list

                valid\_pairs.append(valid\_pair)

            else: # If no keypoints are detected

                invalid\_pairs.append(k)

                valid\_pairs.append([])

        return valid\_pairs, invalid\_pairs

Iterando su tutti i paf\_idx definiti 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 ci indica 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 sue 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 elemento di 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 poi salvare il ramo con paf\_score medio più alto.

Se per uno dei due estremi del ramo non sono stati trovati dei keypoints salvarsi per quale coppia di lembi si è riscontrato questo problema.

## 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 *ghostOut1.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: ghostBinOut1.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

Esempio di immagine con threshold=3.3

Eseguendo successivi test si è arrivati ad un valore di threshold=3.01042 .

        # threshold to detect the keypoint

        self.threshold = 0.301042

Questo porta ad un alleggerimento del programma, dato il numero minore di keypoints rilevati e quindi elaborati, con una piccola perdita in precisione.  
La rete, tendenzialmente, identifica con più sicurezza la coppia di keypoints testa-collo; quindi, la perdita di precisione è contemplabile.

I valori di soglia utilizzati per collegamenti tra keypoints (PAF), si è notato che non cambiano l’output del programma; esclusi naturalmente valori troppo alti che impediscono l’accettazione di qualsiasi coppia di keypoints. Questi valori sono conseguentemente rimasti intoccati.

## Nuovi output

Nuove immagini di output con commento sui cambiamenti dell’output e sull’aumento delle prestazioni (dati i minori keypoints “inutili” registrati)

Immagine che contiene pavimento, interni, edificio, piastrellato

Descrizione generata automaticamente

Immagine: ghostBinOut2.jpg



Immagine: groupOut2.jpg

## Limiti dell’approccio

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

# 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.

# Fonti bibliografiche e sitografia

*-*[*https://github.com/CMU-Perceptual-Computing-Lab/openpose*](https://github.com/CMU-Perceptual-Computing-Lab/openpose) *-*[*https://learnopencv.com/multi-person-pose-estimation-in-opencv-using-openpose/*](https://learnopencv.com/multi-person-pose-estimation-in-opencv-using-openpose/) *-*[*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/) *-*[*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.