**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](#_Toc87003758)

[2 Strumenti utilizzati 3](#_Toc87003759)

[2.1 Anaconda 3](#_Toc87003760)

[2.2 CMake 3](#_Toc87003761)

[2.3 OpenPose 4](#_Toc87003762)

[2.4 OpenCv 4](#_Toc87003763)

[2.5 NumPy 4](#_Toc87003764)

[3 Sviluppo 5](#_Toc87003765)

[3.1 Input 5](#_Toc87003766)

[3.2 Logica 6](#_Toc87003767)

[3.3 Dati Sperimentali 7](#_Toc87003768)

[3.4 Modifiche dei threshold 7](#_Toc87003769)

[3.5 Nuovi output 7](#_Toc87003770)

[3.6 Limiti dell’approccio 8](#_Toc87003771)

[4 Conclusione 8](#_Toc87003772)

# 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 delle 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

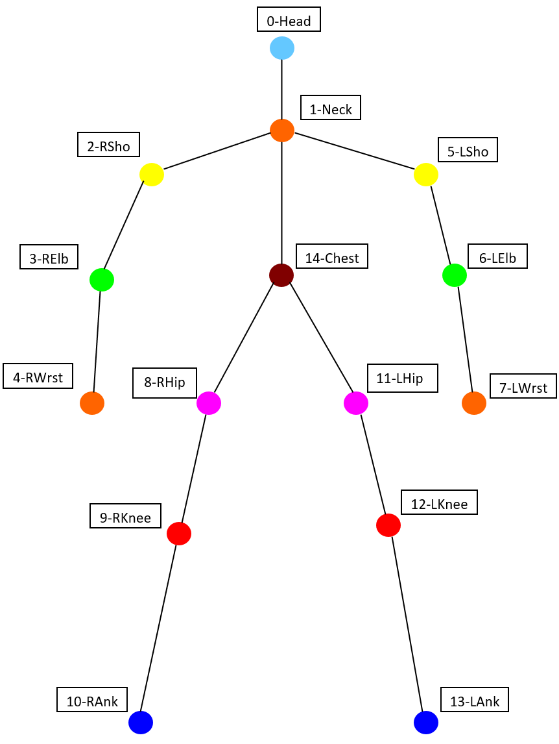
Anaconda è un package manager, un manager di ambienti virtuali, una distribuzione di Python/R improntata alla data-science, con una grande collezione di pacchetti open-source.

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 usata nello specifico utilizza un formato di output dei “keypoints” chiamato MPI.

I keypoint sono le parti del corpo che la rete neurale cerca di riconoscere 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 si esporranno le parti principali del codice.

        # 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.

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 (aka where the prob i higher),

# 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.

## Input

Immagini utilizzate per il testing e problemi con questa rete neurale.

## Logica

Immagini del codice con immagini esplicatorie della logica dietro la probMap, il ritrovamento dei keypoints e 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.

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.



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

## Dati Sperimentali

I seguenti output della rete neurale presentano dei difetti ben identificabili:

* nella prima immagine non viene identificato nessun keypoint della quarta persona da sinistra, ciò indica che bisognerebbe abbassare il valore di threshold per l’identificazione dei keypoint;
* nella seconda immagine vengono identificati dei keypoints legati ad un cestino (rispettivamente petto e anca-sx), suggerendo invece di aumentare il livello di threshold per evitare di identificare falsi positivi.

Immagine che contiene persona, esterni, colorato, carica

Descrizione generata automaticamente

Immagine che contiene pavimento, interni, edificio, piastrellato

Descrizione generata automaticamente

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

## Modifiche dei threshold

Spiegazione della modifica dei valori di threshold per la rilevazione dei keypoints, specificando che solitamente i keypoint meglio riconosciuti sono quelli del collo e del naso/viso, quindi anche perdere qualche keypoint dei piendi non ci interessa allo scopo dell’applicazione.

Immagine che contiene persona

Descrizione generata automaticamente

Esempio di immagine con threshold=3.3

Sucessivamente all’analisi della relazione tra identificazione dei keypoints, valore threshold e prestazioni, si è deciso di aumentare il valore di threshold a

        # threshold to detect the keypoint

        self.threshold = 0.301042

Questo porta ad un alleggerimento del programma, dato il numero minore di keypoints rilevati, con una relativamente piccola perdita in precisione in quanto la rete, tendenzialmente, identifica con più sicurezza la coppia di keypoints viso-petto.

Gli altri valori di soglia utilizzati per PAF si è notato che, sostanzialmente, non cambiano l’output, esclusi valori troppo alti che impedivano l’accettazione di qualsiasi coppia di keypoints; sono quindi rimasti intoccati.

## Nuovi output

Nuove immagini di output degli input 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



## Limiti dell’approccio

La rete neurale è molto pesante, quindi non integrabile con sistemi IOT con potenza computazionale ridotta, inoltre la rete scelta basata su MPI, essendo infatti 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://learnopencv.com/multi-person-pose-estimation-in-opencv-using-openpose/>  
  
<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.