# POLITECNICO DI TORINO


Politecnico di Torino

DIPARTIMENTO DI AUTOMATICA E INFORMATICA

**Ingegneria Informatica**

Corso di Image Processing and Computer Vision

# Report di progetto: riconoscimento posti liberi in un'aula studio

***Docente:***

*Bartolomeo Montrucchio*

***Esercitatore:***

*Luigi De Russis*

***Studenti:***

*Luca Filippetti, 303392*

*Davide Aiello, 303296*

*Anno accademico 2021/2022*

# Sommario

# Introduzione:

Studiare nelle aule dell'ateneo rappresenta per molti universitari una scelta importante, in quanto permette loro di avere uno spazio dedicato dove poter concentrarsi e preparare al meglio gli esami. Conoscere la disponibilità dei posti in aula e la loro collocazione può quindi essere molto utile e vantaggioso.

Il progetto di riconoscimento dei posti liberi nelle aule studio nasce proprio con l'obiettivo di risolvere il problema della mancanza di informazioni sulla disponibilità dei posti. Grazie a questo progetto, sarà possibile per gli studenti conoscere quali posti sono disponibili e scegliere quello più adeguato alle loro esigenze, mentre per i responsabili dell'ateneo sarà più facile gestire la disponibilità delle aule studio e garantire che tutti gli studenti possano trovare un posto dove studiare.

Per avviare il nostro progetto basta far partire il file *main.py* allegato che prende il flusso video d'esempio *CVvideo_aula.MOV* riproducendo in output il medesimo con l'aggiunta di un overlay grafico per sottolineare meglio la distinzione tra posti liberi e occupati.

Inoltre, sono allegati i file utili per riprodurre il risultato sul proprio pc, *result.json* il quale contiene i risultati della rete neurale YOLO, il *VideoPickle* che contiene le coordinate delle posizioni dei quadrati che specificano i posti a sedere e *SpacePicker.py* che contiene il codice per generare le posizioni salvate nel VideoPickle, quest'ultimo file python non è necessario per mettere in azione il prototipo.

# Programma:

Prima di illustrare il funzionamento del programma presentiamo una carrellata delle varie tecniche e approcci provati che non hanno portato ad una soluzione accettabile.

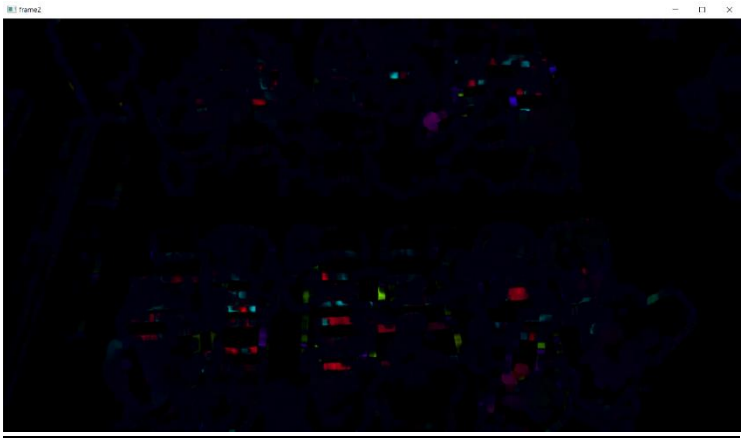
## Prove:

### AdaptiveThreshold:

Siamo partiti dalle tecniche più tradizionali della computer vision cercando di lavorare a basso livello con i pixel. L'idea era quella di prendere il flusso video ed applicare una serie di funzioni per migliorare la qualità e la stabilità della ripresa e infine usare la funzione *adapativeThreshold* della libreria di **openCV[1].** La funzione viene utilizzata per applicare una soglia adattiva su un'immagine ed è un ottimo metodo di soglia soprattutto per via del suo adattamento alle variazioni locali del video, il che può essere utile quando sfruttare l'intensità globale dell'immagine non porta ad ottenere informazioni utili. Nonostante ciò, non siamo riusciti a trovare un modo efficace per capire se il posto fosse occupato o libero: il semplice conteggio dei pixel bianchi, e il relativo confronto con una threshold, non permetteva di indentificare ottimamente il comportamento all'interno di ogni posizione precedentemente definita. I risultati sono stati quindi scarsi e altalenanti, molto probabilmente dovuti anche al fatto che la ripresa non fosse totalmente perpendicolare con il terreno e stabile e quindi era difficile trovare una giusta zona di background da analizzare e separare per verificare così la presenza o meno dello studente.

### Motion estimation:

Un'ulteriore strada che abbiamo deciso di tentare è stata utilizzare la motion estimation tramite il dense optical flow in modo da verificare la presenza di una persona in funzione del suo movimento sul posto a sedere (quindi controllare se c'era movimento all'interno del rettangolo precedentemente definito tramite *VideoPickle)* o quanto meno unire questa tecnica alla precedente per catturare il momento in cui la persona sia alzava o sedeva. Per fare questo ci siamo valsi della funzione di openCV *calcOpticalFlowFarneback* richiamata con gli stessi parametri presenti su un esempio di implementazione nella documentazione (non abbiamo fatto test con ulteriori parametri perché la tecnica non è stata ulteriormente esplorata; i motivi saranno riportati successivamente) in modo da catturare il flow. L'implementazione segue quella presente sulla documentazione di openCV. La parte chiave sfrutta una matrice nello spazio di colori HSV grande quanto l'immagine di input. Il canale di saturazione viene messo al massimo (255). Gli altri canali sono calcolati a partire dalle due componenti del flow restituite da *calcOpticalFlowFarneback* (passandogli in particolare il frame precedente e quello attuale) e poi successivamente convertite in coordinate polari tramite *cartToPolar* ottenendo così l'angolo e l'ampiezza. Il primo è stato sfruttato per calcolare il canale del tono per la nostra matrice in HSV (da qua capiamo che la parte colorata nella nostra immagine finale è indice di movimento) mentre l'ampiezza per il canale dell'illuminazione (valori maggiori di illuminazione indicano pixel in cui c'è un effettivo movimento). Per la visualizzazione finale, l'immagine è convertita nello spazio RGB. Il problema con questa tecnica è l'alto costo computazionale che rende difficile apprezzare il movimento ed una sua effettiva efficacia (il frame rate si abbassa notevolmente, una possibile miglioria sarebbe calcolare il dense-optical flow solo all'interno dei rettangoli, cioè delle aree di interesse) ed il fatto che gli studenti tendono a rimanere per lo più fermi mentre studiano. Piuttosto che tentare di ottenere risultati migliori con questa tecnica, abbiamo deciso di tentare altre strade.



## Cascade classifier:

Ci siamo allora spostati verso tecniche di machine learning. Dato che il task in questione è l'object detection, un primo approccio è stato il cascade classifier, utilizzando le Haar features. Abbiamo quindi richiamato la funzione *CascadeClassifier* e come modello le abbiamo passato *haarcascade_upperbody.xml* dato che all'interno del nostro video era per lo più visibile la parte superiore del corpo. Chiamando *detectMultiScale* sul modello con diversi parametri, i risultati sono stati comunque molto scarsi.

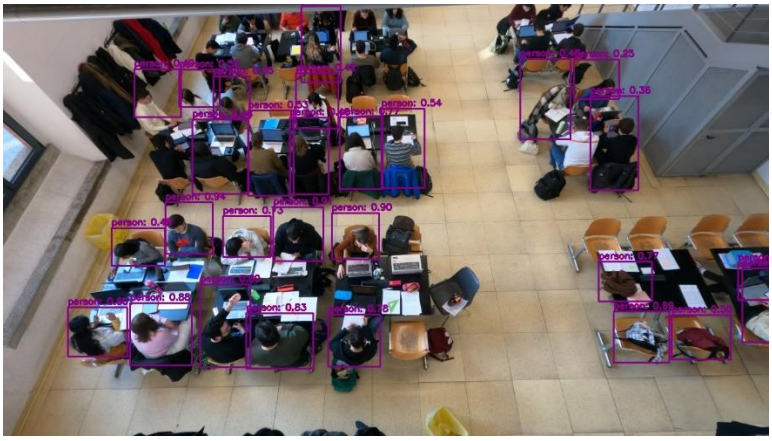
## Neural network:

Successivamente abbiamo deciso di usare una rete neurale per poter riconoscere le persone in modo più preciso e gestire successivamente su script la presenza o meno dei posti liberi. Abbiamo deciso di utilizzare una rete già allenata sul task di object detection per poter risparmiare tempo e potenza di calcolo.

Le prime prove sono state fatte usando YOLOv4 e v7 in locale importando la rete direttamente su file python tramite le funzioni di libreria di openCV ottenendo scarsi risultati sia a livello di accuratezza che di stabilità.

# Soluzione finale:

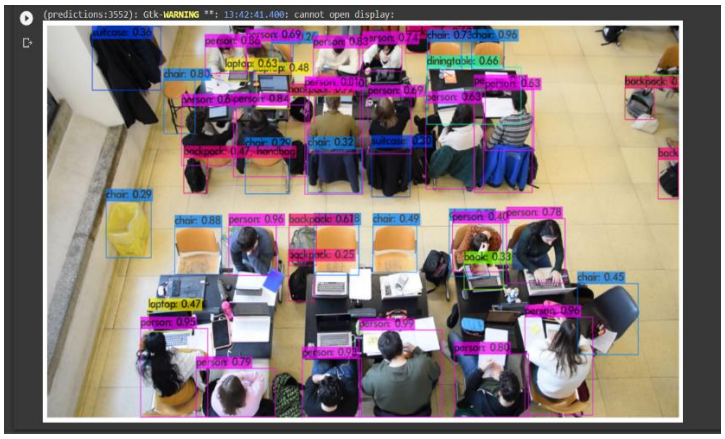
Prima di andare ad analizzare il prototipo dal punto di vista funzionale è utile sapere che la ripresa del video *CVvideo_aula.MOV* è stata realizzata tramite una NIKON D3300 riprendendo a 25fps con risoluzione di 1920x1080 ed è stato scelta un'angolazione sopraelevata per riprendere al meglio le postazioni, mantenendo una sorta di regolarità tra le postazioni e cercando di inquadrare in modo accurato e preciso le figure umane. Abbiamo anche provato a realizzare una ripresa più ampia tramite GOPRO Hero 9 con scarsi risultati per la fase di detection, come si vede in foto.



Per quanto riguarda il flusso del prototipo in sé lo abbiamo diviso in due parti, una parte di detection, realizzata su **google colab[2]** rielaborando il codice gitHub **DarkNet[3]** basato sulla rete neurale YOLOv4 e una parte di logica su file python che prende le predizioni fatte precedentemente e le gestisce per individuare la presenza o meno dei posti liberi.

## Detection:

La scelta di YOLOv4 è dovuta al fatto che quest'ultimo è considerato uno dei modelli di rilevamento degli oggetti più avanzati attualmente disponibili. YOLO è un algoritmo di rilevamento degli oggetti basato su una rete neurale profonda, in particolare una rete neurale convoluzionale. La rete neurale viene addestrata utilizzando una grande quantità di immagini etichettate, cioè immagini in cui gli oggetti presenti sono stati marcati e identificati. Una volta addestrata, la rete neurale può essere utilizzata per riconoscere gli oggetti in nuove immagini o video e individuarne la posizione.




Per prima cosa abbiamo ridotto i tempi di detection riducendo il frame rate del video saltando un frame ogni tre, cercando di non rendere il flusso video troppo a scatti, prendendo quindi 16 frame ogni secondo. Successivamente abbiamo provato a comprimere i frame in jpg prima di passarli alla rete ma ciò non ha ridotto il tempo di calcolo come sperato (che si aggirava intorno ai 50 ms per frame) anzi, i risultati erano decisamente più accurati con le immagini in png; abbiamo deciso così di mantenere il frame in alta qualità mantenendo il formato png, ma comunque abbassando la qualità da Full HD ad HD dato che non era necessaria una qualità superiore né per la nostra visualizzazione né per la rete in sé. Infine, abbiamo caricato su un fork del github della rete tutti i frame necessari e dopo averlo clonato su google colab, abbiamo testato la rete su di essi. In particolare, sono stati scaricati i pesi della rete YOLOv4 e il modello di quest'ultima, che come già detto è stata precedentemente allenata sul dataset di COCO.

Inoltre, abbiamo specificato come output un file json per contenere così tutte le informazioni delle predizioni fatte dalla rete su un unico file potendo così elaborarlo in un secondo momento.

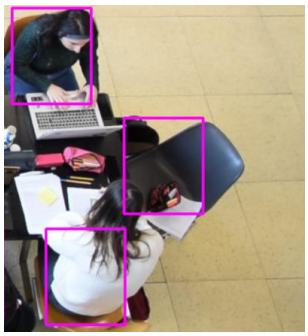
## Logica:

Il codice si divide principalmente in due file, il file di SpacePicker.py e il main.py.

Il primo serve solo in una fase a priori per specificare manualmente i posti presenti nella ripresa. Viene catturato il flusso video e viene definita una funzione che aggiunge o sottrae da una lista le coordinate dei punti che si ottengono premendo rispettivamente il tasto sinistro e destro del mouse. Queste coordinate sono poi salvate sul file **pickle[4]** e usate poi nel file di main.py.

```
23  def mouse_click(events, x, y, flags, params):
24
25      if events == cv2.EVENT_LBUTTONDOWN:
26          posList.append((x, y))
27      if events == cv2.EVENT_RBUTTONDOWN:
28          for i, pos in enumerate(posList):
29              x1, y1 = pos
30              if (x1 - int(side/2)) < x < (x1 + int(side/2)) and (y1 - int((side*1.2)/2)) < y < (y1 + int((side*1.2)/2)):
31                  posList.pop(i)
32
33
34      with open("VideoPickle", "wb") as f:
35          pickle.dump(posList, f)
```

La scelta di realizzare questo file è dovuta ad una maggior velocità ed efficienza nell'adattarsi a diverse riprese di aule studio e nel caso del video usato di aggiunte di posti non previsti dall'aula (come si vede in foto).



Un'anteprima sull'implementazione può essere efficace per comprendere l'intero funzionamento scritto all'interno del file main.py. L'idea di base è quella di sfruttare un dizionario, *posDict,* che ha come chiave le posizioni precedentemente definite tramite SpacePicker (nella scena ripresa, tutte le sedie tranne una non possono essere spostate, ne risulta che i posti sono fissi) e come valore il numero di frame precedenti per cui quella posizione è risultata occupata secondo il nostro algoritmo.

```
15
16  def list2dict(lst: list) -> dict:
17      res_dct = {lst[i]: 0 for i in range(len(lst))}
18      return res_dct
19
```

La funzione crea un dizionario vuoto, "res_dct", e utilizza una *dict comprehension* per assegnare come chiave ogni elemento della lista "lst" e come valore "0".

Ad ogni posizione può essere associato al massimo un valore pari a FRAME_ROBUSTNESS che sta a significare che nei FRAME_ROBUSTNESS frame precedenti, la posizione è risultata occupata secondo l'algoritmo. Se una persona si alza dal suo posto e l'algoritmo riconosce questo comportamento, il valore della posizione verrà decrementato di uno per ogni frame in cui risulterà libera fino a raggiungere lo zero. Se il valore associato ad ogni posizione scende sotto una certa soglia (FRAME_THRESHOLD), il posto sarà considerato come libero ,ciò significa che per FRAME_ROBUSTNESS - FRAME_THRESHOLD frame non è stata riconosciuta nessuna persona seduta. Se invece un posto risulta libero ma successivamente viene occupato per FRAME_THRESHOLD frame, questo risulterà come occupato.

```python
def update_dict(centers: list, posDict: dict) -> None:
    #posList_tmp = deepcopy(posList)
    posList_tmp = list(posDict.keys())

    for center in centers:
        for pos in posList_tmp:
            if occupied(center, pos):
                posList_tmp.remove(pos)
                if posDict[pos] < FRAME_ROBUSTNESS:
                    posDict[pos] += 1
                break

    for pos in posList_tmp:
        if posDict[pos] > 0:
            posDict[pos] -= 1

```

Questa implementazione è atta a garantire più robustezza all'algoritmo ed evitare che i posti risultino immediatamente liberi o occupati anche solo come conseguenza di un semplice passaggio di una persona nella traiettoria del posto. Cosa che può capitare soprattutto se la telecamera non è posizionata perpendicolarmente al piano da riprendere.

Questo comportamento può essere modificato andando ad agire sugli hyperparameters definiti all'inizio del codice. Questi sono:

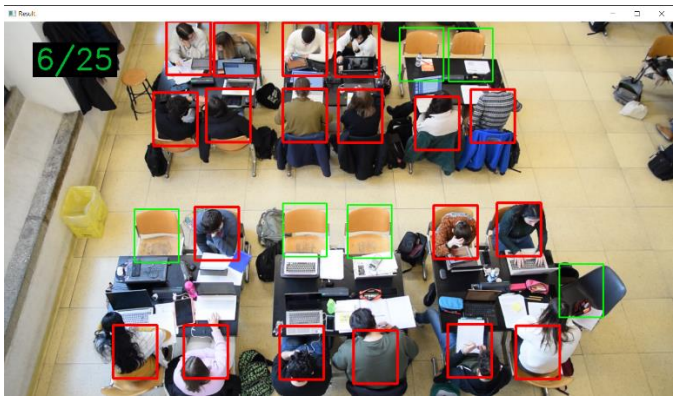
- DRAW_BOX : se definita True, disegna i box (e i loro centri) che la rete neurale YoloV4 è stata in grado di riconoscere.
- FRAME_ROBUSTNESS : numero massimo di frame che possono essere presi in considerazione per ogni posizione per determinare il suo andamento di occupazione.
- FRAME_THRESHOLD : se il valore della data posizione nel dizionario scende sotto questo valore, il posto risulta libero. Se superiore, risulterà occupato. È questa soglia che determina l'effettiva appartenenza del posto ad una delle due categorie e di conseguenza il colore del rettangolo attorno ad ogni posto.
- SIDE : grandezza del box così come viene prelevato dallo SpacePicker. Tramite questo parametro può essere ulteriormente modificato per renderlo più accurato.

Analizzando ulteriormente il codice abbiamo questo tipo di flow. All'interno del main, tramite la funzione json.load() carichiamo il file json su script python come una lista di dizionari, uno per ogni frame, dalla quale recupereremo gli oggetti rilevati dalla rete. A partire dalla lista di posizioni definite tramite SpacePicker, costruiamo il dizionario *posDict*, i cui valori per ogni posizione sono inizializzati a zero. All'interno del ciclo di lettura dei frame, sono stati presi in considerazione gli stessi frame analizzati dalla rete quindi 16 per ogni secondo. Abbiamo preso il dizionario corrispondente al frame corrente e abbiamo filtrato gli oggetti rilevati prendendo in considerazione solo quelli di classe 'person'. Questo filtraggio è stato implementato tramite *list comprehension* in modo da garantire più efficienza. Dopo il filtraggio, abbiamo iterato sugli oggetti rimasti e trasformato le coordinate relative dei box dell'oggetto json in coordinate utili a calcolare i centri di ogni persona rilevata nel frame corrente ed eventualmente disegnarne il box. Non è stata presa in considerazione nessuna threshold per la probabilità che il modello assegnava ad ogni oggetto rilevato.

Ottenuti i centri abbiamo aggiornato il dizionario, aggiungendo 1 al valore corrispondente se questa posizione risultava occupata (il centro del box calcolato era all'interno del box definito tramite SpacePicker) o sottraendo 1 altrimenti (sempre facendo in modo che il valore cadesse tra 0 e FRAME_ROBUSTNESS compresi). Fatto ciò, abbiamo definito le caratteristiche del box da disegnare, color e thickness, per ogni posizione in funzione del rapporto tra il corrispettivo valore e FRAME_THRESHOLD.

Nella fase iniziale dell'algoritmo, i box sono colorati in grigio per FRAME_ROBUSTNESS frame in modo da garantire un minimo di convergenza iniziale.

Dopodiché stampiamo in output e a video il numero di posti disponibili su quelli totali.




## Conclusione:

Il progetto da noi realizzato rimane comunque un prototipo con un giusto margine di miglioramento su più aspetti. Per quanto riguarda soprattutto la realizzazione della ripresa una possibile miglioria è cercare di riprendere l'aula studio dall'alto così che i posti vengono tutti visualizzati con la solita inclinazione, cercando di aver comunque un'illuminazione costante e un'ottima qualità video per permettere anche alla rete di svolgere al meglio il suo lavoro. Per quanto riguarda il codice invece ci sarebbe la possibilità di realizzare la versione real-time disponendo di computer adatti e quindi non passando per il servizio di google colab.

## Riferimenti:

Riportiamo qui le librerie e i framework più importanti usati:

### [1]OpenCV:

OpenCV (Open Source Computer Vision) è una libreria open source di computer vision che fornisce funzionalità di elaborazione delle immagini e del video per diverse applicazioni.

### [2]Google colab-yolov4:

Link per il foglio di google colab nel quale viene clonata il repository creato da noi a partire da quello di darknet per poter caricare più velocemente tutti i frame del video d'esempio e poter così usare la rete senza problemi.

https://colab.research.google.com/drive/1sK-jeNQ-LTnPlY4PSZ3U9lzX3k441nse#scrollTo=MRsBt8Z8obcn

### [3]Darknet:

Darknet è un neural network framework usato per poter riprodurre la rete YOLOv4 sul proprio pc.
https://github.com/AlexeyAB/darknet

### [4]Pickle:

Pickle è un modulo integrato in Python che viene utilizzato per serializzare e deserializzare oggetti Python. La serializzazione consiste nel convertire un oggetto Python in una stringa di byte, mentre la deserializzazione consiste nel convertire una stringa di byte in un oggetto Python