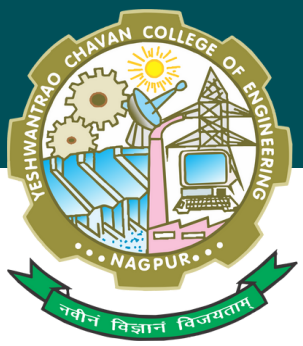
# Intel Unnati

# Project Report

## Social Distancing project using Computer Vision and Deep Learning

## Team: Tech Busters




**YESHWANTRAO CHAVAN COLLEGE OF ENGINEERING**

**TEAM MEMBERS:**

| Names | Student/ Mentor |
|---|---|
| 1. Dr. Gauri Dhopavkar | Mentor |
| 2. Paresh Hambarde | Student |
| 3. Sanket Chamate | Student |

**Date of Submission: 15/07/2023**

# Abstract

The purpose of this project is to provide an effective social distance monitoring solution. The raging coronavirus disease 2019 (COVID-19) brought a global crisis with its deadly spread all over the world. At the time of crisis when there is no effective cure, personal prevention actions are the only way to cope up with the situation. By using computer vision algorithms, our goal is to develop a robust system capable of accurately identifying and tracking individuals in crowded environments such as public spaces, workplaces, and retail settings.

# Introduction:

This project focuses on addressing the crucial issue of social distancing violations using computer vision and deep learning techniques. By using Deep Sort algorithm, MobileNetV2 and OmniScaleNet, the project aims to develop a robust system capable of real-time detection and tracking individuals in crowd environments. By extracting features and estimating proximity, the system can identify instance of violation and provide real-time feedback and alerts.

# Motivation:

The motivation behind this project is to be ready for any future pandemic, there can be situation where we don't have effective cure for certain diseases but due to personal prevention actions like washing hands, maintaining appropriate distance between individuals, etc. can help in preventing the worst situation.
Efficiently detecting and tracking objects, particularly people, in video streams or surveillance footage can provide valuable insights and enable various applications. By accurately detecting and tracking individuals, it becomes possible to analyze their behavior, count their occurrences, and identify potential anomalies or security risks.

# prior work/background:

As a student of computer science Engineering we have the basic knowledge of ML techniques and neural network. To develop this project, we additionally studied different algorithms used for social distancing and its type. Researched about 'openvino' library and implemented it in our project.

# Our approach:

- **For unoptimized model :**

1. Importing necessary libraries: The code begins by importing required libraries such as `time`, `math`, `cv2` (OpenCV), `numpy`, and `collections`.
2. Defining thresholds and parameters: The code sets threshold values for confidence (`confid`) and distance (`thresh`) to determine close pairs. It also defines other parameters such as the video file name (`vid_file_name`), angle factor, and Euclidean distance calculation function (`dist`).
3. Implementing helper functions: The code defines helper functions `tan2sine` and `tan2cosine` to calculate sine and cosine values based on the tangent value.
4. Loading YOLOv3-tiny model and labels: The code loads the pre-trained YOLOv3-tiny model by specifying the paths to the weights and configuration files. It also loads the labels from the COCO dataset.

5. Initializing video capture: The code initializes video capture by creating an instance of `cv2.VideoCapture` and setting the path to the video file.
6. Main loop: The code enters a main loop to process each frame of the video. It reads each frame using `cap.read()` and breaks the loop if no frame is grabbed.
7. Preprocessing the frame: The frame is preprocessed by resizing it to the input size of the YOLO model (416x416) using `cv2.dnn.blobFromImage`. The preprocessed frame is then set as the input to the YOLO model.
8. Object detection: The preprocessed frame is passed through the YOLO model using `net.forward(ln)`, where `ln` contains the names of the output layers. The output of the model is used to extract bounding box coordinates, confidence scores, and class IDs for detected objects.
9. Non-Maximum Suppression (NMS): The code applies NMS to remove duplicate bounding boxes and keep only the most relevant ones using `cv2.dnn.NMSBoxes`.
10. Social distancing calculation: For each detected person, the code calculates the center point and other information. It then iterates over all pairs of detected people and determines if they are close based on predefined criteria. If a close pair is found, it updates the status of the individuals accordingly.
11. Visualization and output: The code displays the FPS and inference time on the frame using `cv2.putText` and draws bounding boxes around individuals, highlighting those at risk in red and safe individuals in green. The processed frame is shown in a window using `cv2.imshow`.

- **For optimized model: (using openvino):**

The given code implements a person tracking and social distancing detection system using OpenVINO toolkit and DeepSORT algorithm. The code imports various libraries and modules required for image and video processing, including OpenCV, NumPy, and Matplotlib. It also imports specific modules and classes from the OpenVINO toolkit, such as Core and Model, to load and run pre-trained models for person detection and re-identification.

The approach starts by defining utility functions for distance calculation and coordinate conversion. Then, the person detection and re-identification models are loaded using the OpenVINO toolkit. The `preprocess` function is used to resize and format the input frame for inference, while the `process_results` function processes the output of the person detection model to obtain bounding box coordinates, labels, and scores for detected persons.

The `draw_boxes` function is responsible for drawing bounding boxes and labels on the original frame, as well as identifying and marking violations of social distancing rules. It uses a centroid-based approach to calculate the distance between detected persons and determines whether they violate the social distancing threshold. The number of violations is counted and displayed on the frame, and the risk level of the scene is determined based on the number of violations.
The code also includes functions for cosine distance calculation, video preprocessing, and batch processing of image crops. The `run_person_tracking` function is the main processing function that performs person tracking and visualization. It uses the DeepSORT algorithm to predict and update the positions of tracked targets based on the detections and re-identification features. The tracked targets' bounding boxes are drawn on the frame, along with their associated identities.

The processed frames are displayed using either OpenCV's `imshow` function or as HTML images in a Jupyter notebook, depending on the `use_popup` parameter. Additionally, the processed frames can be saved to a video file using OpenCV's `VideoWriter` class.

# Results:

- **After the execution of unoptimized model we got the latency and throughput as follows:**

---- For YOLOV3 Tiny:

Average Latency: ~30 ms

Throughput: ~33 FPS

- **After execution of the Optimized model we got the latency and throughput parameters as follows:**

---- For MobileNetV2:

Average Latency: 53.82 ms

Throughput: 74.03 FPS

---- For OmniScaleNet:

Average Latency: 8.10 ms

Throughput: 981.13 FPS

Note: Please refer data/openvino_benchmark.txt (link) for benchmark related information.

# Links:

- Code/Result: (Github link)
- Model: (Github link)