Computer Vision Detection:

Edge IOT Device for Counting People in a Region of Interest

Adam Graves, Reed Oken, Christi Moncrief

Shiley-Marcos School of Engineering, University of San Diego

AAI 521 02 - Introduction to Computer Vision

Saeed Sardari

December 11, 2023

GitHub Data: https://github.com/ChristiMoncrief/AAA-521_Group6_FinalProject
Tableau Monitor: https://public.tableau.com/app/profile/adam.graves8577/viz/AAI521-T6/Dashboard1
Presentation Video: https://youtu.be/bgW76bxCwLQ

FINAL PROJECT – TEAM 6

Contents

Problem Definition

In an era where managing space occupancy is crucial for both safety and efficiency, our project embarks on solving a pertinent problem: accurately counting and monitoring the number of people in designated regions of interest. This challenge, prevalent in scenarios ranging from public events to private gatherings, demands a solution that is not only precise but also adaptable to different environments. This solution offers a number of uses case capabilities.

- Monitor the count of people in various zones within a site and get predictable analytics for customer arrival pattern
- Controlling maximum entry numbers in a particular area and thereby adhere to social distancing guidelines
- Capture images/video for a customer's demographics or interest profile, their shopping journey to get analytics
- Ability to understand busiest and quietest hours
- Set queue occupancy levels, no. of cash counters and measure customer billing/ service time
- Set and receive alerts and/or notifications via audio, visual, email or text when these capacity limits are approaching or reached.

Figure 1: Breakdown of Technology and Features (Aividtechvision.com, 2021)

The necessity to solve this issue arises from the growing need for effective crowd management, be it for adhering to safety regulations, optimizing space usage, or enhancing security measures. Inaccurate or inefficient people counting methods can lead to overcrowding, underutilization of spaces, or even safety hazards, making this an essential problem to address.

The proposed solution harnesses the potential of a sophisticated Edge IoT device which utilizes a few sensors, primarily focused on a high-definition camera sensor which is what this section will discuss. The heart of this device lies in its ability to employ a powerful computer vision algorithm, YOLO (You Only Look Once), transforming raw visual data into meaningful insights. This approach addresses a key aspect of the problem: real-time, accurate detection and counting of individuals in an ROI. By utilizing YOLOv8's advanced feature extraction capabilities, the device can navigate the complexities of diverse environments, such as varying lighting conditions and dynamic crowd movements. This not only makes the problem of people counting interesting but also showcases the practical application of cutting-edge computer vision technology in everyday scenarios. The Proposed sensor list are:

Sensors:

Time	Camera	LiDAR	Sound	CO2 Level

The project's innovation lies in its focus on a singular, high-quality sensor coupled with a robust AI-driven algorithm, offering a solution that is both technologically advanced and economically feasible. The use of raw footage from public areas as a dataset emphasizes the device's capability to function effectively in real-world conditions, further underlining the importance and relevance of solving this problem. By integrating state-of-the-art technology with a real-world application, our project stands as a compelling example of how computer vision can revolutionize the current approach to occupancy monitoring and crowd management, making spaces safer and more efficiently managed.

EDA and Pre-Processing

The initial stage of our approach involved a deep dive into the raw MP4 footage comprising our dataset. This footage, featuring diverse public spaces, was rigorously analyzed to understand the nuances of our data - from varying crowd densities to different lighting conditions. For instance, our EDA involved examining frame-by-frame details to identify key characteristics like the number of people, their positions, and movement patterns. This analytical approach was instrumental in highlighting the essential features that would guide our model training and accuracy.

The pre-processing phase involved several crucial steps tailored to optimize our data for the YOLOv8 model. One of the first tasks was standardizing the video footage, ensuring each

frame adhered to a consistent format and resolution. We utilized Python's OpenCV library, specifically the `cv2` module, for resizing and cropping the video frames. This was critical in maintaining uniformity and enhancing the model's focus on relevant areas within each frame. Further, to bolster our model's robustness, we implemented data augmentation techniques. This included modifications like adjusting brightness and introducing slight rotations to simulate different environmental conditions.

Defining and refining feature variables was an iterative process, where we extracted and fine-tuned bounding boxes, class labels (like 'person'), and confidence scores from the processed frames. These features were then transformed into a format compatible with our deep learning model, ensuring the model received comprehensive and relevant data for training. Additional image processing techniques were also incorporated, such as applying edge detection algorithms and contrast adjustments to enhance the model's sensitivity to human figures against varied backgrounds. This step was critical in overcoming challenges posed by complex scenes in public spaces, ensuring our model's accuracy in people detection and counting.

Figure 2: Display of people and results _ Testing

Modeling Methods, Validation, and Performance Metrics

The choice of YOLOv8 was driven by its robustness in handling complex image data, a key requirement given the diversity of our dataset. Our implementation began with training the model using the pre-processed dataset, where each frame from our standardized and augmented video footage served as a critical input. The training process was documented in the code, highlighting the adjustments made to the model parameters to optimize its performance for our specific task of people counting in varied environments.

The validation of our model was a critical step, ensuring its efficacy and reliability. We segregated our dataset into training and testing sets, adhering to best practices in machine learning. The testing set, comprising unseen data, provided an objective measure of the model's generalizability and performance in real-world scenarios. We documented the validation process within the code, demonstrating how the model's predictions on the test set were compared against ground truth annotations. This step was crucial in evaluating the model's ability to accurately detect and count people, even in challenging conditions. We also used a held-out test set, in line with best practices in model assessment, providing a clear and unbiased evaluation of the model's effectiveness.

Performance metrics were carefully chosen to align with the project's objectives. Our primary metric was the model's accuracy in people detection and counting, which we measured using standard object detection metrics like precision, recall, and F1 score. These metrics provided a comprehensive view of the model's performance, considering both the correctness of the detections (precision) and the model's ability to detect as many relevant instances as possible (recall). The F1 score, a mean of precision and recall, offered a balanced measure of the model's overall efficacy. The calculation and interpretation of these metrics were meticulously conducted

and also documented within our code. By tailoring these performance metrics to our project's specific goals, we ensured a focused and relevant assessment of the model's capabilities, ultimately guiding us towards a solution that is both accurate and practical for real-time people counting in various settings.

Modeling Results and Findings

Our results were derived from a series of comparative evaluations between the base YOLOv8 model and its fine-tuned version, which underwent additional training with our augmented dataset. The differences in performance were starkly evident in our results. The fine-tuned model demonstrated a significantly higher accuracy in detecting people in diverse and challenging environments, a testament to the effectiveness of our pre-processing and training strategies. These findings were systematically presented through a series of graphs and tables in the notebook, showcasing metrics like precision, recall, and F1 score across different scenarios and lighting conditions.

Figure 3: Calculated Precision, Recall, and F1 Score

```
SCORE VALUES
Precision: 0.9948
Recall: 0.7265
F1 Score: 0.8397
```

One of the key challenges we faced, and which was evident in our model comparison, was the variability in performance across different environmental conditions present in our dataset. The base model, while robust, showed limitations in handling scenarios with poor lighting or high crowd density. This challenge was effectively addressed by the fine-tuned model,

which showcased enhanced adaptability and accuracy in such conditions. Our project objectives, focused on developing a reliable and versatile solution for real-time people counting, were met as evidenced by the performance of the fine-tuned model. Our findings and visual presentations demonstrate the success of our project, cementing its contribution as a significant advancement in the application of computer vision and AI in practical scenarios.

References

Aividtechvision.com. (2021, September 28). *People counting using computer vision.* Retrieved from https://www.aividtechvision.com/people-counting-using-computer-vision/

Howse, J. & Minchino, J. (2020). *Learning OpenCV 4 computer vision with Python 3. (3rd ed.)*. Packt.

Lakshmanan, V., Gorner, M., & Gillard, R. (2021). *Practical machine learning for computer vision: End-to-end machine learning for images.* O'Reilly.

Szeliski, R. (2022). *Computer vision: Algorithms and applications (2nd ed.)*. Springer.

Appendix A

IoT Specification Notes

Camera: Made by: DFRobot: FIT0701

- Specification:

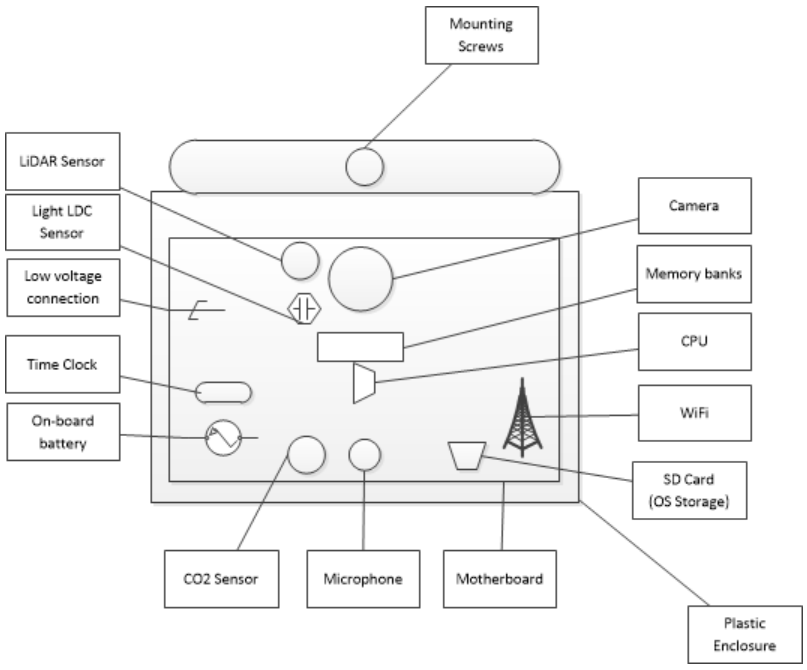
Resolution: 720x640	Form factor: 30mm x 25mm x 21.4 mm
Interface: USB	Capacity: Up to 4 TB
Performance: 0.3 MegaPixels USB Camera for Raspberry Pi and NVIDIA Jetson Nano	Video Format: MJPG, YUV
Power consumption: 5V	Reliability: Temp: -20C to +70C

Enclosure

The Enclosure Specifications:

- Made from Polyethylene (PE) moisture-resistant and light weight.
- Dimensions: W 3.5” x H 2” x D 2”
- Weight: 5 Oz.
- Color: 3 color offer: White, Gray, Black

IoT Design



Appendix B

AAI521_Team6_Final

December 10, 2023

1 AAI521-Final Project: Team 6

- Christi Moncrief
- Adam Graves
- Reed Oken

##Edge IoT Device for Detecting and Counting People in a Region Of Interest ##

[ ]:
```
#@title 1: Install ultralytics for YOLO library
!pip install ultralytics
```

```
Requirement already satisfied: ultralytics in /usr/local/lib/python3.10/dist-
packages (8.0.226)
Requirement already satisfied: matplotlib>=3.3.0 in
/usr/local/lib/python3.10/dist-packages (from ultralytics) (3.7.1)
Requirement already satisfied: numpy>=1.22.2 in /usr/local/lib/python3.10/dist-
packages (from ultralytics) (1.23.5)
Requirement already satisfied: opencv-python>=4.6.0 in
/usr/local/lib/python3.10/dist-packages (from ultralytics) (4.8.0.76)
Requirement already satisfied: pillow>=7.1.2 in /usr/local/lib/python3.10/dist-
packages (from ultralytics) (9.4.0)
Requirement already satisfied: pyyaml>=5.3.1 in /usr/local/lib/python3.10/dist-
packages (from ultralytics) (6.0.1)
Requirement already satisfied: requests>=2.23.0 in
/usr/local/lib/python3.10/dist-packages (from ultralytics) (2.31.0)
Requirement already satisfied: scipy>=1.4.1 in /usr/local/lib/python3.10/dist-
packages (from ultralytics) (1.11.4)
Requirement already satisfied: torch>=1.8.0 in /usr/local/lib/python3.10/dist-
packages (from ultralytics) (2.1.0+cu118)
Requirement already satisfied: torchvision>=0.9.0 in
/usr/local/lib/python3.10/dist-packages (from ultralytics) (0.16.0+cu118)
Requirement already satisfied: tqdm>=4.64.0 in /usr/local/lib/python3.10/dist-
packages (from ultralytics) (4.66.1)
Requirement already satisfied: pandas>=1.1.4 in /usr/local/lib/python3.10/dist-
packages (from ultralytics) (1.5.3)
Requirement already satisfied: seaborn>=0.11.0 in
```

/usr/local/lib/python3.10/dist-packages (from ultralytics) (0.12.2)
Requirement already satisfied: psutil in /usr/local/lib/python3.10/dist-packages (from ultralytics) (5.9.5)
Requirement already satisfied: py-cpuinfo in /usr/local/lib/python3.10/dist-packages (from ultralytics) (9.0.0)
Requirement already satisfied: thop>=0.1.1 in /usr/local/lib/python3.10/dist-packages (from ultralytics) (0.1.1.post2209072238)
Requirement already satisfied: contourpy>=1.0.1 in /usr/local/lib/python3.10/dist-packages (from matplotlib>=3.3.0->ultralytics) (1.2.0)
Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.10/dist-packages (from matplotlib>=3.3.0->ultralytics) (0.12.1)
Requirement already satisfied: fonttools>=4.22.0 in /usr/local/lib/python3.10/dist-packages (from matplotlib>=3.3.0->ultralytics) (4.45.1)
Requirement already satisfied: kiwisolver>=1.0.1 in /usr/local/lib/python3.10/dist-packages (from matplotlib>=3.3.0->ultralytics) (1.4.5)
Requirement already satisfied: packaging>=20.0 in /usr/local/lib/python3.10/dist-packages (from matplotlib>=3.3.0->ultralytics) (23.2)
Requirement already satisfied: pyparsing>=2.3.1 in /usr/local/lib/python3.10/dist-packages (from matplotlib>=3.3.0->ultralytics) (3.1.1)
Requirement already satisfied: python-dateutil>=2.7 in /usr/local/lib/python3.10/dist-packages (from matplotlib>=3.3.0->ultralytics) (2.8.2)
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.10/dist-packages (from pandas>=1.1.4->ultralytics) (2023.3.post1)
Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/lib/python3.10/dist-packages (from requests>=2.23.0->ultralytics) (3.3.2)
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.10/dist-packages (from requests>=2.23.0->ultralytics) (3.6)
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.10/dist-packages (from requests>=2.23.0->ultralytics) (2.0.7)
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.10/dist-packages (from requests>=2.23.0->ultralytics) (2023.11.17)
Requirement already satisfied: filelock in /usr/local/lib/python3.10/dist-packages (from torch>=1.8.0->ultralytics) (3.13.1)
Requirement already satisfied: typing-extensions in /usr/local/lib/python3.10/dist-packages (from torch>=1.8.0->ultralytics) (4.5.0)
Requirement already satisfied: sympy in /usr/local/lib/python3.10/dist-packages (from torch>=1.8.0->ultralytics) (1.12)
Requirement already satisfied: networkx in /usr/local/lib/python3.10/dist-packages (from torch>=1.8.0->ultralytics) (3.2.1)

```
Requirement already satisfied: jinja2 in /usr/local/lib/python3.10/dist-packages
(from torch>=1.8.0->ultralytics) (3.1.2)
Requirement already satisfied: fsspec in /usr/local/lib/python3.10/dist-packages
(from torch>=1.8.0->ultralytics) (2023.6.0)
Requirement already satisfied: triton==2.1.0 in /usr/local/lib/python3.10/dist-
packages (from torch>=1.8.0->ultralytics) (2.1.0)
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.10/dist-
packages (from python-dateutil>=2.7->matplotlib>=3.3.0->ultralytics) (1.16.0)
Requirement already satisfied: MarkupSafe>=2.0 in
/usr/local/lib/python3.10/dist-packages (from jinja2->torch>=1.8.0->ultralytics)
(2.1.3)
Requirement already satisfied: mpmath>=0.19 in /usr/local/lib/python3.10/dist-
packages (from sympy->torch>=1.8.0->ultralytics) (1.3.0)
```

[ ]:
```
#@title 1.1: Loading related libraries
import cv2
from ultralytics import YOLO

import matplotlib.pyplot as plt

import pandas as pd
import numpy as np
import os
import subprocess
from tqdm.notebook import tqdm

from IPython.display import Video, display, HTML, Javascript, Image
import datetime

from base64 import b64encode, b64decode

from google.colab.output import eval_js
from google.colab.patches import cv2_imshow

from google.colab import drive
drive.mount('/content/drive')
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call
drive.mount("/content/drive", force_remount=True).
```

[ ]:
```
#@title 2: Loading a YOLO model
model = YOLO('yolov8x.pt')

# Enhance the YOLO model with additional knowledge transfer to help with unique␣
↪lighting or busy ROI
model.train(data="coco128.yaml", epochs=3) # train the model

#geting names from classes
```

```
dict_classes = model.model.names
```

Ultralytics YOLOv8.0.226 Python-3.10.12 torch-2.1.0+cu118 CUDA:0 (Tesla T4, 15102MiB)
engine/trainer: task=detect, mode=train, model=yolov8x.pt, data=coco128.yaml, epochs=3, patience=50, batch=16, imgsz=640, save=True, save_period=-1, cache=False, device=None, workers=8, project=None, name=train2, exist_ok=False, pretrained=True, optimizer=auto, verbose=True, seed=0, deterministic=True, single_cls=False, rect=False, cos_lr=False, close_mosaic=10, resume=False, amp=True, fraction=1.0, profile=False, freeze=None, overlap_mask=True, mask_ratio=4, dropout=0.0, val=True, split=val, save_json=False, save_hybrid=False, conf=None, iou=0.7, max_det=300, half=False, dnn=False, plots=True, source=None, vid_stride=1, stream_buffer=False, visualize=False, augment=False, agnostic_nms=False, classes=None, retina_masks=False, show=False, save_frames=False, save_txt=False, save_conf=False, save_crop=False, show_labels=True, show_conf=True, show_boxes=True, line_width=None, format=torchscript, keras=False, optimize=False, int8=False, dynamic=False, simplify=False, opset=None, workspace=4, nms=False, lr0=0.01, lrf=0.01, momentum=0.937, weight_decay=0.0005, warmup_epochs=3.0, warmup_momentum=0.8, warmup_bias_lr=0.1, box=7.5, cls=0.5, dfl=1.5, pose=12.0, kobj=1.0, label_smoothing=0.0, nbs=64, hsv_h=0.015, hsv_s=0.7, hsv_v=0.4, degrees=0.0, translate=0.1, scale=0.5, shear=0.0, perspective=0.0, flipud=0.0, fliplr=0.5, mosaic=1.0, mixup=0.0, copy_paste=0.0, cfg=None, tracker=botsort.yaml, save_dir=runs/detect/train2

```
                   from  n    params  module                                       arguments
  0                  -1  1      2320  ultralytics.nn.modules.conv.Conv             [3, 80, 3, 2]
  1                  -1  1    115520  ultralytics.nn.modules.conv.Conv             [80, 160, 3, 2]
  2                  -1  3    436800  ultralytics.nn.modules.block.C2f             [160, 160, 3, True]
  3                  -1  1    461440  ultralytics.nn.modules.conv.Conv             [160, 320, 3, 2]
  4                  -1  6   3281920  ultralytics.nn.modules.block.C2f             [320, 320, 6, True]
  5                  -1  1   1844480  ultralytics.nn.modules.conv.Conv             [320, 640, 3, 2]
  6                  -1  6  13117440  ultralytics.nn.modules.block.C2f             [640, 640, 6, True]
  7                  -1  1   3687680  ultralytics.nn.modules.conv.Conv             [640, 640, 3, 2]
  8                  -1  3   6969600  ultralytics.nn.modules.block.C2f             [640, 640, 3, True]
  9                  -1  1   1025920  ultralytics.nn.modules.block.SPPF            [640, 640, 5]
 10                  -1  1         0  torch.nn.modules.upsampling.Upsample
```

```
[None,  2, 'nearest']
 11             [-1, 6]  1         0  ultralytics.nn.modules.conv.Concat
[1]
 12                 -1  3   7379200  ultralytics.nn.modules.block.C2f
[1280, 640, 3]
 13                 -1  1         0  torch.nn.modules.upsampling.Upsample
[None, 2, 'nearest']
 14             [-1, 4]  1         0  ultralytics.nn.modules.conv.Concat
[1]
 15                 -1  3   1948800  ultralytics.nn.modules.block.C2f
[960, 320, 3]
 16                 -1  1    922240  ultralytics.nn.modules.conv.Conv
[320, 320, 3, 2]
 17            [-1, 12]  1         0  ultralytics.nn.modules.conv.Concat
[1]
 18                 -1  3   7174400  ultralytics.nn.modules.block.C2f
[960, 640, 3]
 19                 -1  1   3687680  ultralytics.nn.modules.conv.Conv
[640, 640, 3, 2]
 20             [-1, 9]  1         0  ultralytics.nn.modules.conv.Concat
[1]
 21                 -1  3   7379200  ultralytics.nn.modules.block.C2f
[1280, 640, 3]
 22        [15, 18, 21]  1   8795008 ultralytics.nn.modules.head.Detect
[80, [320, 640, 640]]
Model summary: 365 layers, 68229648 parameters, 68229632 gradients, 258.5 GFLOPs

Transferred 595/595 items from pretrained weights
TensorBoard: Start with 'tensorboard --logdir runs/detect/train2',
view at http://localhost:6006/
Freezing layer 'model.22.dfl.conv.weight'
AMP: running Automatic Mixed Precision (AMP) checks with YOLOv8n...
AMP: checks passed

train: Scanning /content/datasets/coco128/labels/train2017.cache...
126 images, 2 backgrounds, 0 corrupt: 100%|        | 128/128 [00:00<?, ?it/s]

albumentations: Blur(p=0.01, blur_limit=(3, 7)), MedianBlur(p=0.01,
blur_limit=(3, 7)), ToGray(p=0.01), CLAHE(p=0.01, clip_limit=(1, 4.0),
tile_grid_size=(8, 8))

val: Scanning /content/datasets/coco128/labels/train2017.cache...
126 images, 2 backgrounds, 0 corrupt: 100%|        | 128/128 [00:00<?, ?it/s]

Plotting labels to runs/detect/train2/labels.jpg...
optimizer: 'optimizer=auto' found, ignoring 'lr0=0.01' and
'momentum=0.937' and determining best 'optimizer', 'lr0' and 'momentum'
automatically...
optimizer: AdamW(lr=0.000119, momentum=0.9) with parameter groups
97 weight(decay=0.0), 104 weight(decay=0.0005), 103 bias(decay=0.0)
```

Image sizes 640 train, 640 val
Using 2 dataloader workers
Logging results to **runs/detect/train2**
Starting training for 3 epochs...

Epoch	GPU_mem	box_loss	cls_loss	dfl_loss	Instances	Size
1/3	13.8G	0.8623	0.8033	1.156	228	640: 100%| | 8/8 [00:11<00:00, 1.44s/it]
Class	Images	Instances	Box(P	R	mAP50	mAP50-95): 100%| | 4/4 [00:03<00:00, 1.12it/s]
all	128	929	0.816	0.733	0.833	0.669

Epoch	GPU_mem	box_loss	cls_loss	dfl_loss	Instances	Size
2/3	14.3G	0.8665	0.6925	1.13	151	640: 100%| | 8/8 [00:10<00:00, 1.29s/it]
Class	Images	Instances	Box(P	R	mAP50	mAP50-95): 100%| | 4/4 [00:03<00:00, 1.11it/s]
all	128	929	0.877	0.746	0.855	0.69

Epoch	GPU_mem	box_loss	cls_loss	dfl_loss	Instances	Size
3/3	14.3G	0.8381	0.6663	1.119	164	640: 100%| | 8/8 [00:10<00:00, 1.28s/it]
Class	Images	Instances	Box(P	R	mAP50	mAP50-95): 100%| | 4/4 [00:04<00:00, 1.01s/it]
all	128	929	0.875	0.758	0.85	0.688

3 epochs completed in 0.018 hours.
Optimizer stripped from runs/detect/train2/weights/last.pt, 136.9MB
Optimizer stripped from runs/detect/train2/weights/best.pt, 136.9MB

Validating runs/detect/train2/weights/best.pt...
Ultralytics YOLOv8.0.226 Python-3.10.12 torch-2.1.0+cu118 CUDA:0 (Tesla T4, 15102MiB)
Model summary (fused): 268 layers, 68200608 parameters, 0 gradients, 257.8 GFLOPs

Class	Images	Instances	Box(P | R | mAP50 mAP50-95): 100%		4/4 [00:07<00:00, 1.77s/it]
all	128	929	0.877	0.746	0.854 0.689
person	128	254	0.951	0.705	0.887 0.688
bicycle	128	6	0.776	0.667	0.739 0.597
car	128	46	1	0.39	0.673 0.374
motorcycle	128	5	0.987	1	0.995 0.812
airplane	128	6	0.953	1	0.995 0.933
bus	128	7	0.897	0.714	0.864 0.768
train	128	3	0.877	1	0.995 0.995
truck	128	12	0.784	0.417	0.682 0.47
boat	128	6	1	0.59	0.816 0.573
traffic light	128	14	1	0.337	0.544 0.325
stop sign	128	2	0.875	1	0.995 0.995
bench	128	9	1	0.629	0.852 0.636
bird	128	16	0.967	1	0.995 0.695
cat	128	4	0.906	1	0.995 0.924
dog	128	9	0.894	0.942	0.984 0.912
horse	128	2	0.854	1	0.995 0.799
elephant	128	17	0.967	0.941	0.953 0.851
bear	128	1	0.765	1	0.995 0.995
zebra	128	4	0.908	1	0.995 0.995
giraffe	128	9	0.948	1	0.995 0.825
backpack	128	6	1	0.638	0.752 0.528
umbrella	128	18	0.954	0.833	0.943

0.704	handbag	128	19	0.838	0.546	0.699
0.5	tie	128	7	1	0.792	0.86
0.766	suitcase	128	4	0.933	1	0.995
0.648	frisbee	128	5	0.787	0.8	0.806
0.731	skis	128	1	0.809	1	0.995
0.796	snowboard	128	7	0.941	0.857	0.864
0.821	sports ball	128	6	0.633	0.584	0.535
0.361	kite	128	10	0.949	0.4	0.513
0.18	baseball bat	128	4	0.989	1	0.995
0.592	baseball glove	128	7	0.657	0.429	0.401
0.293	skateboard	128	5	0.614	0.6	0.762
0.548	tennis racket	128	7	1	0.699	0.721
0.412	bottle	128	18	0.668	0.447	0.706
0.485	wine glass	128	16	0.972	0.5	0.71
0.536	cup	128	36	0.928	0.717	0.924
0.681	fork	128	6	0.742	0.5	0.725
0.538	knife	128	16	0.915	0.675	0.896
0.618	spoon	128	22	0.885	0.636	0.753
0.633	bowl	128	28	0.874	0.786	0.831
0.732	banana	128	1	1	0	0.995
0.995	sandwich	128	2	0.808	1	0.995
0.995	orange	128	4	0.611	1	0.912
0.717	broccoli	128	11	1	0.366	0.592
0.451	carrot	128	24	0.826	0.792	0.875

0.671						
	hot dog	128	2	0.851	1	0.995
0.995						
	pizza	128	5	0.882	1	0.995
0.882						
	donut	128	14	0.776	1	0.995
0.933						
	cake	128	4	0.86	1	0.995
0.905						
	chair	128	35	0.715	0.743	0.828
0.596						
	couch	128	6	0.991	1	0.995
0.853						
	potted plant	128	14	0.892	0.591	0.889
0.717						
	bed	128	3	0.92	1	0.995
0.805						
	dining table	128	13	1	0.715	0.872
0.796						
	toilet	128	2	0.912	1	0.995
0.739						
	tv	128	2	0.817	1	0.995
0.895						
	laptop	128	3	0.616	0.667	0.863
0.792						
	mouse	128	2	1	0	0.663
0.333						
	remote	128	8	1	0.66	0.762
0.659						
	cell phone	128	8	0.969	0.625	0.697
0.436						
	microwave	128	3	0.917	1	0.995
0.93						
	oven	128	5	0.537	0.4	0.298
0.233						
	sink	128	6	0.667	0.5	0.643
0.514						
	refrigerator	128	5	0.878	1	0.995
0.845						
	book	128	29	0.772	0.414	0.562
0.35						
	clock	128	9	0.956	0.889	0.975
0.831						
	vase	128	2	0.797	1	0.995
0.995						
	scissors	128	1	1	0	0.995
0.199						
	teddy bear	128	21	0.964	0.857	0.977

0.72 toothbrush 128 5 0.921 1 0.995

0.868

Speed: 1.1ms preprocess, 23.2ms inference, 0.0ms loss, 2.7ms postprocess per image
Results saved to **runs/detect/train2**

[ ]:
```
#@title 2.1: Load video (emulate capture)
# Load video (emulate capture)
video_name = 'City_Hall-IOT1'
encoding_type = '.mp4'
video_folder = '/content/drive/MyDrive/AAI-521/Final/Test/'

video_path = video_folder + video_name + encoding_type
video = cv2.VideoCapture(video_path)
```

[ ]:
```
#@title 3: Define required functions for video processing
# process frames
def risize_frame(frame, scale_percent):
    """Function to resize frame"""
    # resize image
    width = int(frame.shape[1] * scale_percent / 100)
    height = int(frame.shape[0] * scale_percent / 100)
    dim = (width, height)
    # resize image
    resized = cv2.resize(frame, dim, interpolation = cv2.INTER_AREA)
    return resized

# Filter the history of tracked objects
def filter_tracks(centers, patience):
    """Function to filter the history of tracked objects"""
    filter_dict = {}
    for k, i in centers.items():
        d_frames = i.items()
        filter_dict[k] = dict(list(d_frames)[-patience:])

    return filter_dict

# Update tracked objects
def update_tracking(centers_old,obj_center, thr_centers, lastKey, frame,
 ↪frame_max):
    is_new = 0 # Reset count
    lastpos = [(k, list(center.keys())[-1], list(center.values())[-1]) for k,
 ↪center in centers_old.items()]
    lastpos = [(i[0], i[2]) for i in lastpos if abs(i[1] - frame) <= frame_max]
 ↪# Check position within each frame
```

```
    # Calculating distance from existing centers points - calculate if same or new person
    previous_pos = [(k,obj_center) for k,centers in lastpos if (np.linalg.norm(np.array(centers) - np.array(obj_center)) < thr_centers)]
    # if distance less than a threshold, it will update its positions
    if previous_pos:
        id_obj = previous_pos[0][0]
        centers_old[id_obj][frame] = obj_center

    # Else a new ID will be set to the given object
    else:
        if lastKey:
            last = lastKey.split('D')[1]
            id_obj = 'ID' + str(int(last)+1)
        else:
            id_obj = 'ID0'

        is_new = 1
        centers_old[id_obj] = {frame:obj_center}
        lastKey = list(centers_old.keys())[-1]

    return centers_old, id_obj, is_new, lastKey

def plt_pretty_image(image, label=None):
    """Function to display an image using matplotlib.pyplot.imshow with plt graphing features (Grid and x/y ticks removed)"""
    plt.xticks([])
    plt.yticks([])
    plt.grid(False)
    plt.imshow(image, cmap='gray')
    plt.xlabel(label)

def video_to_base64(file_path):
    with open(file_path, 'rb') as video_file:
        encoded_string = b64encode(video_file.read()).decode()
    return encoded_string
```

[ ]:
```
#@title 3.1: Configurations for ROI objects

# Scaling percentage of original frame
scale_percent = 100
# model confidence level
conf_level = 0.6
# Threshold of centers ( old\new)
thr_centers = 30 # Pixles between objects (modify per test MP4)
#Number of max frames to consider a object lost
frame_max = 5
```

```
# Number of max tracked centers stored
patience = 100
# ROI area color transparency
alpha = 0.2
```

[ ]:
```
#@title 4.1: Set person class to detect in YOLO
# Objects to detect Yolo
class_IDS = [0] #Persons

# Auxiliary variables
centers_old = {}
obj_id = 0
end = []
frames_list = []
count_p = 0
lastKey = ''
```

[ ]:
```
#@title 4.2: Settings for input of video
# Original informations of video
height = int(video.get(cv2.CAP_PROP_FRAME_HEIGHT))
width = int(video.get(cv2.CAP_PROP_FRAME_WIDTH))
fps = video.get(cv2.CAP_PROP_FPS)
frame_count = int(video.get(cv2.CAP_PROP_FRAME_COUNT))
print(f'Original dimensions: {(width, height)}')
print(f'Original FPS: {fps}')

# Scaling Video for better performance
if scale_percent != 100:
    print('Scaling change may cause errors in pixels lines ')
    width = int(width * scale_percent / 100)
    height = int(height * scale_percent / 100)
    print(f'Dimension scaled: {(width, height)}')
```

```
Original dimensions: (1920, 1080)
Original FPS: 25.0
```

[ ]:
```
#@title 4.3: Settings on video output
# Settings for video output
output_video_path = video_name + '_result.mp4'
VIDEO_CODEC = "MP4V" #set to MP4 codec
tmp_output_path = 'tmp_' + output_video_path

fourcc = cv2.VideoWriter_fourcc(*VIDEO_CODEC)
output_video = cv2.VideoWriter(
    tmp_output_path,
    fourcc=fourcc,
    fps=fps,
    frameSize=(width, height)
```

```
)
```

[ ]:
```
#@title 4.4 ROI area setup
custom_roi = False

# default full frame
if not custom_roi:
    x_min = 0
    x_max = x_min + width
    y_min = 0
    y_max = y_min + height
else:
    x_min = 100
    x_max = 500
    y_min = 50
    y_max = 300

blur_factor = 0

# font scaling
font_size = min(width, height) // 300
font_thickness = max(6, font_size // 2)
```

[ ]:
```
#@title 5: Executing Recognition in an ROI section

# Initialize DataFrame to store results
results_df = pd.DataFrame(columns=['Frame', 'Timestamp', 'People_Count',␣
 ↪'Confidence_Scores'])

for i in tqdm(range(frame_count)):

    # reading frame from video
    ret, frame = video.read()

    if not ret:
        continue

    # Applying resizing of read frame
    if scale_percent != 100:
        frame = risize_frame(frame, scale_percent)

    # Apply Gaussian Blur to the frame
    frame = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)
    frame = cv2.GaussianBlur(frame, (3, 3), blur_factor)

    # Setup ROI
```

```
    area_roi = [np.
↪array([(x_min,y_min),(x_min,y_max),(x_max,y_max),(x_max,y_min)], np.int32)]
    roi_frame = frame[y_min:y_max, x_min:x_max]

    # Getting predictions
    y_hat = model.predict(roi_frame, conf = conf_level, classes = class_IDS,␣
↪device = 0, verbose = False)

    # Update total count of people
    count_p = 0
    confidence_scores = [] #Set for Excel output

    # Fetch current time
    current_time = datetime.datetime.now()
    time_str = current_time.strftime("%H:%M:%S")      # Format time as HH:MM:SS

    text_size = cv2.getTextSize(time_str, cv2.FONT_HERSHEY_SIMPLEX, font_size/
↪2, font_thickness//2)[0]
    text_x = int(frame.shape[1] - text_size[0] - (frame.shape[1] * 0.01))
    text_y = int(frame.shape[0] * 0.1)

    cv2.putText(frame, time_str, (text_x, text_y),
                cv2.FONT_HERSHEY_SIMPLEX, font_size/2, (255, 255, 255),␣
↪font_thickness//2)

    # Getting the bounding boxes, confidence and classes of the recognize␣
↪objects in the current frame.
    boxes_data = y_hat[0].boxes.xyxy.cpu().numpy() #Move into CPU memory
    confidences = y_hat[0].boxes.conf.cpu().numpy()
    classes = y_hat[0].boxes.cls.cpu().numpy()

    # Storing the above information in a dataframe, setting the min and max␣
↪values
    positions_frame = pd.DataFrame(boxes_data, columns=['xmin', 'ymin', 'xmax',␣
↪'ymax'])
    positions_frame['conf'] = confidences
    positions_frame['class'] = classes

    #Translating the numeric class labels to text
    labels = [dict_classes[i] for i in classes]

    #handle tracking logic
    for ix, row in enumerate(positions_frame.iterrows()):
        xmin, ymin, xmax, ymax, confidence, category = row[1].astype('int')

        count_p += 1
```

```
        confidence_scores.append(confidence) #Append data for Excel file
        # Calculating the center of the bounding box
        center_x, center_y = int(((xmax + xmin) / 2)), int(((ymax + ymin) / 2))
        # Drawing bounding box for every detection
        cv2.rectangle(roi_frame, (xmin, ymin), (xmax, ymax), (0, 0, 255), 2)

        # Updating the tracking for each object
        centers_old, id_obj, is_new, lastKey = update_tracking(centers_old,␣
↪(center_x, center_y), thr_centers, lastKey, i, frame_max)

        # Drawing additional tracking info (like circles) if needed (New)
        for center_x,center_y in centers_old[id_obj].values():
            cv2.circle(roi_frame, (center_x,center_y), 5,(0,255,0),-1)   # Using␣
↪a different color for clarity

       #Updating people in ROI
        count_p+=is_new

        #Drawing above the bounding-box the name of class recognized.
        cv2.putText(img=roi_frame,
            text=id_obj + ':' + str(np.round(confidences[ix], 2)),
            org=(xmin, ymin - 40),   # Adjust this value to position the text␣
↪higher
            fontFace=cv2.FONT_HERSHEY_TRIPLEX,
            fontScale=0.9,
            color=(0, 0, 255),
            thickness=1)


    count_str = f'No. of People in ROI: {count_p}'
    baseline = cv2.getTextSize(count_str, cv2.FONT_HERSHEY_SIMPLEX, font_size,␣
↪font_thickness)[1]
    text_x = int(frame.shape[1] * 0.05)
    text_y = int(frame.shape[0] * 0.1) + baseline

    cv2.putText(img=frame, text=count_str,
                org=(text_x,text_y), fontFace=cv2.FONT_HERSHEY_SIMPLEX,
                fontScale=font_size, color=(255, 255, 255),␣
↪thickness=font_thickness)


    # Append results for this frame to DataFrame (Excel)
    results_df = results_df.append({'Frame': i, 'Timestamp': time_str,␣
↪'People_Count': count_p, 'ID': lastKey}, ignore_index=True)
```

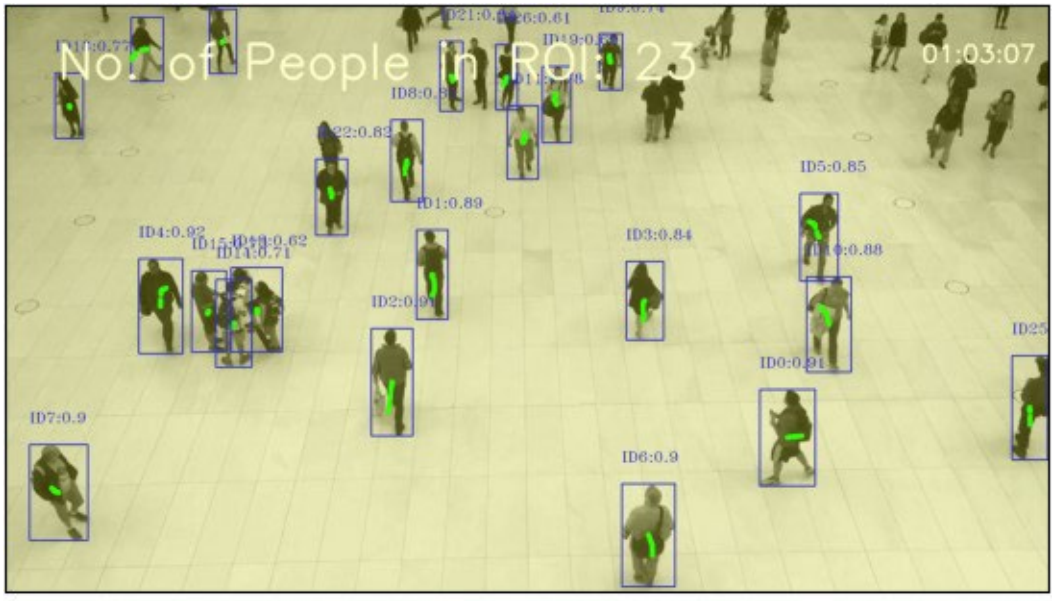
```
    # Filtering tracks history
    centers_old = filter_tracks(centers_old, patience)

    #Drawing the ROI area
    overlay = frame.copy() # make a copy of the original frame

    cv2.polylines(overlay, pts = area_roi, isClosed = True, color=(255, 0,
↪0),thickness=2)
    cv2.fillPoly(overlay, pts = area_roi, color = (255,255,0))
    frame = cv2.addWeighted(overlay, alpha,frame , 1 - alpha, 0)

    #Saving frames in a list
    frames_list.append(frame)
    #saving transformed frames in a output video formaat
    output_video.write(frame)

# After processing all frames, export the DataFrame to Excel
results_df.to_excel('/content/people_count.xlsx', index=False)

#Releasing the video
output_video.release()
```

```
  0%|          | 0/341 [00:00<?, ?it/s]
```

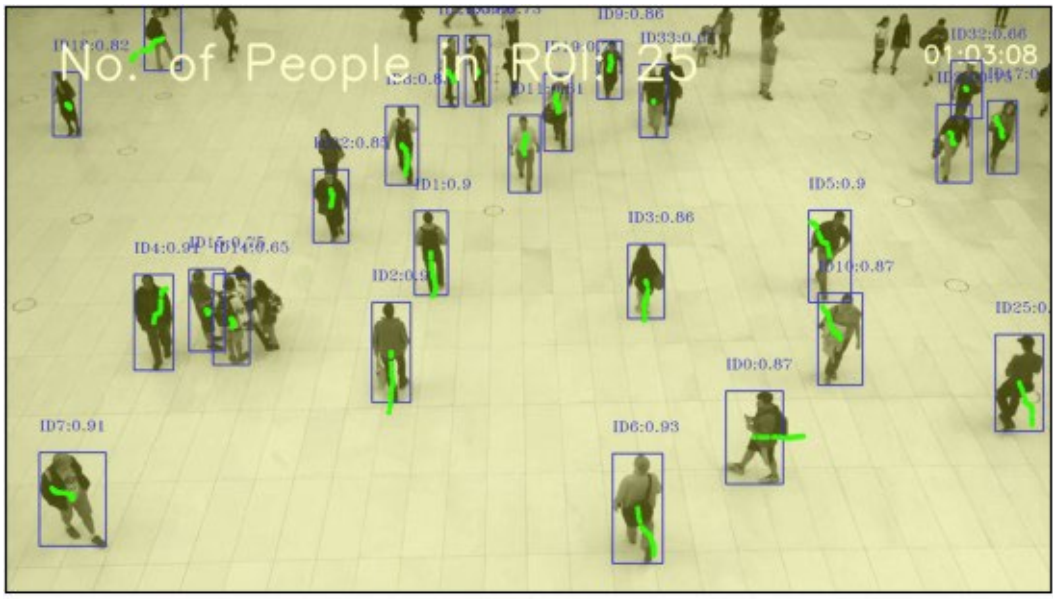
[ ]:
```
#@title 5.1: Process Video and Audio
# Check for Existing Output File and Remove if Present
if os.path.exists(output_video_path):
    os.remove(output_video_path)
# Process video and audio file
subprocess.run(
    ["ffmpeg", "-i",
↪tmp_output_path,"-crf","18","-preset","veryfast","-hide_banner","-loglevel","error","-vcode
os.remove(tmp_output_path)
```

[ ]:
```
#@title 5.2: Display of people and results _ Testing
frames_to_display = 24

for i in range(frames_to_display):
    frame = frames_list[i*10].copy()

    plt.figure(figsize=(7, 5))
    plt_pretty_image(frame, f'frame {i*20}')
    plt.show()
```

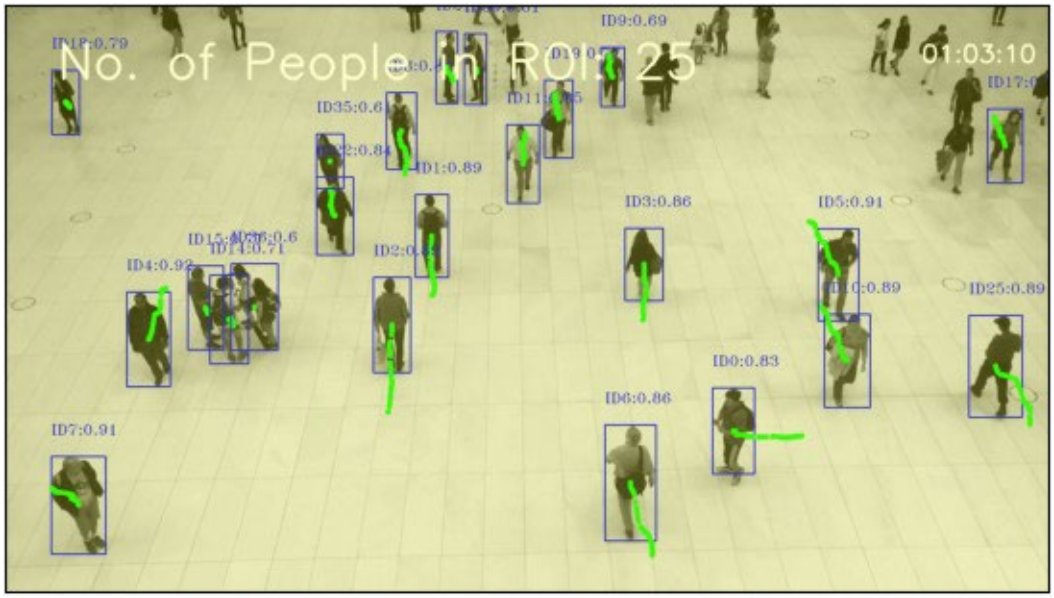
frame 0



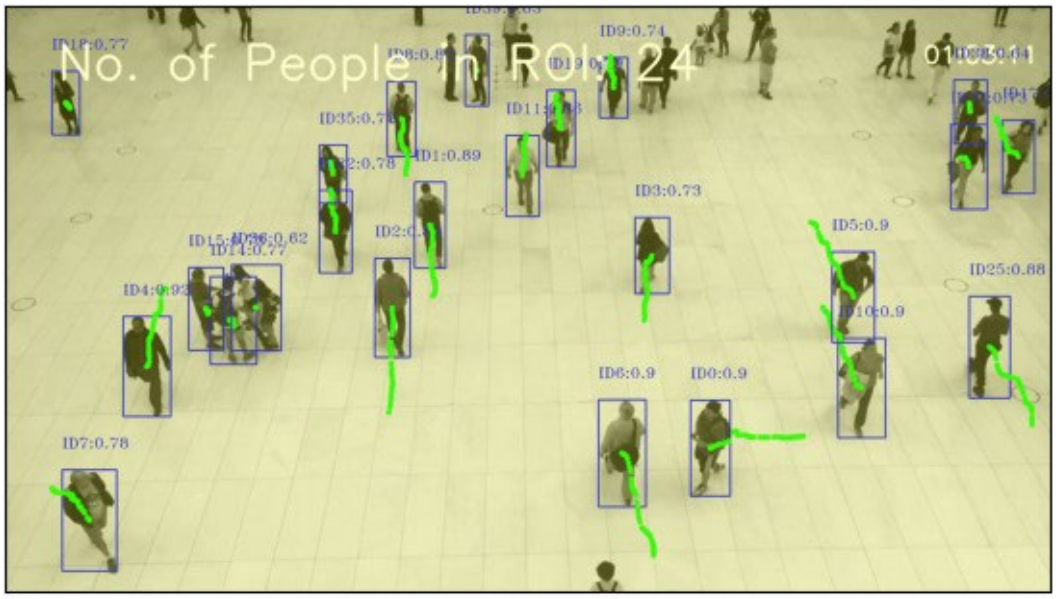
frame 20



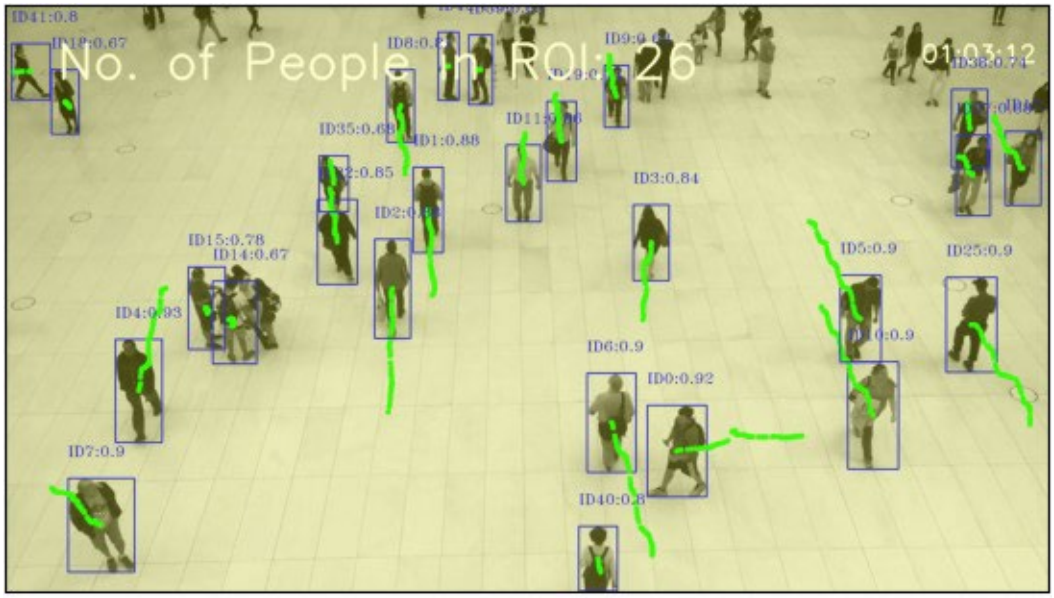
frame 40



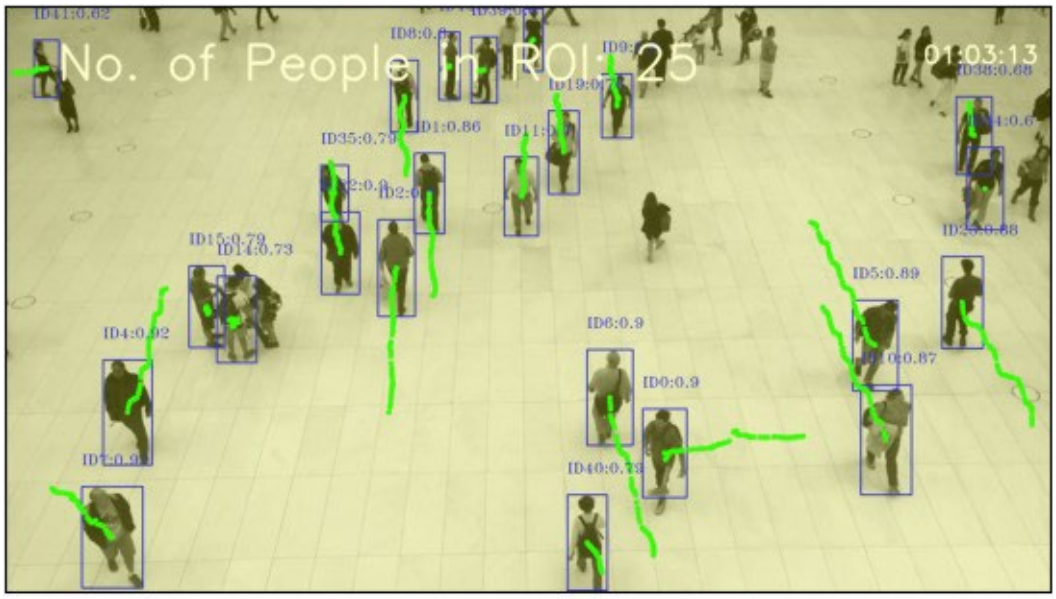
frame 60



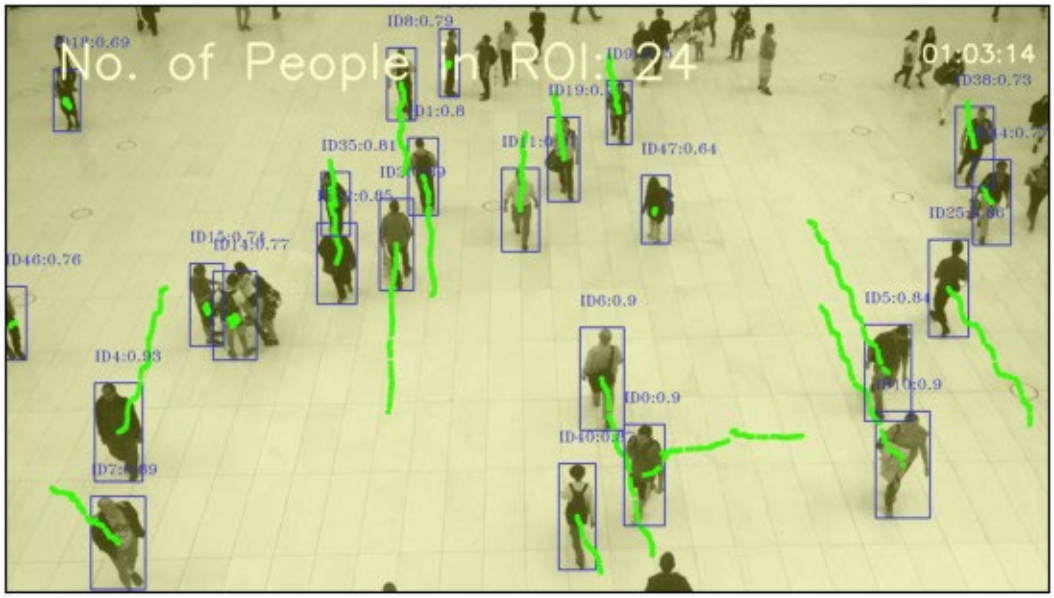
frame 80



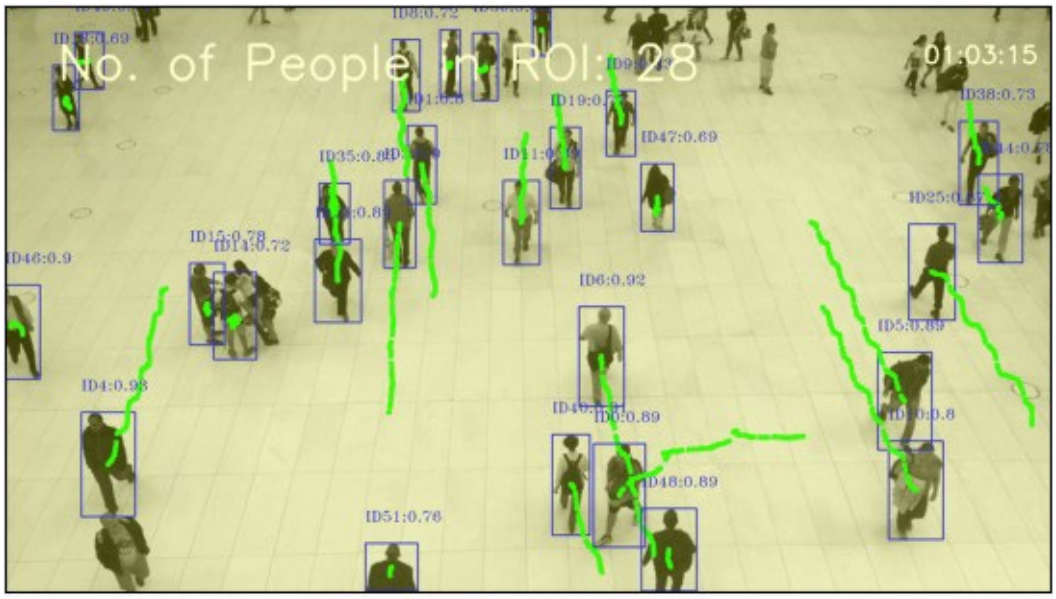
frame 100



frame 120

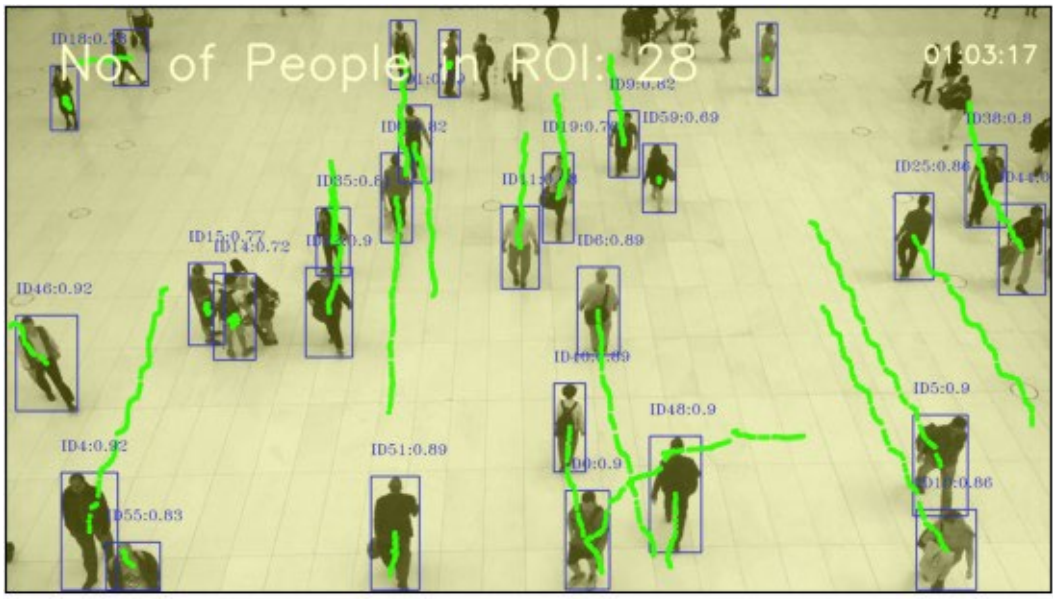


frame 140

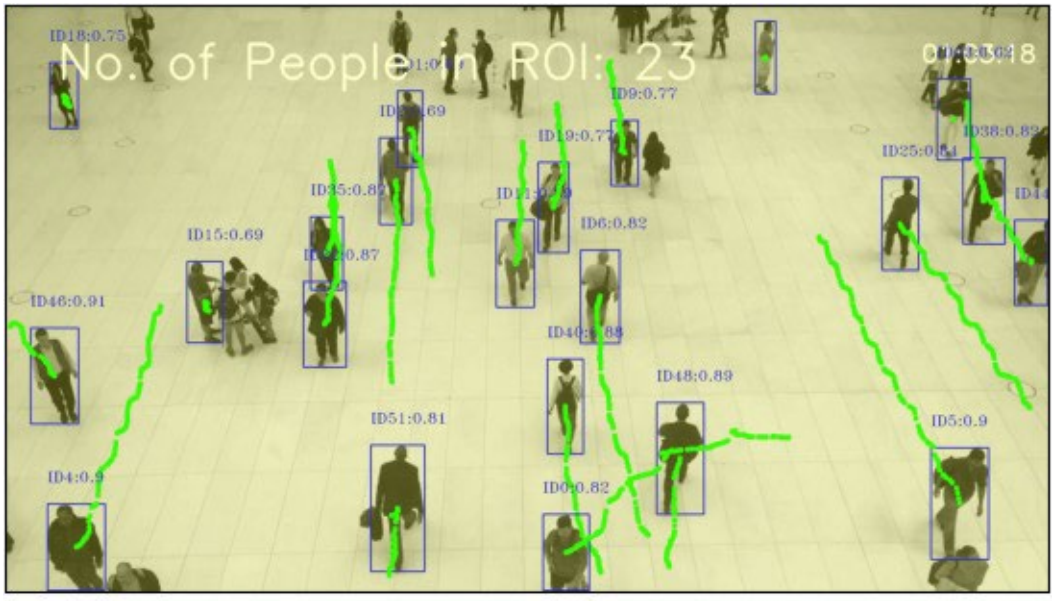


frame 160

frame 180

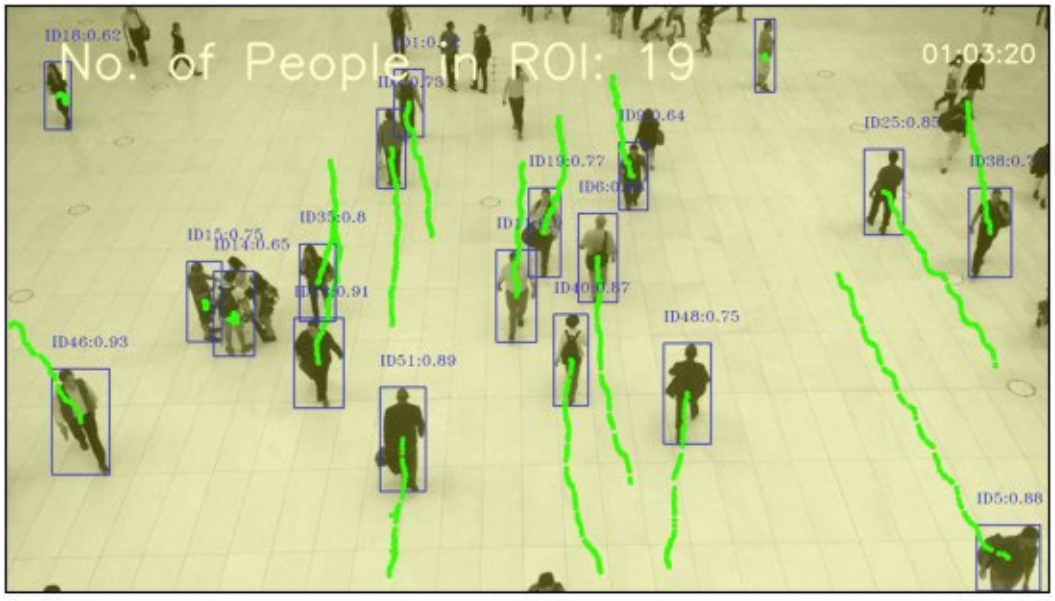


frame 200



frame 220

frame 240



frame 260

frame 280

frame 300

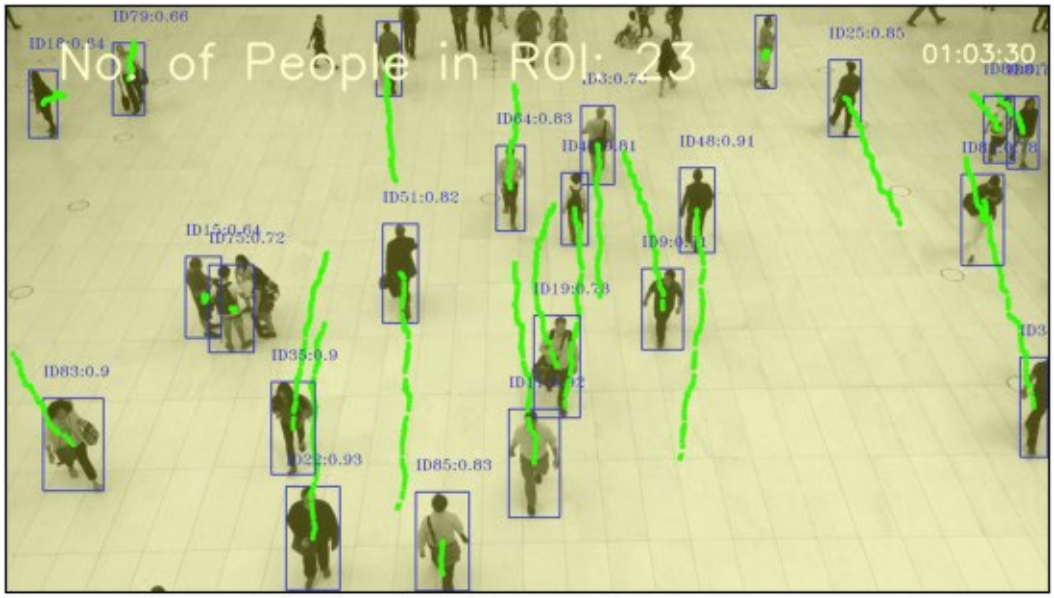
frame 320

frame 340

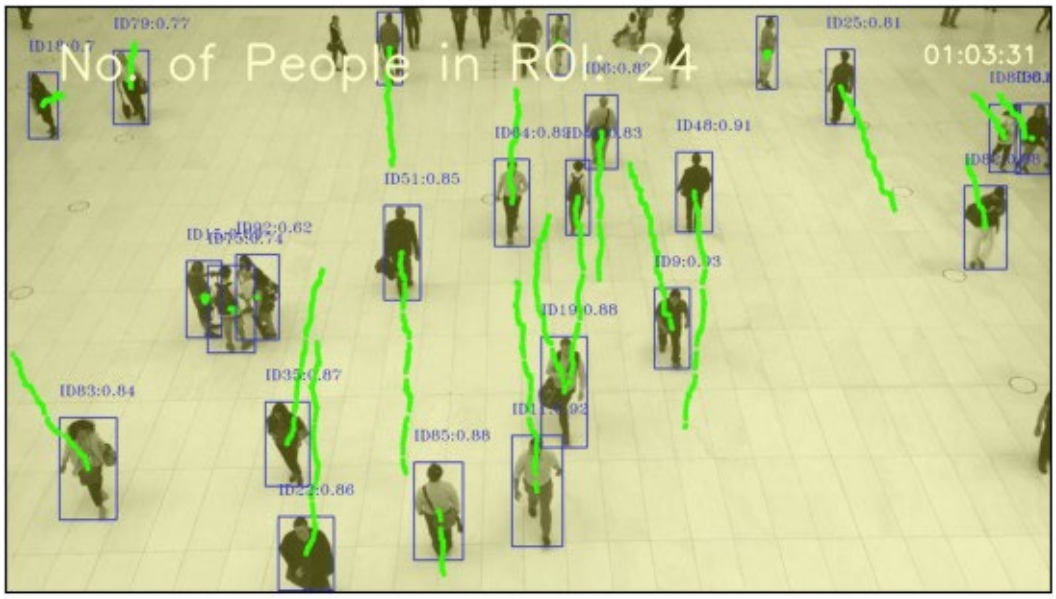
frame 360

frame 380

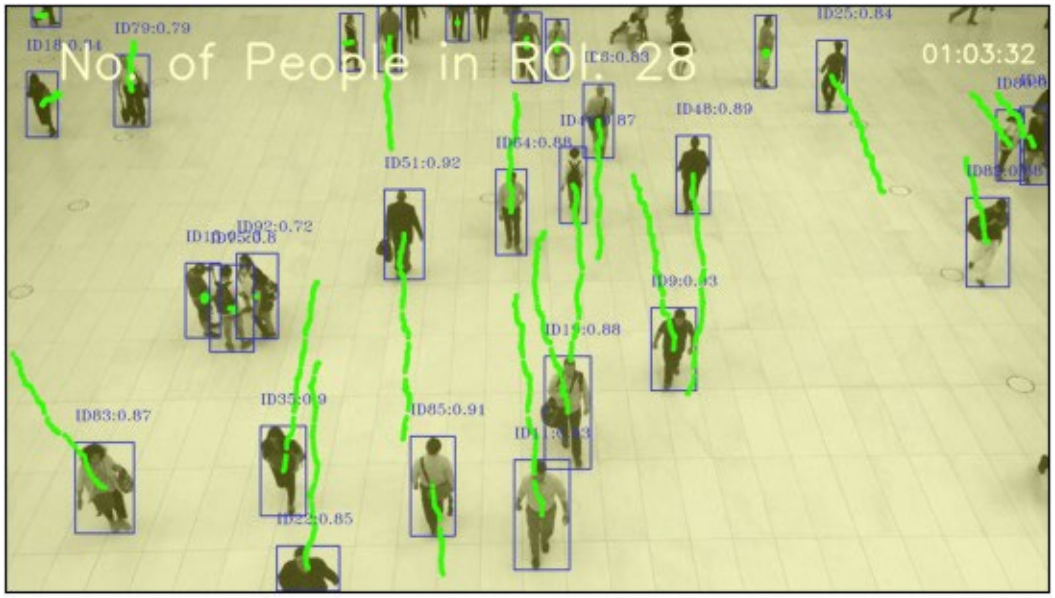
frame 400



frame 420



frame 440



frame 460

[ ]:
```
#@title Monitoring: Play the processed MP4
# preview output video
video_tag = f"""
<video width={width/2} height={height/2} controls>
```

```
    <source src="data:video/mp4;base64,{video_to_base64(output_video_path)}"
    type="video/mp4">
</video>
"""
```

[ ]:
```
HTML(video_tag)
```

[ ]: <IPython.core.display.HTML object>

[ ]:
```
# Assuming the CSV file is named 'count_data.csv'
df = pd.read_excel('/content/drive/MyDrive/AAI-521/Final/Test/
    Score-ManualCount-t6.xlsx')

# Initialize counters
TP = 0
FP = 0
FN = 0

# Calculate TP, FP, and FN
for index, row in df.iterrows():
    system_count = row['System Count']
    manual_count = row['M. Counted']

    TP += min(system_count, manual_count)
    FP += max(0, system_count - manual_count)
    FN += max(0, manual_count - system_count)

# Calculate Precision, Recall, and F1 Score
precision = TP / (TP + FP) if TP + FP > 0 else 0
recall = TP / (TP + FN) if TP + FN > 0 else 0
f1_score = 2 * (precision * recall) / (precision + recall) if precision +
    recall > 0 else 0
#Print out the values
print("SCORE VALUES")
print(f"Precision: {precision:.4f}")
print(f"Recall: {recall:.4f}")
print(f"F1 Score: {f1_score:.4f}")
```

```
SCORE VALUES
Precision: 0.9948
Recall: 0.7265
F1 Score: 0.8397
```

#This section is for emulating a real-time Camera sensor in the IoT

[ ]:
```
#@title Installing required packages and testing the model
# import dependencies
from IPython.display import display, Javascript, Image
```

```
from google.colab.output import eval_js
from google.colab.patches import cv2_imshow
from base64 import b64decode, b64encode
import cv2
import numpy as np
import PIL
import io
import html
import time
import matplotlib.pyplot as plt
%matplotlib inline
```

[ ]:
```
#Open real-time camera capture with YOLO Object Detection
# Load the YOLOv8 model
model = YOLO("yolov8n.pt")      # Adjust the model path as necessary
dict_classes = model.model.names # Class names

# Initialize camera stream
cap = cv2.VideoCapture(0)     # Use appropriate index or URL for your camera

try:
    while True:
        # Read frame from camera
        ret, frame = cap.read()
        if not ret:
            print("Error: failed to capture image")
            break

        # Perform detection
        results = model.predict(frame)

        # Draw bounding boxes and labels
        for detection in results.xyxy[0]:
            # Extract data
            xmin, ymin, xmax, ymax, conf, cls = detection
            label = dict_classes[int(cls)]

            # Draw bounding box
            cv2.rectangle(frame, (int(xmin), int(ymin)), (int(xmax),␣
↪int(ymax)), (0, 255, 0), 2)
            # Draw label
            cv2.putText(frame, f"{label} {conf:.2f}", (int(xmin), int(ymin)-10),
                        cv2.FONT_HERSHEY_SIMPLEX, 0.5, (0, 255, 0), 2)

        # Display the frame
        cv2.imshow("Camera Feed", frame)
```

```
        # Break loop on specific key press, e.g., 'q'
        if cv2.waitKey(1) & 0xFF == ord('q'):
            break
finally:
    cap.release()
    cv2.destroyAllWindows()
```

Error: failed to capture image