International Journal of Engineering Applied Sciences and Technology, 2024
Vol. 9, Issue 04, ISSN No. 2455-2143, Pages 87-94
Published Online August 2024 in IJEAST (http://www.ijeast.com)

BEYOND THE ROLLCALL: SMART STRATEGY FOR ATTENDANCE

Mr. Tejas Varute, Mr. Shripad Rajput, Mr. Swapnil Shinde, Mr. Parth Shinde
Department of Electronics and Computer Engineering,
Sharad Institute of Technology, College of engineering,
Maharashtra, India

Abstract— **Attendance tracking is a crucial aspect of educational institutions, yet the traditional manual methods are time-consuming and error-prone. This project presents an innovative solution to this problem through the development of an automated attendance system. Utilizing advanced technologies such as face detection and optical character recognition (OCR), this system aims to streamline the attendance tracking process in schools and colleges. By automatically detecting student faces and extracting roll numbers from stickers affixed to the left side of their faces, the system eliminates the need for manual intervention, thereby improving efficiency and accuracy. Additionally, a user-friendly GUI created using customtkinter in Python enhances the system's usability and accessibility. Attendance records are automatically stored in an Excel file, allowing for easy tracking and analysis by educators.**

Keywords— **Automated attendance system, face detection, optical character recognition (OCR), Automated database management.**

I. INTRODUCTION

Traditional methods of attendance-taking in schools and colleges often lead to inefficiencies and errors. The introduction of an automated attendance system marks a significant advancement in this domain. By harnessing the power of face detection and OCR technologies, coupled with an intuitive GUI, the system aims to streamline the attendance tracking process. This report delves into the development and implementation of this innovative solution, highlighting its benefits for educational institutions.
This project report describes the development of a software solution to automate the attendance process. The software is implemented using,

Python and takes input image file as:
- Capture live image on Webcam or CCTV cameras that connected to system.
- Provided pre-captured image from system.

After getting the image it will detect face of students and extract roll numbers from stickers affixed to the left side of their faces.
Stores all the collected roll numbers on dynamically arranged database file which created with help of Microsoft Excel.

II. LITERATURE REVIEW

Manual attendance systems are rife with inefficiencies, prompting the exploration of automated alternatives. Existing literature reveals the promise of face detection algorithms like YOLO for real-time detection and OCR techniques such as Easy OCR for text extraction. Moreover, the integration of GUIs plays a pivotal role in enhancing user experience and system usability. This review sets the stage for the integration of customtkinter into the project, emphasizing the importance of user interaction.

III. PROBLEM STATEMENT

The manual process of attendance-taking poses significant challenges, including time consumption and inaccuracies. This project aims to address these challenges by automating the attendance tracking process. The objective is to develop a robust system capable of seamlessly detecting student faces and extracting roll numbers from stickers affixed to the left side of their faces. By doing so, the system seeks to improve efficiency and accuracy in classroom management.

IV. OBJECTIVE

The primary objective of the project is to develop an automated attendance system leveraging face detection and OCR technologies. Key goals include real-time detection of student faces, extraction of roll numbers from stickers, and automatic logging of attendance records into an Excel file. Additionally, the system aims to provide a user-friendly GUI to facilitate ease of use for educators.

A. Software Used:

The development of the automated attendance system relies on a suite of software tools and libraries, including,

- **Software:**
  - Python IDLE/VS Code.

- Microsoft Excel.

- **Libraries:**
- Ultralytics: Provide module for YOLO.
- EasyOCR: Provide module for Optical Character Recognition.
- Openpyxl: Provide module to handle Excel file.
- OS: It is inbuilt python library help to communication with Operating system.
- OpenCV: Provide module for capture image, edit image, process on image.
- Customtkinter / tkinter: Provide module to create GUL.

These tools are seamlessly integrated to create a robust and efficient system capable of accurately tracking student attendance.

B. **Algorithm:**

Algorithm for face detection & capturing roll number:

1. START.
2. Set class strength.
3. Get image:
   i. Capture live image using Camera connected to System.
   ii. Provide pre-capture image from System storage.

4. Process on image:
   i. Using YOLO:
   - Detect all the student faces in image.
   - Mark a rectangular border around that detected face.

5. In loop:
   i. Get detected face one by one.
   ii. Marking the imaginary rectangular border on right side of detected face and cropping it.
   iii. Providing that cropped image to OCR.
   - Return the detected roll number in image.
   iv. Storing the detected roll number in temporary list.
   v. Storing the detected roll number into the Excel database.

6. STOP.

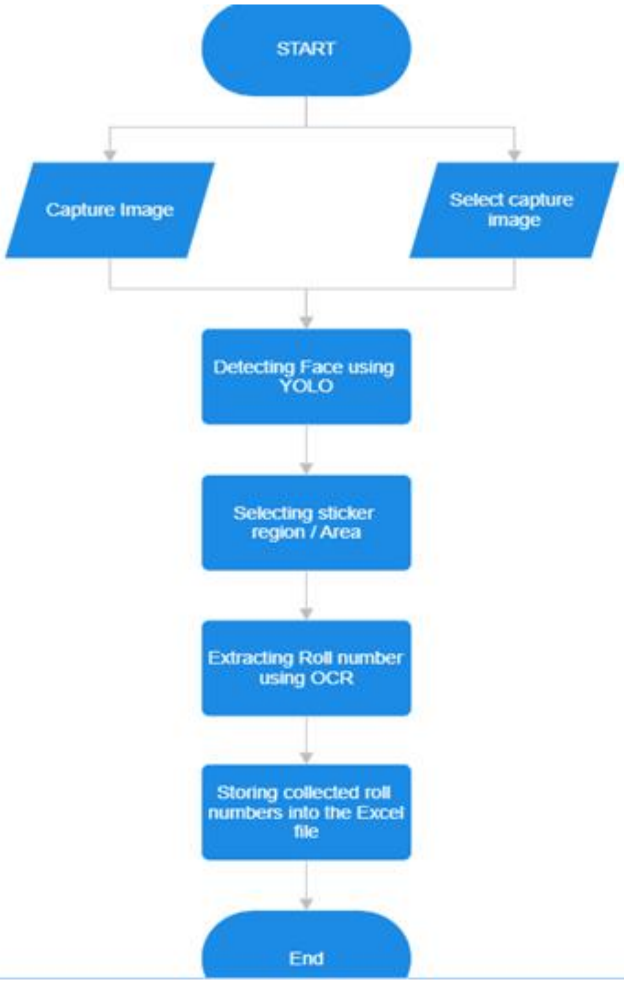

Fig. 1. Flowchart

- **Algorithm for database insertion:**

1. File creation:
- This ensures that there is always a database file available for use.
- Check if the file exists.

i. If not,
- create it and initialize it with ‘workbook 1.xlsx’.

ii. Else:
- Check if week ended:

iii. If yes,
- create new file with ‘workbook {count+1}.xlsx’.

iv. Else:
- Use current present ‘workbook {count}.xlsx’.

2. Open the Excel file:
- Check if sheet is created or not:

i. If not, create sheet with attribute ‘Roll No’ and add tuples from 1 to class strength.

3. Day Check:
- Check if the current day's sheet exists in the database. If not, create a new sheet for the current day.
- Ensure that the database is properly organized with separate sheets for each day's attendance.

4. Attendance Processing:
- Process attendance records based on the roll list.
- Update attendance status (Present/Absent) for each roll number in the database.

5. Result Calculation:
- Calculate daily and weekly attendance results based on the attendance recorded in the database.
- Daily result calculation involves iterating over each row in the current day's sheet and computing the attendance percentage.
- Weekly result calculation involves aggregating daily results to compute the average attendance for the week.

C. **Model Training:**

Training:

To train the YOLOv8 model with a custom dataset of 3669 images with subject face are pre-trained to YOLOv8 which was already pre-trained for the COCO dataset with YOLOv8n.pt This model was trained for this model 100 epochs. All 3669 images were annotated using Roboflow. The data set was trained with the help of PyTorch library. The images are labelled YOLO. A total of _ images were used to validation and _ images were used for training.

To train a model using labelled images, the custom dataset images are in three folders.

1. Train
2. Test
3 Valid

Each folder has images with labels. Labels are saved in .txt format, The yaml file (custom_data.yaml) specifies the location of the folders to call to train a model.

“yolo task=detect, model=train, model=yolov8n.pt, data=custom_data.yaml, epochs=100, imgsz=640, plos=True”

After the 100 epochs, we get our best model (best.pt).

The trained custom dataset (see Fig. 2)

Epochs	Box loss	Class loss	DFL loss (Distributional focal loss)
1	1.119	1.626	1.534
100	0.493	0.414	1.071

Table 1: Loss of custom dataset

Segment Anything model (SAM) is already pre-trained with 1+billion mask and 11 million images of SA-1B dataset. So that we can use the SAM without labelling them. It segments all the things they’ve been trained in. Now, we have to segment the custom-specific object so we give our trained model as input and integrated with SAM, so it only focuses on the model that we trained. So, the latest model is more productive than all other segmentation models.

```
New https://pypi.org/project/ultralytics/8.2.92 available 😃 Update with 'pip install -U ultralytics'
Ultralytics YOLOv8.0.196 🚀 Python-3.10.12 torch-2.4.0+cu121 CUDA:0 (Tesla T4, 15102MiB)
engine/trainer: task=detect, mode=train, model=yolov8s.pt, data=/content/datasets/Human-Face-and-Hand-Detection-2/data.yaml, epochs=100, patience=50, batch=16, imgsz
Downloading https://ultralytics.com/assets/Arial.ttf to '/root/.config/Ultralytics/Arial.ttf'...
100% 755k/755k [00:00<00:00, 20.4MB/s]
2024-09-12 10:12:53.629660: E external/local_xla/xla/stream_executor/cuda/cuda_fft.cc:485] Unable to register cuFFT factory: Attempting to register factory for plugi
2024-09-12 10:12:53.651101: E external/local_xla/xla/stream_executor/cuda/cuda_dnn.cc:8454] Unable to register cuDNN factory: Attempting to register factory for plug
2024-09-12 10:12:53.658100: E external/local_xla/xla/stream_executor/cuda/cuda_blas.cc:1452] Unable to register cuBLAS factory: Attempting to register factory for pl
Overriding model.yaml nc=80 with nc=3

                   from  n    params  module                                       arguments
  0                  -1  1       928  ultralytics.nn.modules.conv.Conv             [3, 32, 3, 2]
  1                  -1  1     18560  ultralytics.nn.modules.conv.Conv             [32, 64, 3, 2]
  2                  -1  1     29056  ultralytics.nn.modules.block.C2f             [64, 64, 1, True]
  3                  -1  1     73984  ultralytics.nn.modules.conv.Conv             [64, 128, 3, 2]
  4                  -1  2    197632  ultralytics.nn.modules.block.C2f             [128, 128, 2, True]
  5                  -1  1    295424  ultralytics.nn.modules.conv.Conv             [128, 256, 3, 2]
  6                  -1  2    788480  ultralytics.nn.modules.block.C2f             [256, 256, 2, True]
  7                  -1  1   1180672  ultralytics.nn.modules.conv.Conv             [256, 512, 3, 2]
  8                  -1  1   1838080  ultralytics.nn.modules.block.C2f             [512, 512, 1, True]
  9                  -1  1    656896  ultralytics.nn.modules.block.SPPF            [512, 512, 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  1    591360  ultralytics.nn.modules.block.C2f             [768, 256, 1]
 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  1    148224  ultralytics.nn.modules.block.C2f             [384, 128, 1]
 16                  -1  1    147712  ultralytics.nn.modules.conv.Conv             [128, 128, 3, 2]
 17            [-1, 12]  1         0  ultralytics.nn.modules.conv.Concat           [1]
 18                  -1  1    493056  ultralytics.nn.modules.block.C2f             [384, 256, 1]
 19                  -1  1    590336  ultralytics.nn.modules.conv.Conv             [256, 256, 3, 2]
 20             [-1, 9]  1         0  ultralytics.nn.modules.conv.Concat           [1]
 21                  -1  1   1969152  ultralytics.nn.modules.block.C2f             [768, 512, 1]
 22        [15, 18, 21]  1   2117209  ultralytics.nn.modules.head.Detect           [3, [128, 256, 512]]
Model summary: 225 layers, 11136761 parameters, 11136745 gradients, 28.7 GFLOPs

Transferred 349/355 items from pretrained weights
TensorBoard: Start with 'tensorboard --logdir runs/detect/train', view at http://localhost:6006/
Freezing layer 'model.22.dfl.conv.weight'
AMP: running Automatic Mixed Precision (AMP) checks with YOLOv8n...
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))
/usr/lib/python3.10/multiprocessing/popen_fork.py:66: RuntimeWarning: os.fork() was called. os.fork() is incompatible with multithreaded code, and JAX is multithreade
  self.pid = os.fork()
val: Scanning /content/datasets/Human-Face-and-Hand-Detection-2/valid/labels... 191 images, 0 backgrounds, 0 corrupt: 100% 191/191 [00:00<00:00, 1362.74it/s]
val: New cache created: /content/datasets/Human-Face-and-Hand-Detection-2/valid/labels.cache
Plotting labels to runs/detect/train/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=0.001429, momentum=0.9) with parameter groups 57 weight(decay=0.0), 64 weight(decay=0.0005), 63 bias(decay=0.0)
Image sizes 800 train, 800 val
Using 2 dataloader workers
Logging results to runs/detect/train
Starting training for 100 epochs...

      Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
      1/100      6.14G      1.119      1.626      1.534         73        800: 100% 126/126 [01:24<00:00,  1.50it/s]
                 Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100% 6/6 [00:05<00:00,  1.03it/s]
                   all        191        458       0.56      0.598      0.574      0.283

      Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
      2/100      6.29G      1.068      1.362      1.488         81        800: 100% 126/126 [01:20<00:00,  1.57it/s]
                 Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100% 6/6 [00:03<00:00,  1.84it/s]
                   all        191        458      0.428      0.432      0.388      0.185

      Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
      3/100      5.82G      1.058      1.357      1.495         72        800: 100% 126/126 [01:18<00:00,  1.61it/s]
                 Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100% 6/6 [00:03<00:00,  1.52it/s]
                   all        191        458      0.502      0.501      0.475      0.225

      Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
      4/100      5.86G      1.044       1.32      1.479         74        800: 100% 126/126 [01:17<00:00,  1.63it/s]
                 Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100% 6/6 [00:05<00:00,  1.18it/s]
                   all        191        458       0.71      0.391      0.456      0.237

      Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
      5/100      5.81G      0.996      1.271      1.448         90        800: 100% 126/126 [01:20<00:00,  1.57it/s]
                 Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100% 6/6 [00:03<00:00,  1.75it/s]
                   all        191        458      0.671      0.573      0.635      0.346

      Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
      6/100       5.9G     0.9864      1.244      1.437         77        800: 100% 126/126 [01:20<00:00,  1.57it/s]
                 Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100% 6/6 [00:03<00:00,  1.71it/s]
                   all        191        458      0.647      0.534       0.57      0.327

      Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
      7/100       5.9G     0.9498      1.191      1.407         73        800: 100% 126/126 [01:17<00:00,  1.62it/s]
                 Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100% 6/6 [00:06<00:00,  1.02s/it]
                   all        191        458      0.624       0.68      0.704      0.384

      Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
      8/100      5.87G     0.9398      1.174      1.402         97        800: 100% 126/126 [01:21<00:00,  1.54it/s]
                 Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100% 6/6 [00:05<00:00,  1.08it/s]
                   all        191        458      0.802      0.688      0.761      0.411

      Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
      9/100      5.86G     0.9187      1.138      1.387         69        800: 100% 126/126 [01:17<00:00,  1.62it/s]
                 Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100% 6/6 [00:03<00:00,  1.67it/s]
                   all        191        458       0.72       0.66      0.734      0.417


      Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
     91/100      5.88G     0.5602     0.4827      1.117         41        800: 100% 126/126 [01:17<00:00,  1.62it/s]
                 Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100% 6/6 [00:05<00:00,  1.09it/s]
                   all        191        458      0.874      0.864      0.846      0.536

      Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
     92/100       5.9G     0.5422     0.4608      1.106         33        800: 100% 126/126 [01:11<00:00,  1.75it/s]
                 Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100% 6/6 [00:03<00:00,  1.99it/s]
                   all        191        458      0.825      0.876      0.847      0.534

      Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
     93/100      5.87G     0.5363     0.4511      1.109         31        800: 100% 126/126 [01:16<00:00,  1.65it/s]
                 Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100% 6/6 [00:03<00:00,  1.77it/s]
                   all        191        458      0.864      0.832      0.847      0.538

      Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
     94/100      5.88G      0.527      0.438      1.097         31        800: 100% 126/126 [01:14<00:00,  1.70it/s]
                 Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100% 6/6 [00:03<00:00,  1.81it/s]
                   all        191        458      0.871      0.854      0.863      0.541

      Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
     95/100      5.87G     0.5166     0.4406      1.092         41        800: 100% 126/126 [01:14<00:00,  1.69it/s]
                 Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100% 6/6 [00:05<00:00,  1.18it/s]
                   all        191        458      0.875      0.861      0.863      0.541

      Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
     96/100      5.87G     0.5155      0.435      1.091         44        800: 100% 126/126 [01:18<00:00,  1.61it/s]
                 Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100% 6/6 [00:03<00:00,  1.72it/s]
                   all        191        458      0.866       0.85      0.852      0.548

      Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
     97/100      5.92G     0.5161     0.4313      1.091         40        800: 100% 126/126 [01:13<00:00,  1.72it/s]
                 Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100% 6/6 [00:03<00:00,  1.74it/s]
                   all        191        458      0.861       0.86       0.86      0.544

      Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
     98/100      5.89G     0.5078     0.4273      1.084         41        800: 100% 126/126 [01:12<00:00,  1.73it/s]
                 Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100% 6/6 [00:03<00:00,  1.79it/s]
                   all        191        458      0.852      0.881      0.858      0.549

      Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
     99/100      5.87G     0.4994     0.4169      1.078         39        800: 100% 126/126 [01:14<00:00,  1.70it/s]
                 Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100% 6/6 [00:05<00:00,  1.02it/s]
                   all        191        458      0.876       0.86      0.867      0.552

      Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
    100/100      5.88G      0.493      0.414      1.071         33        800: 100% 126/126 [01:11<00:00,  1.77it/s]
                 Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100% 6/6 [00:04<00:00,  1.45it/s]
                   all        191        458      0.873      0.861      0.864       0.55

100 epochs completed in 2.315 hours.
Optimizer stripped from runs/detect/train/weights/last.pt, 22.6MB
Optimizer stripped from runs/detect/train/weights/best.pt, 22.6MB

Validating runs/detect/train/weights/best.pt...
Ultralytics YOLOv8.0.196 🚀 Python-3.10.12 torch-2.4.0+cu121 CUDA:0 (Tesla T4, 15102MiB)
Model summary (fused): 168 layers, 11126745 parameters, 0 gradients, 28.4 GFLOPs
                 Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100% 6/6 [00:09<00:00,  1.55s/it]
                   all        191        458      0.891      0.879      0.874      0.559
            Human Body        191         43          1      0.969      0.978      0.652
            Human Face        191        363      0.962      0.956      0.974       0.75
            Human Hand        191         52      0.711      0.711       0.67      0.275
Speed: 0.5ms preprocess, 8.3ms inference, 0.0ms loss, 5.0ms postprocess per image
Results saved to runs/detect/train
```

Fig. 2. Train with custom dataset.

International Journal of Engineering Applied Sciences and Technology, 2024
Vol. 9, Issue 04, ISSN No. 2455-2143, Pages 87-94
Published Online August 2024 in IJEAST (http://www.ijeast.com)

Experimental result:
Evaluation Metrics: The following metrics are used to evaluate the classification performance of the algorithm,

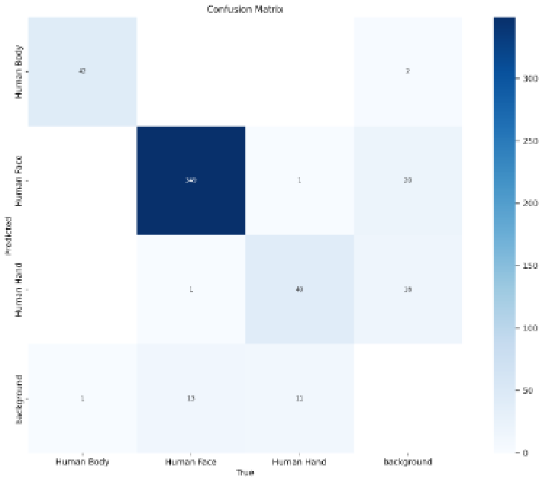

Fig. 3. Confusion matrix

Accuracy:
It is defined as the number of correct predictions made by the model over the total number of predictions. This is good measure, especially when objective variables in the data are balance. This can be illustrated as,

Accuracy = (TP + TN) ÷ (TP + TN + FP + FN)

Where, True positive (TP) is defined as the correct recognition of training group of objects. A True negative (TN) is defined as a grossly incorrect unspecified factor, i.e. knowing nothing when something should be known. A false positive (FP) is defined as false detection, meaning that there is detection even though no object should be detected. A false negative (FN) is defined as not detecting any ground truth, i.e. the algorithm failed to find the object to be found.

F1 score:
The precision of test is determined by the balanced F-measure. If both false positive and false negative rate are low, the F1 score is considered positive. It is defined as the harmonic medium of precision and recall.

F1 = 2 (precision × recall) ÷ (precision + recall)
= TP ÷ (TP + ½ (FP + FN))

Precision:
It is the number of positives divided by the total number of positives predicted by the classifiers.

Precision = (True positive) ÷ (True positive + False positive)

Recall:
It is defined as the number of true positives divided by the sum of true positives and false negatives.

Recall = (True positive) ÷ (True positive + False Negative)
Validation:
The validation process involves running the model on validation data set and comparing the model predictions to the ground truth labels. Several metrics such as mean average precision (mAP), intercept over union (IoU), and false positive rate (FPR) can be used to evaluate model performance. Validation results can be used to tune model hyperparameters such as number of trails, batch size, and number of epochs. The goal is to find different hyperparameters that lead to the best performance on the validation dataset.

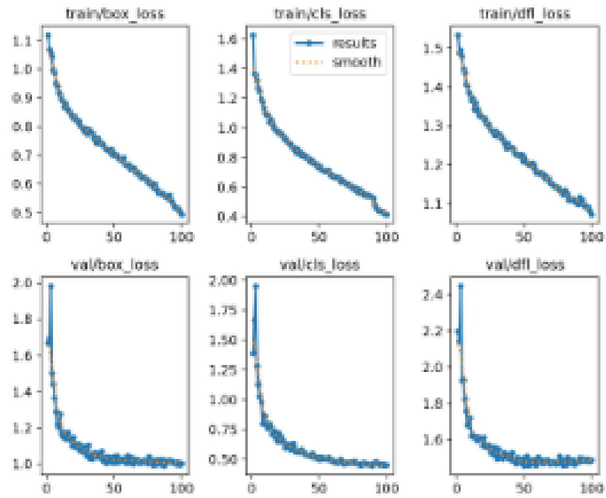

Fig. 4. Loss graph for training and validation dataset.

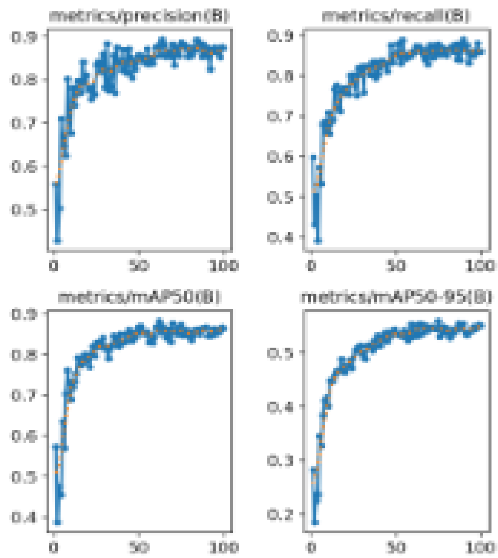

Fig. 5. Metrics graph for custom dataset

	P	R	mAP50	MAP50-95
All	0.891	0.879	0.874	0.559
Face	0.962	0.956	0.974	0.75

Table 2: Metrics of custom dataset

V. SOFTWARE INTERFACE

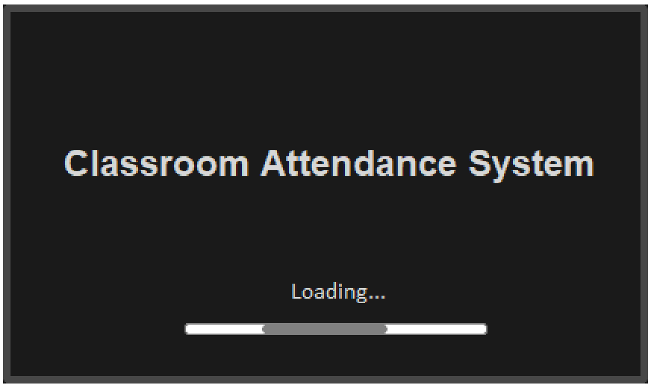

Fig. 1 Loading window

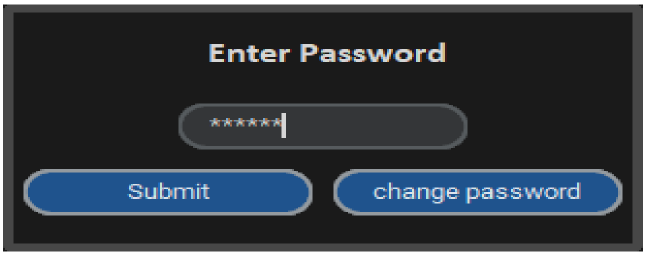

Fig. 2 Password Window

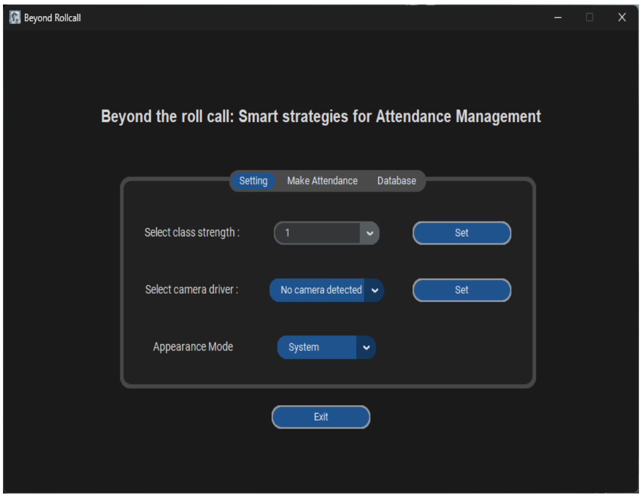

Fig. 3 Main setting window

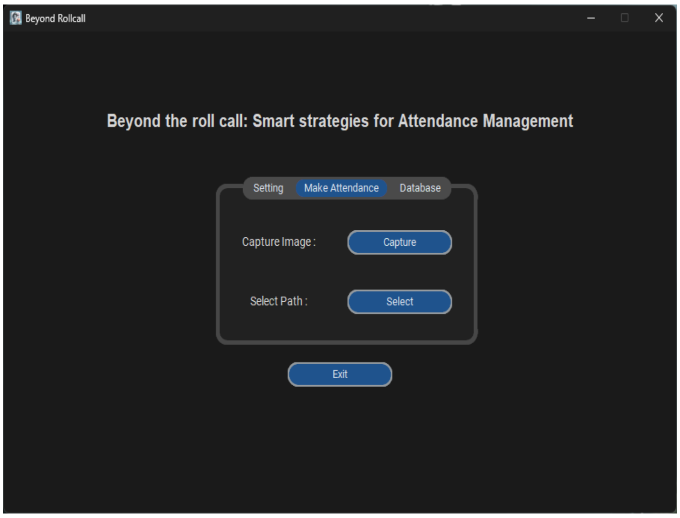

Fig. 4 Take attendance window

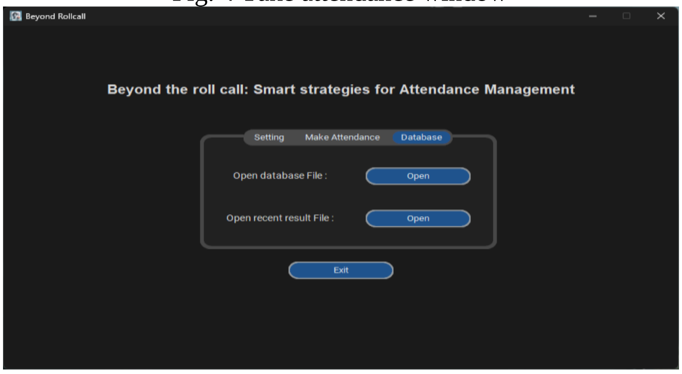

Fig. 5 Database window

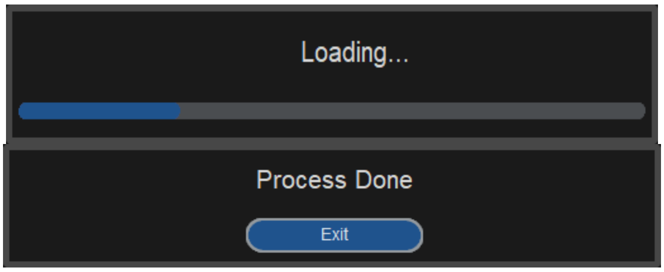

Fig. 6 Loading window while performing face detection and OCR and notifying as process done.

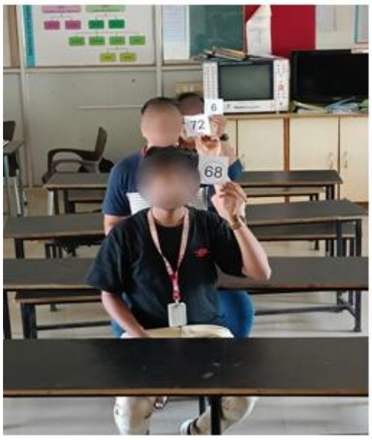

Fig. 7 Image provided/captured by camera

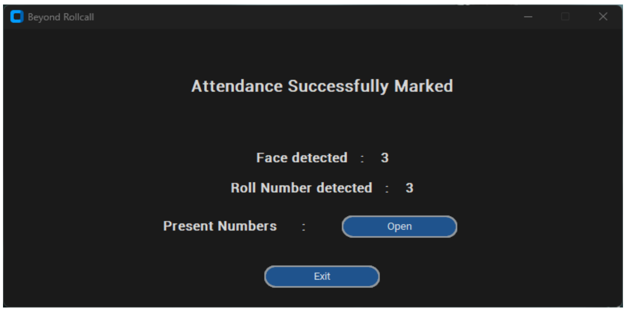

Fig. 8 Output window showing face and roll count with collected roll numbers

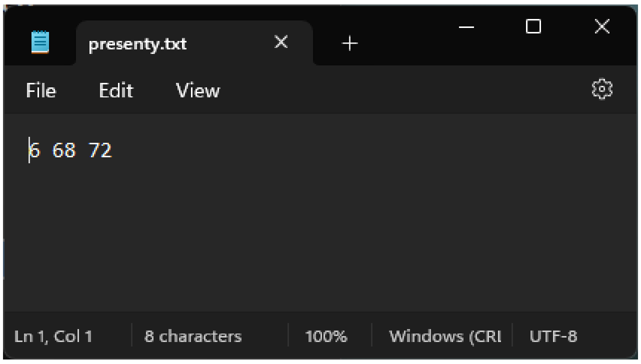

Fig. 9 'Presenty' text file showing collected present numbers

VI. WORKING

1. Initialization:
   a. The system initializes the YOLO (You Only Look Once) model for face detection.
   b. It loads the pre-trained model for YOLO.

2. Capture Mode Selection:
   a. The system prompts the user to choose between live capture mode and image input mode.
   b. In live capture mode, the system continuously captures video feed from the webcam.
   c. In image input mode, the user provides an image containing student faces.
3. Face Detection:
   a. For each frame captured in live mode or provided image, the system performs face detection using YOLO.
   b. YOLO identifies the bounding boxes around detected faces with high accuracy.

4. Region of Interest (ROI) Selection:
   a. Once a face is detected, the system determines the region of interest (ROI) around the face.
   b. The system identifies the left side of the face as the area where the roll number sticker is affixed.

5. Roll Number Extraction:
   a. The system extracts the ROI containing the roll number sticker.
   b. Using the EasyOCR library, the system performs optical character recognition (OCR) on the ROI to extract the roll number.
   c. The extracted roll number is stored for further processing.

6. Attendance Recording:
   a. The system matches the extracted roll number with the corresponding student record.
   b. If a match is found, the system marks the student as present in the attendance record.
   c. The attendance record is updated in real-time or saved in memory for later processing.

7. Excel Logging:
   a. The attendance records are automatically logged into an Excel sheet.
   b. Each student's attendance is recorded along with the date and time stamp.
   c. The Excel sheet serves as a centralized database for attendance tracking & analysis.

8. User Interface Interaction:
   a. In live capture mode, the system displays the video feed with overlaid bounding boxes around detected faces.
   b. In image input mode, the system provides feedback on the processing of the provided image and displays the extracted roll numbers.

9. Error Handling:
   a. The system includes error handling mechanisms to address potential issues during face detection or OCR.
   b. If a face or roll number cannot be detected, appropriate error messages are displayed to the user.

10. Feedback and Output:
    a. After processing each frame or image, the system provides feedback on the attendance status and any errors encountered.

b. Once all frames are processed or the image input is complete, the system may generate a summary report of attendance status.

11. Termination:
a. The system terminates once all frames are processed in live mode or after completing image input.
b. In live mode, the system continues to run until the user manually stops the capture.

12. GUI Interaction:
a. If using the system with a GUI, the user interacts with the system through the graphical interface.
b. The GUI provides options for mode selection, input file selection, and displays real-time feedback on attendance status.

VII. FEATURES

- Include real-time face detection.
- Automatically create reports.
- Considers one face one sticker.
- Prevent duplicity and provide more accuracy.
- Less time consuming compared to other techniques.
- Automatic logging of attendance records and a user-friendly GUI.

These features make the system suitable for application in schools, colleges, and other educational institutions, where efficient attendance tracking is essential for maintaining productivity and accountability.

ADVANTAGES

- Simple and Easy to handle user-friendly interface.
- Having advanced programming which provide dynamic working.
- Provide more accuracy and less time consumption compared to other attendance techniques like Traditional attendance techniques, RFID attendance techniques, Fingerprint attendance techniques, Face recognition attendance techniques, etc.
- No requirement of internet after installation.

Accessing and storing data in Excel file which is well known to any technical and non-technical person.

VIII. CONCLUSION

The development of an automated attendance system using face detection and OCR technologies represents a significant milestone in classroom management.
By automating the attendance tracking process and providing a user-friendly interface, the system improves,

- Efficiency
- Accuracy
- Usability

With further refinements and enhancements, this system has the potential to transform attendance management and contribute to the overall improvement of teaching and learning experiences.

IX. REFERENCE

[1] Z. Wang, B. Jiao and L. Xu, "Visual Object Detection: A Review," 2021 40th Chinese Control Conference (CCC), Shanghai, China, 2021, pp. 7106-7112, doi: 10.23919/CCC52363.2021.9550689. M. Young, The Technical Writer's Handbook. Mill Valley, CA: University Science, 1989.
[2] “Roboflow,” Available: https://app.roboflow.com/.
[3] "GoogleColab,"Colab.research.google.com.2019[Online]Available: https://colab.research.google.com/.
[4] “Ultralytics,” Available: https://www.ultralytics.com/
[5] “EasyOCR,”Available:https://github.com/JaidedAI/EasyOCR/
[6] “Customtkinter,”Available:https://github.com/TomSchimansky/CustomTkinter/
[7] “OpenCV,” Available:https://github.com/opencv/opencv
[8] Alexander Kirillov, Eric Mintun, Nikhila Ravi1, Hanzi Mao2 Chloe Rolland, Laura Gustafson, Tete Xiao, Spencer Whitehead Alexander C. Berg, Wan-Yen Lo, Piotr Dollar, Ross Girshick “Segment Anything” (Apr 2023). https://doi.org/10.48550/arXiv.2304.02643
[9] M. B. Blaschko and C. H. Lampert. Learning to localize objects with structured output regression. In Computer Vision – ECCV 2008, pages 2–15. Springer, 2008.
[10] H. Shen and J. Coughlan, “Reading LCD/LED Displays with a Camera Cell Phone”, Proceedings of the 2006 Conference on Computer Vision and Pattern Recognition Workshop, 2006.
[11] C. L. Zitnick and P. Doll´ar. Edge boxes: Locating object proposals from edges. In Computer Vision–ECCV 2014, pages 391–405. Springer, 2014.
[12] J. Yan, Z. Lei, L. Wen, and S. Z. Li. The fastest deformable part model for object detection. In Computer Vision and Pattern Recognition (CVPR), 2014 IEEE Conference on, pages 2497–2504. IEEE, 2014.
[13] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich. Going deeper with convolutions. CoRR, abs/1409.4842, 2014.
[14] S. Ren, K. He, R. B. Girshick, X. Zhang, and J. Sun. Object detection networks on convolutional feature maps. CoRR, abs/1504.06066, 2015.
[15] S. Gidaris and N. Komodakis. Object detection via a multiregion & semantic segmentation-aware CNN model. CoRR, abs/1505.01749, 2015.