DLIP Final Project Report

Real-Time Warning System Using Pose Detection for Raised Arm Monitoring

Date: 2025-06-22

Author: Yuha Park, SiYoung Lee

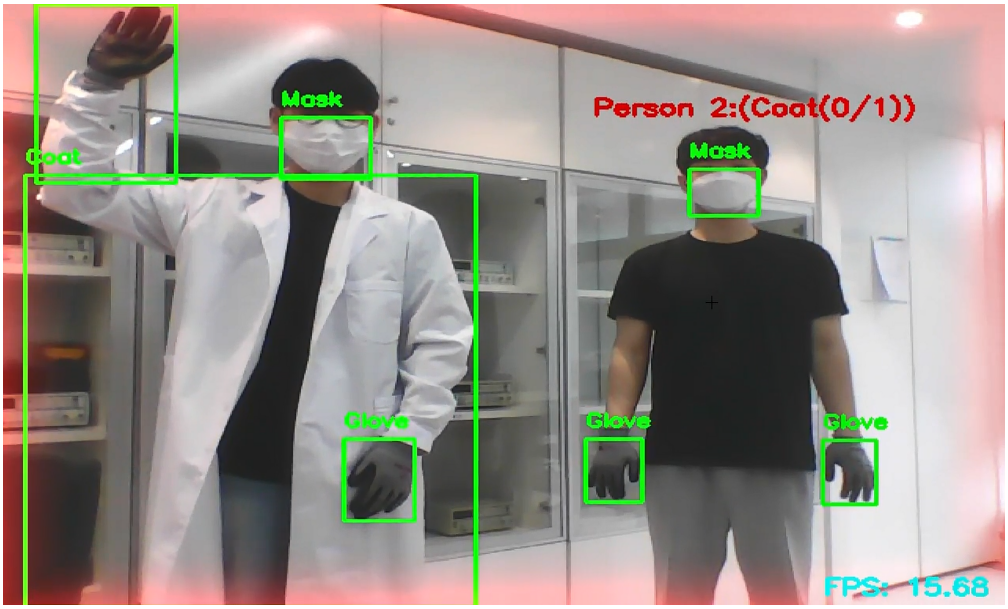
Github: repository link

Demo: Youtube link

I. Introduction

1. Introduction

In this LAB, we designed and implemented an AI-based Lab Safety Monitoring System to enhance safety awareness and real-time risk detection in laboratory environments. The system utilizes deep learning models to automatically detect the presence or absence of essential personal protective equipment (PPE), including lab coats, gloves, and masks. Beyond PPE monitoring, the system is equipped with a critical emergency response feature: if a person raises their hand and maintains the gesture for a few seconds, the system interprets it as a potential emergency signal. In such cases, it immediately triggers a visual warning and sends an alert email to the safety manager. This hands-free, automated approach aims to reduce human oversight and provide faster responses to dangerous situations, ultimately creating a safer and more intelligent laboratory space.



2. Problem Statement

- **Project Objectives**

  This project aims to develop an AI-based real-time laboratory safety monitoring system that addresses two primary goals:

  1. **Automated PPE Compliance Detection**
     Detect whether individuals in the lab are wearing required personal protective equipment (PPE) — such as masks, gloves, and lab coats — using computer vision models.
  2. **Emergency Gesture Recognition**
     Recognize raised-arm gestures as emergency signals using multi-person pose estimation and respond by automatically logging the event, sending an alert email, and overlaying visual warnings.
  3. **Non-Invasive, Real-Time Operation**
     Ensure the system functions in real time (≥15 FPS), supports multiple users simultaneously, and operates without intrusive sensors or facial recognition.

- **Expected Outcome and Evaluation**

  The system is expected to meet the following functional and performance requirements:

Task	Metric	Target Value
Glove Detection	F1-score	≥ 0.90
Mask Detection	F1-score	≥ 0.90
Lab Coat Detection	F1-score	≥ 0.90
Overall Real-time FPS	FPS	≥ 15
Arm Raise Detection	Correctly identify raised-hand gestures and trigger alerts	All features successfully triggered
Alert System	Automatically send email alerts and save logs/images upon detection	All features successfully triggered

  The project will be considered successful if it achieves high PPE detection accuracy, reliable emergency detection, and stable real-time performance across varied lab conditions.

II. Requirements

Hardware

- Laptop or Desktop PC

Software

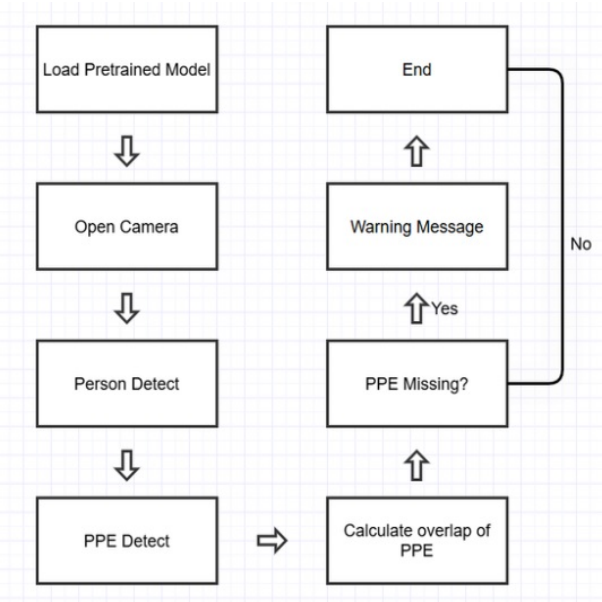
- Python 3.9.21

- Tensorflow 2.9.1
- numpy 1.26.4
- OpenCV 4.5.5
- MoveNet

III. Flow Chart

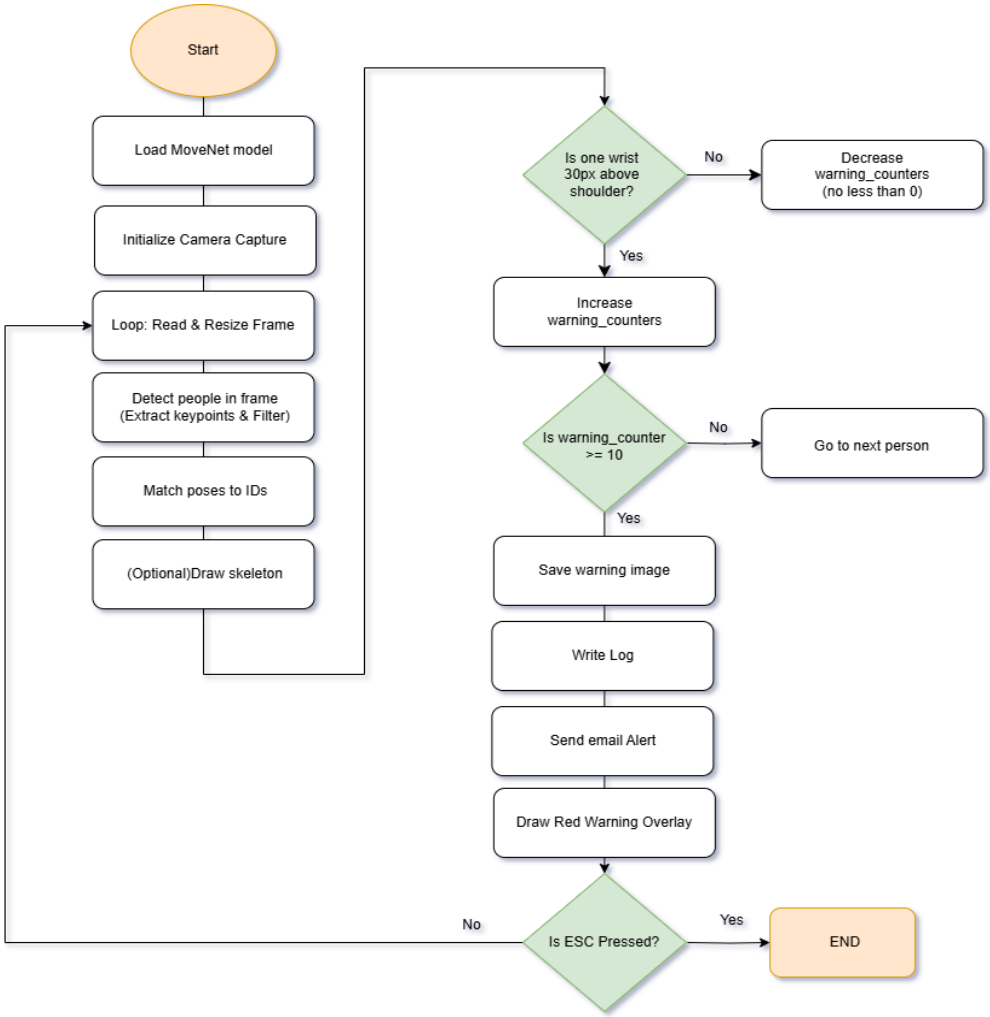
- **Flow Chart of PPE Detection**

The system first loads the pretrained model and opens the camera to capture real-time video. It then detects people in the video, followed by detecting PPE items. Next, it calculates how much each PPE overlaps with the detected person. Based on the overlap values, the system checks whether any required PPE is missing. If PPE is missing, it displays a warning message. Otherwise, the system ends.



- **Flow Chart of Arm Raise Detection**

The system begins by loading the MoveNet model and initializing the camera. Each frame is captured and resized, and then the keypoints of people in the frame are extracted. Detected poses are matched to consistent IDs across frames. If a person's wrist is detected 30 pixels above their shoulder, a warning counter is increased. Once the counter reaches 10, the system saves an image, logs the event, sends an email alert, and shows a red warning overlay. The overlay turns off when the person lowers their hand and the counter drops below the threshold.



IV. Procedure

1. Environment Setup

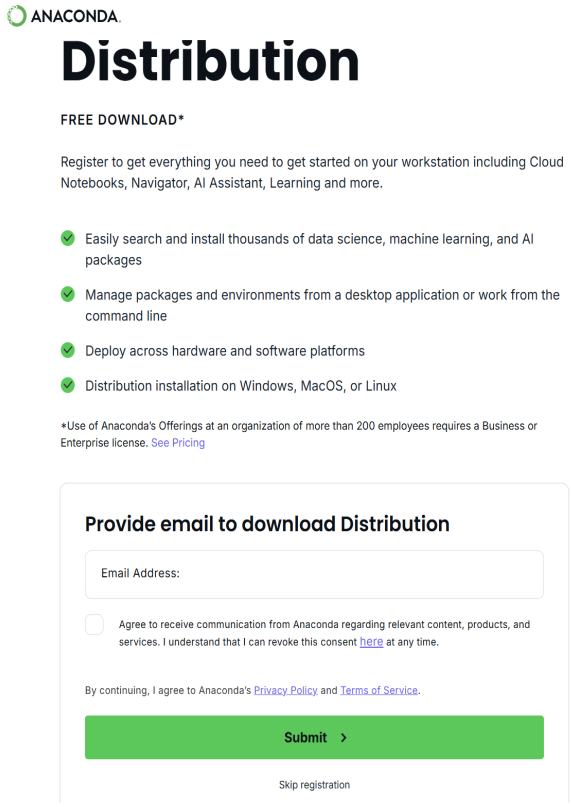
All installations are performed using **Anaconda Prompt** to ensure environment management and compatibility. Separate virtual environments are created for different purposes (MoveNet, YOLOv8, etc.).

2. Installation (Anaconda, Python, YOLO v8, MoveNet)

2.1. Install Anaconda

Anaconda : Python and libraries package installer.

Click here Download the installer on window to Windows 64-Bit Graphical Installer



Follow the following steps

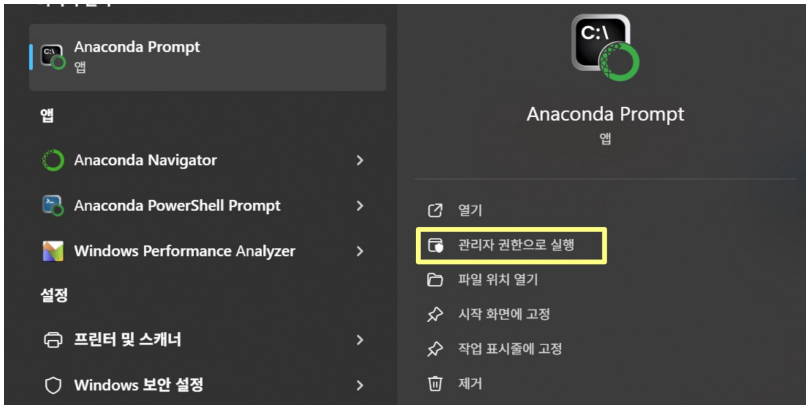
- Double click the installer to launch.
- Select an install for "Just Me"(recommended)
- Select a destination folder to install Anaconda and click the Next button.
- Do NOT add Anaconda to my PATH environment variable
- Check to register Anaconda as your default Python.

2.2. Install Python

Python 3.9

Python is already installed by installing Anaconda. But, we will make a virtual environment for a specific Python version.

- Open Anaconda Prompt(admin mode)



- First, update **conda** and **pip** (If the message "Press Y or N" appears, press Y and Enter key)

```
# Conda Update
conda update -n base -c defaults conda

# pip Update
python -m pip install --upgrade pip
```

- Then, Create virtual environment for Python 3.9, Name the $ENV as `py39`. If you are in base, enter `conda activate py39`

```
# Install Python 3.9
conda create -n py39 python=3.9.12
```

- After installation, activate the newly created environment

```
# Activate py39
conda activate py39
```

2.3. Install Libraries

- Install Numpy, OpenCV, Jupyter (opencv-python MUST use 4.5.5 NOT 4.6.0)

```
conda activate py39
conda install -c anaconda seaborn jupyter
pip install opencv-python
```

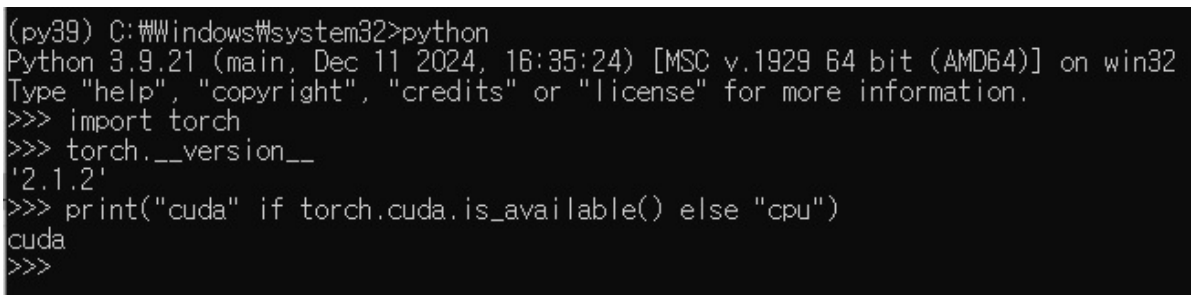
- Install pytorch

```
# CPU Only
conda install -c anaconda cudatoolkit=11.8 cudnn seaborn jupyter
conda install pytorch=2.1 torchvision torchaudio pytorch-cuda=11.8 -c pytorch -c
nvidia
pip install torchsummary
```

- Check GPU in PyTorch

```
conda activate py39
python
import torch
torch.__version__
print("cuda" if torch.cuda.is_available() else "cpu")
```

The output should be the same as the figure below (It's okay if the version is slightly different)



2.4. Install Visual Studio Code

Download from: https://ykkim.gitbook.io/dlip/installation-guide/ide/vscode#installation

Also, read about

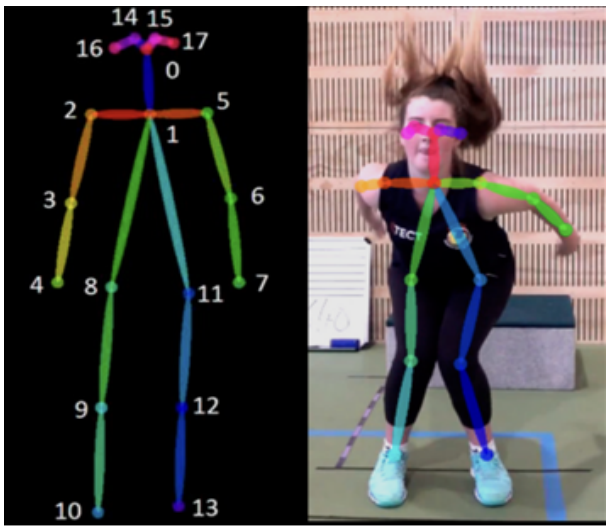
- https://ykkim.gitbook.io/dlip/installation-guide/ide/vscode/python-vscode

2.5. Install TensorFlow

- **TensorFlow** - DL library, developed by Google.

```
conda activate py39
pip install tensorflow==2.9.1
```

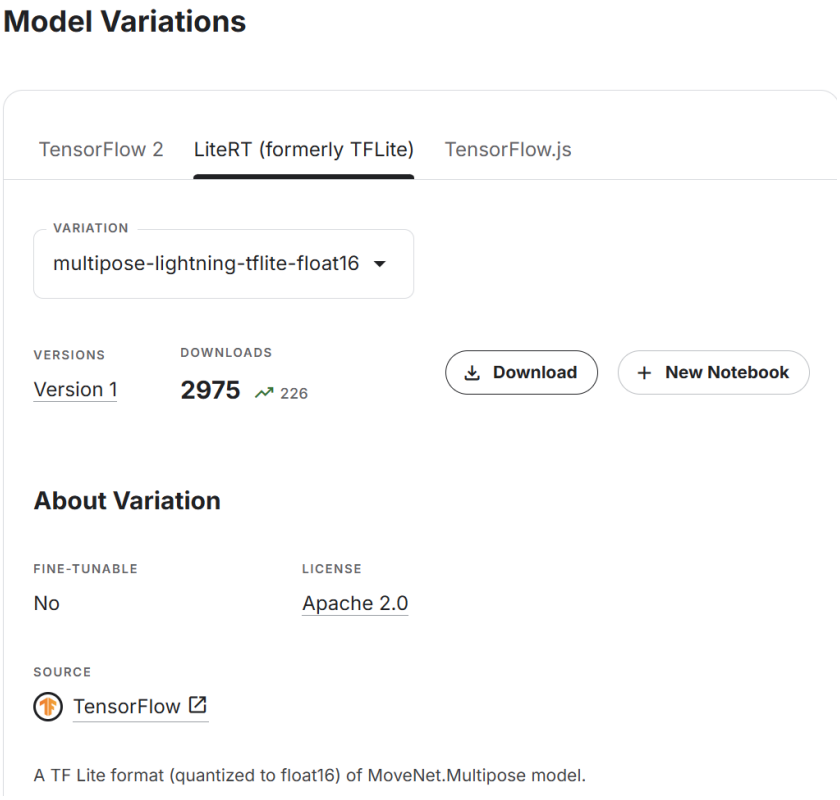
2.6. Install MoveNet Model



Download the MoveNet Thunder model:

- Go to: https://www.kaggle.com/models/google/movenet/tfLite/multipose-lightning-tflite-float16
- Save it locally as a `saved_model` folder using the following code (once):

This creates a `saved_model/` directory which is required to run the script.



2.7. Install YOLO v8

- Create a new environment for YOLOv8 in Anaconda Prompt.

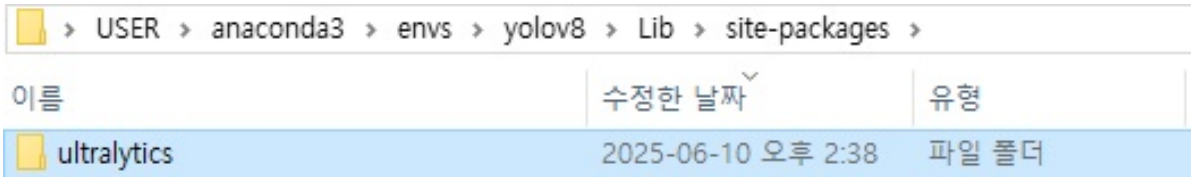
```
e.g. $myENV$ = yolov8
```

- install the latest ONNX

```
conda activate yolov8
pip install ultralytics
pip install onnx
```

- Check for YOLO8 Installation

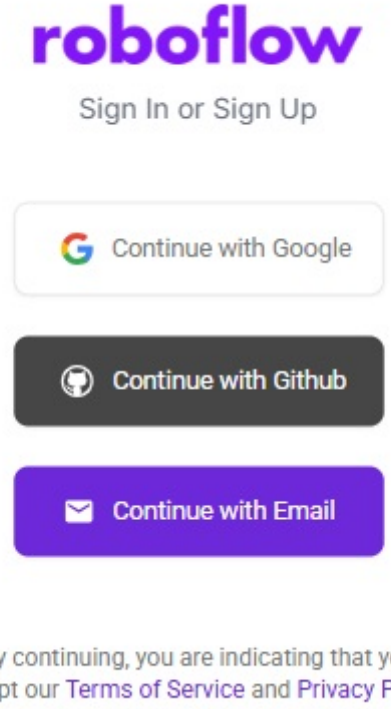
After the installation, you can check the saved source code and libs of YOLOv8 in the local folder



3. Part 1. PPE Detection

3.1. Sign up with email or GitHub

This website provides a data transformation method: https://app.roboflow.com

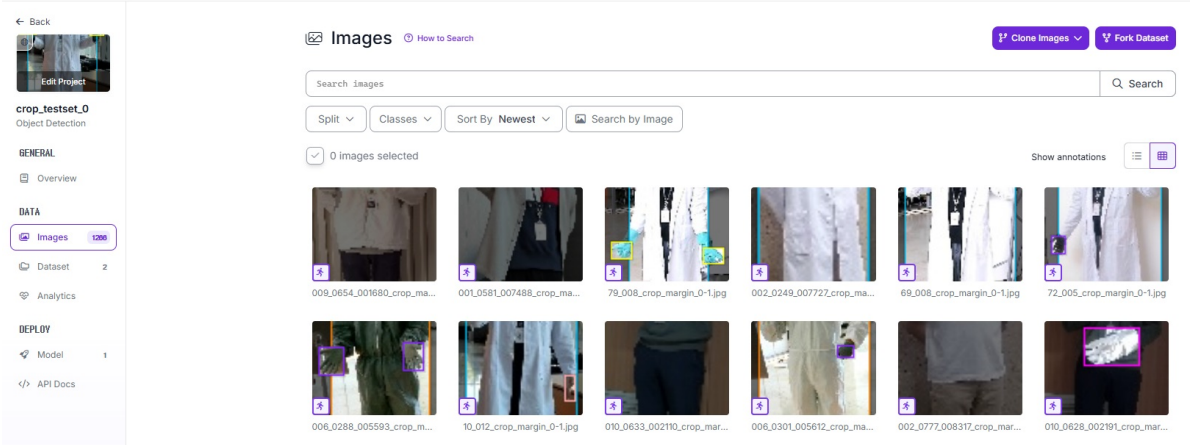


3.2. Download Image and Annotation files

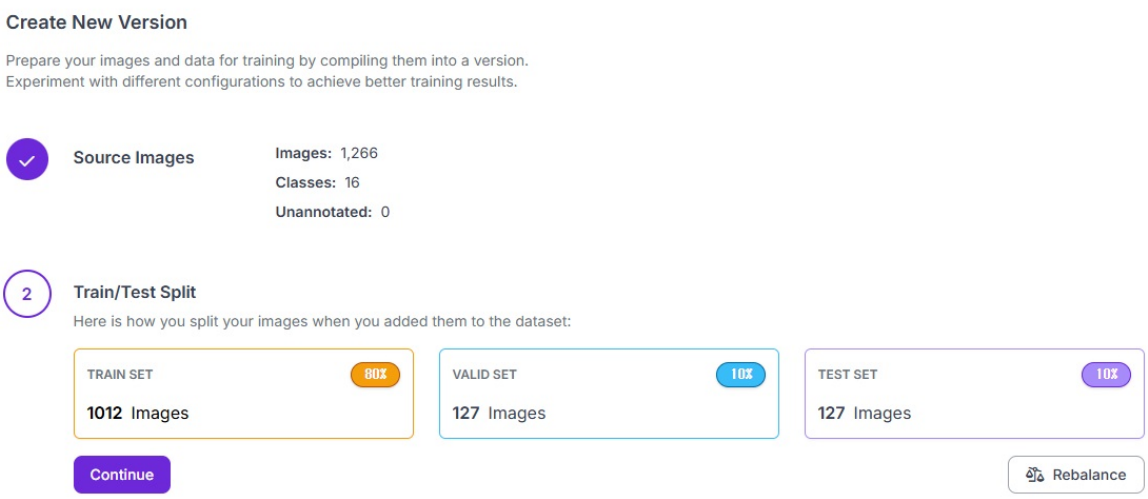
After signing up for Roboflow, follow the link below

Download Link: [Go to Roboflow Dataset]

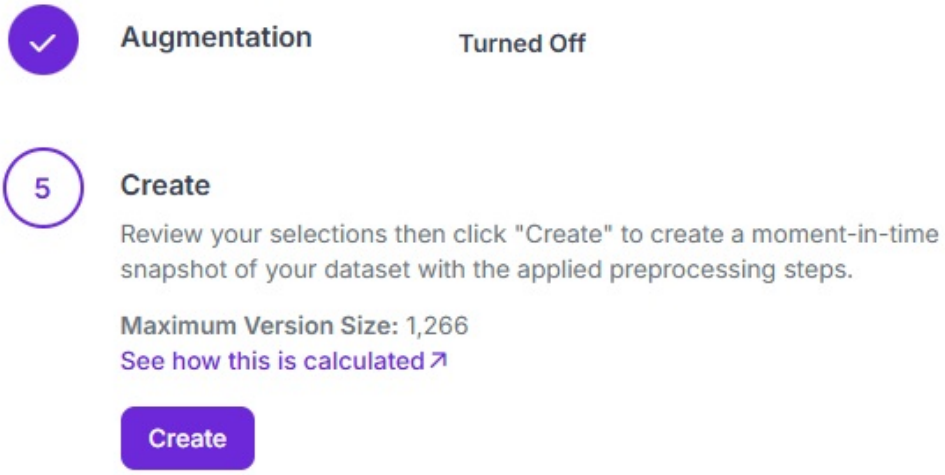
Click "Images" from the left sidebar, and then click the "Fork Dataset" button located at the top right corner



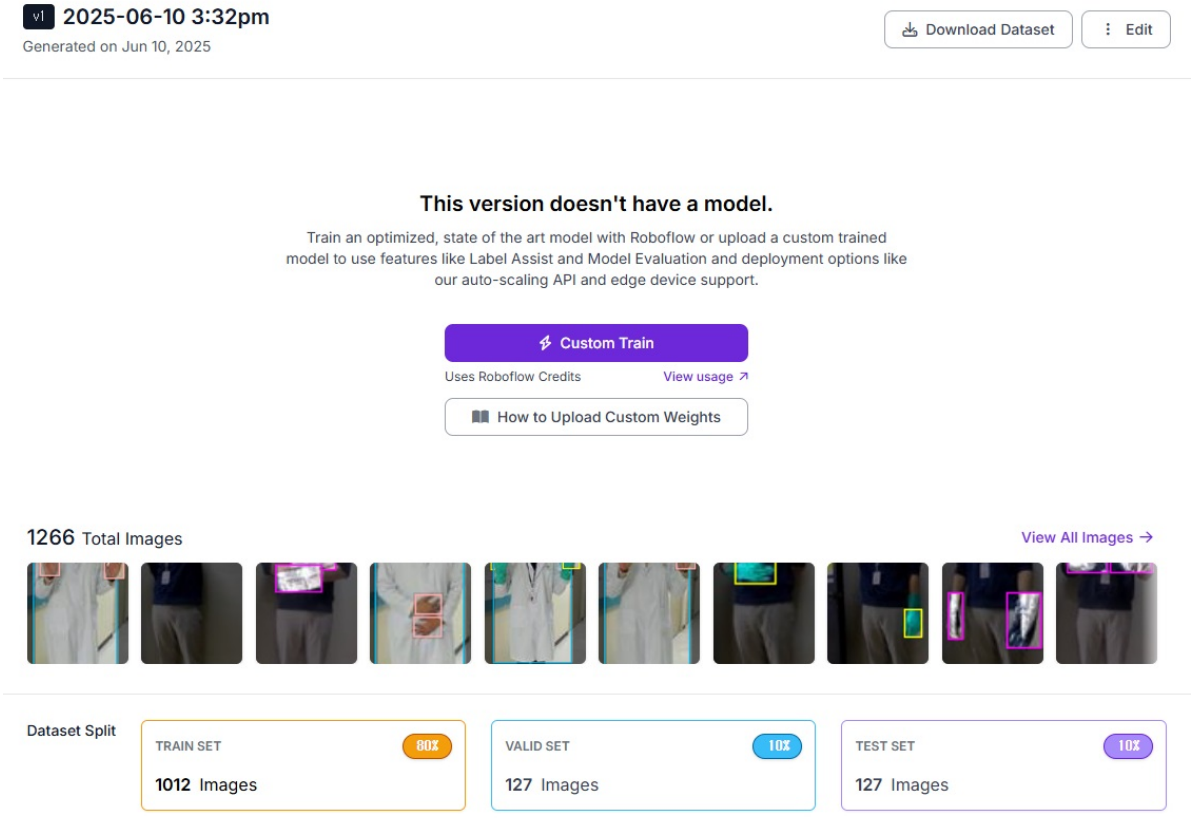
Then, split the dataset into training, validation, and test sets with a ratio of 80/10/10.



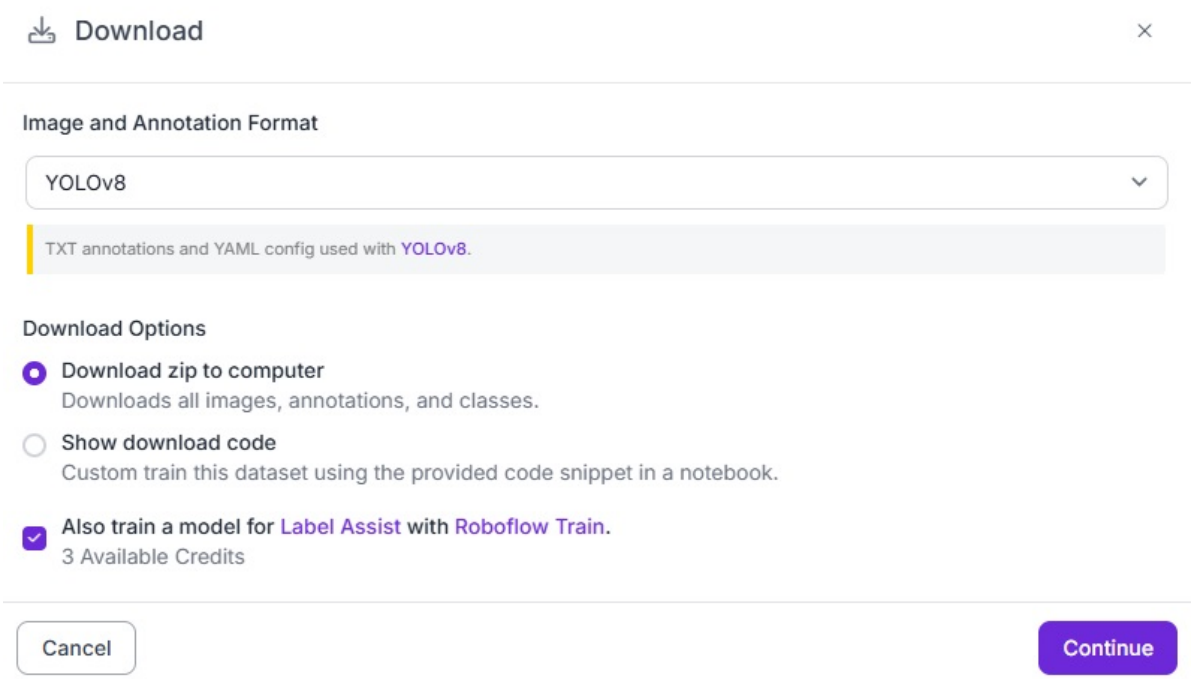
Skip steps 3 and 4, then click the "Create" button in step 5.



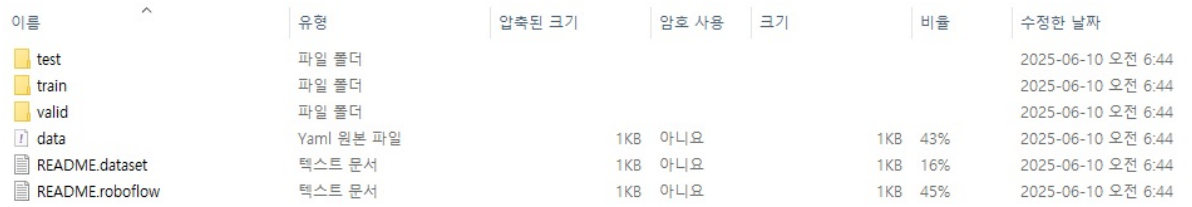
Click the "Download Dataset"



Next, select the YOLOv8 format and click “Download as zip to computer”.



The dataset format must be the same as shown in the figure below.



이름	유형	압축된 크기	암호 사용	크기	비율	수정한 날짜
test	파일 폴더					2025-06-10 오전 6:44
train	파일 폴더					2025-06-10 오전 6:44
valid	파일 폴더					2025-06-10 오전 6:44
data	Yaml 원본 파일	1KB	아니요	1KB	43%	2025-06-10 오전 6:44
README.dataset	텍스트 문서	1KB	아니요	1KB	16%	2025-06-10 오전 6:44
README.roboflow	텍스트 문서	1KB	아니요	1KB	45%	2025-06-10 오전 6:44

3.3. Labeling

The `data.yaml` file defines a total of 15 classes, as shown below.
In this project, only 3 classes **mask**, **Lab_coat**, and **gloves** will be used.

```yaml
train: ../train/images
val: ../valid/images
test: ../test/images

nc: 15
names: ['Face_Shield', 'Fire_Gloves', 'Gas_Mask_Full', 'Gas_Mask_Half', 'Glasses', 'Hazmat_Coat', 'Lab_Coat', 'Latex_Gloves', 'Mask',
 'No_Gloves', 'Normal_Shoes', 'Safety_Glasses', 'Safety_Hat', 'Sandal', 'Work_Gloves']
```

The label data is stored in `.txt` files in the following format

The first number on each line represents the class ID

```
12 0.5212082262210797 0.08427267847557703 0.35347043701799485 0.12130971551261406
6 0.4582262210796915 0.6065485775630703 0.916452442159383 0.703166935050993
2 0.5295629820051414 0.17337627482555018 0.442159383033419 0.15995705850778313
14 0.6638817480719794 0.37278582930756843 0.2480719794344473 0.09715512614063339
```

3.3.1. Removing Class

As the first step, we should remove the unnecessary classes by running the code below.

By inserting the dataset path into the code and running it three times for the `valid`, `train`, and `test` folders, you can remove all the unwanted labels.

Specifically, classes 0, 3, 4, 9, 10, 11, 12, 13 were excluded in this project.

```
import os

label_dir = './datasets/mix2/valid/labels'  # valid/test/train path
remove_ids = ['0', '3', '4', '9', '10', '11', '12', '13']   # Remove ID number

for name in os.listdir(label_dir):
    if not name.endswith('.txt'):
        continue

    path = os.path.join(label_dir, name)
    with open(path, 'r') as f:  # Open files in reading modes
        lines = f.readlines()

    new_lines = []
    for line in lines:
        if line.strip() == "":
            continue
        parts = line.strip().split()    # Split by space

        # if class number not in remove_ids, append it to new lines
        if parts[0] not in remove_ids:
            new_lines.append(line)

    # save result
    with open(path, 'w') as f:
        f.writelines(new_lines)
```

3.3.2. Merging classes

After removing unnecessary classes, we merged similar types of classes to simplify the dataset.

In this project, the remaining classes 1, 2, 5, 6, 7, 8, 14 were grouped as follows: (1, 7, 14), (2, 8), and (5, 6).

We used the following code to relabel similar classes: 1, 7, and 14 were converted to class 1;

2 and 8 to class 2; and 5 and 6 to class 5.

```
import os

label_dir = './datasets/mix2/valid/labels'  # path(valid/test/train)
merge_ids = ['5', '6']    # Class IDs to merge
target_id = '5'           # Merged class ID

for name in os.listdir(label_dir):
    if not name.endswith('.txt'):  # if not txt file, ignore
```

```
        continue
    path = os.path.join(label_dir, name)
    with open(path, 'r') as f:  # Open files in reading modes
        lines = f.readlines()
    new_lines = []
    for line in lines:
        parts = line.strip().split()    # Split by space
        if parts[0] in merge_ids:       # if id in merge_ids, change it to
target_id
            parts[0] = target_id
        new_lines.append(' '.join(parts))
    with open(path, 'w') as f:
        f.write('\n'.join(new_lines))
```

3.3.3. Add Person label using YOLO8

Since there is no class for human Detection in this dataset, we will add it to the label using YOLO8.

Apply the code below to all test/valid/train paths to detect all images and add class 3 to labels if they are people.

```
from ultralytics import YOLO
import cv2
import os

# 1. Path (Train/Test/Valid)
image_dir = 'datasets/mix2/test/images'
label_dir = 'datasets/mix2/test/labels'

# 2. YOLOv8 pre-trained model load
model = YOLO('yolov8n.pt')

# 3. Detection Class: person only (class 0 in COCO)
TARGET_CLASS_ID = 0
NEW_CLASS_ID = 3  # Class ID

# 4. Detect for all images
for name in os.listdir(image_dir):
    if not name.lower().endswith(('.jpg', '.jpeg', '.png')):
        continue

    img_path = os.path.join(image_dir, name)
    label_path = os.path.join(label_dir, name.replace('.jpg',
'.txt').replace('.png', '.txt'))

    results = model(img_path, verbose=False)
    boxes = results[0].boxes

    # 5. load original labels
    if os.path.exists(label_path):
        with open(label_path, 'r') as f:
            original_labels = f.read().strip().split('\n')
    else:
        original_labels = []

    new_labels = []
```

```
    # 6. Add box(detected person)
    for box in boxes:
        cls = int(box.cls[0])
        if cls != TARGET_CLASS_ID:
            continue

        x1, y1, x2, y2 = box.xyxy[0].tolist()
        img = cv2.imread(img_path)
        h, w = img.shape[:2]

        # Transform box to YOLO type
        x_center = (x1 + x2) / 2 / w
        y_center = (y1 + y2) / 2 / h
        width = (x2 - x1) / w
        height = (y2 - y1) / h

        new_labels.append(f"{NEW_CLASS_ID} {x_center:.6f} {y_center:.6f}
{width:.6f} {height:.6f}")

    # 7. Save new person label
    combined = original_labels + new_labels
    with open(label_path, 'w') as f:
        f.write('\n'.join(combined))
```

After this step, you can find person class number(3) in labels file

3.3.4. Remapping Class ID

Since the class numbers in the data.yaml file must be entered sequentially from 0, the class numbers must be remapped.

The class id that was integrated before was remapped as follows:

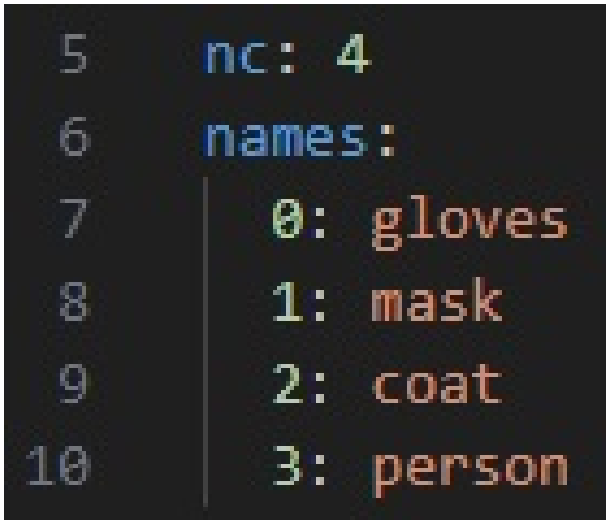
1(gloves) -> 0 / 2(mask) -> 1 / 5(coat) -> 2

```
import os

# 1. Path
label_dir = './datasets/mix2/valid/labels'

# 2. remapping class: exist ID → new ID
remap_dict = {
    '5': '2',
}

# 3. repeat for every file
for name in os.listdir(label_dir):
    if not name.endswith('.txt'): # Open files in reading modes
        continue

    path = os.path.join(label_dir, name)
    with open(path, 'r') as f:
        lines = f.readlines()

    new_lines = []
```

```
    for line in lines:
        parts = line.strip().split()
        # Remapping class if class in remap_dict
        parts[0] = remap_dict.get(parts[0], parts[0])
        new_lines.append(' '.join(parts))

    # 4. overwrite
    with open(path, 'w') as f:
        f.write('\n'.join(new_lines))
```

3.3.5. Change data.yaml

The class numbers have changed to 0,1,2,3, so we should modify data.yaml accordingly.

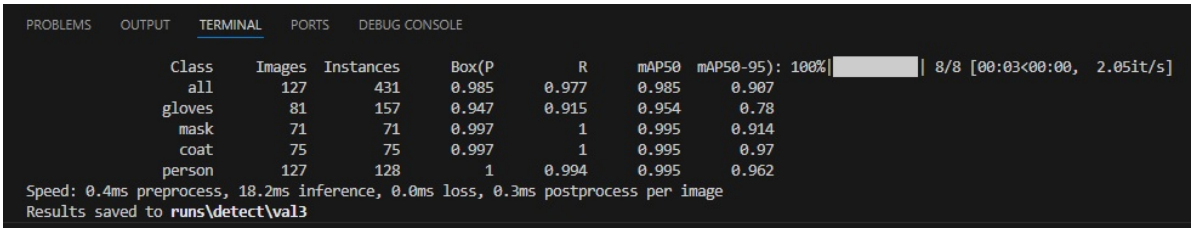


3.4. Training

Once you have finished labeling with the desired class, train the model using the code below.

```
from ultralytics import YOLO

def train():
    # Load a pretrained YOLO model
    model = YOLO('yolov8n.pt')

    # Train the model using the 'maskdataset.yaml' dataset for 30 epochs
    results = model.train(data='data.yaml', epochs=30)

if __name__ == '__main__':
    train()
```

When train is finished, the file where the train results are is printed on the terminal, so you need to remember this. In addition, evaluation results such as precision and recall are output, so you can see the model's performance.



3.5. Test model using Cam

Let's test the model using the laptop's webcam.

In this project, the code was designed to detect four classes—person, gloves, lab coat, and mask—by drawing bounding boxes around them.

If any required safety equipment is missing, a warning message is displayed.

Functions

`box_region`: A function that calculates how much two boxes overlap, expressed as a value

between 0 and 1

`overlapping_degree`: A function that computes how much one box overlaps with another, used to determine whether a person is wearing PPE.

`cam_ppe_check`: A function that activates the webcam, checks whether a person is wearing PPE, and displays a warning message if any item is missing.

Code(PPE Detection)

```
from ultralytics import YOLO
import cv2
import numpy as np

# calculate boxes are overlapped
def box_region(boxA, boxB):
    xA = max(boxA[0], boxB[0])
    yA = max(boxA[1], boxB[1])
    xB = min(boxA[2], boxB[2])
    yB = min(boxA[3], boxB[3])

    # Compute overlapping Area
    interArea = max(0, xB - xA) * max(0, yB - yA)
    boxAArea = (boxA[2] - boxA[0]) * (boxA[3] - boxA[1])
    boxBArea = (boxB[2] - boxB[0]) * (boxB[3] - boxB[1])

    # To prevent division by zero, a very small number is added
    return interArea / float(boxAArea + boxBArea - interArea + 1e-6)

# calculate PPE is on the person
def overlapping_degree(object_box, person_box, threshold=0.3):
    xA = max(object_box[0], person_box[0])
    yA = max(object_box[1], person_box[1])
    xB = min(object_box[2], person_box[2])
    yB = min(object_box[3], person_box[3])

    # Compute overlapping Area
    interArea = max(0, xB - xA) * max(0, yB - yA)
    objArea = (object_box[2] - object_box[0]) * (object_box[3] - object_box[1])

    # To prevent division by zero, a very small number is added
    overlapping_percent = (interArea / (objArea + 1e-6))
```

```
     # if overlapping_percent > threshold, it is considered overlapping
    return overlapping_percent > threshold

def cam_ppe_check():
    model = YOLO('runs/detect/train9/weights/best.pt')  # pretrained model path

    # Class index(data.yaml)
    CLASS_GLOVE = 0
    CLASS_MASK = 1
    CLASS_COAT = 2
    CLASS_PERSON = 3

    # opne Camera
    cap = cv2.VideoCapture(0)

    if not cap.isOpened():
        print("fail to open Camera")
        return

    while True:
        # Read frame at cap
        ret, frame = cap.read()

        # if can't read frame, break
        if not ret:
            break

        # Detect using pretrained model
        results = model.predict(source=frame, save=False, conf=0.3)


        for r in results:

            # Made images with bounding box and labels
            dst = r.plot()
            boxes = r.boxes
            class_ids = boxes.cls.cpu().numpy() # Class ID for each boxes
            xyxy = boxes.xyxy.cpu().numpy() # Get coordinates [x1, y1, x2, y2]

            person_boxes = []
            other_objects = []

            # Repeat for detected boxes
            for cls_id, box in zip(class_ids, xyxy):

                # Classify Person box and PPE box
                if int(cls_id) == CLASS_PERSON:
                    person_boxes.append(box)
                else:
                    other_objects.append((int(cls_id), box))

            # Repeat for detected person
            for i, person_box in enumerate(person_boxes):

                # Sepearate coordinates
                px1, py1, px2, py2 = person_box
```

```
                # PPE variables
                gloves = 0
                masks = 0
                coats = 0

                # Repeat for detected PPE
                for cls_id, obj_box in other_objects:

                    # Calculate the degree of overlap
                    iou = box_region(person_box, obj_box)
                    overlap = overlapping_degree(obj_box, person_box,
threshold=0.3)

                    # If overlap, judge PPE worn by a person
                    if iou > 0.1 or overlap:
                        if cls_id == CLASS_GLOVE:
                            gloves += 1
                        elif cls_id == CLASS_MASK:
                            masks += 1
                        elif cls_id == CLASS_COAT:
                            coats += 1

                missing_items = []
                # If there are any missing items, append it to missing_items
                if gloves < 2:
                    missing_items.append(f"Gloves({gloves}/2)")
                if masks < 1:
                    missing_items.append(f"Mask({masks}/1)")
                if coats < 1:
                    missing_items.append(f"Coat({coats}/1)")

                # If there are missing items, display warning messages
                if missing_items:
                    warning_message = f"Person {i+1}: PPE Missing ({',
                            '.join(missing_items)})"
                    print(warning_message)
                    # Draw Red boxe on the Person boxe
                    cv2.rectangle(dst, (int(px1), int(py1)), (int(px2),
int(py2)), (0, 0,                    255), 2)
                    # location of text
                    text_x = int(px1)
                    text_y = max(30, int(py1) - 10)
                    # Output warning text to image
                    cv2.putText(dst, warning_message, (text_x, text_y),
                                cv2.FONT_HERSHEY_SIMPLEX, 0.6, (0, 0, 255), 2)

            # Display result image
            cv2.imshow("PPE Check (CAM)", dst)

        # if push 'q', quit
        if cv2.waitKey(1) & 0xFF == ord('q'):
            break

    # Close the window
    cap.release()
    cv2.destroyAllWindows()

if __name__ == '__main__':
```

```
    cam_ppe_check()
```

4. Part 2. Arm-Raise Detection

4.1. The library you need

```
import tensorflow as tf
import numpy as np
import cv2 as cv
import os
from datetime import datetime

import smtplib
from email.mime.text import MIMEText
from email.mime.multipart import MIMEMultipart
import threading
```

4.2. Definition of Function

4.2.1. Email Alert Function

```
def send_email_alert(ppe_status: str, person_id: int, timestamp: str):
    SENDER_EMAIL = "SENDER_EMAIL@gmail.com"
    APP_PASSWORD = "PASSWORD"                    # App password for Gmail
    RECEIVER_EMAIL = "RECEIVER_EMAIL@gmail.com"

    try:
        msg = MIMEMultipart()
        msg["Subject"] = "DLIP Alert - PPE Violation Detected"
        msg["From"] = SENDER_EMAIL
        msg["To"] = RECEIVER_EMAIL

        body = f"""
        [DLIP Real-Time Safety Monitoring System]

        Time: {timestamp}
        Person: Person {person_id + 1}
        Detected: {ppe_status}

        ※ This alert was sent automatically by the deep learning-based
surveillance system.
        """

        msg.attach(MIMEText(body, "plain"))

        with smtplib.SMTP_SSL("smtp.gmail.com", 465) as smtp:
            smtp.login(SENDER_EMAIL, APP_PASSWORD)
            smtp.send_message(msg)
            print(f"[Email Sent] To: {RECEIVER_EMAIL} / Content: {ppe_status}")

    except Exception as e:
```

```
        print(f"[Email Failed] {e}")
```

Purpose: Sends an automatic email when a safety violation (e.g., raised arm) is detected.

Inputs:

- `ppe_status`: A string describing the detected violation (e.g., "Arm raise detected").
- `person_id`: Integer ID of the person who triggered the alert.
- `timestamp`: Time when the event occurred (formatted).

Description:

- This function constructs an email with details about the detected event.
- Uses Gmail's SMTP server to send the message securely.
- It runs inside a separate thread to avoid blocking the main video processing loop.

4.2.2. Frame Preprocessing Function

```
def preprocess_frame(frame):
    img = tf.image.resize_with_pad(tf.convert_to_tensor(frame), 256, 256)
    input_tensor = tf.expand_dims(tf.cast(img, dtype=tf.int32), axis=0)
    return input_tensor
```

Purpose: Prepares the input image frame for inference by resizing and padding it to the model's required input shape.

Inputs:

- `frame`: The original image captured from the camera (NumPy array format).

Outputs:

- `input_tensor`: A TensorFlow tensor with size (1, 256, 256, 3), ready to be fed into the model.

Description:

- The original frame is resized and padded to 256×256 pixels.
- Maintains the aspect ratio using TensorFlow's `resize_with_pad`.
- The result is batched using `expand_dims` to match the model's expected input shape.

4.2.3. Pose Detection Function

```
def detect_people(frame, original_shape):
    input_tensor = preprocess_frame(frame)
    output = inference_func(input=input_tensor)
    people = output["output_0"].numpy()[0]
    h, w, _ = original_shape
    keypoints_list = []

    for person in people:
        score = person[55]
        if score < 0.2:
            continue
        keypoints = person[:51].reshape((17, 3))
        shaped = np.multiply(keypoints, [h, w, 1])
        keypoints_list.append(shaped)
```

```
    return keypoints_list
```

Purpose: Uses the MoveNet model to detect keypoints of people in the frame.

Inputs:

- `frame`: Preprocessed image for the model.
- `original_shape`: Shape of the original image to map coordinates back.

Outputs:

- A list of keypoints for each detected person.

Description:

- The MoveNet model returns up to 6 persons' poses with 17 keypoints each.
- Filters out persons with low confidence scores.
- Rescales coordinates to fit the original image size.

4.2.4. Pose Similarity Function

```
def pose_similarity(pose1, pose2):
    dists = []
    for k1, k2 in zip(pose1, pose2):
        if k1[2] > CONFIDENCE_THRESHOLD and k2[2] > CONFIDENCE_THRESHOLD:
            dists.append(np.linalg.norm(k1[:2] - k2[:2]))
    return np.mean(dists) if dists else float('inf')
```

Purpose: Calculates how similar two poses are based on joint locations.

Inputs:

- `pose1`, `pose2`: Two lists of 17 keypoints (each with y, x, confidence).

Output:

- A float value representing the average distance between corresponding keypoints.

Description:

- Compares only the keypoints with confidence above a defined threshold.
- Uses Euclidean distance.
- A smaller value means the poses are more similar.

4.2.5. ID Matching Function

```
def match_poses_with_ids(current_poses, tracked_poses, threshold=40):
    id_map = [-1] * len(current_poses)
    used = [False] * MAX_TRACKED

    for i, curr_pose in enumerate(current_poses):
        best_id = -1
        min_sim = float('inf')
        for j in range(MAX_TRACKED):
            if tracked_poses[j] is None or used[j]:
                continue
            sim = pose_similarity(curr_pose, tracked_poses[j])
```

```
            if sim < threshold and sim < min_sim:
                min_sim = sim
                best_id = j
        if best_id != -1:
            id_map[i] = best_id
            used[best_id] = True

    for i, pose in enumerate(current_poses):
        if id_map[i] == -1:
            for j in range(MAX_TRACKED):
                if not used[j] and tracked_poses[j] is None:
                    id_map[i] = j
                    used[j] = True
                    tracked_poses[j] = pose
                    break

    return id_map
```

Purpose: Matches each current detected person to a unique ID across frames.

Inputs:

- `current_poses`: Detected keypoints in the current frame.
- `tracked_poses`: Previously saved poses.
- `threshold`: Distance threshold to determine matching (default: 40).

Output:

- `id_map`: A list of matched IDs for the current poses.

Description:

- Keeps track of up to 6 people using their pose similarity.
- Ensures consistent ID assignment between frames.

4.2.6. Arm Raise Detection Function

```
def is_one_arm_raised(shaped):
    l_sh_y, _, l_sh_c = shaped[LEFT_SHOULDER]
    r_sh_y, _, r_sh_c = shaped[RIGHT_SHOULDER]
    l_wr_y, _, l_wr_c = shaped[LEFT_WRIST]
    r_wr_y, _, r_wr_c = shaped[RIGHT_WRIST]

    if l_sh_c < CONFIDENCE_THRESHOLD or r_sh_c < CONFIDENCE_THRESHOLD:
        return False
    if l_wr_c < CONFIDENCE_THRESHOLD or r_wr_c < CONFIDENCE_THRESHOLD:
        return False

    left_up = l_wr_y < l_sh_y - 30
    right_up = r_wr_y < r_sh_y - 30
    return left_up or right_up
```

Purpose: Checks if a person's left or right arm is raised above shoulder height.

Input:

- `shaped`: List of 17 keypoints (y, x, confidence) for a person.

Output:

- `True` if at least one arm is raised, otherwise `False`.

Description:

- Compares y-coordinates of wrists and shoulders.
- Uses a 30-pixel threshold to determine raised arms.

4.2.7. Warning Overlay Drawing Function

```
def draw_warning_overlay(frame):
    h, w, _ = frame.shape
    mask = np.zeros((h, w, 3), dtype=np.uint8)
    border_thickness = 50

    for i in range(border_thickness):
        alpha = (1 - i / border_thickness) ** 2
        intensity = int(255 * alpha)
        color = (0, 0, intensity)
        cv.rectangle(mask, (i, i), (w - i, i + 1), color, -1)
        cv.rectangle(mask, (i, h - i - 1), (w - i, h - i), color, -1)
        cv.rectangle(mask, (i, i), (i + 1, h - i), color, -1)
        cv.rectangle(mask, (w - i - 1, i), (w - i, h - i), color, -1)

    frame[:] = cv.addWeighted(frame, 1.0, mask, 0.5, 0)
```

Purpose: Adds a red glowing border to alert viewers during a warning event.

Input:

- `frame`: Current camera frame.

Description:

- Creates a red gradient border to signal danger.
- Enhances the visibility of the warning in real-time monitoring.

4.2.8. Asynchronous Image Save Function

```
def save_image_async(img, filename):
    cv.imwrite(filename, img)
```

Purpose: Saves an image without delaying the main program.

Inputs:

- `img`: Image to be saved.
- `filename`: Name and path of the output file.

Description:

- Intended to be used with Python's `threading.Thread`.

4.2.9. Warning Log Function

```
def log_warning(person_id, frame):
    now = datetime.now()
    timestamp = now.strftime("date_%Y-%m-%d_time_%Hh_%Mm_%Ss")
    readable_time = now.strftime("%Y-%m-%d %H:%M:%S")

    img_copy = frame.copy()
    y, x, _ = prev_poses[person_id][0]
    cv.putText(img_copy, f"WARNING {person_id+1}", (int(x)-50, int(y)-50),
cv.FONT_HERSHEY_SIMPLEX, 0.8, (0,0,255), 2)
    cv.rectangle(img_copy, (int(x)-60, int(y)-70), (int(x)+100, int(y)-30),
(0,0,255), 2)

    filename = f"warning_logs/warning_{timestamp}_person{person_id+1}.jpg"
    threading.Thread(target=save_image_async, args=(img_copy, filename)).start()

    with open("warning_logs/warning_log.txt", "a") as f:
        f.write(f"[{readable_time}] Person {person_id+1} WARNING →
{filename}\n")

    threading.Thread(target=send_email_alert, args=("Arm raise detected",
person_id, readable_time)).start()
    print(f"[LOG] Warning for Person {person_id+1} at {readable_time}")
```

Purpose: Logs a warning event by saving the image, writing to a log file, and sending an email alert.

Inputs:

- `person_id`: ID of the person being tracked.
- `frame`: The frame when the warning was triggered.

Description:

- Draws a red rectangle and label on the warning frame.
- Saves the image asynchronously.
- Appends the event to a text log file.
- Sends an email alert in a separate thread.

4.3. Main Code of Arm-Raise Detection

```
# 1. Initialize tracking state
warning_counters = [0] * MAX_TRACKED
log_saved = [False] * MAX_TRACKED
prev_poses = [None] * MAX_TRACKED
frame_count = 0

# 2. Start video capture
cap = cv.VideoCapture(0)
if not cap.isOpened():
    print("Camera not found.")
    exit()

# 3. Main Loop
while True:
```

```
    frame_count += 1
    ret, frame = cap.read()
    if not ret:
        break

    frame = cv.resize(frame, (640, 480))

    # 4. Pose detection & ID matching
    keypoints_list = detect_people(frame, frame.shape)
    id_map = match_poses_with_ids(keypoints_list, prev_poses)

    # 5. Check if any person is currently triggering a warning
    warning_triggered = any(counter >= WARNING_HOLD_FRAMES for counter in
warning_counters)

    for i, shaped in enumerate(keypoints_list):
        person_id = id_map[i]
        if person_id == -1:
            continue

        prev_poses[person_id] = shaped

        # 6. Detect raised arm
        if is_one_arm_raised(shaped):
            warning_counters[person_id] += 1
        else:
            warning_counters[person_id] = max(0, warning_counters[person_id] -
2)

        # 7. Trigger warning if condition is held long enough
        if warning_counters[person_id] >= WARNING_HOLD_FRAMES:
            if not log_saved[person_id]:
                log_warning(person_id, frame)
                log_saved[person_id] = True
        else:
            log_saved[person_id] = False

    # 8. Draw warning overlay (blink effect)
    if warning_triggered and (frame_count % 20 < 10):
        draw_warning_overlay(frame)

    # 9. Show frame
    cv.imshow("MultiPose Warning", frame)
    if cv.waitKey(10) & 0xFF == 27:
        break

# 10. Clean exit
cap.release()
cv.destroyAllWindows()
```

Purpose:
This block contains the entire real-time logic for the safety monitoring system. It processes live video, detects human poses, tracks individuals, checks for safety violations, and triggers alerts and visual overlays.

Description of the Flow:

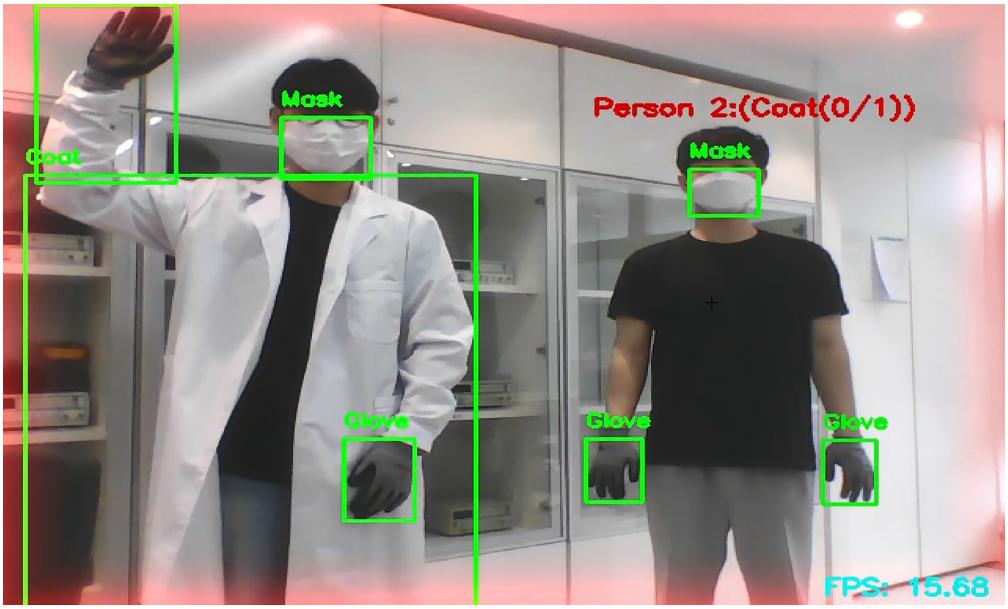
1. **Initialization**:

- Sets up counters for each tracked person.
- Initializes logging and pose tracking memory.

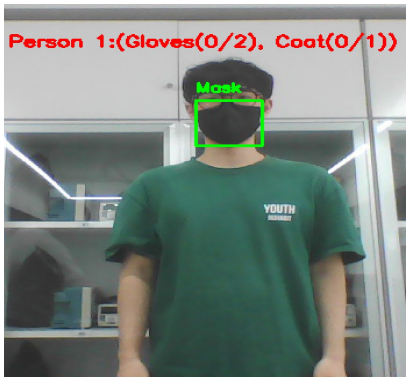
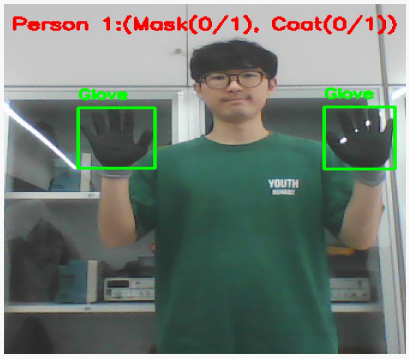
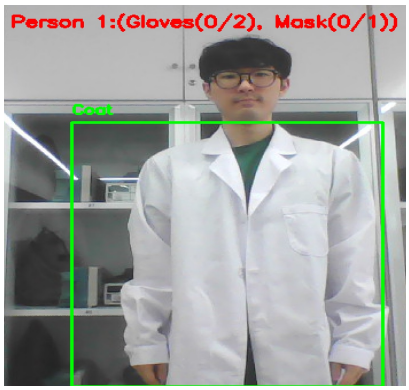
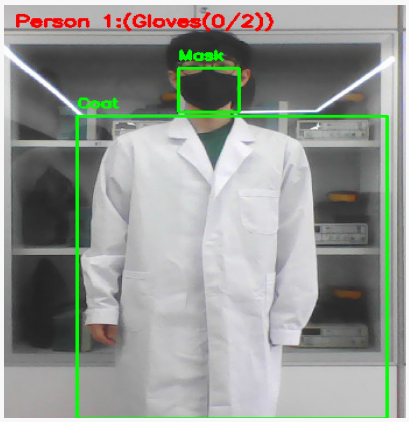
2. **Video Input**:
   - Opens the default camera (index 0).
   - Exits if the camera is not found.
3. **Main Loop** (repeats every frame):
   - Reads and resizes each frame.
   - Detects people and their poses using `detect_people()`.
   - Matches each person to a consistent ID using `match_poses_with_ids()`.
4. **Pose Processing**:
   - Checks if the person is raising their arm using `is_one_arm_raised()`.
   - Updates the warning counter.
   - If a person raises their arm for more than `WARNING_HOLD_FRAMES` frames:
     - Logs the event using `log_warning()`.
     - Sends an alert and saves an image.
5. **Visual Warning**:
   - If any warning is active, a blinking red border is shown using `draw_warning_overlay()`.
6. **Output & Exit**:
   - The processed frame is displayed.
   - The loop continues until the ESC key is pressed.
   - Camera is released, and OpenCV windows are closed cleanly.

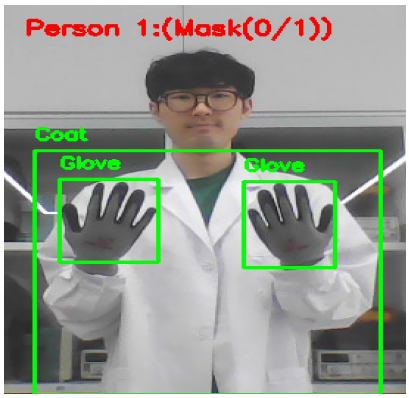
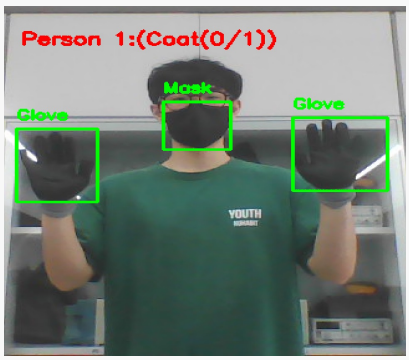
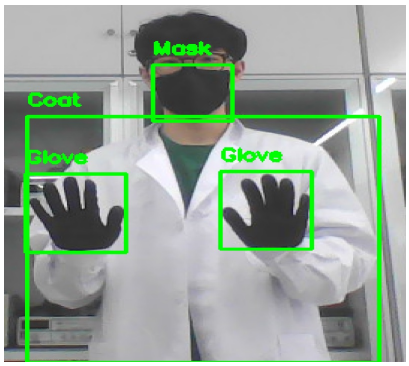
V. Results

0. Final Result



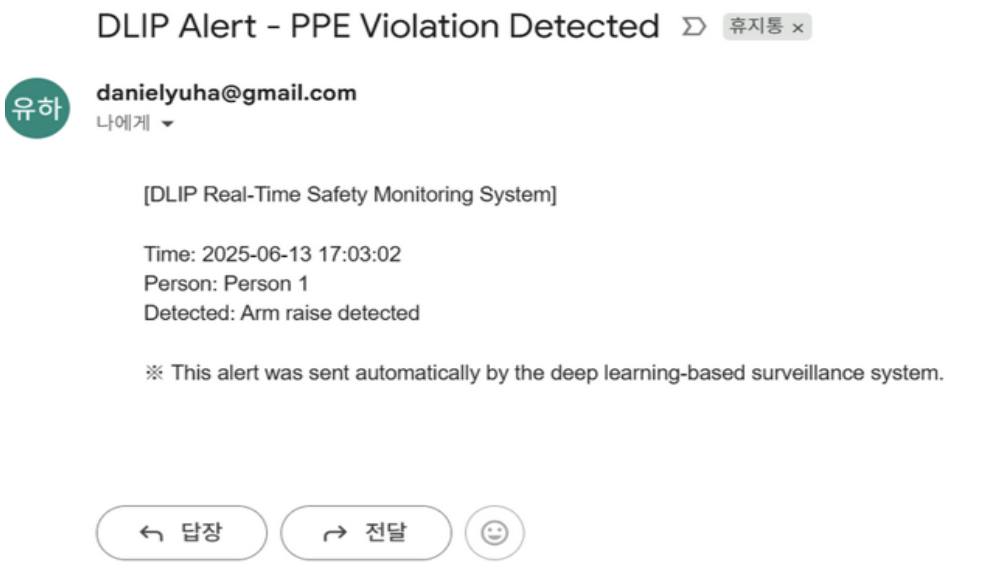
1. Table of seven situations (PPE Detection)

	Images
Situation 1 (Only Mask)	
Situation 2 (Only gloves)	
Situation 3 (Only Coat)	
Situation 4 (Mask & Coat)	

	Images
Situation 5 (Gloves & Coat)	
Situation 6 (Gloves & Mask)	
Situation 7 (Coat & Gloves & Mask)	

2. Result of Arm-Raise Detection

2.1. Send Email Alert



2.2. Write Log with Timestamp

```
[2025-06-13 14:51:23] Person 2 WARNING → warning_logs/warning_date_2025-06-13_time_14h_51m_23s_person2.jpg
[2025-06-13 14:52:56] Person 1 WARNING → warning_logs/warning_date_2025-06-13_time_14h_52m_56s_person1.jpg
[2025-06-13 14:53:41] Person 1 WARNING → warning_logs/warning_date_2025-06-13_time_14h_53m_41s_person1.jpg
[2025-06-13 16:00:25] Person 1 WARNING → warning_logs/warning_date_2025-06-13_time_16h_00m_25s_person1.jpg
[2025-06-13 16:01:49] Person 1 WARNING → warning_logs/warning_date_2025-06-13_time_16h_01m_49s_person1.jpg
[2025-06-13 16:02:24] Person 1 WARNING → warning_logs/warning_date_2025-06-13_time_16h_02m_24s_person1.jpg
[2025-06-13 16:02:49] Person 1 WARNING → warning_logs/warning_date_2025-06-13_time_16h_02m_49s_person1.jpg
```

2.3. Save Warning Image

VI. Evaluation

1. PPE Detection

To evaluate the practical performance of our PPE detection system, we conducted a series of **real-world tests (100 cases per class)** under diverse conditions. Each test focused on a specific class — **Mask**, **Coat**, **Gloves**, and **Person** — and the results were recorded using a confusion matrix. From these matrices, we calculated Accuracy, Precision, Recall, and F1-score for each class.

- **coat**

		Predicted	
		Negative	Positive
Actual	Negative	41	9
	Positive	0	50

Based on the confusion matrix, the following performance metrics were calculated:

- **Accuracy**: 91.0%
- **Precision**: 84.75%
- **Recall**: 100.0%
- **F1-score**: 91.7%

The coat detection system showed perfect recall (100%), meaning it successfully detected all instances where a coat was actually worn. However, the precision dropped to 84.75% due to some false positives. These errors often occurred when the subject was wearing **white or bright clothing** that visually resembled a lab coat. This suggests that the model may confuse similar color textures under certain conditions. To improve precision, further training with hard negative samples (e.g., white shirts, reflective clothing) is recommended.

- **mask**

		Predicted	
		Negative	Positive
Actual	Negative	42	8
	Positive	1	49

Based on the confusion matrix, the following performance metrics were calculated:

- **Accuracy**: 91.0%
- **Precision**: 85.96%
- **Recall**: 98.0%
- **F1-score**: 91.4%

The results indicate that the mask detection system performs well overall, especially in correctly identifying individuals who are wearing masks (Recall = 98.0%). However, the relatively lower precision (85.96%) suggests that there were a few cases where the model incorrectly detected a mask when none was present. These false positives were mainly observed under strong lighting conditions or when white objects partially occluded the face. To further improve precision, it may be necessary to adjust the detection threshold or include more challenging negative samples in training that resemble masks under adverse lighting conditions.

- **gloves**

		Predicted	
		Negative	Positive
Actual	Negative	45	5
	Positive	4	46

Based on the confusion matrix, the following performance metrics were calculated:

- **Accuracy**: 91.0%
- **Precision**: 90.2%
- **Recall**: 92.0%
- **F1-score**: 91.0%

The glove detection system performed reliably overall, achieving a balanced precision and recall around 90%. Some false negatives occurred when only part of the glove was visible (e.g., edge of the hand), and false positives were sometimes caused by **reflections on glass** or **skin-colored objects** in the frame. These results suggest that glove detection is sensitive to hand angle and partial occlusion. Including more diverse hand positions and lighting conditions in training data could help reduce such errors.

- **people**

		Predicted	
		Negative	Positive
Actual	Negative	50	0
	Positive	0	50

Based on the confusion matrix, the following performance metrics were calculated:

- **Accuracy**: 100.0%
- **Precision**: 100.0%
- **Recall**: 100.0%
- **F1-score**: 100.0%

The person detection component achieved perfect performance in this test, with no false positives or false negatives observed. This indicates that the model is highly reliable for identifying human presence in laboratory settings.

2. Summary of Evaluation Results

Class	Precision	Recall	F1-score	Accuracy
Gloves (0)	0.902	0.92	0.91	0.91
Mask (1)	0.86	0.98	0.914	0.91
Coat (2)	0.848	1	0.917	0.91
Person (3)	1	1	1	1

- **Accuracy Analysis**:
  All four classes successfully met the performance requirement of **F1-score > 0.90**, demonstrating that the PPE detection system is suitable for practical deployment. These results confirm that the model performs reliably under diverse real-world conditions.
- **System FPS**:
  The integrated system maintained a real-time processing speed of **16 FPS**, exceeding the minimum requirement of 15 FPS for live monitoring applications.

---

3. Feature Functionality Test

We also validated each system feature functionally. The results are summarized below:

Feature	Status	Notes
PPE Detection	✅Accurate	All PPE classes were detected with high confidence
Bounding Box / Label Display	✅ Implemented	Color-coded labels are shown on-screen
Warning Overlay Display	✅ Working	A red blinking border appears upon alert
Email Alert Function	✅ Tested OK	Sent via Gmail SMTP with an auto message
Warning Image & Log Save	✅ Logged	Image saved + timestamped `.txt` log

---

VII. Discussion

- **ID Matching**

  One of the difficulties was ensuring consistent identity tracking of multiple individuals in real time. Because the system does not use facial recognition or clothing features, we relied solely on pose data. Initially, the system would frequently assign incorrect IDs, causing the warning sign to appear on the wrong person or bounce between people. To resolve this, we implemented a pose-matching algorithm that calculates the average Euclidean distance between keypoints of the current and previous frames. This allowed us to track up to 6 people with stable IDs and display the correct warning to the correct individual.

- **Arm Raise Detection**

  We also encountered issues with false positives from brief movements, especially when someone accidentally lifted their arm slightly. To address this, we introduced a warning counter mechanism, which only triggers a warning if the raised arm is detected for at least 10 consecutive frames. Furthermore, we applied hysteresis logic so the warning doesn't disappear immediately when the arm comes down, reducing flickering and improving the reliability of alerts.

- **Improving Visual Clarity**

  In the PPE detection module powered by YOLO, one significant challenge was the instability of bounding boxes. The bounding boxes often flickered due to minor camera movements or model fluctuations, making the display cluttered and difficult to interpret. To overcome this, we removed the visual bounding boxes and instead displayed text-based status labels above each person (e.g., “Mask: Not Wearing”). This improved both readability and user experience.

- **Detection Challenges Due to Angles and Occlusions**

  We also found that PPE detection was highly angle-dependent, especially for gloves. For instance, if only the side of a hand (the "knife edge") was visible, the model often failed to detect the glove. Similarly, if a person was sitting, partially occluded, or positioned differently than the standing examples in the dataset, detection accuracy dropped. This was due to the dataset bias, which mostly contained images of standing individuals in open view.

- **Filtering False Detections Using Spatial Constraints**

  To reduce false detection of PPE not actually being worn, we applied a spatial filtering method: even if a glove or mask appeared in the frame, it was only counted as "worn" if it was located inside the bounding box of a detected person. This way, stray or background PPE items were not mistakenly considered valid.

- **Overall System Robustness**

  Through iterative testing and refinement, we developed a system that is robust to environmental variation, accurate in tracking, and clear in visual output. These improvements collectively made the system suitable for real-world deployment in lab safety monitoring, emergency response, or smart surveillance scenarios.

VIII. Conclusion

In this project, we developed a real-time AI-based safety monitoring system that detects raised arms as a form of emergency signaling and evaluates PPE compliance using pose estimation and object detection techniques. The system integrates TensorFlow MoveNet for multi-person pose tracking and YOLO-based PPE detection, combined with a custom logic to ensure accurate and consistent monitoring.

We addressed several practical challenges throughout development. These included tracking individuals consistently across frames, eliminating visual clutter from unstable bounding boxes, handling detection errors caused by object angle or occlusion, and reducing false positives due to lighting conditions. Through iterative testing and optimization, we introduced solutions such as pose-based ID tracking, text-based alert displays, frame-based warning counters, and spatial filtering for PPE validation.

As a result, the system is capable of accurately detecting when a person raises their arm for help, identifying missing PPE items, and alerting supervisors through visual cues, image logging, and automated email notifications. The modular design allows it to be extended to other environments or use cases such as industrial safety, smart classrooms, or healthcare.

Overall, this project demonstrates the feasibility of applying deep learning-based computer vision techniques to real-time human monitoring tasks and lays a solid foundation for future enhancements such as fall detection, voice integration, or improved environmental adaptation.

IX. References

1. Google. (2021). *MoveNet: Ultra fast and accurate pose detection model*. TensorFlow Hub. https://www.tensorflow.org/hub/tutorials/movenet
2. Ultralytics. (2024). *YOLOv8 Documentation*. Ultralytics. https://docs.ultralytics.com/
3. OpenCV Team. (2023). *OpenCV: Open Source Computer Vision Library*. https://docs.opencv.org/4.5.5/
4. Roboflow. (2023). *Roboflow: Annotate, Train, Deploy Computer Vision Models*. https://roboflow.com/
5. Gmail Help. (n.d.). *Check Gmail through other email platforms (IMAP)*. https://support.google.com/mail/answer/7126229?hl=en
6. NVIDIA. (2023). *CUDA Toolkit 11.8 Documentation*. NVIDIA Developer. https://docs.nvidia.com/cuda/
7. DLIP Past Project. (2021). *Mask Detection using YOLOv5*. Mask Detection using YOLOv5

X. Appendix

1. Full Code

```
from ultralytics import YOLO
import cv2 as cv
import numpy as np
import os
from datetime import datetime
import tensorflow as tf
import time
import smtplib
from email.mime.text import MIMEText
from email.mime.multipart import MIMEMultipart
import threading

# Keypoint indices for MoveNet
LEFT_SHOULDER = 5
RIGHT_SHOULDER = 6
LEFT_WRIST = 9
RIGHT_WRIST = 10

CONFIDENCE_THRESHOLD = 0.2
WARNING_HOLD_FRAMES = 10
MAX_TRACKED = 6

# Create directory for logs
os.makedirs("warning_logs", exist_ok=True)

# Load MoveNet model
model = tf.saved_model.load("saved_model")
inference_func = model.signatures["serving_default"]

# Send email alert
def send_email_alert(ppe_status: str, person_id: int, timestamp: str):
    SENDER_EMAIL = "danielyuha@gmail.com"
    APP_PASSWORD = "hchv zbsy hekj kuuz"
    RECEIVER_EMAIL = "danielyuha@gmail.com"

    try:
        msg = MIMEMultipart()
        msg["Subject"] = "DLIP Alert - PPE Violation Detected"
        msg["From"] = SENDER_EMAIL
        msg["To"] = RECEIVER_EMAIL

        body = f"""
        [DLIP Real-Time Safety Monitoring System]

        Time: {timestamp}
        Person: Person {person_id + 1}
        Detected: {ppe_status}
        """
        msg.attach(MIMEText(body, "plain"))

        with smtplib.SMTP_SSL("smtp.gmail.com", 465) as smtp:
            smtp.login(SENDER_EMAIL, APP_PASSWORD)
            smtp.send_message(msg)
```

```
            print(f"[Email Sent] To: {RECEIVER_EMAIL} / Content: {ppe_status}")
    except Exception as e:
        print(f"[Email Failed] {e}")

# Resize and pad input image
def preprocess_frame(frame):
    img = tf.image.resize_with_pad(tf.convert_to_tensor(frame), 256, 256)
    input_tensor = tf.expand_dims(tf.cast(img, dtype=tf.int32), axis=0)
    return input_tensor

# Detect people using MoveNet
def detect_people(frame, original_shape):
    input_tensor = preprocess_frame(frame)
    output = inference_func(input=input_tensor)
    people = output["output_0"].numpy()[0]
    h, w, _ = original_shape
    keypoints_list = []

    for person in people:
        score = person[55]
        if score < 0.2:
            continue
        keypoints = person[:51].reshape((17, 3))
        shaped = np.multiply(keypoints, [h, w, 1])
        keypoints_list.append(shaped)

    return keypoints_list

# Calculate similarity between poses
def pose_similarity(pose1, pose2):
    dists = []
    for k1, k2 in zip(pose1, pose2):
        if k1[2] > CONFIDENCE_THRESHOLD and k2[2] > CONFIDENCE_THRESHOLD:
            dists.append(np.linalg.norm(k1[:2] - k2[:2]))
    return np.mean(dists) if dists else float('inf')

# Match current poses with previous IDs
def match_poses_with_ids(current_poses, tracked_poses, threshold=40):
    id_map = [-1] * len(current_poses)
    used = [False] * MAX_TRACKED

    for i, curr_pose in enumerate(current_poses):
        best_id = -1
        min_sim = float('inf')
        for j in range(MAX_TRACKED):
            if tracked_poses[j] is None or used[j]:
                continue
            sim = pose_similarity(curr_pose, tracked_poses[j])
            if sim < threshold and sim < min_sim:
                min_sim = sim
                best_id = j
        if best_id != -1:
            id_map[i] = best_id
            used[best_id] = True

    for i, pose in enumerate(current_poses):
        if id_map[i] == -1:
            for j in range(MAX_TRACKED):
```

```
                if not used[j] and tracked_poses[j] is None:
                    id_map[i] = j
                    used[j] = True
                    tracked_poses[j] = pose
                    break

    return id_map

# Check if at least one arm is raised
def is_one_arm_raised(shaped):
    l_sh_y, _, l_sh_c = shaped[LEFT_SHOULDER]
    r_sh_y, _, r_sh_c = shaped[RIGHT_SHOULDER]
    l_wr_y, _, l_wr_c = shaped[LEFT_WRIST]
    r_wr_y, _, r_wr_c = shaped[RIGHT_WRIST]

    if l_sh_c < CONFIDENCE_THRESHOLD or r_sh_c < CONFIDENCE_THRESHOLD:
        return False
    if l_wr_c < CONFIDENCE_THRESHOLD or r_wr_c < CONFIDENCE_THRESHOLD:
        return False

    left_up = l_wr_y < l_sh_y - 30
    right_up = r_wr_y < r_sh_y - 30
    return left_up or right_up

# Draw red warning overlay
def draw_warning_overlay(frame):
    h, w, _ = frame.shape
    mask = np.zeros((h, w, 3), dtype=np.uint8)
    border_thickness = 50

    for i in range(border_thickness):
        alpha = (1 - i / border_thickness) ** 2
        intensity = int(255 * alpha)
        color = (0, 0, intensity)

        cv.rectangle(mask, (i, i), (w - i, i + 1), color, -1)
        cv.rectangle(mask, (i, h - i - 1), (w - i, h - i), color, -1)
        cv.rectangle(mask, (i, i), (i + 1, h - i), color, -1)
        cv.rectangle(mask, (w - i - 1, i), (w - i, h - i), color, -1)

    frame[:] = cv.addWeighted(frame, 1.0, mask, 0.5, 0)

# Asynchronously save image
def save_image_async(img, filename):
    cv.imwrite(filename, img)

# Log and alert when warning is triggered
def log_warning(person_id, frame):
    now = datetime.now()
    timestamp = now.strftime("date_%Y-%m-%d_time_%Hh_%Mm_%Ss")
    readable_time = now.strftime("%Y-%m-%d %H:%M:%S")

    img_copy = frame.copy()
    y, x, _ = prev_poses[person_id][0]
    cv.putText(img_copy, f"WARNING {person_id+1}", (int(x)-50, int(y)-50),
cv.FONT_HERSHEY_SIMPLEX, 0.8, (0,0,255), 2)
    cv.rectangle(img_copy, (int(x)-60, int(y)-70), (int(x)+100, int(y)-30),
(0,0,255), 2)
```

```
    filename = f"warning_logs/warning_{timestamp}_person{person_id+1}.jpg"
    threading.Thread(target=save_image_async, args=(img_copy, filename)).start()

    with open("warning_logs/warning_log.txt", "a") as f:
        f.write(f"[{readable_time}] Person {person_id+1} WARNING →
{filename}\n")

    threading.Thread(target=send_email_alert, args=("Arm raise detected",
person_id, readable_time)).start()
    print(f"[LOG] Warning for Person {person_id+1} at {readable_time}")

# Initialize tracking state
warning_counters = [0] * MAX_TRACKED
log_saved = [False] * MAX_TRACKED
prev_poses = [None] * MAX_TRACKED

# Calculate IoU
def box_region(boxA, boxB):
    xA = max(boxA[0], boxB[0])
    yA = max(boxA[1], boxB[1])
    xB = min(boxA[2], boxB[2])
    yB = min(boxA[3], boxB[3])
    interArea = max(0, xB - xA) * max(0, yB - yA)
    boxAArea = (boxA[2] - boxA[0]) * (boxA[3] - boxA[1])
    boxBArea = (boxB[2] - boxB[0]) * (boxB[3] - boxB[1])
    return interArea / float(boxAArea + boxBArea - interArea + 1e-6)

# Check overlap ratio
def is_overlapping(obj_box, person_box, threshold=0.3):
    xA = max(obj_box[0], person_box[0])
    yA = max(obj_box[1], person_box[1])
    xB = min(obj_box[2], person_box[2])
    yB = min(obj_box[3], person_box[3])
    interArea = max(0, xB - xA) * max(0, yB - yA)
    objArea = (obj_box[2] - obj_box[0]) * (obj_box[3] - obj_box[1])
    return (interArea / (objArea + 1e-6)) > threshold

# Main function
def cam_ppe_check():
    frame_count = 0
    model = YOLO('runs/detect/train9/weights/best.pt')

    CLASS_GLOVE = 0
    CLASS_MASK = 1
    CLASS_COAT = 2
    CLASS_PERSON = 3

    cap = cv.VideoCapture(0)
    if not cap.isOpened():
        print("Camera open failed.")
        return

    while True:
        start_time = time.time()
        frame_count += 1
        ret, frame = cap.read()
        if not ret:
```

```
            break

        results = model.predict(source=frame, save=False, conf=0.3)

        for r in results:
            dst = frame.copy()
            boxes = r.boxes
            class_ids = boxes.cls.cpu().numpy()
            xyxy = boxes.xyxy.cpu().numpy()

            person_boxes = []
            other_objects = []

            for cls_id, box in zip(class_ids, xyxy):
                if int(cls_id) == CLASS_PERSON:
                    person_boxes.append(box)
                else:
                    other_objects.append((int(cls_id), box))
                    x1, y1, x2, y2 = map(int, box)
                    label = "Glove" if cls_id == CLASS_GLOVE else \
                            "Mask" if cls_id == CLASS_MASK else \
                            "Coat" if cls_id == CLASS_COAT else f"Class
{cls_id}"
                    cv.rectangle(dst, (x1, y1), (x2, y2), (0, 255, 0), 2)
                    cv.putText(dst, label, (x1, y1 - 10),
                                cv.FONT_HERSHEY_SIMPLEX, 0.5, (0, 255, 0), 2)

            for i, person_box in enumerate(person_boxes):
                px1, py1, px2, py2 = person_box
                gloves = 0
                masks = 0
                coats = 0

                for cls_id, obj_box in other_objects:
                    iou = box_region(person_box, obj_box)
                    overlap = is_overlapping(obj_box, person_box, threshold=0.3)

                    if iou > 0.1 or overlap:
                        if cls_id == CLASS_GLOVE:
                            gloves += 1
                        elif cls_id == CLASS_MASK:
                            masks += 1
                        elif cls_id == CLASS_COAT:
                            coats += 1

                missing_items = []
                if gloves < 2:
                    missing_items.append(f"Gloves({gloves}/2)")
                if masks < 1:
                    missing_items.append(f"Mask({masks}/1)")
                if coats < 1:
                    missing_items.append(f"Coat({coats}/1)")

                if missing_items:
                    warning_msg = f"Person {i+1}: ({', '.join(missing_items)})"
                    print(warning_msg)
                    text_x = int(px1)
                    text_y = max(30, int(py1) - 10)
```

```
                    cv.putText(dst, warning_msg, (text_x, text_y),
                               cv.FONT_HERSHEY_SIMPLEX, 0.6, (0, 0, 255), 2)

        keypoints_list = detect_people(dst, dst.shape)
        id_map = match_poses_with_ids(keypoints_list, prev_poses)
        warning_triggered = any(counter >= WARNING_HOLD_FRAMES for counter in
warning_counters)

        for i, shaped in enumerate(keypoints_list):
            person_id = id_map[i]
            if person_id == -1:
                continue

            prev_poses[person_id] = shaped

            if is_one_arm_raised(shaped):
                warning_counters[person_id] += 1
            else:
                warning_counters[person_id] = max(0, warning_counters[person_id]
- 2)

            if warning_counters[person_id] >= WARNING_HOLD_FRAMES:
                if not log_saved[person_id]:
                    log_warning(person_id, dst)
                    log_saved[person_id] = True
            else:
                log_saved[person_id] = False

        if warning_triggered and (frame_count % 20 < 10):
            draw_warning_overlay(dst)

        fps = 1.0 / (time.time() - start_time + 1e-6)
        h, w, _ = dst.shape
        fps_text = f"FPS: {fps:.2f}"
        text_size = cv.getTextSize(fps_text, cv.FONT_HERSHEY_SIMPLEX, 0.6, 2)[0]
        text_x = w - text_size[0] - 10
        text_y = h - 10
        cv.putText(dst, fps_text, (text_x, text_y), cv.FONT_HERSHEY_SIMPLEX,
0.6, (255, 255, 0), 2)

        cv.namedWindow("DLIP System", cv.WINDOW_NORMAL)
        cv.setWindowProperty("DLIP System", cv.WND_PROP_FULLSCREEN,
cv.WINDOW_FULLSCREEN)
        cv.imshow("DLIP System", dst)

        if cv.waitKey(1) & 0xFF == ord('q'):
            break

    cap.release()
    cv.destroyAllWindows()

if __name__ == '__main__':
    cam_ppe_check()
```