PPE detection in construction work

AI Assignment

Prerana Bora

Introduction

The objective of this project is to develop a robust object detection system for identifying Personal Protective Equipment (PPE) using the YOLO model.

Work Flow:

1: Total 416 images and their labels are given. The labels are in PascalVoc format.

2: The first task is changing the formatting of the PascalVoc format to yolov8 format. This is acheived by "pascalVOC_to_yolo.py" script.

3: After this step from ultralytics imported yolo package. From yolo package, yolov8 model, specifically the "YOLOv8x.2" (YOLOv8 large 2) variant is choosen for the detction task. For large objects, higher-stride models like yolov8x.2 often work well. These models effectively sum up larger areas of the image, which can help them detect and classify large objects. By importing yolov8x.2 model with pre-trained weights, transfer learning technique is used.

4: This yolov8 model is first trained with the provided image set with modified annotation in yolo format. First Person detected in images and by the corrected annotation the detected person is cropped by the boundary box and those cropped images are stored for the further training process of the PPE detection model.

5: After cropping annotations are adjusted to fit these cropped regions. Then by using a function these adjusted annotations are converted to YOLO format

6: With the annotations converted and images cropped, the YOLO model was trained on the dataset to detect different PPE: hard-hat, gloves, boots, vest, ppe-suit.

7: After this we run "inference.py" script, that leverages both person detection model and PPE detection model to perform object detection in images,First detects people and then detects Personal Protective Equipment (PPE) within the cropped regions containing people.

Changing annotation PascalVoc to Yolo format:

YOLO expects annotations in a specific format that optimizes its performance and training process. That is why it is necessary to Convert annotations from Pascal VOC format to YOLO format.

Feature	Pascal VOC Format	YOLO Format
File Format	XML	Plain text (.txt)
Bounding Box Coordinates	Absolute pixel values	Normalized relative to image size
Bounding Box Definition	Top-left and bottom-right corners (xmin, ymin, xmax, ymax)	Center coordinates and dimensions (x_center, y_center, width, height)
Coordinate Range	Image pixel dimensions	[0, 1] (normalized)
Class Label	Text label (e.g., "person")	Numeric index (e.g., 0, 1, 2, ...)
Metadata	Contains additional image metadata (e.g., size, filename)	No additional metadata, only bounding box and class

Table 1: Comparison between Pascal VOC and YOLO formats

Listing 1: Python script to convert Pascal VOC annotations to YOLO format.

```
1 import xml.etree.ElementTree as ET  # In order to parse the XML file
```

```
import os
import argparse


classes = ["person", "hard-hat", "gloves", "mask", "glasses", "boots", "vest", "ppe-suit", "ear-protector", "safety-harness"]

def convert(size, box):
    dw = 1. / size[0]
    dh = 1. / size[1]
    x = (box[0] + box[1]) / 2.0 - 1
    y = (box[2] + box[3]) / 2.0 - 1
    w = box[1] - box[0]
    h = box[3] - box[2]
    x = x * dw
    w = w * dw
    y = y * dh
    h = h * dh
    return (x, y, w, h)

def convert_annotation(xml_file, output_dir):
    tree = ET.parse(xml_file)
    root = tree.getroot()
    size = root.find('size')
    w = int(size.find('width').text)
    h = int(size.find('height').text)

    with open(f"{output_dir}/{root.find('filename').text.split('.')[0]}.txt", 'w') as out_file:
        for obj in root.iter('object'):
            cls = obj.find('name').text
            if cls not in classes:
                continue
            cls_id = classes.index(cls)
            xmlbox = obj.find('bndbox')
            b = (float(xmlbox.find('xmin').text), float(xmlbox.find('xmax').text),
                 float(xmlbox.find('ymin').text), float(xmlbox.find('ymax').text))
            bb = convert((w, h), b)
            out_file.write(f"{cls_id} " + " ".join([f"{a:.6f}" for a in bb]) + '\n')

if __name__ == "__main__":
    parser = argparse.ArgumentParser(description='Convert Pascal VOC annotations to YOLO format.')
    parser.add_argument('xml_dir', type=str, help='Directory containing Pascal VOC XML files.')
```

```
43      parser.add_argument('output_dir', type=str, help='Directory to save
           YOLO format annotations.')
44
45      args = parser.parse_args()
46
47      os.makedirs(args.output_dir, exist_ok=True)
48
49      for xml_file in os.listdir(args.xml_dir):
50          if xml_file.endswith(".xml"):
51              convert_annotation(os.path.join(args.xml_dir, xml_file),
                   args.output_dir)
52
53      print(f"Conversion completed! YOLO annotations saved to {args.output
           _dir}")
```

The script first import Necessary Libraries : `xml.etree.ElementTree as ET` For parsing XML files, `os`: For file and directory handling, `argparse`: For handling command-line arguments.

Then it defines Classes, a list that contains the object classes that the script will recognize and convert. Each object in the XML file that matches a class in this list will be included in the YOLO annotation.

Then the `convert` function converts bounding box coordinates from the Pascal VOC format to the YOLO format.

Input Parameters: `1.`size, which is A tuple containing the width and height of the image. `2.`box: A tuple containing the bounding box coordinates in the Pascal VOC format (`xmin`, `xmax`, `ymin`, `ymax`).

The bounding box center coordinates (`x`, `y`) and dimensions (`w`, `h`) are calculated relative to the image size. These values are normalized to be within the range [0, 1]. The function returns the normalized coordinates and dimensions in the YOLO format.

The `convert_annotation` Function parses the XML file and writes the converted YOLO annotations to a text file.

The XML file is parsed to extract image size and object information. For each object in the XML file: The object class is checked against the defined `classes`. If it matches, the bounding box is extracted and converted using the `convert` function. The converted bounding box and class ID are written to a text file in the YOLO format.

In the main block the script uses `argparse` to handle command-line arguments, which allow users to specify the directories containing the Pascal VOC XML files and the directory where the YOLO annotations will be saved.

Training the YOLOv8x Model with Converted Annotations

After converting the annotation files from Pascal VOC format (XML) to YOLO format, the next step involved training the YOLOv8x model using the new annotations and the corresponding image set. By doing this model now interprets and utilizes the bounding box coordinates and class labels in a format compatible with YOLOv8x. The model was trained using the newly prepared annotations in YOLO format, which were specified in a YAML configuration file.

The training process was executed using the `model.train()` function, where the `data` parameter was linked to the YAML configuration file located at `/content/drive/My Drive/yolo/person_data.ya` The model was trained for 100 epochs (`epochs=100`), which allowed sufficient iterations for the model to learn from the data. An image size of 640 pixels (`imgsz=640`) was chosen to balance detail and computational efficiency, and a batch size of 16 (`batch=16`) was used to manage the model's memory usage effectively during training.This setup provided a robust environment for refining the YOLOv8x model to achieve optimal detection performance on the person class.

Processing Annotations and Cropping Images for PPE Detection

After training the YOLOv8x model to detect persons, further process is to crop the detected bounding boxes to prepare for the detection of various PPE items on the detected persons. The code first convert Pascal VOC annotations to YOLO format, cropping images based on the detected bounding boxes of persons, and adjusting the annotations for the detected PPE items within these cropped images.

The primary goal is to provide a more focused detection of PPE items by isolating regions of interest (persons) from the full images and refining the corresponding annotations.

The code performs the following steps:

1. **Class Mapping and Annotation Conversion:** It maps PPE classes to numerical IDs suitable for YOLO format, including items such as hard hats, gloves, boots, vests, PPE suits, etc. It converts bounding box coordinates from the Pascal VOC format to YOLO format.

2. **Cropping Images Based on Detected Persons:** For each person detected in the original image, the code crops the image around the bounding box of the person. This step isolates the person from the surrounding environment, allowing subsequent detection models to focus solely on the person and the PPE items worn by them.

3. **Adjusting Annotations:** For each PPE item within the cropped person region, the bounding box coordinates are adjusted relative to the new image dimensions. Only PPE items that are fully contained within the detected person's bounding box are considered. The adjusted bounding boxes are then converted to YOLO format.

4. **Saving Cropped Images and Adjusted Annotations:** The cropped images and their corresponding annotations are saved into specified output folders. The image filenames are modified to include an identifier (e.g., `person1`, `person2`, etc.) to indicate the individual persons detected in each image.

This approach is essential for accurately identifying PPE items on individuals, as it minimizes the impact of extraneous elements in the images.

```
# yaml to yolo format

import os
from PIL import Image
import xml.etree.ElementTree as ET

class_mapping = {
                 'hard-hat': 0,
                 'gloves': 1,
                 'boots': 2,
                 'vest': 3,
                 'ppe-suit': 4
                 }

def convert_to_yolo_format(bbox, img_width, img_height):
    x_center = (bbox['xmin'] + bbox['xmax']) / (2 * img_width)
    y_center = (bbox['ymin'] + bbox['ymax']) / (2 * img_height)
    width = (bbox['xmax'] - bbox['xmin']) / img_width
    height = (bbox['ymax'] - bbox['ymin']) / img_height
    return x_center, y_center, width, height

def crop_and_adjust_annotations(image_path, annotation, person_bbox):

    image = Image.open(image_path)


    cropped_image = image.crop((person_bbox['xmin'], person_bbox['ymin'
       ], person_bbox['xmax'], person_bbox['ymax']))


    crop_width = person_bbox['xmax'] - person_bbox['xmin']
    crop_height = person_bbox['ymax'] - person_bbox['ymin']


    adjusted_objects = []
    for obj in annotation['object']:
        if obj['name'] != 'person' and obj['name'] in class_mapping:
            bbox = obj['bndbox']

            if (bbox['xmin'] >= person_bbox['xmin'] and bbox['xmax'] <=
               person_bbox['xmax'] and
                bbox['ymin'] >= person_bbox['ymin'] and bbox['ymax'] <=
                   person_bbox['ymax']):

                adjusted_bbox = {
                    'xmin': bbox['xmin'] - person_bbox['xmin'],
                    'ymin': bbox['ymin'] - person_bbox['ymin'],
                    'xmax': bbox['xmax'] - person_bbox['xmin'],
                    'ymax': bbox['ymax'] - person_bbox['ymin']
```

```
                }

                x_center, y_center, width, height = convert_to_yolo_
                    format(adjusted_bbox, crop_width, crop_height)
                class_id = class_mapping[obj['name']]
                adjusted_objects.append(f"{class_id} {x_center} {y_
                    center} {width} {height}")

    return cropped_image, adjusted_objects

def process_annotation(annotation_path, image_folder, output_folder):

    tree = ET.parse(annotation_path)
    root = tree.getroot()


    filename = root.find('filename').text


    annotation = {
        'filename': filename,
        'object': []
    }

    for obj in root.findall('object'):
        name = obj.find('name').text
        bbox = obj.find('bndbox')
        annotation['object'].append({
            'name': name,
            'bndbox': {
                'xmin': int(bbox.find('xmin').text),
                'ymin': int(bbox.find('ymin').text),
                'xmax': int(bbox.find('xmax').text),
                'ymax': int(bbox.find('ymax').text)
            }
        })


    person_count = 0
    for obj in annotation['object']:
        if obj['name'] == 'person':
            person_count += 1
            person_bbox = obj['bndbox']


            image_path = os.path.join(image_folder, filename)
            cropped_image, adjusted_objects = crop_and_adjust_
                annotations(image_path, annotation, person_bbox)

```

```
146
147             output_image_name = f"{os.path.splitext(filename)[0]}_person
                    {person_count}.jpg"
148             output_image_path = os.path.join(output_folder, 'images',
                    output_image_name)
149             cropped_image.save(output_image_path)
150
151
152             output_annotation_path = os.path.join(output_folder, 'labels
                    ', f"{os.path.splitext(output_image_name)[0]}.txt")
153             with open(output_annotation_path, 'w') as f:
154                 f.write('\n'.join(adjusted_objects))
155
156 def main():
157     input_folder = '/content/drive/My Drive/yolo/person'
158     output_folder = '/content/drive/My Drive/yolo/ppe/images'
159     image_folder = os.path.join(input_folder, 'images')
160     annotation_folder = os.path.join(input_folder, 'labels')
161
162
163     os.makedirs(os.path.join(output_folder, 'images'), exist_ok=True)
164     os.makedirs(os.path.join(output_folder, 'labels'), exist_ok=True)
165
166
167     for annotation_file in os.listdir(annotation_folder):
168         if annotation_file.endswith('.xml'):
169             annotation_path = os.path.join(annotation_folder, annotation
                    _file)
170             process_annotation(annotation_path, image_folder, output_
                    folder)
171
172 if __name__ == '__main__':
173     main()
```

PPE detection model:

After preparing the dataset by converting the annotations and cropping the images as detailed in the previous section, the next step is to train a YOLOv8x model for detecting PPE.

The YOLOv8x model was trained using the following parameters:

- **Data Configuration:** The model utilizes a YAML configuration file located at `/content/drive/My Drive/yolo/ppe_data.yaml`, which defines the dataset paths for training and validation images and annotations. This file specifies the necessary classes for PPE detection, such as hard-hats, gloves, boots, vests, and PPE suits.
- **Number of Epochs:** The training was conducted for 100 epoch.
- **Image Size:** An image size of 640 pixels was used.

- **Batch Size:** A batch size of 64 was employed.

The model training was executed using the `model.train` method with the parameters specified above.

The primary objective of this training process is to fine-tune the YOLOv8x model specifically for PPE detection, so that it accurately identify and localize items such as hard-hats, gloves, and vests in images of individuals.

Directory Weights

After this a directory weights created for storing weights and moves the trained model weights for person detection and PPE detection to this directory.

Inference Pipeline

The inference script is created to perform object detection on images using a two-step approach: first detecting persons, and then identifying personal protective equipment (PPE) within the detected person regions. The workflow involves the following key steps:

1. **Model Loading:** The script loads the pre-trained YOLO models for person detection and PPE detection using the specified model paths.
2. **Person Detection:** For each input image, the person detection model is applied to identify bounding boxes around persons. The function `perform_inference` processes each detected bounding box and extracts the corresponding region from the original image.
3. **PPE Detection:** The cropped images containing persons are then passed through the PPE detection model. Bounding boxes are drawn around detected PPE items within these regions, and these coordinates are adjusted relative to the original image.
4. **Annotation and Visualization:** Both the person and PPE bounding boxes are drawn on the original image using OpenCV. For PPE items, labels such as "hard-hat", "gloves", and other classes defined in the model are displayed near their respective bounding boxes.
5. **Batch Processing:** The `process_directory` function facilitates batch processing of images from a specified input directory. It saves the output images with the annotated detections into an output directory. The results are saved with filenames prefixed by "inference_" to distinguish them from the originals.
6. **Command Line Interface:** The script uses `argparse` to handle command-line arguments for input and output directories, as well as paths to the person and PPE detection models. This makes the script versatile and easy to execute on different datasets.

```
174 import os
175 import cv2
176 import argparse
177 import numpy as np
```

```
178 from ultralytics import YOLO
179
180 def load_models(person_model_path, ppe_model_path):
181     print(f"Loading person model from {person_model_path}")
182     print(f"Loading PPE model from {ppe_model_path}")
183     person_model = YOLO(person_model_path)
184     ppe_model = YOLO(ppe_model_path)
185     return person_model, ppe_model
186
187 def perform_inference(image_path, person_model, ppe_model):
188
189     image = cv2.imread(image_path)
190     original_image = image.copy()
191
192
193     person_results = person_model(image)[0]
194
195
196     for person_box in person_results.boxes.xyxy:
197         x1, y1, x2, y2 = map(int, person_box[:4])
198
199
200         person_crop = image[y1:y2, x1:x2]
201
202
203         ppe_results = ppe_model(person_crop)[0]
204
205
206         for ppe_box in ppe_results.boxes.data:
207             ppe_x1, ppe_y1, ppe_x2, ppe_y2, conf, cls = map(int, ppe_box
                    .tolist())
208             full_ppe_x1 = x1 + ppe_x1
209             full_ppe_y1 = y1 + ppe_y1
210             full_ppe_x2 = x1 + ppe_x2
211             full_ppe_y2 = y1 + ppe_y2
212
213
214             cv2.rectangle(original_image, (full_ppe_x1, full_ppe_y1), (
                    full_ppe_x2, full_ppe_y2), (0, 255, 0), 2)
215
216
217             label = ppe_model.names[cls]
218             label_text = f"{label}"
219             cv2.putText(original_image, label_text, (full_ppe_x1, full_
                    ppe_y1 - 10),
220                         cv2.FONT_HERSHEY_SIMPLEX, 0.5, (0, 255, 0), 2)
221
222
223         cv2.rectangle(original_image, (x1, y1), (x2, y2), (255, 0, 0),
```

```
            2)
224         cv2.putText(original_image, "Person", (x1, y1 - 10),
225                     cv2.FONT_HERSHEY_SIMPLEX, 0.5, (255, 0, 0), 2)
226
227     return original_image
228
229
230 def process_directory(input_dir, output_dir, person_model, ppe_model):
231     os.makedirs(output_dir, exist_ok=True)
232
233     for filename in os.listdir(input_dir):
234         if filename.lower().endswith(('.png', '.jpg', '.jpeg')):
235             input_path = os.path.join(input_dir, filename)
236             output_path = os.path.join(output_dir, f"inference_{filename
                }")
237
238             result_image = perform_inference(input_path, person_model,
                ppe_model)
239             if result_image is not None:
240                 cv2.imwrite(output_path, result_image)
241                 print(f"Processed: {filename}")
242             else:
243                 print(f"Skipping: {filename} due to errors")
244
245 def main():
246     parser = argparse.ArgumentParser(description='Run object detection
        and PPE detection on images.')
247     parser.add_argument('input_dir', type=str, help='Directory
        containing input images')
248     parser.add_argument('output_dir', type=str, help='Directory to save
        output images with bounding boxes')
249     parser.add_argument('person_det_model', type=str, help='Path to the
        person detection model')
250     parser.add_argument('ppe_detection_model', type=str, help='Path to
        the PPE detection model')
251     args = parser.parse_args()
252
253
254     person_model, ppe_model = load_models(args.person_det_model, args.
        ppe_detection_model)
255
256
257     process_directory(args.input_dir, args.output_dir, person_model, ppe
        _model)
258
259 if __name__ == '__main__':
260     main()
```

Results and interpretation

Person detection model:

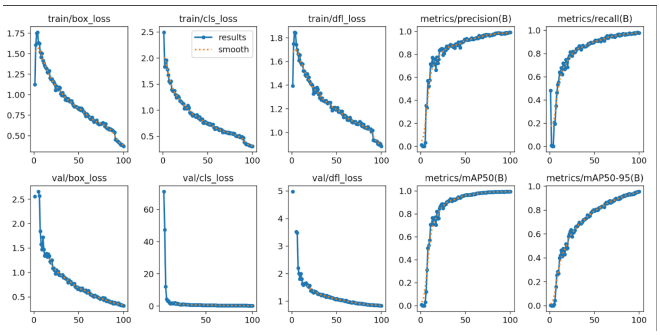

Figure 1: Result of person detection model

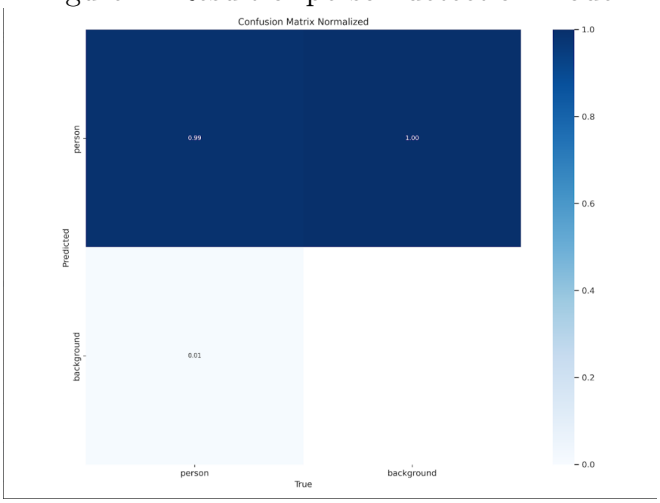

Figure 2: Confusion matrix of person detection model

Examples:

Figure 3: example of person detection

PPE detection model

Examples:

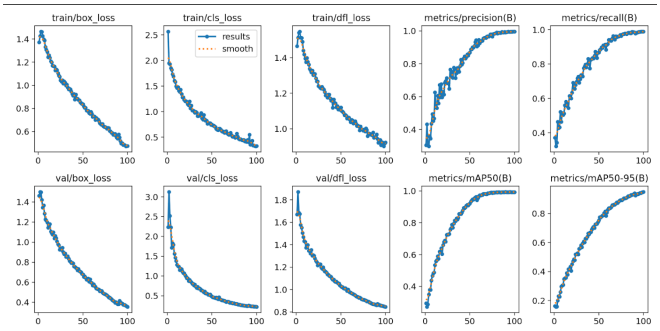

Figure 4: Result

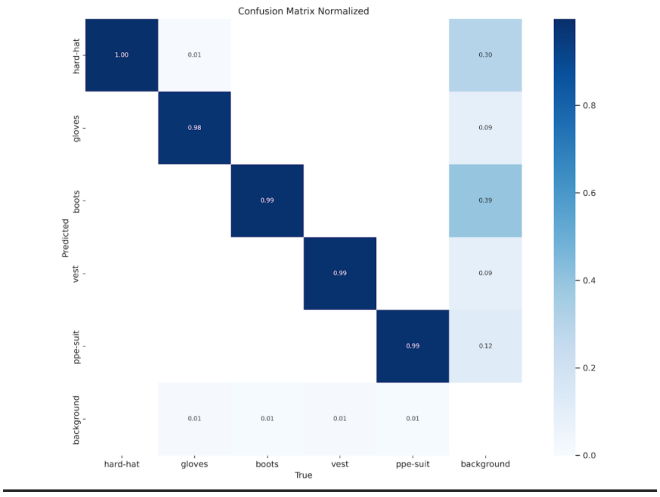

Figure 5: Confusion matrix of person detection model

Final Result:

Figure 6: example of PPE detection

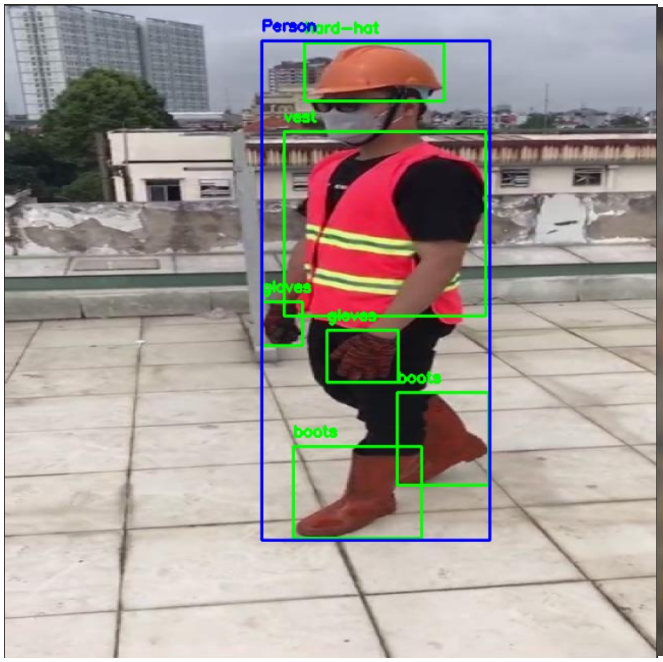

Figure 7: Final result after inference

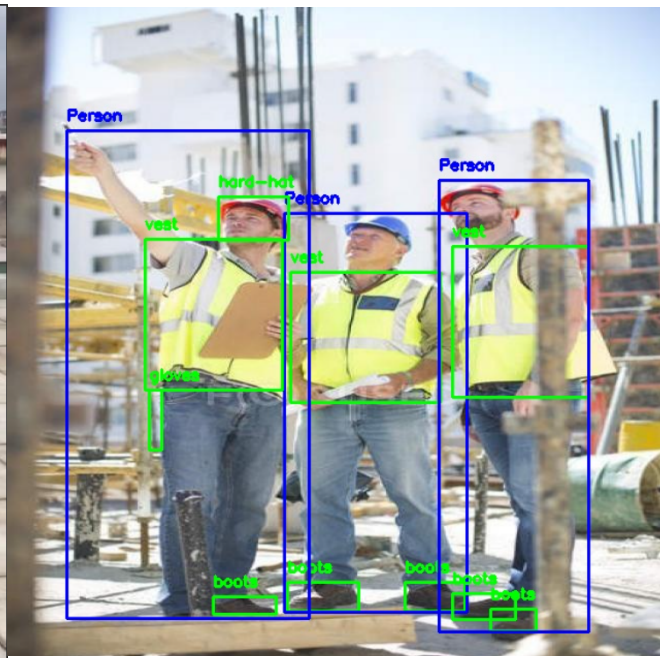

Figure 8: Final result after inference

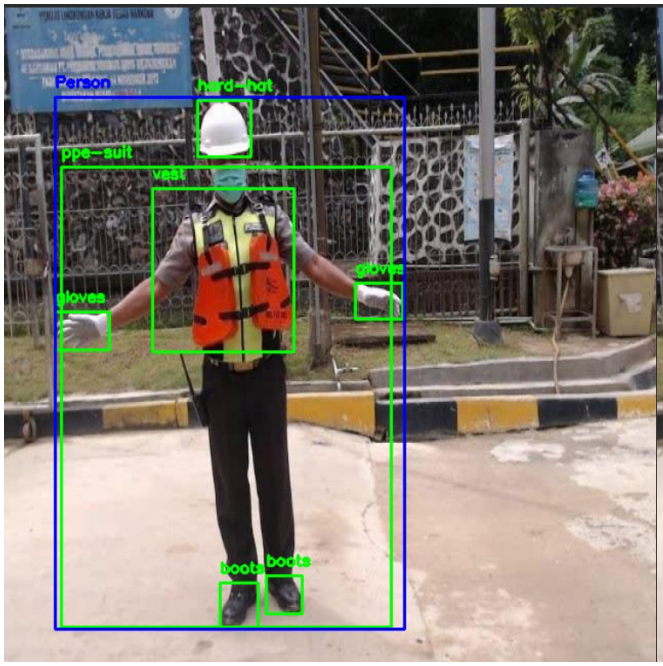

Figure 9: Final result after inference

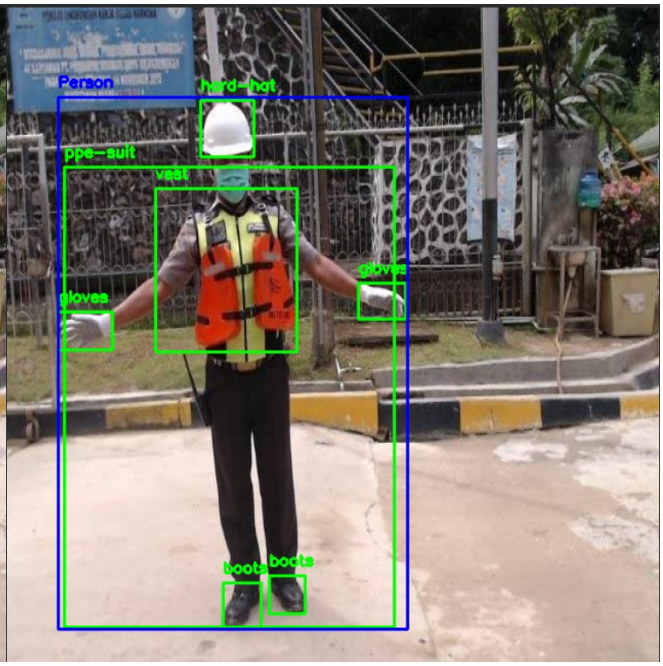

Figure 10: Final result after inference

Figure 11: Final Result