Applied Computer Vision Intern - Assignment

Report

Gourav Singh

August 10, 2024

Contents

1 Abstract

This report describes the implementation of a Applied Computer Vision Intern - Assignment. The approaches, learning process, and evaluation metrics are discussed in detail.

Information :

You will find a dataset directory named "Datasets.zip" which have images along with annotations for the following classes: Person, hard-hat, gloves, mask, glasses, boots, vest, ppe-suit, ear-protector, safety-harness Inside the dataset, you will find two directories, namely images and annotations. Annotations are in PascalVOC format which needs to be converted into yolov8 format for training the "Object Detection" model. Along with these two directories, we have also provided classes.txt for class mapping

Important Instruction :

- You have to train a person detection model on the whole image.
- For PPE detection, please train another model on cropped images after cropping the person's bounding box.
- Please make suitable assumptions regarding class filtering/balancing imbalance classes etc. for reaching an optimized solution.
- You can drop some classes as well if you are seeing inconsistent results but atleast 5 classes model must be trained for ppe detection.
- Please zip everything and share in the email. Please refer to the submission section for instructions on submission.

2 Introduction

Object detection is a crucial task in computer vision with applications in various fields such as security, autonomous driving, and more. This project focuses on customizing the YOLOv8 model for detecting specific objects in images.

The provided dataset.zip looks like

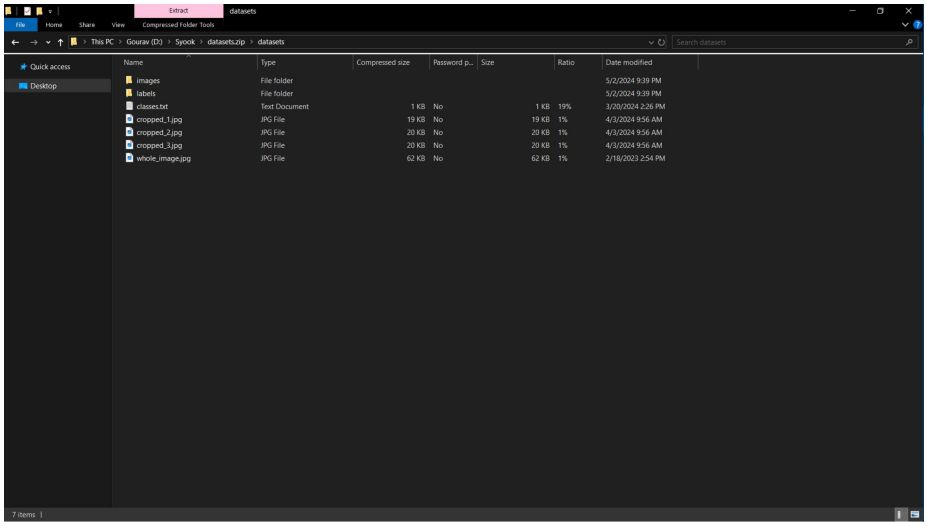

Figure 1: Initial Dataset file structure

Where images contains 416 images and labels contains their labelling in PascalVOC format.The classes.txt file contains the class names that is to be used to detect the objects.

3 Approaches

3.1 PascalVOC to Yolov8 model

PascalVOC annotations are stored in XML files where each object is described with attributes such as name, xmin, ymin, xmax, and ymax, which define the bounding box in pixel coordinates. In contrast, YOLOv8 requires annotations in a text file, where each line corresponds to an object in the image, described by the object class and normalized bounding box coordinates $\langle center_x, center_y, width, height \rangle$.

The Python script accomplish the following tasks:

- Argument Parsing: Using argparse, the script accepts the input directory containing PascalVOC XML files, the output directory where YOLOv8 formatted files will be saved, and an optional class list file that maps class names to class indices.
- Parsing PascalVOC XML Files: The script reads each XML file, extracting relevant information for each object.
- Conversion to YOLOv8 Format: The bounding box coordinates are converted from pixel values to normalized values relative to the image dimensions.
- Saving the Annotations: The converted annotations are saved in a text file with the same base name as the original image file, in the specified output directory.

3.2 Separating Person and PPE annotations

Suppose the folder where the yolo format dataset is saved to be "yolo_labels" and I want to separate the dataset in Person and PPE folder throught the below code :

Listing 1: Python Code

```
import os
import pandas as pd

def action_function(dataframe, file_name):
"""
1) dataframe input
2) filtering for person and ppe
3) exporting in the folder
"""
df = dataframe
df.set_index(df.columns[0], inplace=True)

# Define destination folders
destination_person = "person"
destination_ppe = "ppe"

```

```
    # Ensure destination folders exist
    os.makedirs(destination_person, exist_ok=True)
    os.makedirs(destination_ppe, exist_ok=True)

    # Define file paths for saving
    file_path_p = os.path.join(destination_person, file_name)
    file_path_ppe = os.path.join(destination_ppe, file_name)

    # Filter and adjust data
    person_df = df[df.index == 0]
    ppe_dataframe = df[df.index != 0]
    new_index = [idx - 1 if idx != 0 else idx for idx in ppe_dataframe.index]
    ppe_dataframe.index = new_index

    # Save filtered data to files
    person_df.to_csv(file_path_p, sep=' ', header=False)
    ppe_dataframe.to_csv(file_path_ppe, sep=' ', header=False)

    def setting_data(folder_name):
    for file_name in os.listdir(folder_name):
    file_path = os.path.join(folder_name, file_name)
    if os.path.isfile(file_path):
    print(f"Processing file: {file_name}")
    try:
    # Read the file with the full path
    dataframe = pd.read_csv(file_path, header=None, delimiter=' ')
    action_function(dataframe=dataframe, file_name=file_name)
    except Exception as e:
    print(f"Error processing file {file_name}: {e}")

    # Example usage
    setting_data('yolo_labels')
```

3.3 Compare and Copy

Now we have folder name yolo_labels that consist of yolov8 format annotation data so by comparing through person and ppe folder we would copy images through their name by the below code for both person and ppe separatly through same function :

Listing 2: Compare and Copy Code

```
        import os
        import shutil

        def get_base_name(file_name):
        return os.path.splitext(file_name)[0]

        def compare_and_copy(src_folder1, src_folder2, target_folder):
        # Get lists of files in both source folders
        files1 = [get_base_name(f) for f in os.listdir(src_folder1)]
        files2 = [get_base_name(f) for f in os.listdir(src_folder2)]

        # Find common base filenames
        common_base_names = set(files1).intersection(files2)
```

```
14
15          # Create the target folder if it does not exist
16          os.makedirs(target_folder, exist_ok=True)
17
18          for base_name in common_base_names:
19          # Copy files from src_folder1 to target_folder
20          for file_name in os.listdir(src_folder1):
21          if get_base_name(file_name) == base_name:
22          src_file = os.path.join(src_folder1, file_name)
23          target_file = os.path.join(target_folder, file_name)
24          shutil.copy(src_file, target_file)
25          print(f"Copied {file_name} to {target_folder}")
```

We would run this code twice as we are doing this for person as well as for ppe.

Listing 3: Code for Person

```
1           src_folder1 = 'images'
2           src_folder2 = 'person'
3           target_folder = 'mod_images_person'
4
5           compare_and_copy(src_folder1, src_folder2, target_folder)
```

Listing 4: Code for PPE

```
1           src_folder1 = 'images'
2           src_folder2 = 'ppe'
3           target_folder = 'mod_images_ppe'
4
5           compare_and_copy(src_folder1, src_folder2, target_folder)
```

3.4 Moving Require Files in Structured Format

We are going to create a function that would accept source folder and destination folder and number of files for splitting the data for training yolov8n model for custom dataset.

Listing 5: Code for Structuring

```
1           import os
2           import shutil
3
4           def move_files(src_folder, dest_folder, num_files):
5           # List all files in the source folder
6           files = os.listdir(src_folder)
7
8           # Sort the files (optional: to ensure a consistent order)
9           files.sort()
10
11          # Ensure the destination folder exists
12          os.makedirs(dest_folder, exist_ok=True)
13
14          # Move the specified number of files
15          for i, file_name in enumerate(files[:num_files]):
16          src_file = os.path.join(src_folder, file_name)
```

```
        dest_file = os.path.join(dest_folder, file_name)

        # Move the file
        shutil.move(src_file, dest_file)
        print(f"Moved {file_name} to {dest_folder}")
```

we are going to use this function for 4 times each that is for both Person and ppe setup we would run this below code :

Listing 6: Merged Code for Structuring Person

```
        #moving images for person
        src_folder = 'mod_images_person'
        dest_folder = 'person/dataset/images/train'
        num_files = 329

        move_files(src_folder, dest_folder, num_files)

        src_folder = 'mod_images_person'
        dest_folder = 'person/dataset/images/val'
        num_files = 80

        move_files(src_folder, dest_folder, num_files)

        #moving labels for person
        src_folder = 'person'
        dest_folder = 'person/dataset/labels/train'
        num_files = 329

        move_files(src_folder, dest_folder, num_files)

        src_folder = 'person'
        dest_folder = 'person/dataset/labels/val'
        num_files = 80

        move_files(src_folder, dest_folder, num_files)
```

Listing 7: Merged Code for Structuring Person

```

        #moving images for ppe

        src_folder = 'mod_images_ppe'
        dest_folder = 'ppe/dataset/images/train'
        num_files = 329

        move_files(src_folder, dest_folder, num_files)


        src_folder = 'mod_images_ppe'
        dest_folder = 'ppe/dataset/images/val'
        num_files = 80

        move_files(src_folder, dest_folder, num_files)

        #moving labels for person
        src_folder = 'ppe'
```

```
        dest_folder = 'ppe/dataset/labels/train'
        num_files = 329

        move_files(src_folder, dest_folder, num_files)

        src_folder = 'ppe'
        dest_folder = 'ppe/dataset/labels/val'
        num_files = 80

        move_files(src_folder, dest_folder, num_files)
```

3.5 config.yaml file generation

Training Path : The 'train' parameter specifies the directory path where your training images are stored. YOLOv8 requires access to these images to learn the features and characteristics of the objects you want to detect. This path should point to a folder that contains all the training images, typically in JPEG, PNG, or another supported image format. The training data is used to teach the model how to recognize patterns and features in these images, which is fundamental for achieving accurate object detection.

Validation Path : The val parameter specifies the directory path for validation images. Validation images are separate from training images and are used to evaluate the model's performance during the training process. This helps in monitoring how well the model is generalizing to new, unseen data. By comparing performance on validation data, you can assess whether the model is overfitting or underfitting . This separation is crucial for fine-tuning the model and avoiding biases that may occur if the model only sees the training data.

Number of Classes : The nc parameter denotes the number of distinct classes or categories in your dataset. This value should match the total number of unique objects or categories that the model needs to learn to detect. For instance, if your dataset includes two categories: "person" and "PPE" (personal protective equipment), then nc would be set to 2. This parameter informs the model about how many different objects it should be able to identify and distinguish between.

Class Names : The names parameter provides a list of class names corresponding to the indices defined by the nc parameter. Each class is assigned a unique index starting from 0. For example, in a dataset with "person" and "PPE," you would list these class names with indices like 0: 'person' and 1: 'ppe'. This list allows the model to map its predictions to human-readable class names. Accurate class naming is essential for interpreting the model's output and ensuring that detected objects are correctly labeled.

The config.yaml file serves as the blueprint for training the YOLOv8 model, ensuring that all necessary information about the dataset and training parameters is provided. Proper configuration of this file is vital for successful model training and accurate object detection.

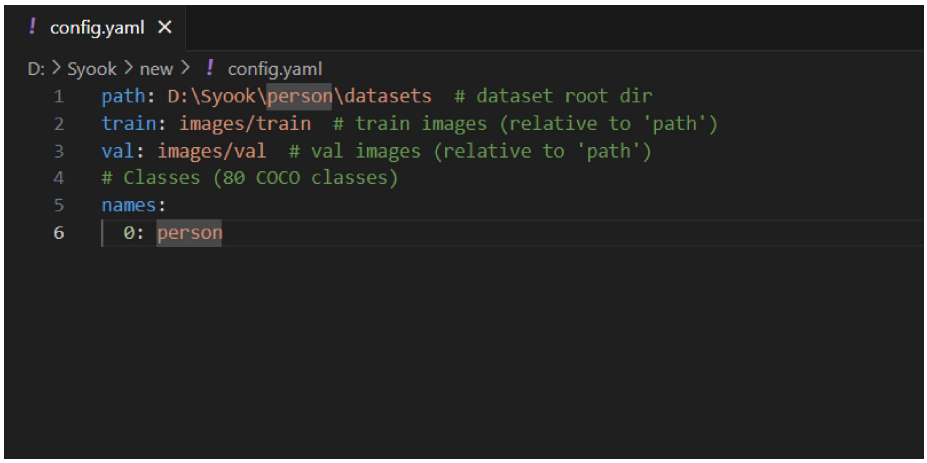

Figure 2: Person

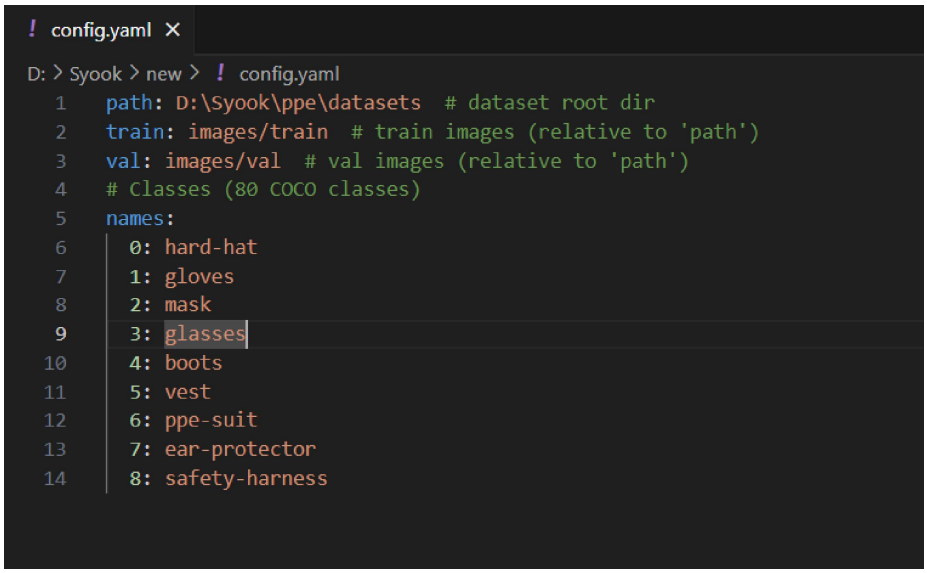

Figure 3: PPE

4 Training Dataset

Training was conducted using the following hyperparameters: learning rate, batch size, and number of epochs. Data augmentation techniques were applied to improve model generalization.Training an object detection model using YOLO (You Only Look Once) involves several key steps that collectively enable the model to accurately identify and locate objects within images. The process begins with preparing a well-labeled dataset, where each image is annotated

with bounding boxes that define the objects' positions and their corresponding labels. This labeled dataset is crucial as it serves as the foundation for the model's learning process. During training, the YOLO model divides each input image into a grid, and within each grid cell, it predicts bounding boxes and class probabilities. The model is trained to minimize the difference between its predictions and the actual annotated data, learning to recognize patterns and features that correspond to the objects of interest.

A significant advantage of YOLO is its ability to perform real-time object detection due to its single-stage architecture, which combines object localization and classification in one go. This contrasts with other models that use a multi-stage approach, leading to slower detection speeds. Throughout the training process, the model adjusts its internal parameters through backpropagation, optimizing its ability to detect objects with increasing accuracy. The model's performance is regularly evaluated on a separate validation set, allowing for the adjustment of hyperparameters and techniques like data augmentation to improve generalization.

Upon completion of training, the model is capable of detecting and classifying objects in new, unseen images with a high degree of accuracy and speed. YOLO's efficiency and effectiveness make it a popular choice for various real-world applications, from autonomous vehicles to surveillance systems, where fast and reliable object detection is essential.

We have to go to the root directory which is person diretory and then execute the below command :

```
!yolo train model=yolov8n.pt data=config.yaml epochs=50
```

We have to go to the root directory which is ppe directory and then execute the below command :

```
    !yolo train model=yolov8n.pt data=config.yaml epochs=50
```

5 Evaluation Metrics

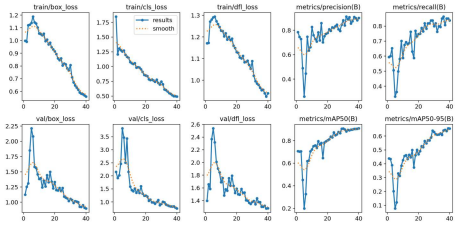

Figure 4: Person Result

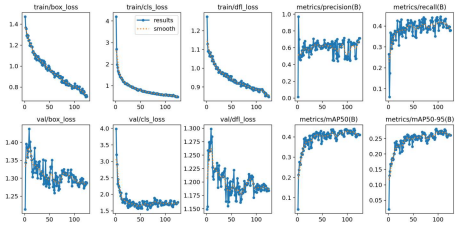

Figure 5: PPE Result

6 Conclusion

There is a file called inference.py the task of that file is to take the models/weights of both people and ppe detection and a directory full of images to detect and an output directory. The flow of this code is something it first detect people in the image then crops the image and detect ppe objects and then make a bounding box around them and again unite the image back to the main form.

Note :

- At first execute the PascalVOC_to_yolo.py properly.
- For customly execute the detection follow the processes as defind in Approaches.
- Please kindly modify some dynamic codes according to your environment in yaml files.
- You can go through yolov8 documentation for training the dataset in your environment.
- As explained earlier you can use inference.py to predict the detection.

- Both the python scripts have their command at the last line to execute through cli

6.1 Final Directory Submission :

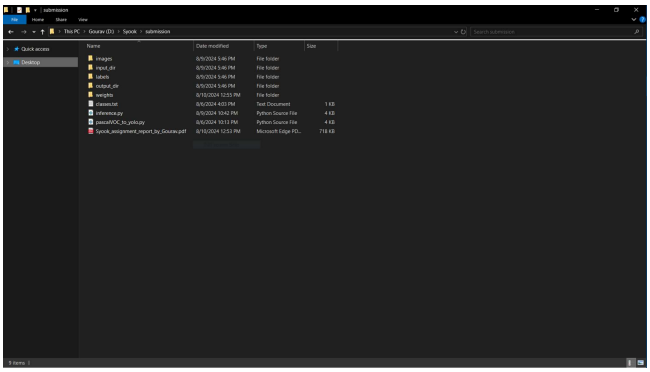

Figure 6: Submission Directory Structure

- **images** folder contains the images that you provided
- **labels** folder contains the labels that you provided
- **weights** folder contains two models one for person detection and one for ppe detection
- **input_dir** folder contains demo images for detection. You can insert your images in this folder.
- **output_dir** folder contains the detected objects images that is being detected from input_dir.You can delete the images as the images gets generated when you run inference.py
- **classes.txt** file contains the classes of the labels.
- **pascal_to_yolo.py** and **inference.py** execution technique is written in the last line of the programs
- **Syook_assignment_report_by_Gourav.pdf** file contains the Report of the Assignment

7 References

- Yolov8, Year. Available at: https://docs.ultralytics.com/datasets/detect/
- Google-Colab,for training using GPU. Available at: https://colab.research.google.com/