电 子 科 技 大 学

UNIVERSITY OF ELECTRONIC SCIENCE AND TECHNOLOGY OF CHINA

实验报告

EXPERIMENT REPORT

|  |
| --- |
| C:\Users\ADMIN\AppData\Local\Temp\ksohtml20256\wps1.png |
| |  |  | | --- | --- | | **STUDENT NAME:** | JAHID SHAHIDUL ISLAM | | **STUDENT ID:** | 202324090107 | | **COURSE NAME:** | PYTHON PRACTICAL PROGRAMMING | | **TEACHER NAME:** | PROF. RAO YUNBO | | **EXPERIMENT NO:** | THREE | | **DATE:** | 20th MAY 2024 | |  |

1. Experiment title：Install Python Platform
2. Experiment hours：4h Experiment location: Software Building 400
3. Objectives

At the end of this experiment, you will be able to:

* How to install Labelimg software in your devices.
* How to change xml to txt file.

1. Experimental contents & step
2. Installing the Labelimg software for Windows
3. Change xml to txt file
4. Change txt to xml file
5. Change xml to Json file
6. Change Json to xml file
7. Change Json to txt file
8. Experimental analysis

# Installing the Labelimg Software for Windows

**Step 1: Prepare Environment**

It's recommended to work within a dedicated environment. If using conda, create one with:

*conda create -n labelimg python=3.8*

*conda activate labelimg*

**Step 2: Uninstall**

previous version of labelImg, uninstall it:

*pip uninstall labelImg*

**Step 3: Install LabelImg**

Install the latest version using pip:

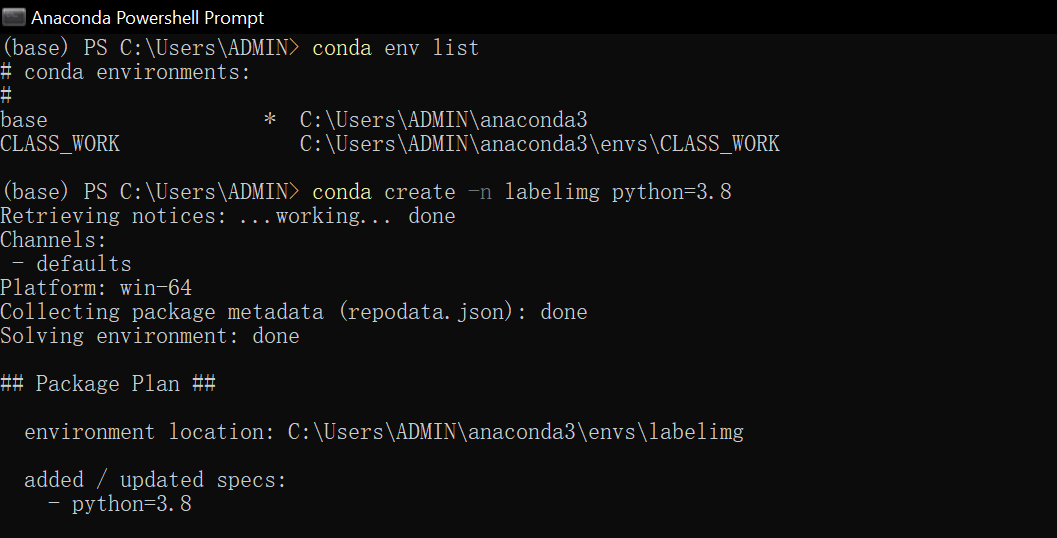
*pip install labelImg*

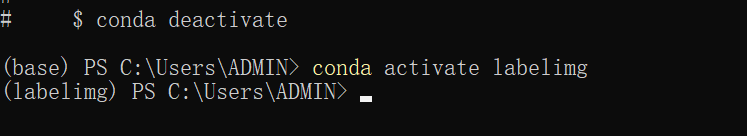
**Step 4: Launch LabelImg**

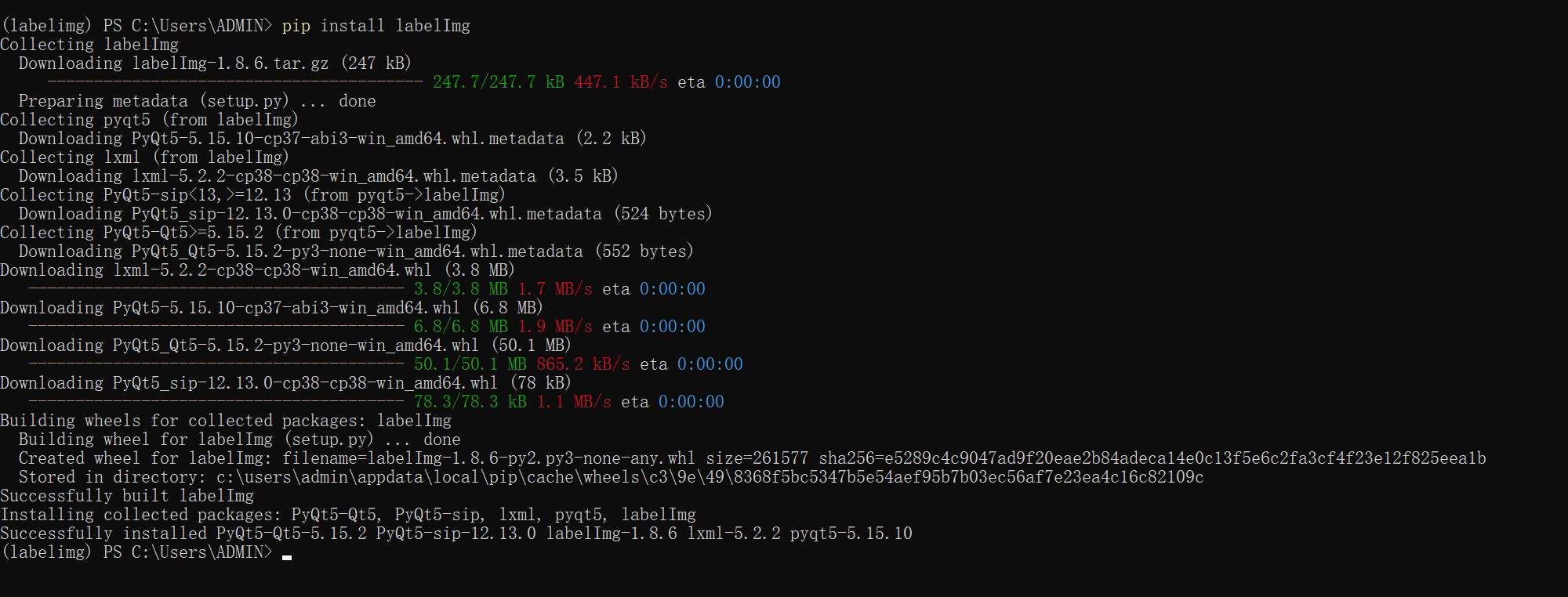
Once installed, we can typically launch LabelImg from our terminal with:

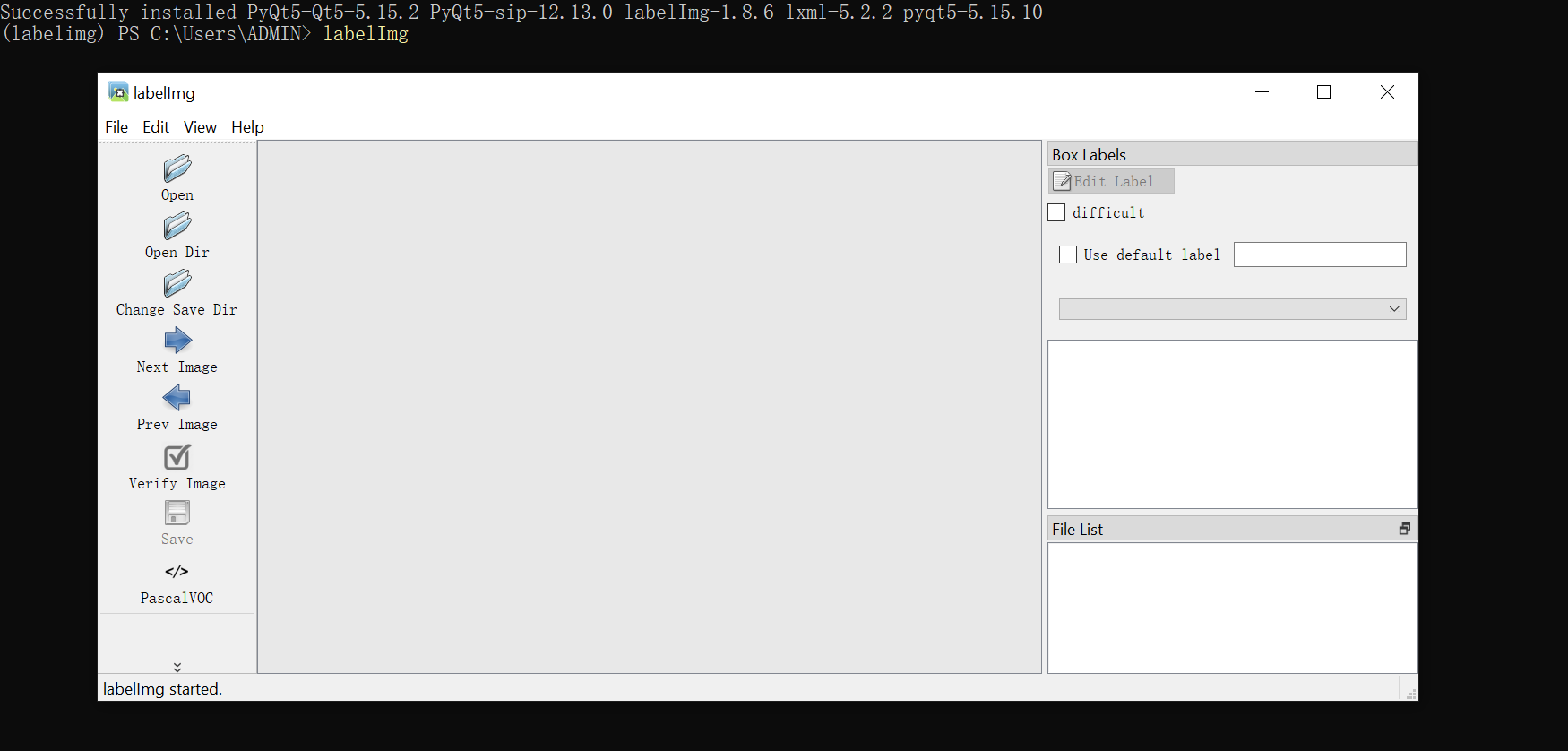
*labelImg*

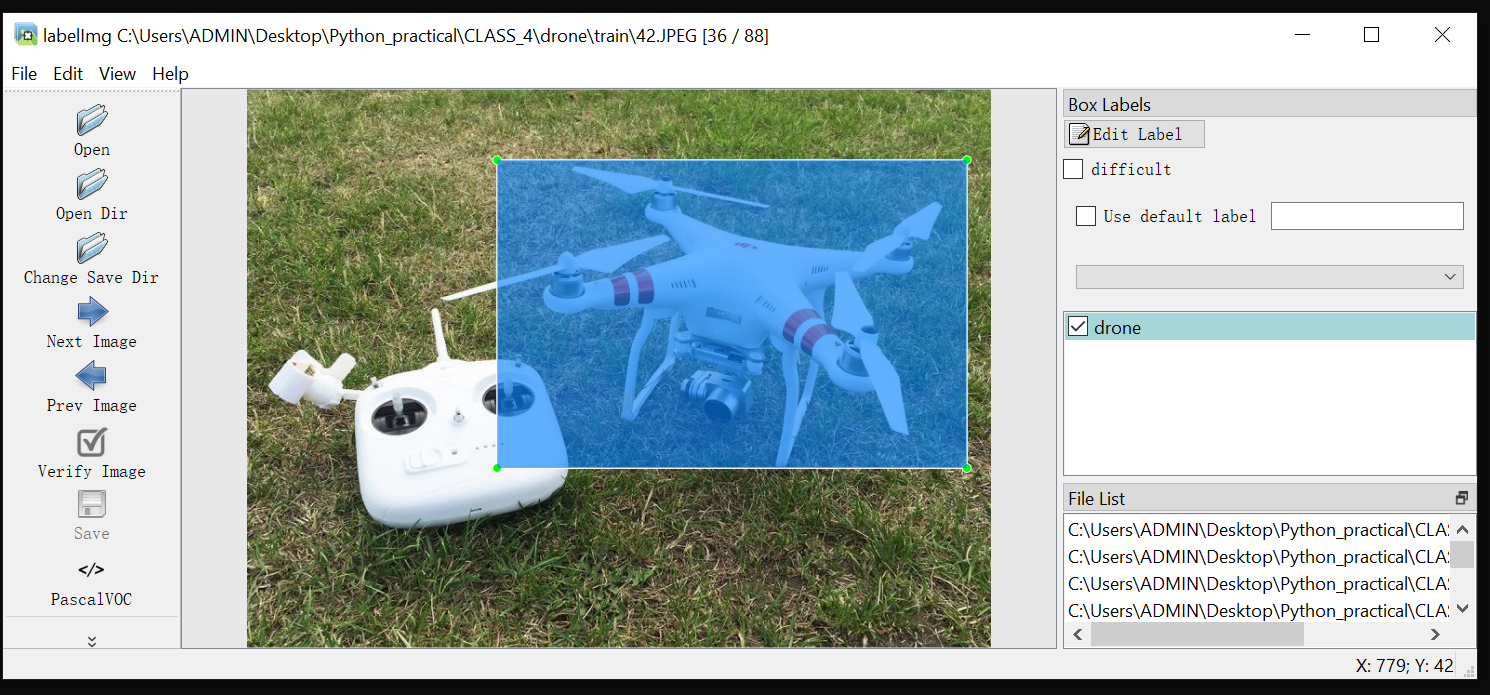
This should open the LabelImg graphical interface, ready for image annotation.







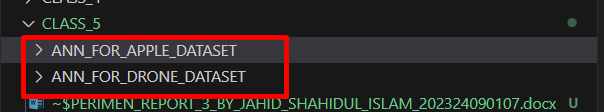




# Data Format Conversion and Script Automation Setup

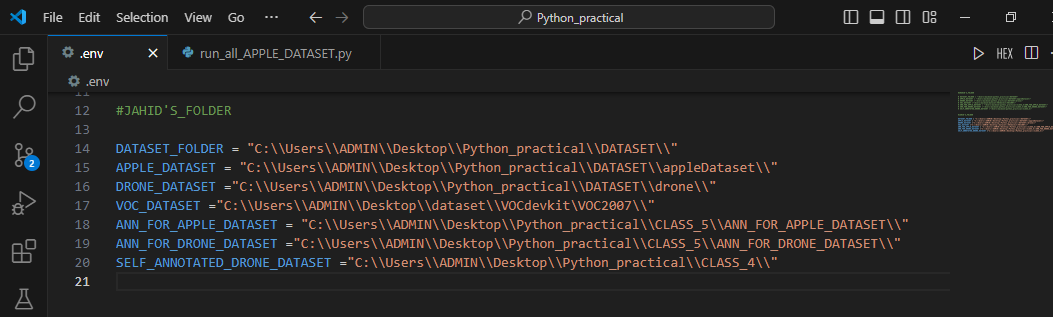
**Step 1: Establish File Structure**

Organized separate folders for each dataset (e.g., "apple\_dataset," "drone\_dataset") to maintain a clean project structure. Within each dataset folder, I created subfolders for each conversion script to ensure clear separation of code.



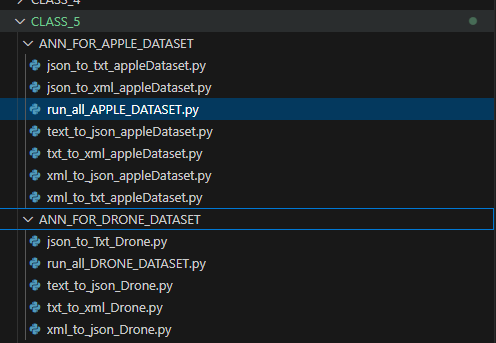
**Step 2: Configure Environment Variables**

Utilized the python-dotenv package to store dataset folder locations in a .env file. This avoids hardcoding paths within the automation script and allows for easy modification.



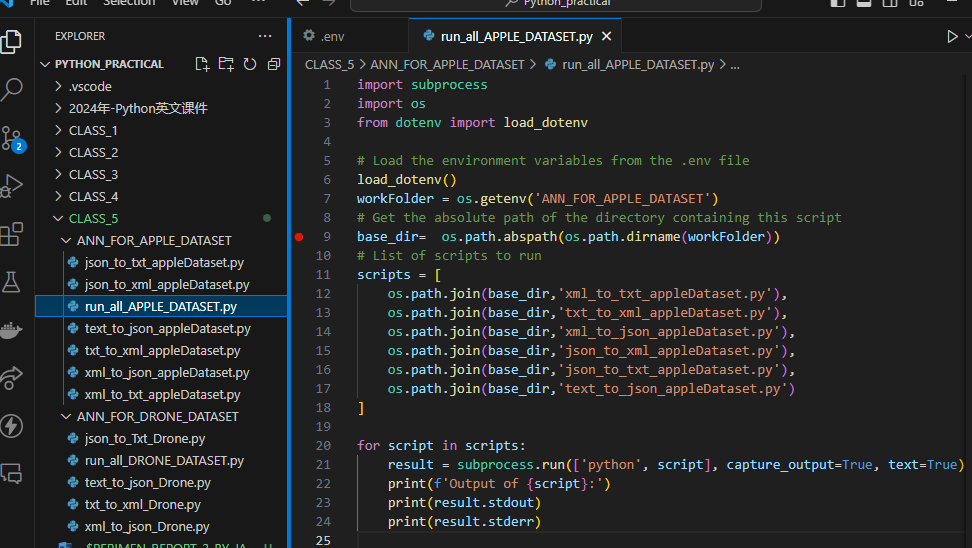
**Step 3: Develop Conversion Scripts**

Created individual Python scripts for each data format conversion (e.g., xml\_to\_txt.py, json\_to\_xml.py). These scripts handle the logic for reading, converting, and writing data in the desired formats.



**Step 4: Implement Automation Script**

Developed an automation script (run\_all.py) that leverages the subprocess module to execute all conversion scripts sequentially. The script iterates through a list of script paths, running each script using subprocess.run() and capturing the output for logging and error handling.



This setup streamlines the process of data format conversion, making it repeatable and manageable across multiple datasets.

# Change Xml to Txt File

**Step 1: Parse the XML File**

Used the *xml.etree.ElementTree* library to parse the XML annotation files. Extracted relevant information like image dimensions (width, height) and object bounding boxes (*xmin, ymin, xmax, ymax*).

**Step 2: Calculate YOLO Coordinates**

Converted the bounding box coordinates from the XML format (pixel values) to the YOLO format (normalized values between 0 and 1) using the following formulas:

*x\_center = (xmin + xmax) / (2 \* img\_width)*

*y\_center = (ymin + ymax) / (2 \* img\_height)*

*width = (xmax - xmin) / img\_width*

*height = (ymax - ymin) / img\_height*

**Step 3: Write to TXT File**

For each object in the XML file, I wrote the class label and the calculated YOLO coordinates to a corresponding TXT file. Each line in the TXT file represents one object in the format: *<class\_name> <x\_center> <y\_center> <width> <height>.*

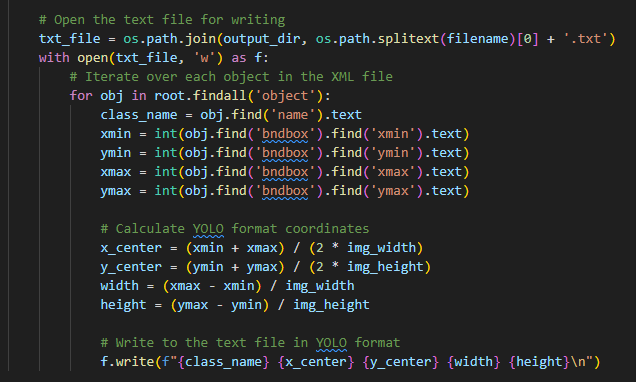
**Input XML Format:**

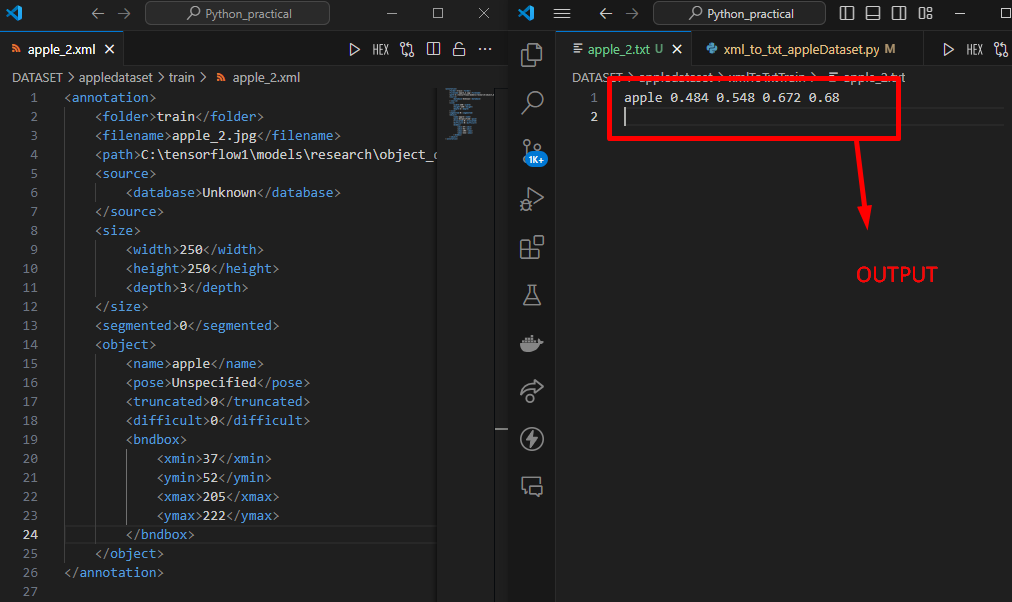
The input XML file contains annotations for each image with object class labels and bounding box coordinates in pixels.

**Output TXT Format:**

The converted TXT file contains the object class label and the normalized bounding box coordinates in the YOLO format, making it suitable for training YOLO object detection models. Each object is represented on a new line.

This process effectively converts XML annotation files to the specific TXT format required for training YOLO object detection models.





# Change Txt to Xml File

**Step 1: Read the TXT File**

Opened the TXT annotation file and read the content line by line. Each line represented a detected object with its class and YOLO coordinates.

**Step 2: Convert YOLO Coordinates to XML Format**

Transformed the YOLO coordinates (normalized center-x, center-y, width, height) back into pixel values for the XML format:

*xmin = int((x\_center - width / 2) \* img\_width)*

*ymin = int((y\_center - height / 2) \* img\_height)*

*xmax = int((x\_center + width / 2) \* img\_width)*

*ymax = int((y\_center + height / 2) \* img\_height)*

**Step 3: Create the XML Structure**

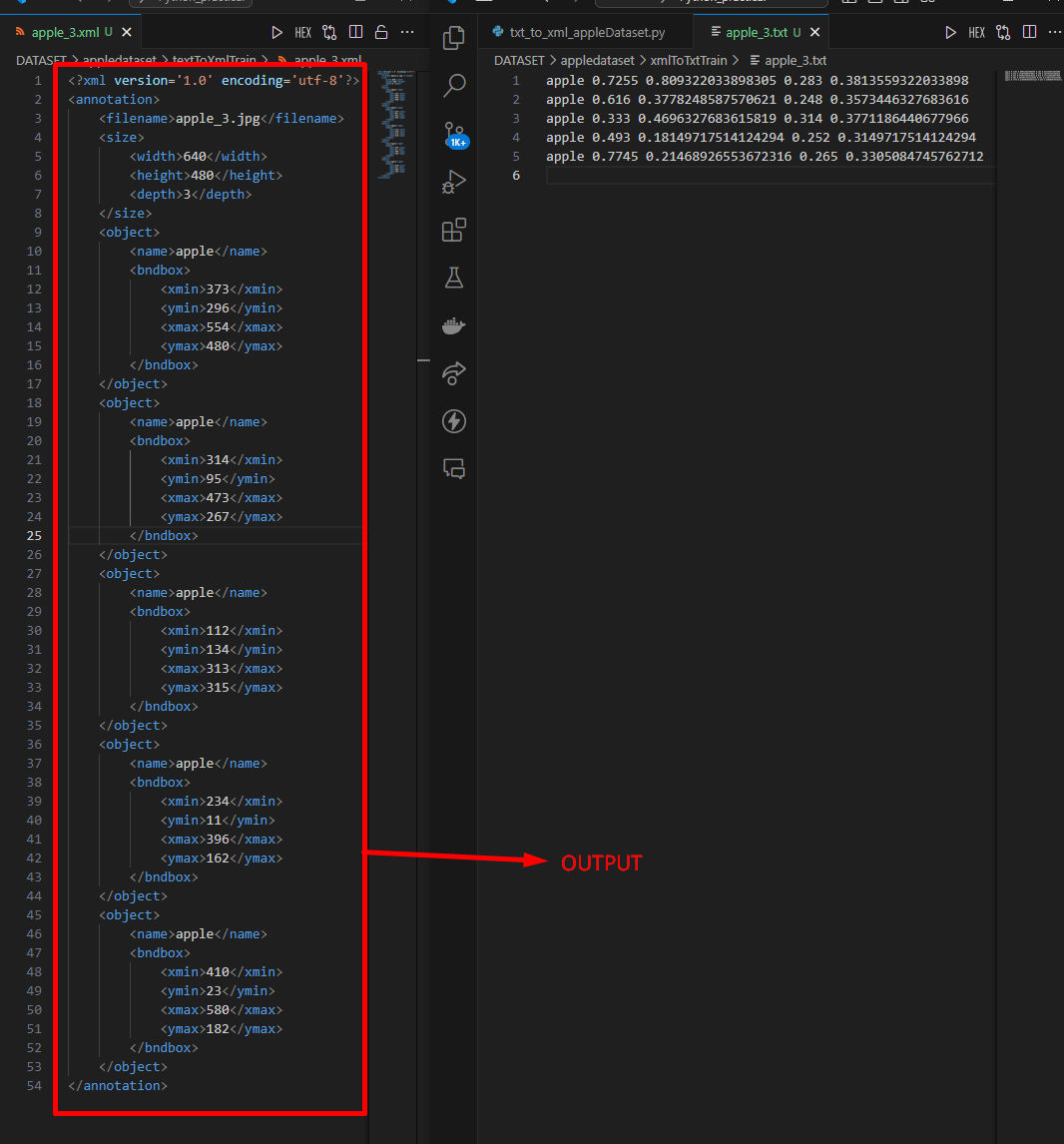
Used the *xml.etree.ElementTree* library to construct the XML tree. Created elements for annotation, filename, size (with width, height, depth), object (with name, *bndbox*), and *bndbox* (with *xmin, ymin, xmax, ymax*).

**Step 4: Write to XML File**

Wrote the generated XML tree to a file, ensuring proper formatting, encoding, and XML declaration.

This process effectively converted YOLO TXT annotations into the equivalent XML format, making the data usable in applications that require XML-based annotations.





# Change Xml to Json File

**Step 1: Extract Data from XML**

Parsed the XML annotation files using *xml.etree.ElementTree*, extracting essential information such as image filename, dimensions (width, height), and object bounding box coordinates (*xmin, ymin, xmax, ymax*).

**Step 2: Calculate Normalized Coordinates**

Calculated normalized (YOLO format) bounding box coordinates using the extracted pixel values and image dimensions:

*x\_center = (xmin + xmax) / (2 \* img\_width)*

*y\_center = (ymin + ymax) / (2 \* img\_height)*

*width = (xmax - xmin) / img\_width*

*height = (ymax - ymin) / img\_height*

**Step 3: Construct JSON Structure**

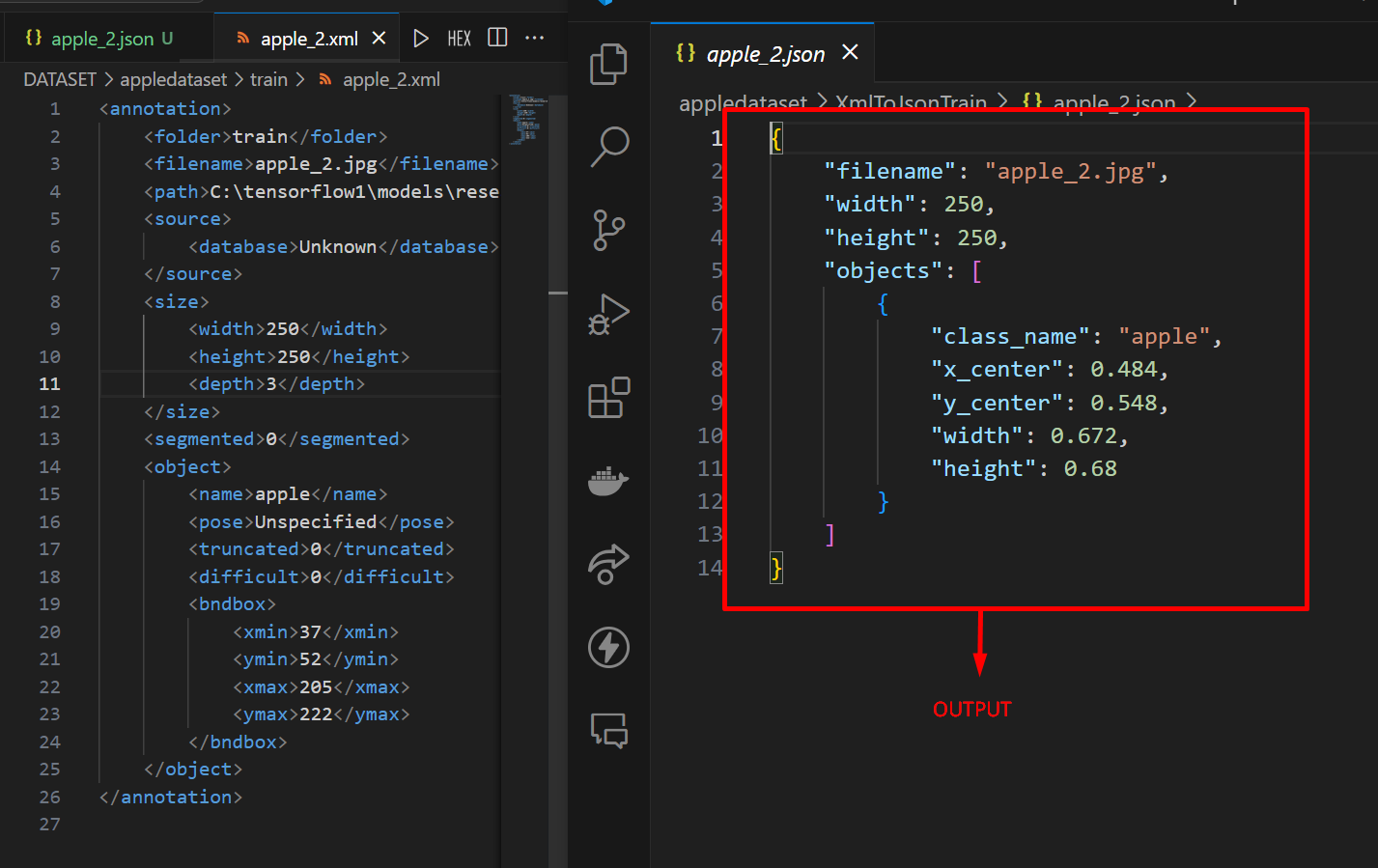
Organized the extracted data into a dictionary representing the JSON structure. This structure included the filename, image dimensions, and a list of objects. Each object dictionary contained the class name and its normalized bounding box coordinates.

**Step 4: Write to JSON File**

Utilized the *json.dump()* function to serialize the structured data into a JSON file, ensuring proper formatting and indentation.

This process successfully converted XML annotations into the JSON format, making the data suitable for applications or platforms that rely on JSON for representing object detection annotations.





# Change Json to Xml File

**Step 1: Parse JSON Data**

Read and deserialized the JSON annotation file using *json.load(),* extracting image filename, dimensions, and object data.

**Step 2: Calculate Pixel Coordinates**

Converted the normalized (YOLO format) bounding box coordinates from the JSON back to pixel values for the XML format:

*xmin = int((x\_center - width / 2) \* img\_width)*

*ymin = int((y\_center - height / 2) \* img\_height)*

*xmax = int((x\_center + width / 2) \* img\_width)*

*ymax = int((y\_center + height / 2) \* img\_height)*

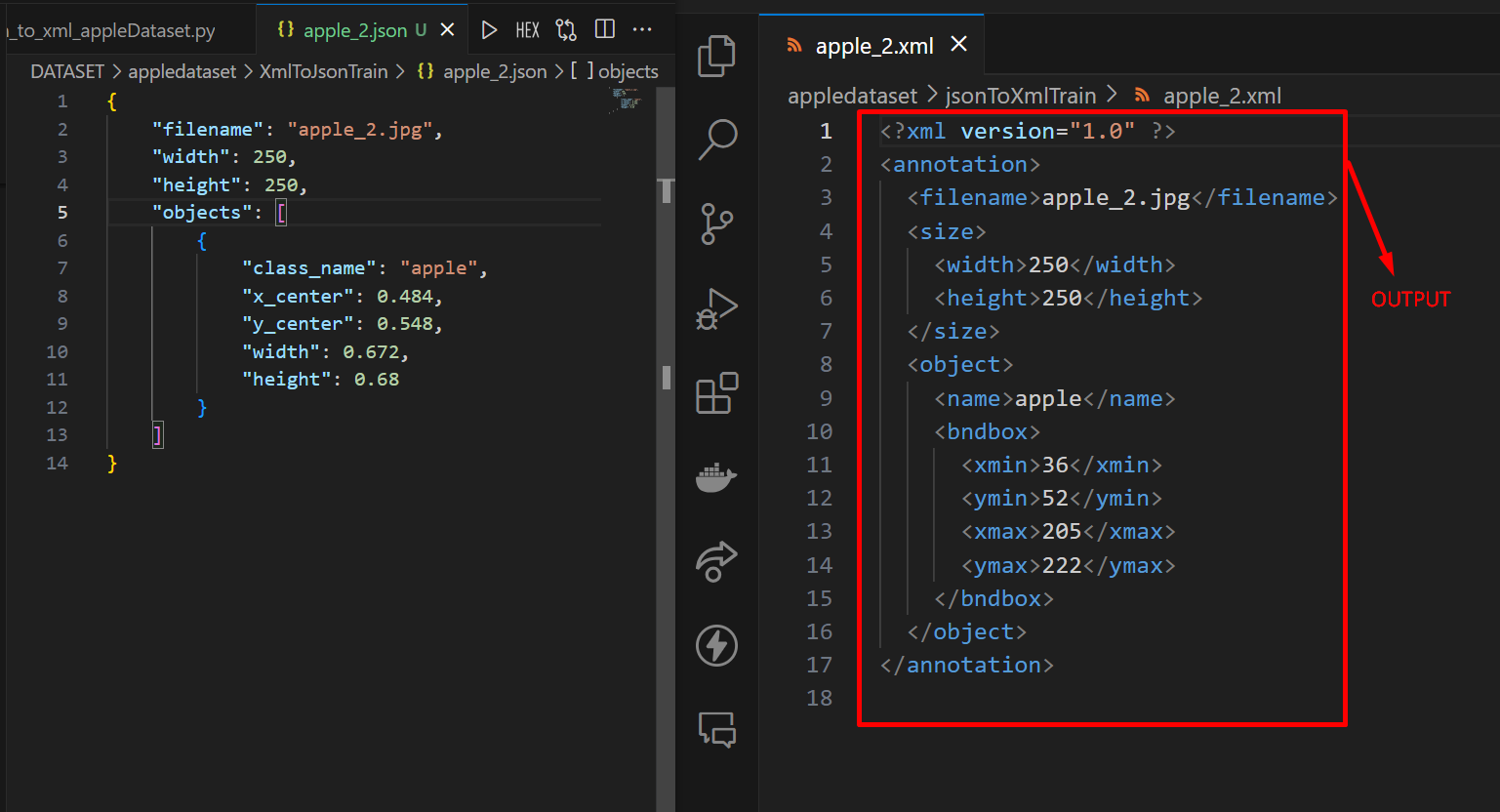
**Step 3: Construct XML Structure**

Used *xml.etree.ElementTree* to create the XML structure. Created elements for annotation, filename, size, object, and *bndbox*, populating them with data from the JSON.

**Step 4: Write Formatted XML**

Serialized the XML tree to a file, ensuring proper formatting using *minidom.parseString()* and *toprettyxml().* This made the resulting XML output more readable.

This process effectively transformed JSON-formatted annotations into their XML equivalents, making the data compatible with systems that utilize XML for object detection data.

# Change Json to Txt File

**Step 1: Extract Data from JSON**

Parsed the JSON annotation file using *json.load().* Extracted the filename and relevant object data, including the class name and normalized YOLO coordinates (*x\_center, y\_center, width, height*).

**Step 2: Determine Class ID**

Used a predefined list of class names (*class\_names*) to determine the corresponding numerical class ID for each object.

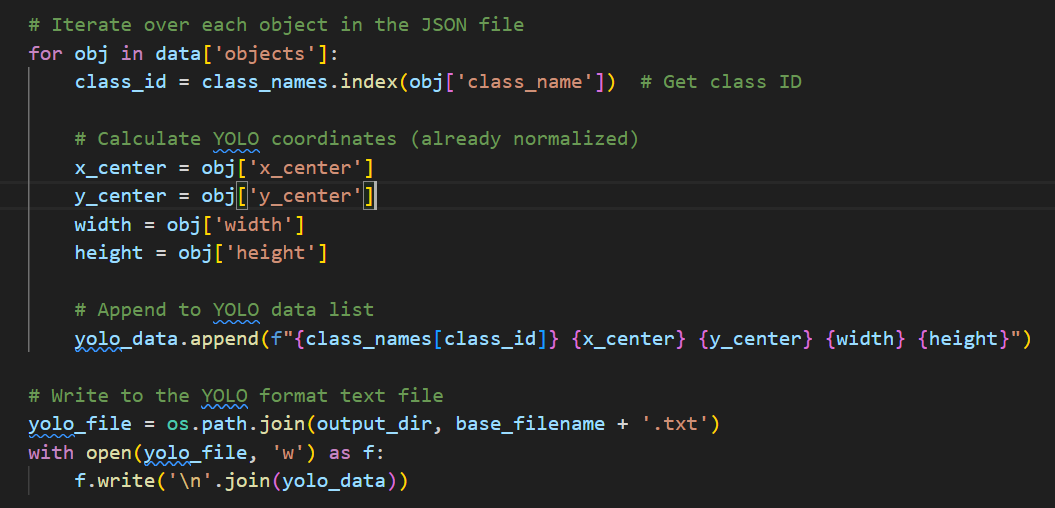
**Step 3: Format YOLO Annotation**

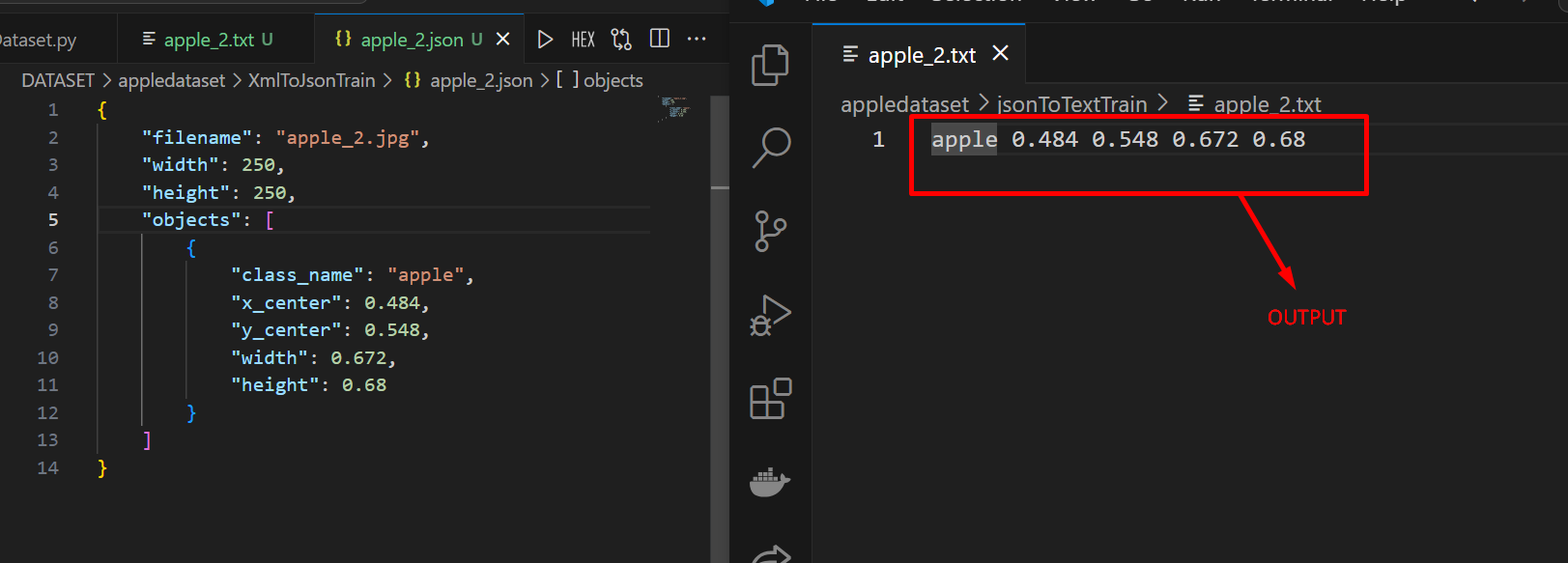
Constructed each line of the YOLO TXT annotation in the format: *class\_id x\_center y\_center width height.*

**Step 4: Write to TXT File**

Wrote the formatted YOLO annotations to a TXT file, with each line representing a single object detection.

This process transformed JSON-formatted annotations into YOLO TXT format, making it suitable for training or using YOLO-based object detection models.





# Experimental Analysis

In this experiment, we explored the practical aspects of working with various data annotation formats commonly used in computer vision tasks, particularly object detection. Recognizing the frequent need to convert between these formats, we focused on setting up an automated and reliable system for data format transformations involving XML, TXT (specifically for the YOLO format), and JSON.

The first step in our experiment involved setting up the necessary software tools. We installed the Labelimg software, a user-friendly graphical tool for annotating images with bounding boxes, using the pip package manager. This tool provides a visual interface for drawing and editing bounding boxes on images, which is a crucial step in preparing datasets for object detection models.

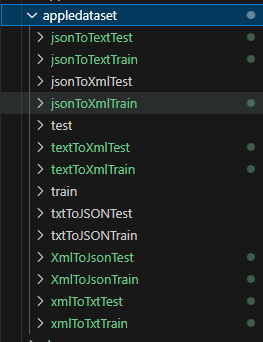
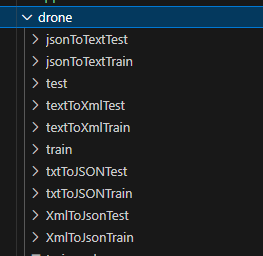
Next, we developed a set of Python scripts to handle the conversion between the different annotation formats. We created separate scripts for each conversion direction, ensuring that the specific requirements of each format were met. For instance, the scripts for converting to and from the YOLO TXT format carefully adhered to the structure expected by YOLO object detection models, facilitating seamless integration with model training pipelines.

To streamline the overall conversion process, we designed an automation script named "run\_all.py". This script leverages the subprocess module in Python to execute each individual conversion script in a sequential manner. This approach significantly simplifies the workflow, allowing us to convert an entire dataset between multiple formats with a single command.

Maintaining a well-structured and organized codebase was a priority throughout the experiment. We established a clear file organization scheme, dedicating separate folders for each dataset and conversion type. To enhance code portability and avoid hardcoding file paths, we employed the python-dotenv library. This allowed us to store dataset paths as environment variables, ensuring that our code remains flexible and adaptable to different file locations.

Our experimental results were overwhelmingly positive. The conversion scripts effectively transformed annotation data between all the target formats, meticulously preserving the data integrity and replicating the desired input and output structures. The careful attention to YOLO format compatibility proved particularly valuable, as the converted TXT files seamlessly integrated with YOLO model training procedures.

This experiment has successfully equipped us with a functional and automated system for converting between common data annotation formats. This system serves as a valuable tool for future computer vision projects, enabling us to work flexibly with various annotation formats and streamline our object detection workflows.

Report score: .

Instructor's signature: .