电 子 科 技 大 学

UNIVERSITY OF ELECTRONIC SCIENCE AND TECHNOLOGY OF CHINA

实验报告

EXPERIMENT REPORT

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.

4. Experimental contents & step

   1) Installing the Labelimg software for Windows
   2) Change xml to txt file
   3) Change txt to xml file
   4) Change xml to Json file
   5) Change Json to xml file
   6) Change Json to txt file

5. Experimental analysis

1. Installing the Labeling 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 labeling python=3.8

conda activate labeling

Step 2: Uninstall

previous version of labelImg, uninstall it:

pip uninstall labelImg

Step 3: Install LabelImg

Install the latest version using pip:

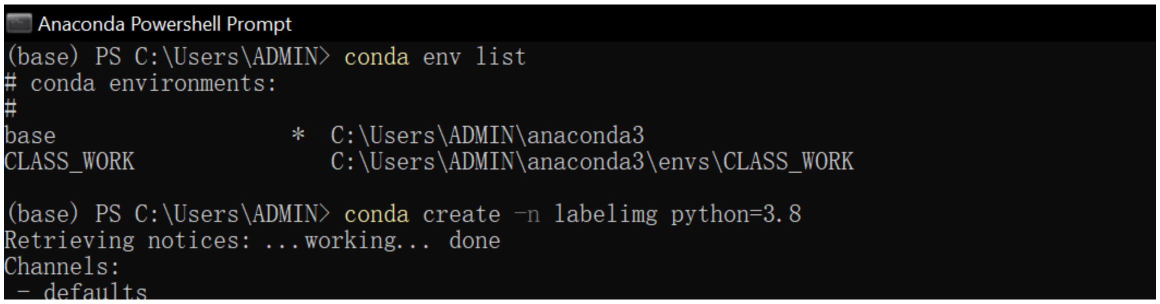
pip install labelImg

Step 4: Launch LabelImg

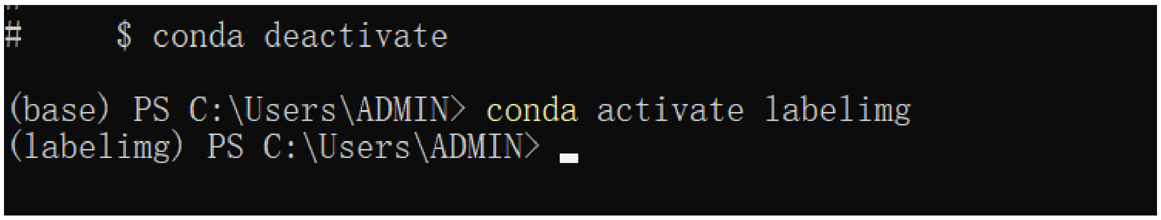
Once installed, we can typically launch LabelImg from our terminal with:

labelImg

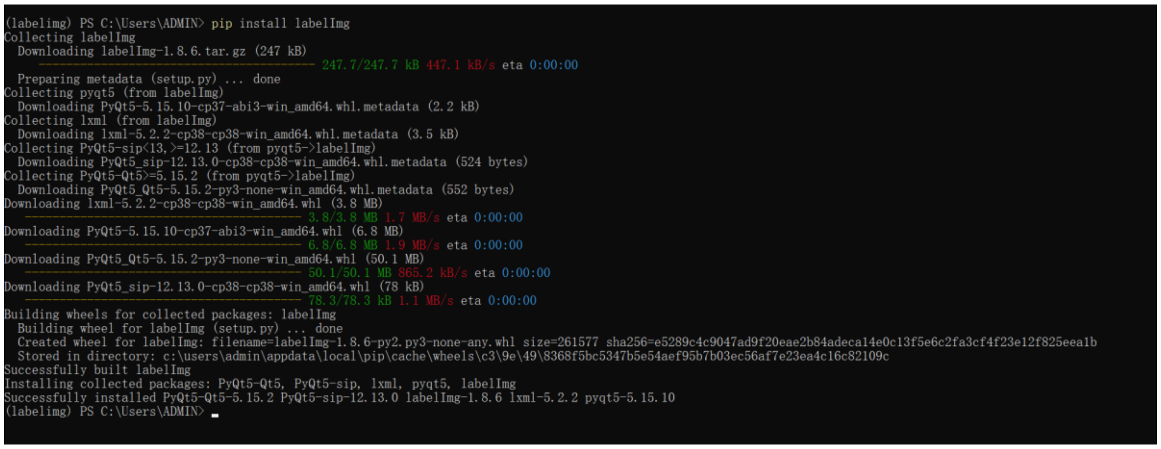
This should open the LabelImg graphical interface, ready for image annotation.



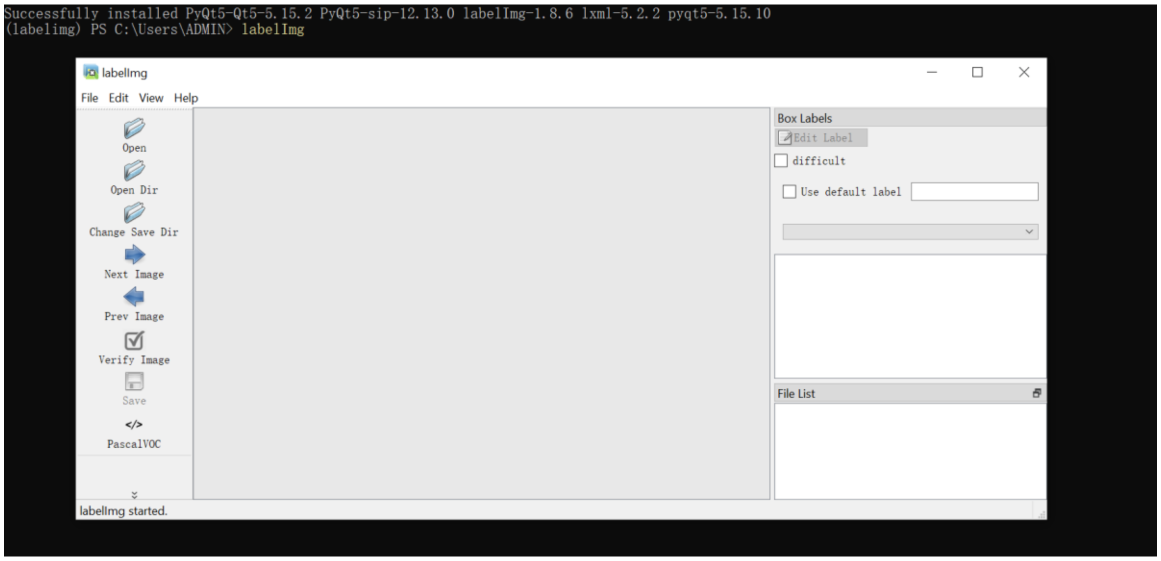
```
Anaconda Powershell Prompt
(base) PS C:\Users\ADMIN> conda env list
# conda environments:
#
base                  *  C:\Users\ADMIN\anaconda3
CLASS_WORK               C:\Users\ADMIN\anaconda3\envs\CLASS_WORK

(base) PS C:\Users\ADMIN> conda create -n labeling python=3.8
Retrieving notices: ...working... done
Channels:
 - defaults
```



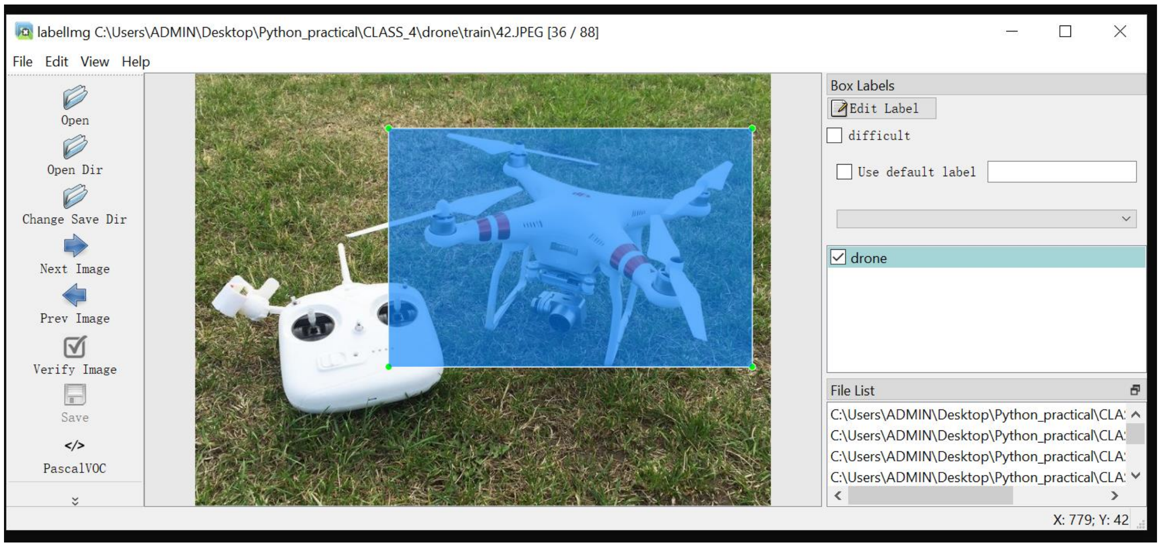
```
#     $ conda deactivate

(base) PS C:\Users\ADMIN> conda activate labeling
(labeling) PS C:\Users\ADMIN> _
```



```
(labelimg) PS C:\Users\ADMIN> pip install labelImg
Collecting labelImg
  Downloading labelImg-1.8.6.tar.gz (247 kB)
     ---------------------------------------- 247.7/247.7 kB 447.1 kB/s eta 0:00:00
  Preparing metadata (setup.py) ... done
Collecting pyqt5 (from labelImg)
  Downloading PyQt5-5.15.10-cp37-abi3-win_amd64.whl.metadata (2.2 kB)
Collecting lxml (from labelImg)
  Downloading lxml-5.2.2-cp38-cp38-win_amd64.whl.metadata (3.5 kB)
Collecting PyQt5-sip<13,>=12.13 (from pyqt5->labelImg)
  Downloading PyQt5_sip-12.13.0-cp38-cp38-win_amd64.whl.metadata (524 bytes)
Collecting PyQt5-Qt5>=5.15.2 (from pyqt5->labelImg)
  Downloading PyQt5_Qt5-5.15.2-py3-none-win_amd64.whl.metadata (552 bytes)
Downloading lxml-5.2.2-cp38-cp38-win_amd64.whl (3.8 MB)
   ---------------------------------------- 3.8/3.8 MB 1.7 MB/s eta 0:00:00
Downloading PyQt5-5.15.10-cp37-abi3-win_amd64.whl (6.8 MB)
   ---------------------------------------- 6.8/6.8 MB 1.9 MB/s eta 0:00:00
Downloading PyQt5_Qt5-5.15.2-py3-none-win_amd64.whl (50.1 MB)
   ---------------------------------------- 50.1/50.1 MB 865.2 kB/s eta 0:00:00
Downloading PyQt5_sip-12.13.0-cp38-cp38-win_amd64.whl (78 kB)
   ---------------------------------------- 78.3/78.3 kB 1.1 MB/s eta 0:00:00
Building wheels for collected packages: labelImg
  Building wheel for labelImg (setup.py) ... done
  Created wheel for labelImg: filename=labelImg-1.8.6-py2.py3-none-any.whl size=261577 sha256=e5289c4c9047ad9f20eae2b84adeca14e0c13f5e6c2fa3cf4f23e12f825eea1b
  Stored in directory: c:\users\admin\appdata\local\pip\cache\wheels\c3\9e\49\8368f5bc5347b5e54aef95b7b03ec56af7e23ea4c16c82109c
Successfully built labelImg
Installing collected packages: PyQt5-Qt5, PyQt5-sip, lxml, pyqt5, labelImg
Successfully installed PyQt5-Qt5-5.15.2 PyQt5-sip-12.13.0 labelImg-1.8.6 lxml-5.2.2 pyqt5-5.15.10
(labelimg) PS C:\Users\ADMIN> _
```



```
Successfully installed PyQt5-Qt5-5.15.2 PyQt5-sip-12.13.0 labelImg-1.8.6 lxml-5.2.2 pyqt5-5.15.10
(labelimg) PS C:\Users\ADMIN> labelImg
```

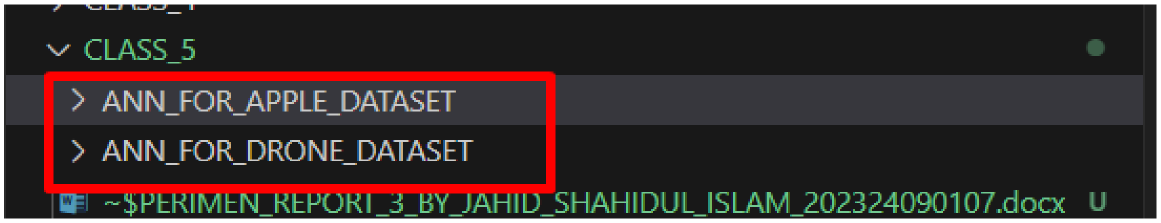


labelImg C:\Users\ADMIN\Desktop\Python_practical\CLASS_4\drone\train\42.JPEG [36 / 88]
File Edit View Help
Open
Open Dir
Change Save Dir
Next Image
Prev Image
Verify Image
Save
PascalVOC
Box Labels
Edit Label
difficult
Use default label
drone
File List
C:\Users\ADMIN\Desktop\Python_practical\CLA
C:\Users\ADMIN\Desktop\Python_practical\CLA
C:\Users\ADMIN\Desktop\Python_practical\CLA
C:\Users\ADMIN\Desktop\Python_practical\CLA
X: 779; Y: 42

2. Data Format Conversion and Script Automation Setup

Step 1: Establish File Structure

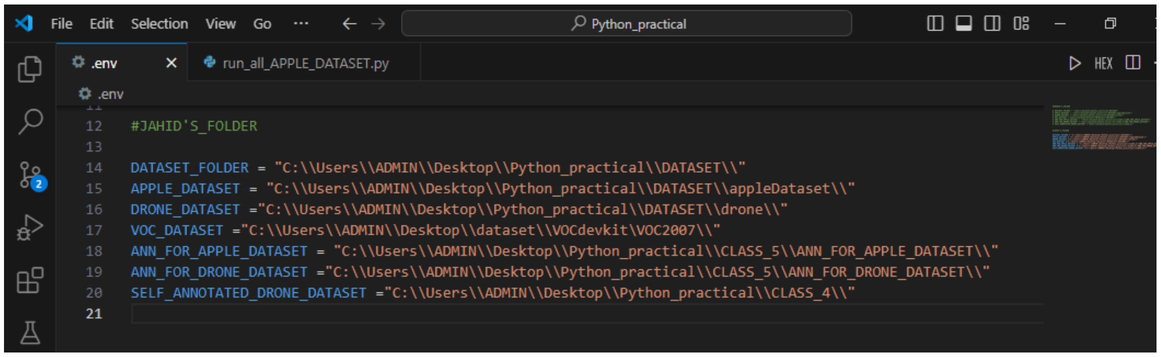
Organized separate folders for each dataset (e.g., "apple_dataset," "drone_dataset") to maintain a clean project structure. Within each dataset folder, I created subfolders for each conversion script to ensure clear separation of code.



```
CLASS_5
  > ANN_FOR_APPLE_DATASET
  > ANN_FOR_DRONE_DATASET
~$PERIMEN_REPORT_3_BY_JAHID_SHAHIDUL_ISLAM_202324090107.docx  U
```

Step 2: Configure Environment Variables

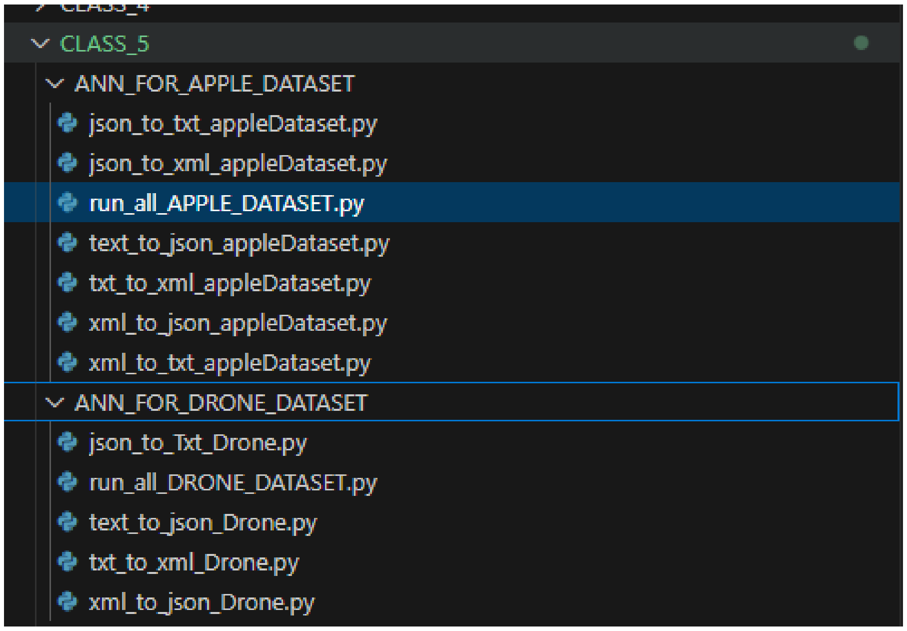
Utilized the python-dotenv package to store dataset folder locations in a .env file. This avoids hardcoding paths within the automation script and allows for easy modification.



```
#JAHID'S_FOLDER

DATASET_FOLDER = "C:\\Users\\ADMIN\\Desktop\\Python_practical\\DATASET\\"
APPLE_DATASET = "C:\\Users\\ADMIN\\Desktop\\Python_practical\\DATASET\\appleDataset\\"
DRONE_DATASET ="C:\\Users\\ADMIN\\Desktop\\Python_practical\\DATASET\\drone\\"
VOC_DATASET ="C:\\Users\\ADMIN\\Desktop\\dataset\\VOCdevkit\VOC2007\\"
ANN_FOR_APPLE_DATASET = "C:\\Users\\ADMIN\\Desktop\\Python_practical\\CLASS_5\\ANN_FOR_APPLE_DATASET\\"
ANN_FOR_DRONE_DATASET ="C:\\Users\\ADMIN\\Desktop\\Python_practical\\CLASS_5\\ANN_FOR_DRONE_DATASET\\"
SELF_ANNOTATED_DRONE_DATASET ="C:\\Users\\ADMIN\\Desktop\\Python_practical\\CLASS_4\\"
```

Step 3: Develop Conversion Scripts

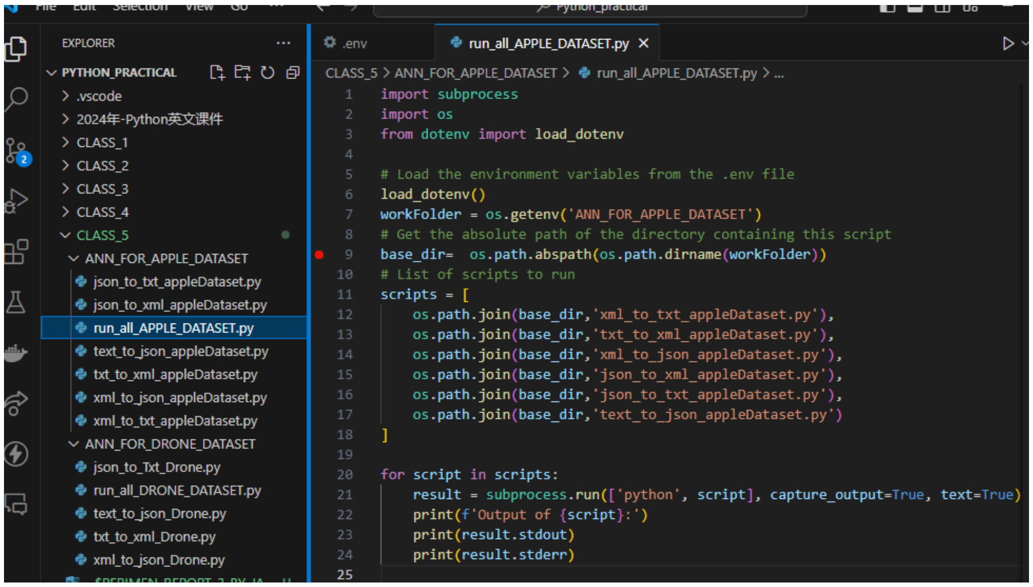
Created individual Python scripts for each data format conversion (e.g., xml_to_txt.py, json_to_xml.py). These scripts handle the logic for reading, converting, and writing data in the desired formats.



```
CLASS_5
  ANN_FOR_APPLE_DATASET
    json_to_txt_appleDataset.py
    json_to_xml_appleDataset.py
    run_all_APPLE_DATASET.py
    text_to_json_appleDataset.py
    txt_to_xml_appleDataset.py
    xml_to_json_appleDataset.py
    xml_to_txt_appleDataset.py
  ANN_FOR_DRONE_DATASET
    json_to_Txt_Drone.py
    run_all_DRONE_DATASET.py
    text_to_json_Drone.py
    txt_to_xml_Drone.py
    xml_to_json_Drone.py
```

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.



CLASS_5 > ANN_FOR_APPLE_DATASET > run_all_APPLE_DATASET.py > ...

```python
import subprocess
import os
from dotenv import load_dotenv

# Load the environment variables from the .env file
load_dotenv()
workFolder = os.getenv('ANN_FOR_APPLE_DATASET')
# Get the absolute path of the directory containing this script
base_dir=  os.path.abspath(os.path.dirname(workFolder))
# List of scripts to run
scripts = [
    os.path.join(base_dir,'xml_to_txt_appleDataset.py'),
    os.path.join(base_dir,'txt_to_xml_appleDataset.py'),
    os.path.join(base_dir,'xml_to_json_appleDataset.py'),
    os.path.join(base_dir,'json_to_xml_appleDataset.py'),
    os.path.join(base_dir,'json_to_txt_appleDataset.py'),
    os.path.join(base_dir,'text_to_json_appleDataset.py')
]

for script in scripts:
    result = subprocess.run(['python', script], capture_output=True, text=True)
    print(f'Output of {script}:')
    print(result.stdout)
    print(result.stderr)
```

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

3. 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>.*

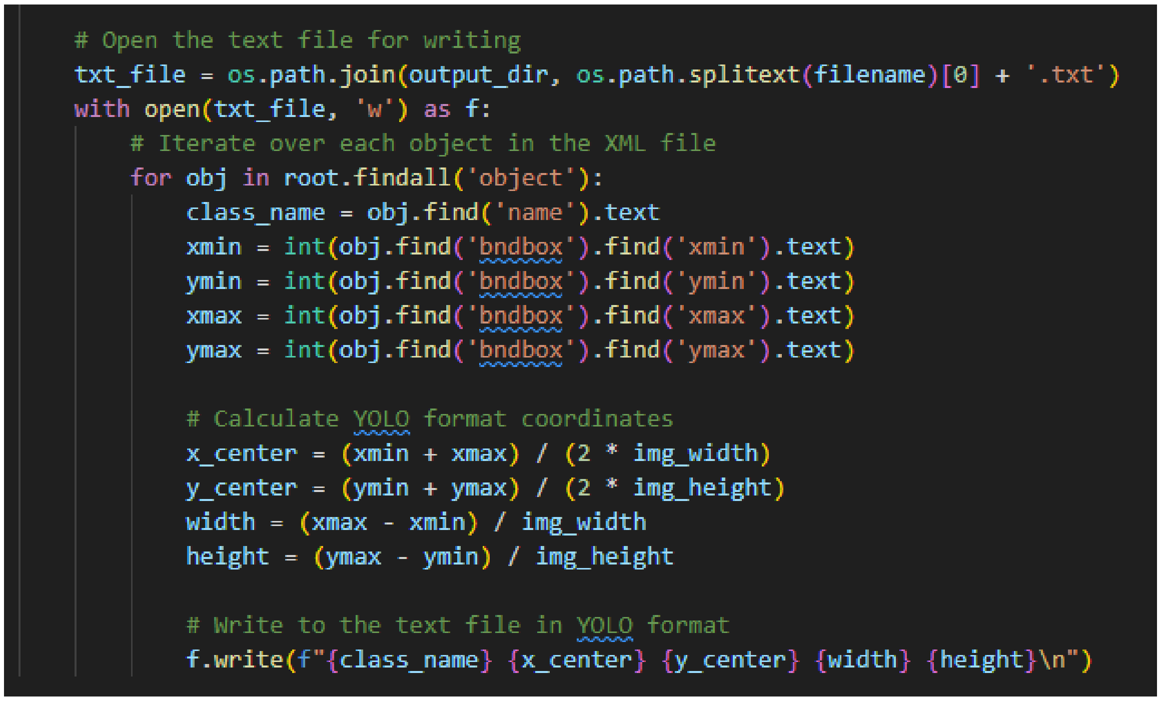
Input XML Format:

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

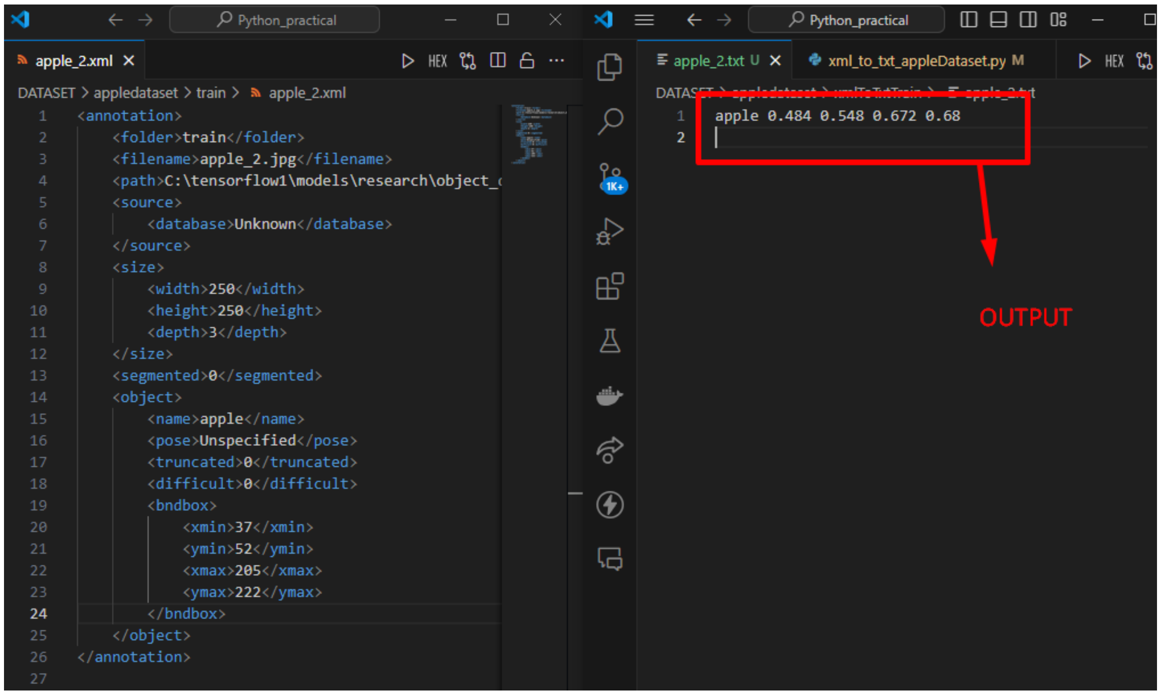
Output TXT Format:

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.



```python
    # Open the text file for writing
    txt_file = os.path.join(output_dir, os.path.splitext(filename)[0] + '.txt')
    with open(txt_file, 'w') as f:
        # Iterate over each object in the XML file
        for obj in root.findall('object'):
            class_name = obj.find('name').text
            xmin = int(obj.find('bndbox').find('xmin').text)
            ymin = int(obj.find('bndbox').find('ymin').text)
            xmax = int(obj.find('bndbox').find('xmax').text)
            ymax = int(obj.find('bndbox').find('ymax').text)

            # Calculate YOLO format coordinates
            x_center = (xmin + xmax) / (2 * img_width)
            y_center = (ymin + ymax) / (2 * img_height)
            width = (xmax - xmin) / img_width
            height = (ymax - ymin) / img_height

            # Write to the text file in YOLO format
            f.write(f"{class_name} {x_center} {y_center} {width} {height}\n")
```


Python_practical
apple_2.xml
DATASET > appledataset > train > apple_2.xml
apple_2.txt U
xml_to_txt_appleDataset.py M
OUTPUT

```xml
<annotation>
    <folder>train</folder>
    <filename>apple_2.jpg</filename>
    <path>C:\tensorflow1\models\research\object_
    <source>
        <database>Unknown</database>
    </source>
    <size>
        <width>250</width>
        <height>250</height>
        <depth>3</depth>
    </size>
    <segmented>0</segmented>
    <object>
        <name>apple</name>
        <pose>Unspecified</pose>
        <truncated>0</truncated>
        <difficult>0</difficult>
        <bndbox>
            <xmin>37</xmin>
            <ymin>52</ymin>
            <xmax>205</xmax>
            <ymax>222</ymax>
        </bndbox>
    </object>
</annotation>
```

```
apple 0.484 0.548 0.672 0.68
```

4. 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)*

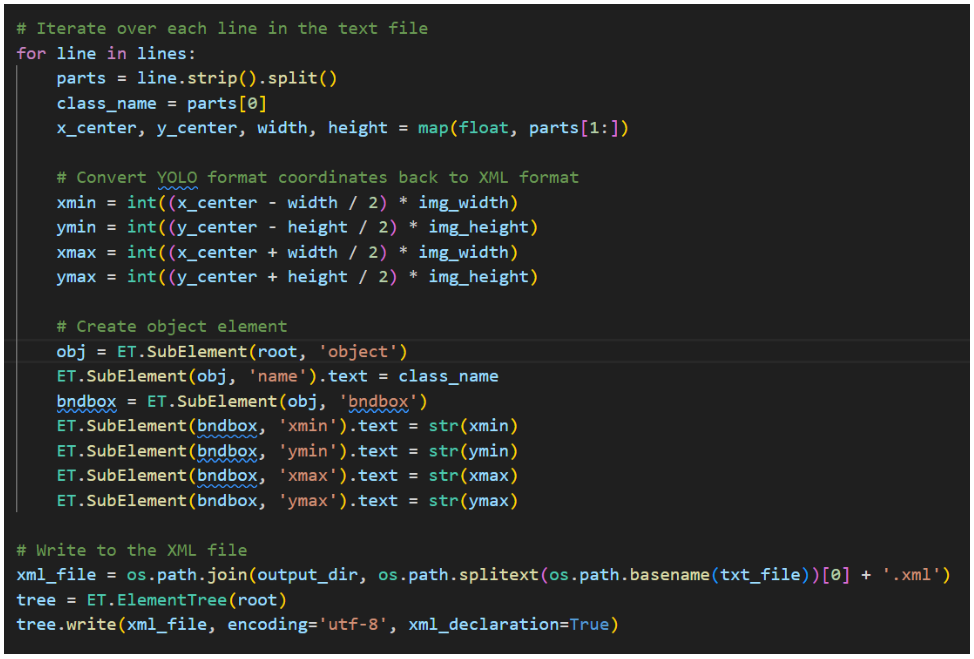
Step 3: Create the XML Structure

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*).

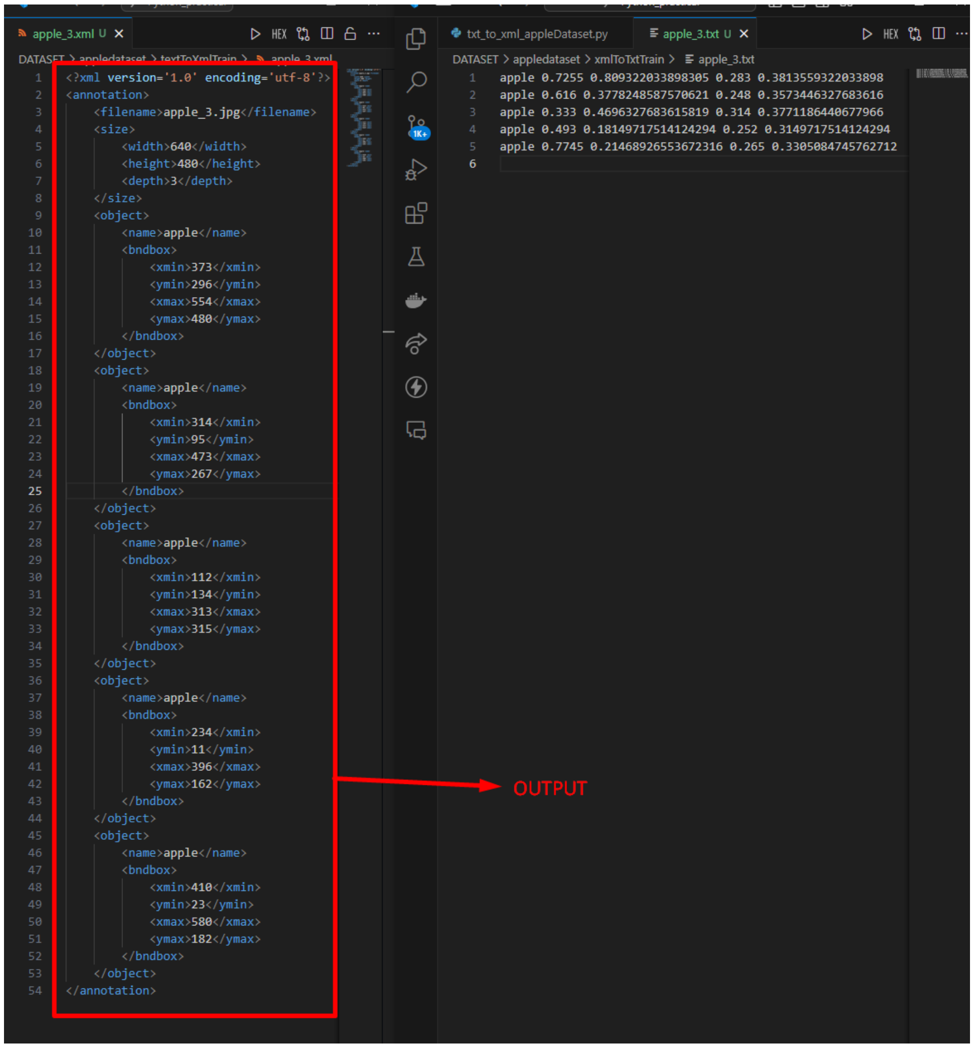
Step 4: Write to XML File

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.



```python
# Iterate over each line in the text file
for line in lines:
    parts = line.strip().split()
    class_name = parts[0]
    x_center, y_center, width, height = map(float, parts[1:])

    # Convert YOLO format coordinates back to 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)

    # Create object element
    obj = ET.SubElement(root, 'object')
    ET.SubElement(obj, 'name').text = class_name
    bndbox = ET.SubElement(obj, 'bndbox')
    ET.SubElement(bndbox, 'xmin').text = str(xmin)
    ET.SubElement(bndbox, 'ymin').text = str(ymin)
    ET.SubElement(bndbox, 'xmax').text = str(xmax)
    ET.SubElement(bndbox, 'ymax').text = str(ymax)

# Write to the XML file
xml_file = os.path.join(output_dir, os.path.splitext(os.path.basename(txt_file))[0] + '.xml')
tree = ET.ElementTree(root)
tree.write(xml_file, encoding='utf-8', xml_declaration=True)
```


apple_3.xml U
DATASET > appledataset > textToXmlTrain > apple_3.xml
txt_to_xml_appleDataset.py
apple_3.txt U
DATASET > appledataset > xmlToTxtTrain > apple_3.txt
OUTPUT

```xml
<?xml version='1.0' encoding='utf-8'?>
<annotation>
    <filename>apple_3.jpg</filename>
    <size>
        <width>640</width>
        <height>480</height>
        <depth>3</depth>
    </size>
    <object>
        <name>apple</name>
        <bndbox>
            <xmin>373</xmin>
            <ymin>296</ymin>
            <xmax>554</xmax>
            <ymax>480</ymax>
        </bndbox>
    </object>
    <object>
        <name>apple</name>
        <bndbox>
            <xmin>314</xmin>
            <ymin>95</ymin>
            <xmax>473</xmax>
            <ymax>267</ymax>
        </bndbox>
    </object>
    <object>
        <name>apple</name>
        <bndbox>
            <xmin>112</xmin>
            <ymin>134</ymin>
            <xmax>313</xmax>
            <ymax>315</ymax>
        </bndbox>
    </object>
    <object>
        <name>apple</name>
        <bndbox>
            <xmin>234</xmin>
            <ymin>11</ymin>
            <xmax>396</xmax>
            <ymax>162</ymax>
        </bndbox>
    </object>
    <object>
        <name>apple</name>
        <bndbox>
            <xmin>410</xmin>
            <ymin>23</ymin>
            <xmax>580</xmax>
            <ymax>182</ymax>
        </bndbox>
    </object>
</annotation>
```

```
apple 0.7255 0.809322033898305 0.283 0.3813559322033898
apple 0.616 0.3778248587570621 0.248 0.3573446327683616
apple 0.333 0.4696327683615819 0.314 0.3771186440677966
apple 0.493 0.18149717514124294 0.252 0.3149717514124294
apple 0.7745 0.21468926553672316 0.265 0.3305084745762712
```

5. 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

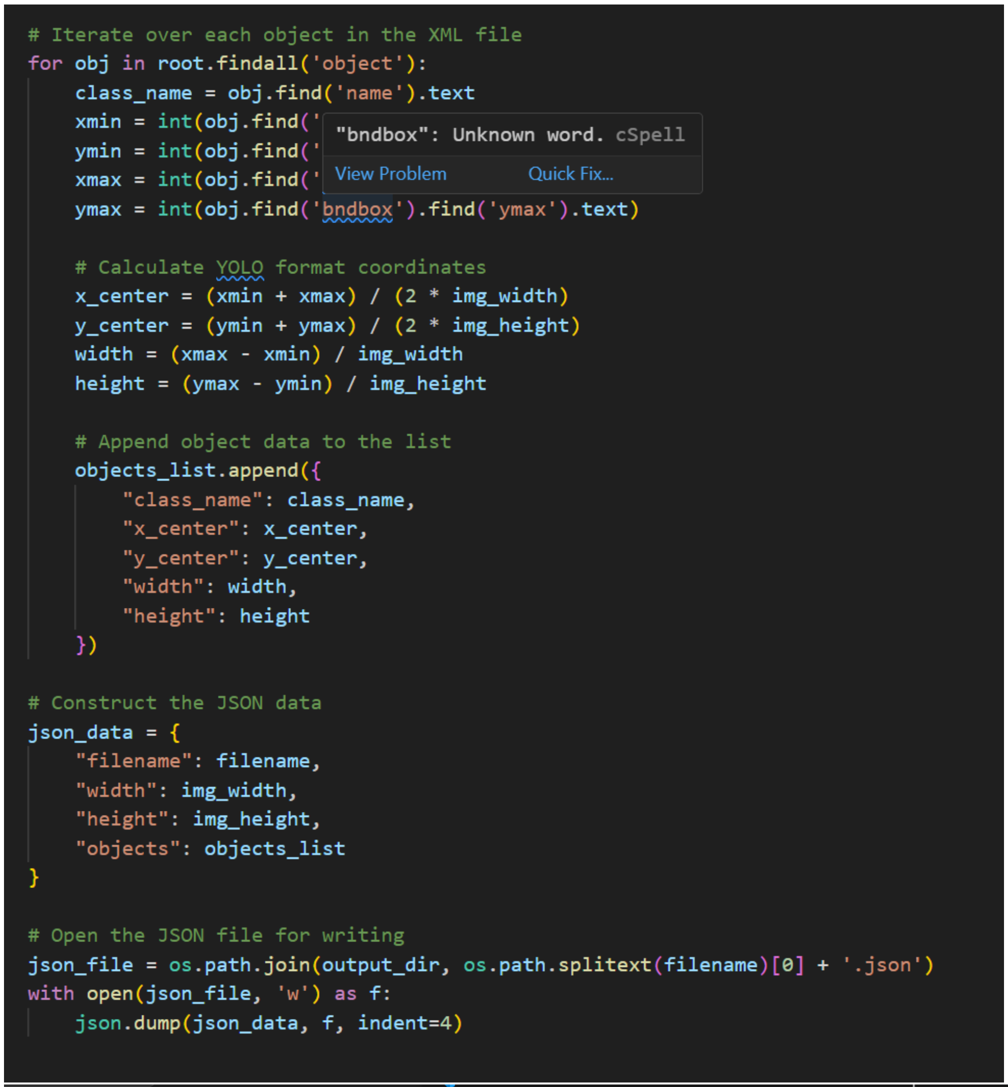
Step 3: Construct JSON Structure

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

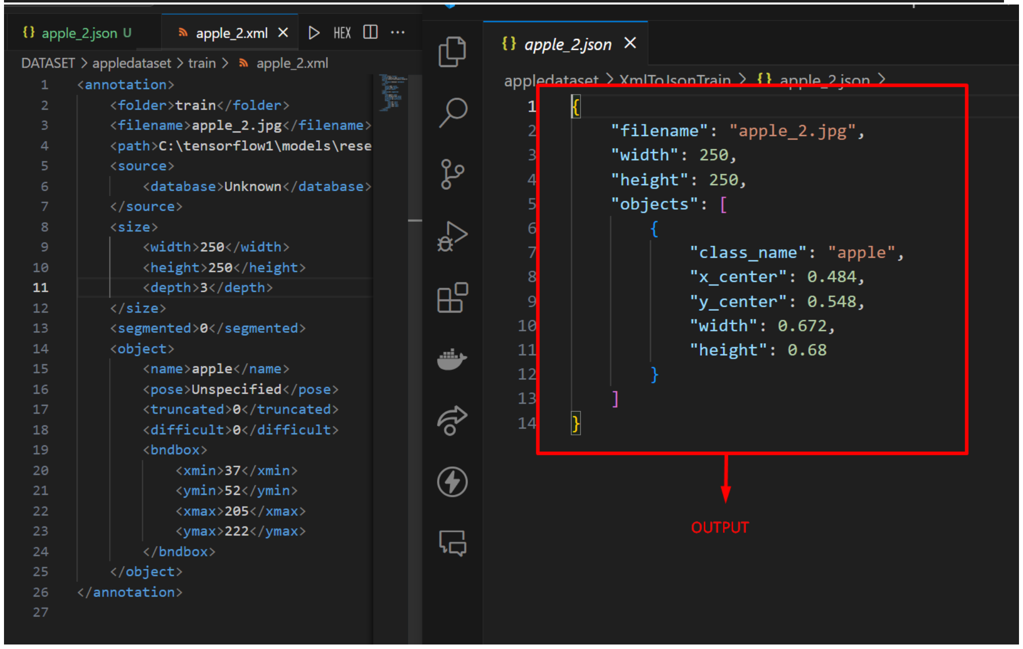
Utilized the *json.dump()* function to serialize the structured data into a JSON file, ensuring proper formatting and indentation.

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.



```python
# Iterate over each object in the XML file
for obj in root.findall('object'):
    class_name = obj.find('name').text
    xmin = int(obj.find('
    ymin = int(obj.find('
    xmax = int(obj.find('
    ymax = int(obj.find('bndbox').find('ymax').text)

    # Calculate YOLO format coordinates
    x_center = (xmin + xmax) / (2 * img_width)
    y_center = (ymin + ymax) / (2 * img_height)
    width = (xmax - xmin) / img_width
    height = (ymax - ymin) / img_height

    # Append object data to the list
    objects_list.append({
        "class_name": class_name,
        "x_center": x_center,
        "y_center": y_center,
        "width": width,
        "height": height
    })

# Construct the JSON data
json_data = {
    "filename": filename,
    "width": img_width,
    "height": img_height,
    "objects": objects_list
}

# Open the JSON file for writing
json_file = os.path.join(output_dir, os.path.splitext(filename)[0] + '.json')
with open(json_file, 'w') as f:
    json.dump(json_data, f, indent=4)
```

"bndbox": Unknown word. cSpell
View Problem
Quick Fix...



apple_2.json U
apple_2.xml
HEX
DATASET > appledataset > train > apple_2.xml
apple_2.json
appledataset > XmlToJsonTrain > apple_2.json >
OUTPUT

```xml
<annotation>
    <folder>train</folder>
    <filename>apple_2.jpg</filename>
    <path>C:\tensorflow1\models\rese
    <source>
        <database>Unknown</database>
    </source>
    <size>
        <width>250</width>
        <height>250</height>
        <depth>3</depth>
    </size>
    <segmented>0</segmented>
    <object>
        <name>apple</name>
        <pose>Unspecified</pose>
        <truncated>0</truncated>
        <difficult>0</difficult>
        <bndbox>
            <xmin>37</xmin>
            <ymin>52</ymin>
            <xmax>205</xmax>
            <ymax>222</ymax>
        </bndbox>
    </object>
</annotation>
```

```json
{
    "filename": "apple_2.jpg",
    "width": 250,
    "height": 250,
    "objects": [
        {
            "class_name": "apple",
            "x_center": 0.484,
            "y_center": 0.548,
            "width": 0.672,
            "height": 0.68
        }
    ]
}
```

6. 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)*

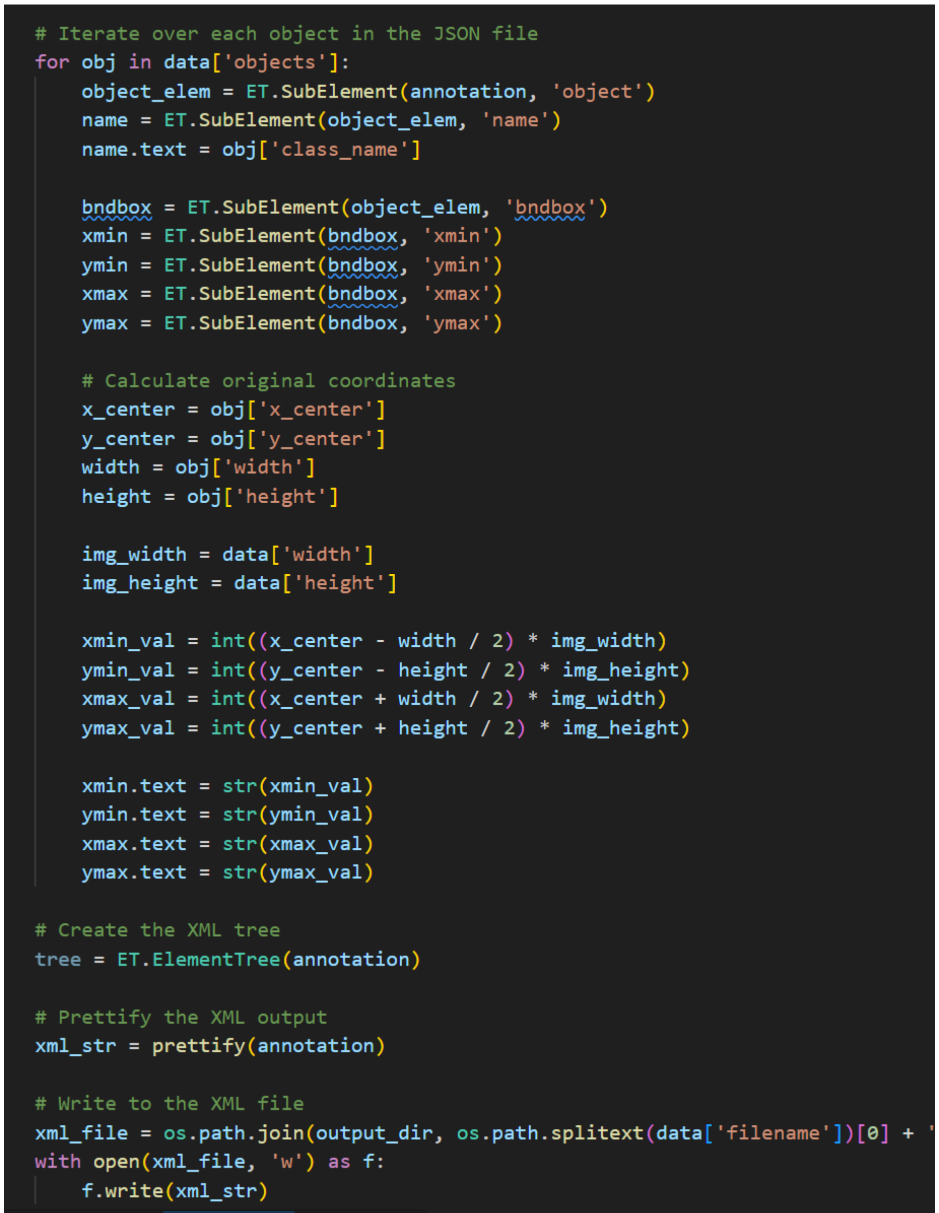
Step 3: Construct XML Structure

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.

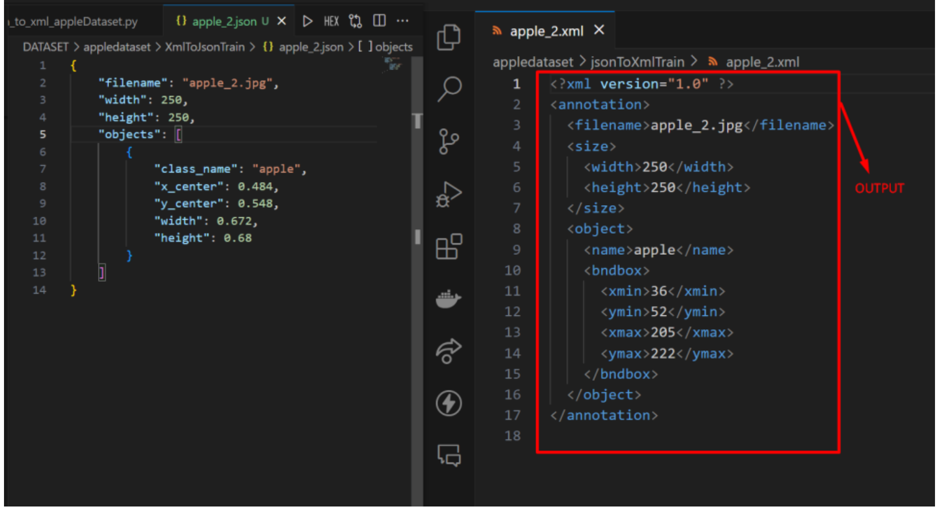
Step 4: Write Formatted XML

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.



```python
# Iterate over each object in the JSON file
for obj in data['objects']:
    object_elem = ET.SubElement(annotation, 'object')
    name = ET.SubElement(object_elem, 'name')
    name.text = obj['class_name']

    bndbox = ET.SubElement(object_elem, 'bndbox')
    xmin = ET.SubElement(bndbox, 'xmin')
    ymin = ET.SubElement(bndbox, 'ymin')
    xmax = ET.SubElement(bndbox, 'xmax')
    ymax = ET.SubElement(bndbox, 'ymax')

    # Calculate original coordinates
    x_center = obj['x_center']
    y_center = obj['y_center']
    width = obj['width']
    height = obj['height']

    img_width = data['width']
    img_height = data['height']

    xmin_val = int((x_center - width / 2) * img_width)
    ymin_val = int((y_center - height / 2) * img_height)
    xmax_val = int((x_center + width / 2) * img_width)
    ymax_val = int((y_center + height / 2) * img_height)

    xmin.text = str(xmin_val)
    ymin.text = str(ymin_val)
    xmax.text = str(xmax_val)
    ymax.text = str(ymax_val)

# Create the XML tree
tree = ET.ElementTree(annotation)

# Prettify the XML output
xml_str = prettify(annotation)

# Write to the XML file
xml_file = os.path.join(output_dir, os.path.splitext(data['filename'])[0] + '
with open(xml_file, 'w') as f:
    f.write(xml_str)
```



_to_xml_appleDataset.py
apple_2.json U
DATASET > appledataset > XmlToJsonTrain > apple_2.json > [ ] objects

```json
{
    "filename": "apple_2.jpg",
    "width": 250,
    "height": 250,
    "objects": [
        {
            "class_name": "apple",
            "x_center": 0.484,
            "y_center": 0.548,
            "width": 0.672,
            "height": 0.68
        }
    ]
}
```

apple_2.xml
appledataset > jsonToXmlTrain > apple_2.xml

```xml
<?xml version="1.0" ?>
<annotation>
  <filename>apple_2.jpg</filename>
  <size>
    <width>250</width>
    <height>250</height>
  </size>
  <object>
    <name>apple</name>
    <bndbox>
      <xmin>36</xmin>
      <ymin>52</ymin>
      <xmax>205</xmax>
      <ymax>222</ymax>
    </bndbox>
  </object>
</annotation>
```

7. 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.

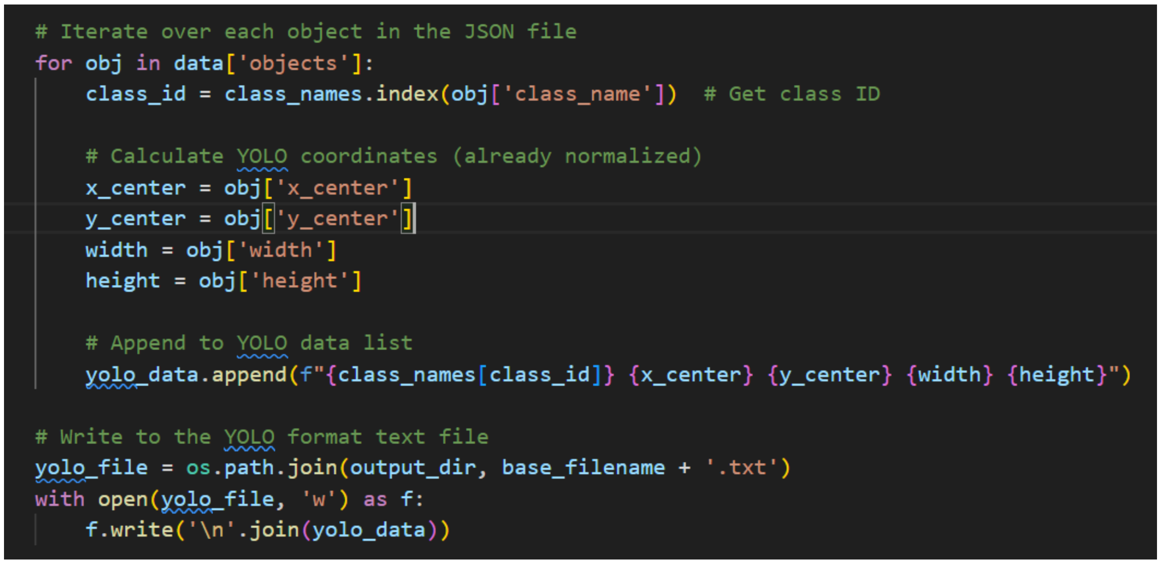
Step 3: Format YOLO Annotation

Constructed each line of the YOLO TXT annotation in the format: *class_id x_center y_center width height.*

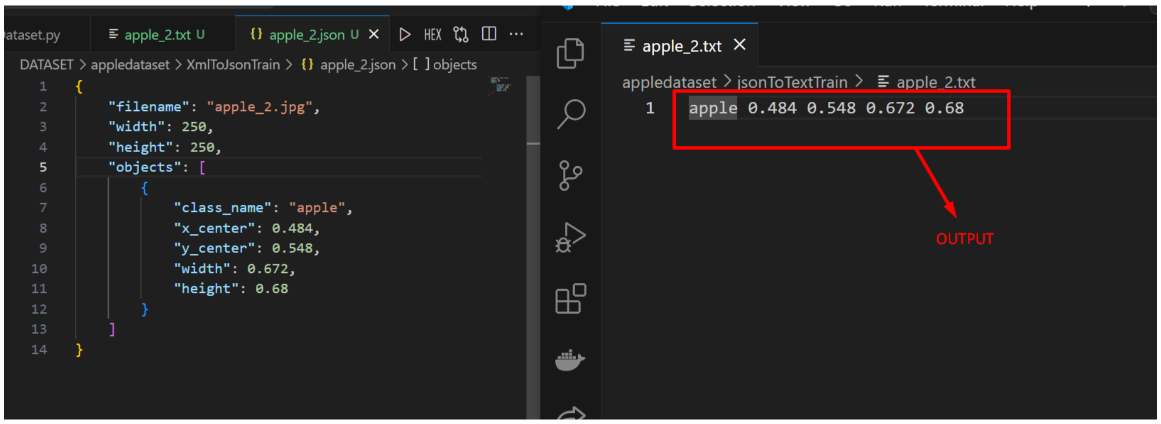
Step 4: Write to TXT File

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.



```
# Iterate over each object in the JSON file
for obj in data['objects']:
    class_id = class_names.index(obj['class_name'])  # Get class ID

    # Calculate YOLO coordinates (already normalized)
    x_center = obj['x_center']
    y_center = obj['y_center']
    width = obj['width']
    height = obj['height']

    # Append to YOLO data list
    yolo_data.append(f"{class_names[class_id]} {x_center} {y_center} {width} {height}")

# Write to the YOLO format text file
yolo_file = os.path.join(output_dir, base_filename + '.txt')
with open(yolo_file, 'w') as f:
    f.write('\n'.join(yolo_data))
```


ataset.py
apple_2.txt U
apple_2.json U
DATASET > appledataset > XmlToJsonTrain > apple_2.json > [ ] objects
apple_2.txt
appledataset > jsonToTextTrain > apple_2.txt
OUTPUT

```
{
    "filename": "apple_2.jpg",
    "width": 250,
    "height": 250,
    "objects": [
        {
            "class_name": "apple",
            "x_center": 0.484,
            "y_center": 0.548,
            "width": 0.672,
            "height": 0.68
        }
    ]
}
```

```
apple 0.484 0.548 0.672 0.68
```

8. 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.

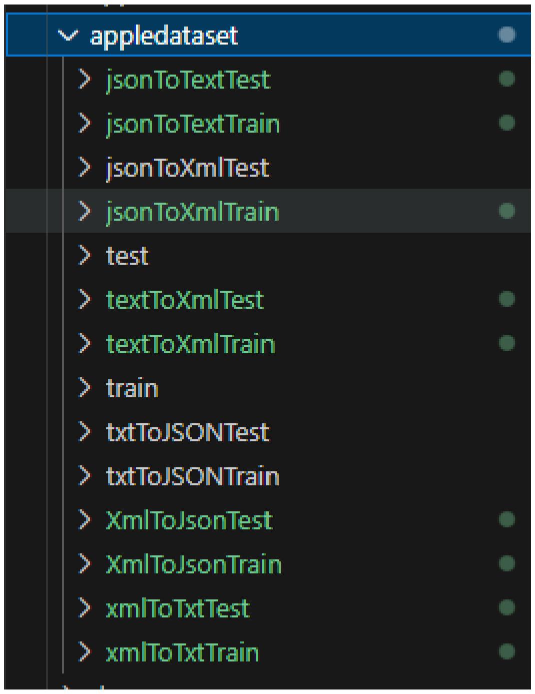
Next, we developed a set of Python scripts to handle the conversion between the different annotation formats. We created separate scripts for each conversion direction, ensuring that the specific requirements of each format were met. For instance, the scripts for converting to and from the YOLO TXT format 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.

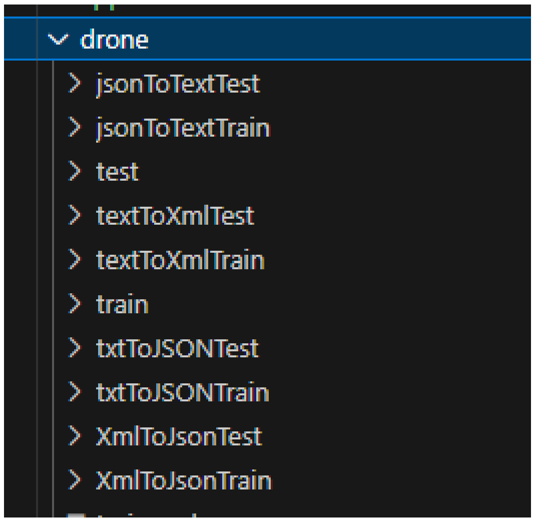
Maintaining a well-structured and organized codebase was a priority throughout the experiment. We established a clear file organization scheme, dedicating separate folders for each dataset and conversion type. To enhance code portability and avoid hardcoding file paths, we employed the python-dotenv 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.



```
⌄ appledataset
  > jsonToTextTest
  > jsonToTextTrain
  > jsonToXmlTest
  > jsonToXmlTrain
  > test
  > textToXmlTest
  > textToXmlTrain
  > train
  > txtToJSONTest
  > txtToJSONTrain
  > XmlToJsonTest
  > XmlToJsonTrain
  > xmlToTxtTest
  > xmlToTxtTrain
```



```
⌄ drone
  > jsonToTextTest
  > jsonToTextTrain
  > test
  > textToXmlTest
  > textToXmlTrain
  > train
  > txtToJSONTest
  > txtToJSONTrain
  > XmlToJsonTest
  > XmlToJsonTrain
```

Report score: ___________

Instructor's signature: ___________