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:** | TWO | | **DATE:** | 9th 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 Pytorch in your devices.
* How to use Yolo detection object by Pytorch.
* How to install PaddlePaddle in your devices.
* How to install TensorFlow in your devices.

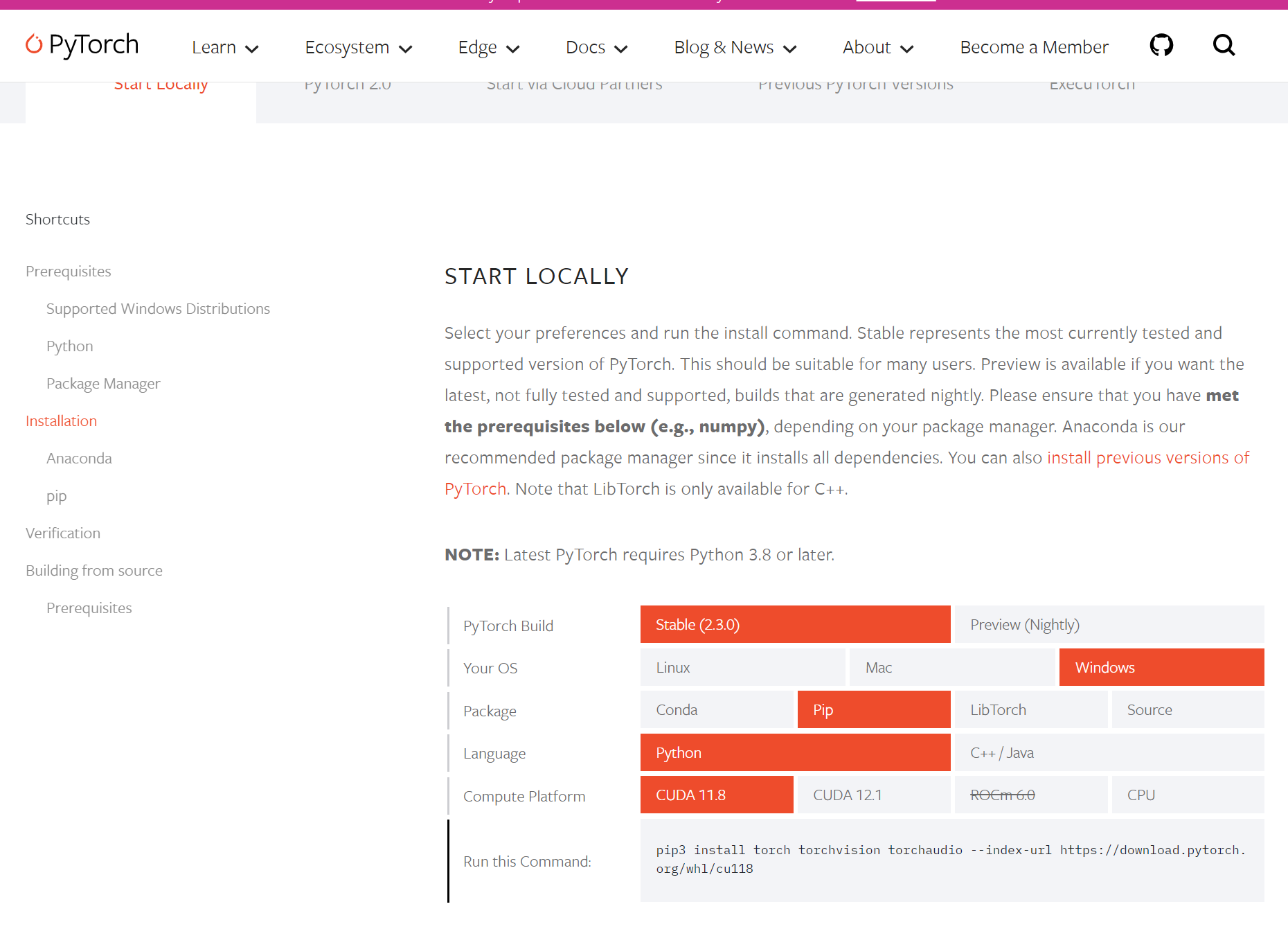
1. Experimental contents & step
2. Installing the Pytorch for Windows
3. Install PaddlePaddle for Windows
4. Install TensorFlow for Windows
5. Create Yolo detection object by Pytorch
6. other

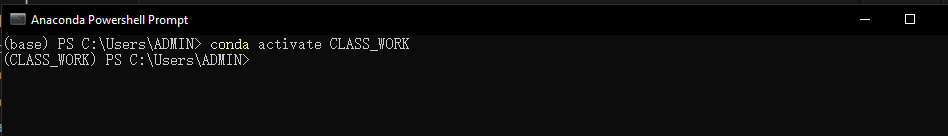
# Installing the Pytorch for Windows

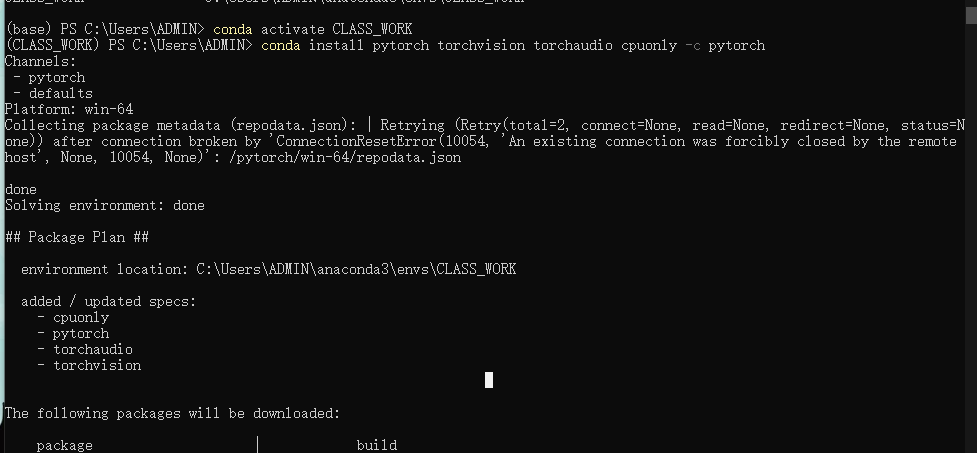
Eager to dive into the world of PyTorch, I installed it on my Windows machine. Here's a quick rundown of my installation process:

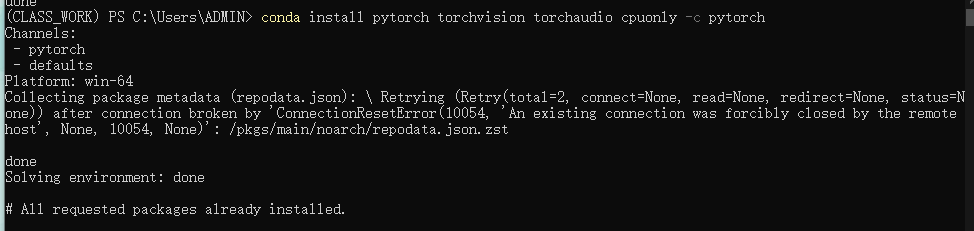
* **Step 1**: I accessed the official PyTorch website at *https://pytorch.org/get-started/locally/* for installation instructions.
* **Step 2**: Using Anaconda Prompt, I activated my "CLASS\_WORK" environment: conda activate CLASS\_WORK.
* **Step 3**: I installed PyTorch, torchvision, and torchaudio for CPU-only operations: *conda install pytorch torchvision torchaudio cpuonly -c pytorch.*
* **Step 4**: I confirmed the installation by creating a Python file with import torch, which ran without errors, signaling a successful installation!

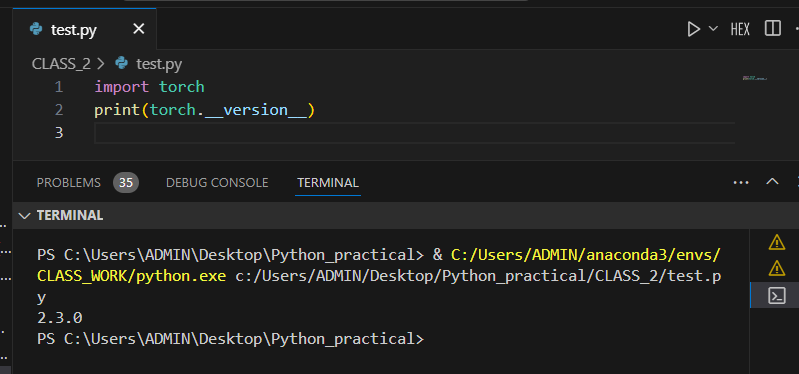
Now, I'm all set to explore the power of PyTorch!











# Install PaddlePaddle for Windows

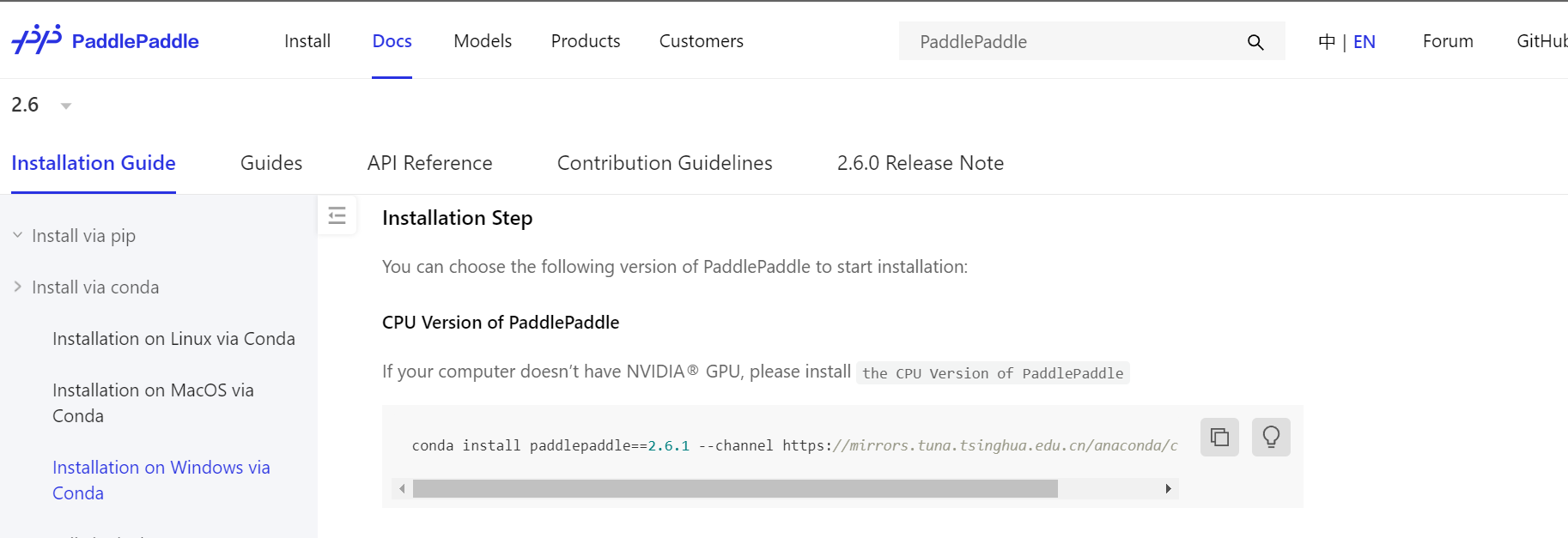
To get started with PaddlePaddle, I followed these installation steps:

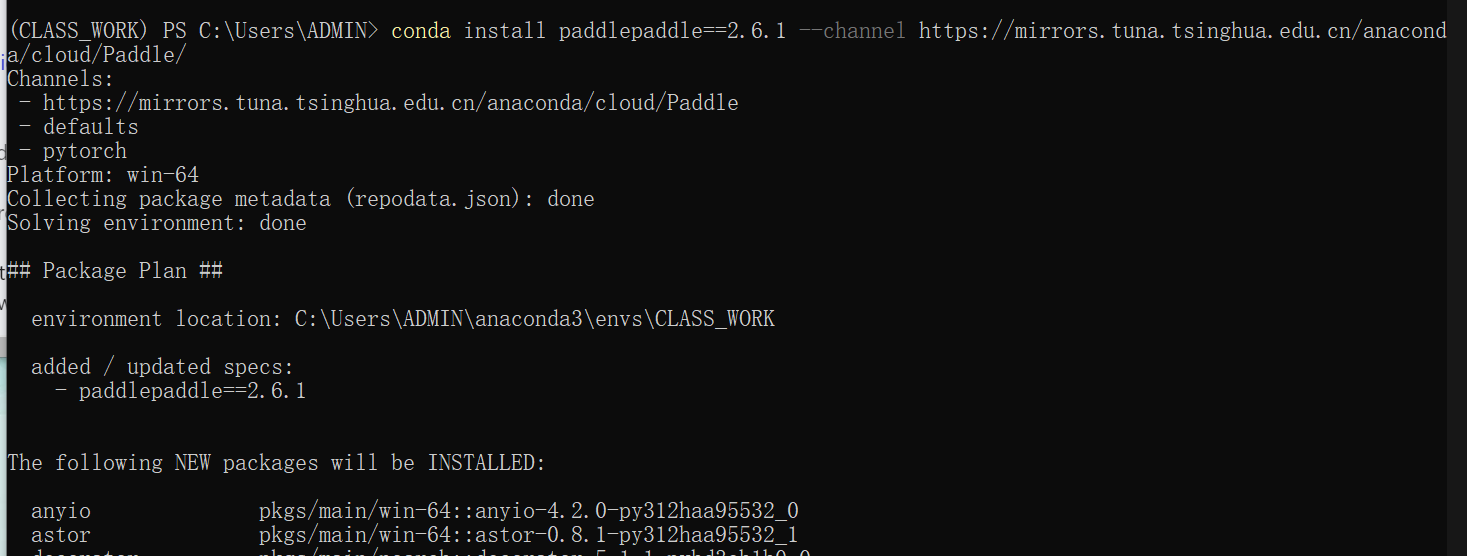
* **Step 1**: I referred to the PaddlePaddle documentation at *https://www.paddlepaddle.org.cn/documentation/docs/en/install/conda/windows-conda\_en.html* for Windows-specific instructions.
* **Step 2**: I activated my Anaconda "CLASS\_WORK" environment: conda activate CLASS\_WORK.
* **Step 3**: Using the Tsinghua mirror channel, I installed PaddlePaddle:

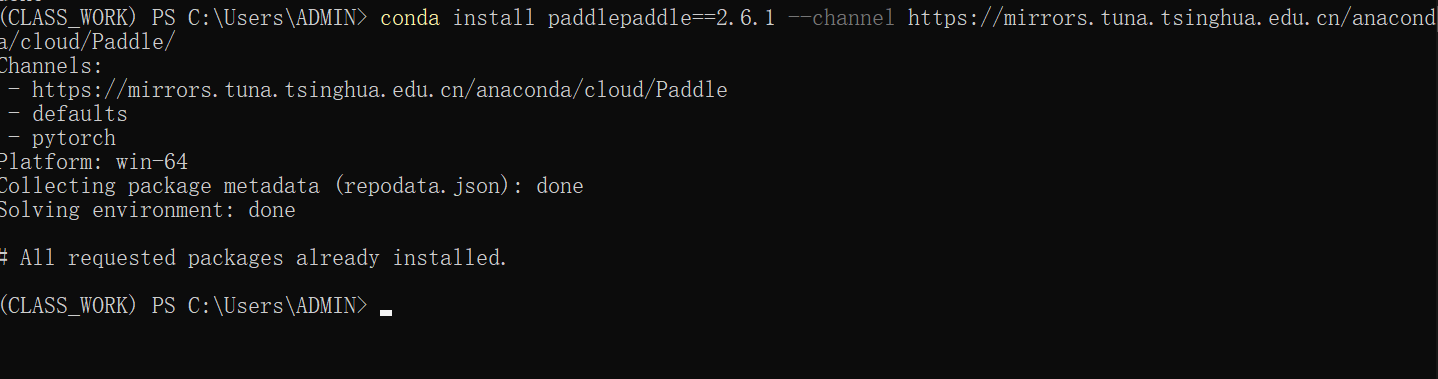
*conda install paddlepaddle==2.6.1 –channel* [*https://mirrors.tuna.tsinghua.edu.cn/anaconda/cloud/Paddle/*](https://mirrors.tuna.tsinghua.edu.cn/anaconda/cloud/Paddle/).

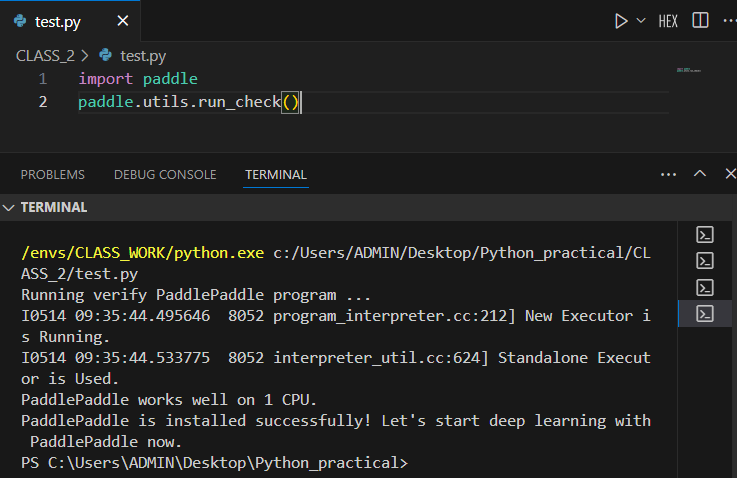
* **Step 4**: To confirm, I created a Python file with import paddlepaddle. The import worked flawlessly, validating my successful PaddlePaddle installation!

Now I'm ready to explore the capabilities of PaddlePaddle!







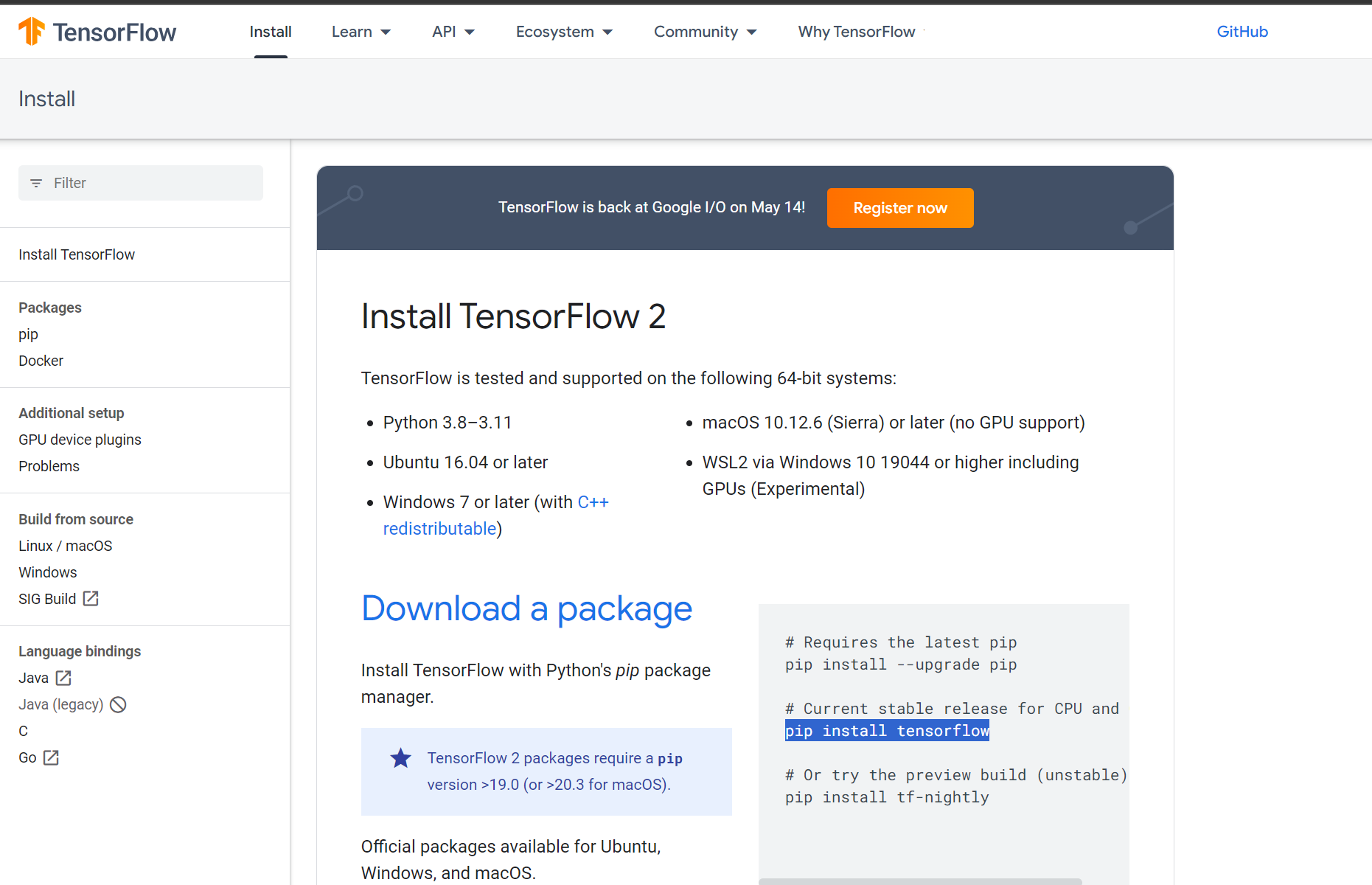


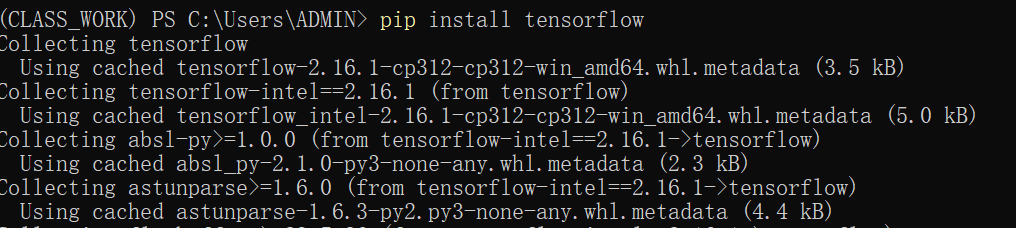
# Install TensorFlow for Windows

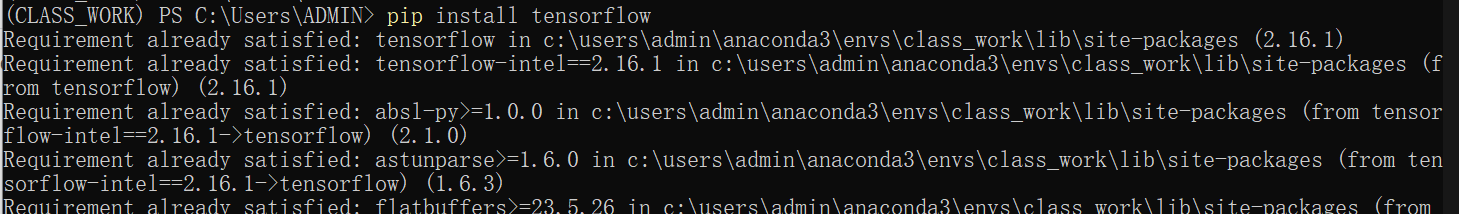
Here's how I installed TensorFlow on my Windows computer:

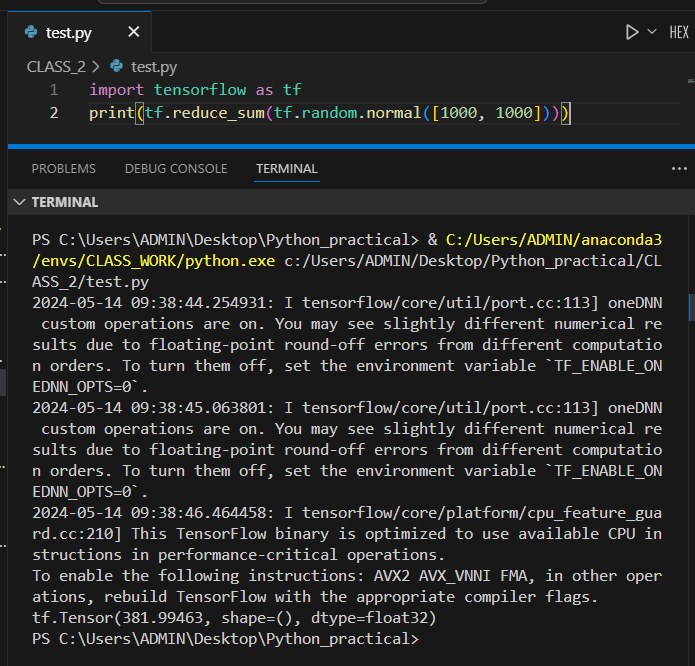
* **Step 1**: I visited the official TensorFlow installation guide: *https://www.tensorflow.org/install* for detailed instructions.
* **Step 2**: In my Anaconda Prompt, I activated my "CLASS\_WORK" environment: conda activate CLASS\_WORK.
* **Step 3**: I used pip to install TensorFlow*: pip install tensorflow*.
* **Step 4**: I verified the installation by creating a Python file and importing TensorFlow: import tensorflow. The successful import confirmed that TensorFlow was installed correctly!

Now I'm ready to start using TensorFlow for my machine learning projects!









# Create Yolo detection object by Pytorch

Here's how I built a YOLO object detection model using PyTorch:

* **Step 1**: Install Ultralytics YOLOv8

I began by navigating to the Ultralytics website: *https://docs.ultralytics.com/quickstart/#install-ultralytics* for installation instructions. Then, within my activated "CLASS\_WORK" environment in the Anaconda Prompt, I installed the Ultralytics library using pip:

*pip install ultralytics*

* **Step 2**: Initialize the YOLO model

In my preferred IDE, I started by importing the YOLO class from the Ultralytics package:

*from ultralytics import YOLO*

Next, I initialized a new YOLO model, choosing the yolov8n.yaml configuration for its balance of speed and accuracy:

*model = YOLO('yolov8n.yaml') # build a new model from YAML*

* **Step 3**: Prepare the Drone Image Dataset

I organized my dataset of drone images and created a YAML file (drone.yaml) defining the dataset's structure and classes. I placed this file within my project directory.

* **Step 4**: Train the YOLO Model

With the dataset ready, I defined the dataset path variable:

*#DATASET\_PATH*

*PATH = 'C:\\Users\\ADMIN\\Desktop\\Python\_practical\\pythonpa\\dataset\\drone.yaml'*

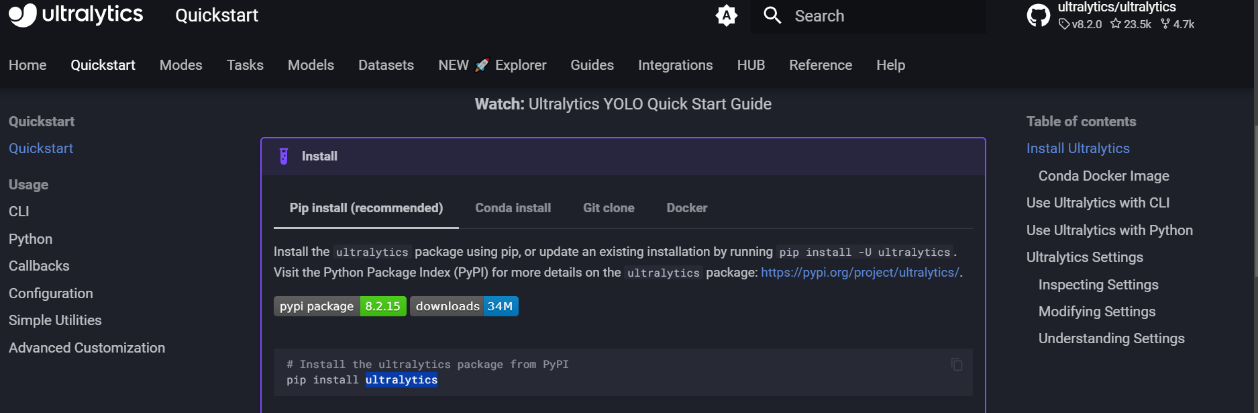
Then, I used the model.train() function to train my YOLO model on the drone image dataset:

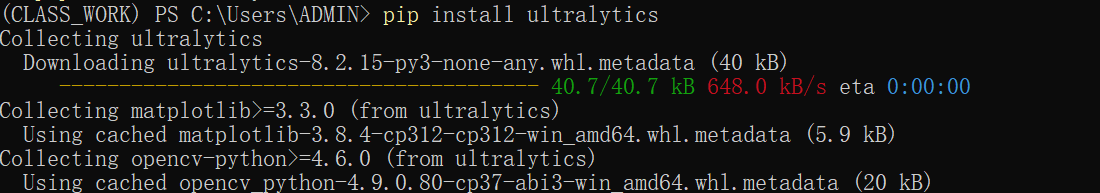
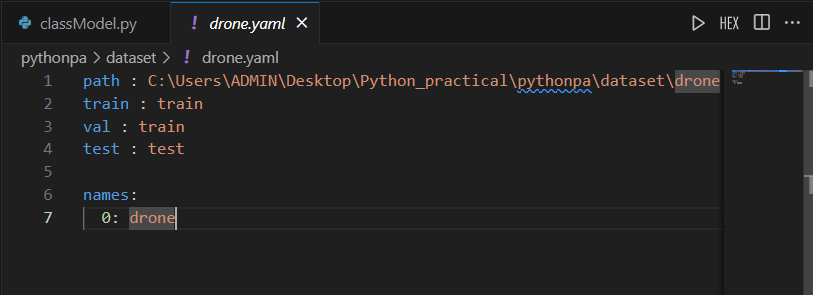
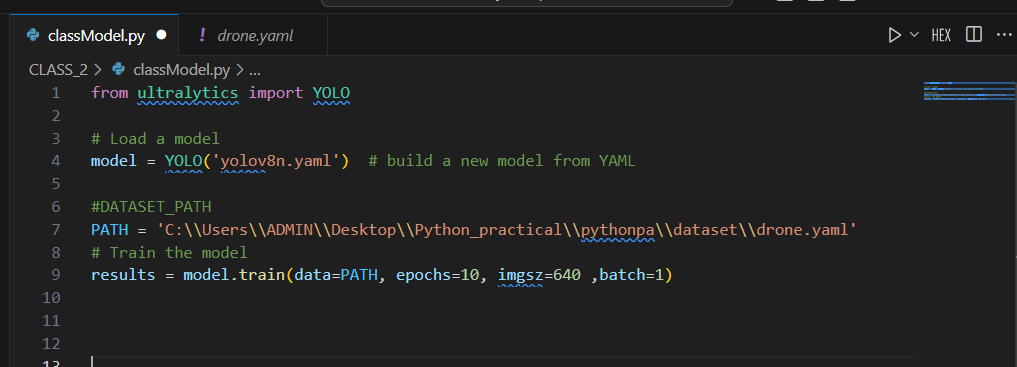
*# Train the model*

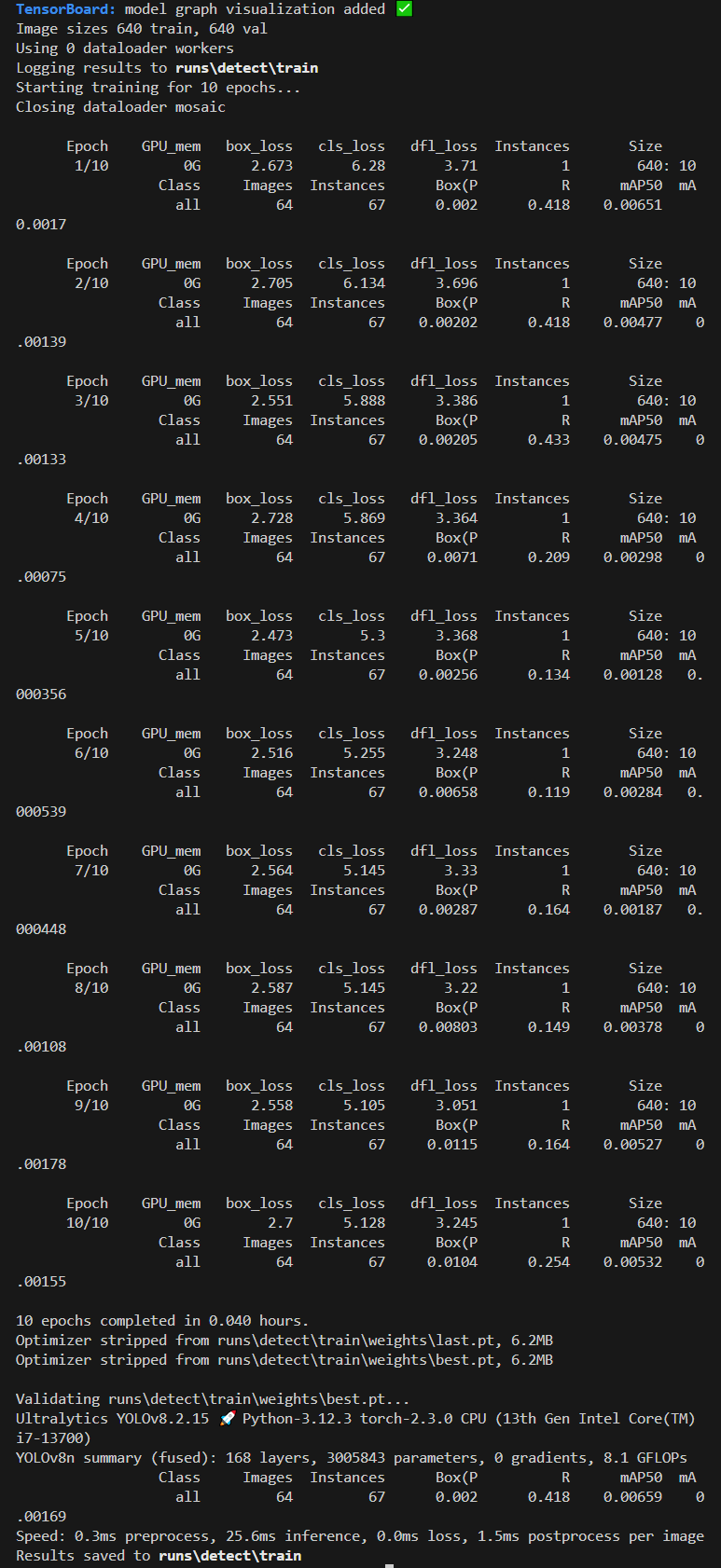
*results = model.train(data=PATH, epochs=10, imgsz=640, batch=1)*

I set the training parameters to 10 epochs, an image size of 640 pixels, and a batch size of 1.

* **Step 5**: Model Training and Results

Running this code successfully trained my YOLO model on the drone image dataset. I obtained satisfactory results with the trained model successfully detecting objects within the drone images.





# Use Different Yolo Model With Different Batch

To explore the impact of different YOLO models and batch sizes, I modified my previous code as follows:

* **Step 1**: Load a Pretrained YOLO Model

Instead of building a new model from scratch, I loaded a pretrained yolov8n model using:

*model = YOLO('yolov8n.yaml').load('yolov8n.pt')*

This allowed me to leverage a model that was already trained on a vast dataset.

* **Step 2**: Adjust the Batch Size

In the training parameters, I increased the batch size to 2:

*#DATASET\_PATH*

*PATH = 'C:\\Users\\ADMIN\\Desktop\\Python\_practical\\pythonpa\\dataset\\drone.yaml'*

*# Train the model*

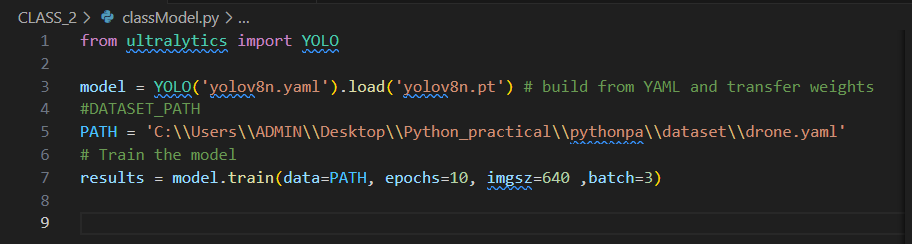
*results = model.train(data=PATH, epochs=10, imgsz=640, batch=2)*

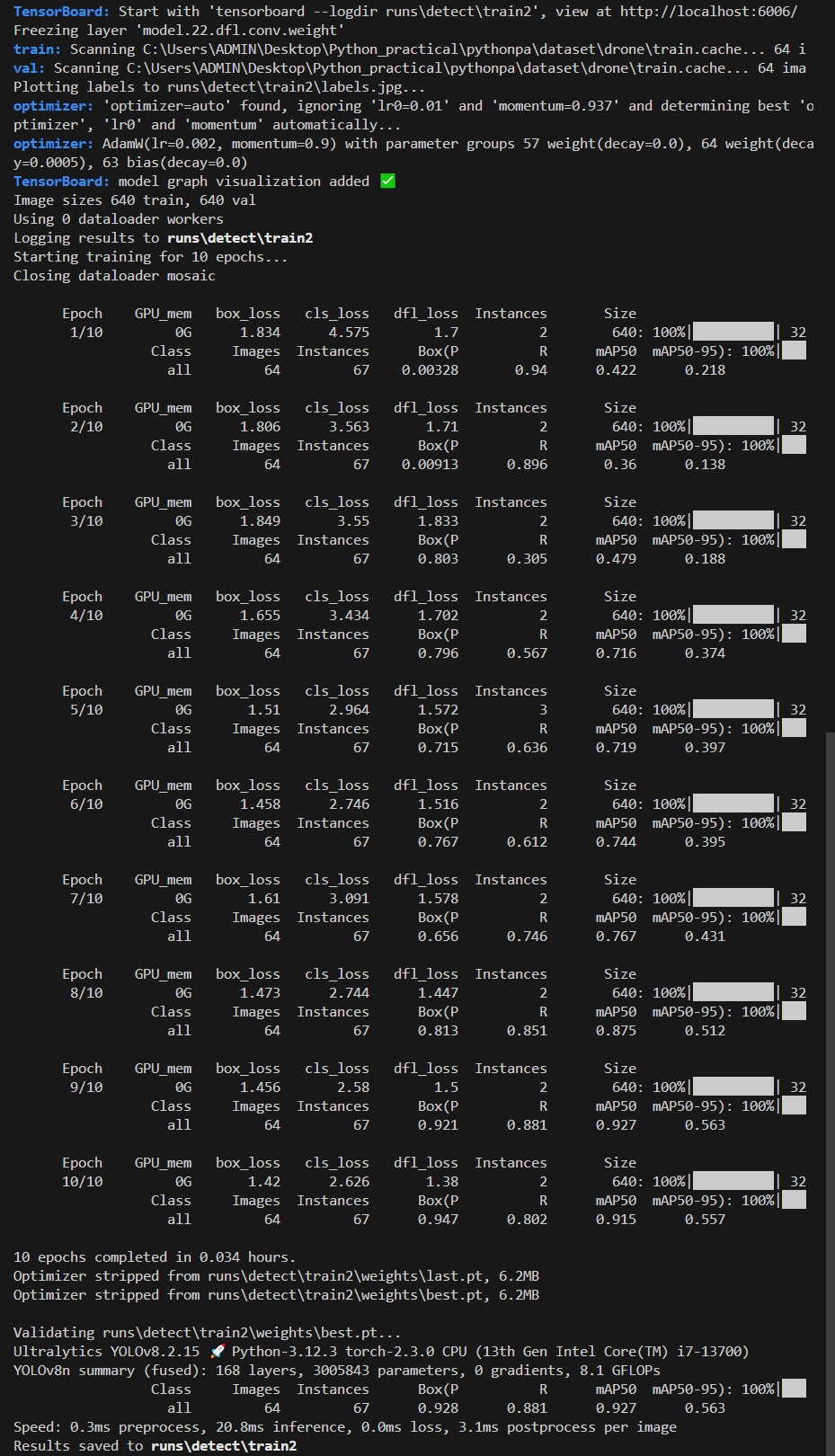
This modification allowed me to process two images simultaneously during each training iteration.

* **Step 3**: Experiment with Different Models and Batch Sizes

By changing the model YAML file (yolov8n.yaml, yolov8s.yaml, etc.) and adjusting the batch size, I could easily experiment with different configurations. This allowed me to observe the impact on training speed, memory usage, and overall model performance.

Through these modifications, I gained insights into the tradeoffs between different YOLO model architectures and the effects of varying batch sizes on the training process.





# Experimental analysis

This experimental focuses on exploring the capabilities of the YOLOv8 object detection model for identifying objects within drone imagery. The research involved a series of steps: setting up the necessary software environment, installing the required libraries, building and training YOLOv8 models, and experimenting with different model configurations and batch sizes.

The initial phase focused on establishing a robust environment for deep learning on a Windows machine. This involved installing popular deep learning frameworks like PyTorch, PaddlePaddle, and TensorFlow. Leveraging the Anaconda platform, a dedicated "CLASS\_WORK" environment was created to manage dependencies effectively. Each framework was installed using either conda or pip, ensuring a smooth and successful installation process.

Next, the Ultralytics library, which provides a user-friendly implementation of the YOLOv8 model, was installed using pip within the "CLASS\_WORK" environment. This library simplifies the process of building, training, and evaluating YOLOv8 models. A dataset consisting of drone imagery was curated and organized, with a corresponding YAML file defining its structure and object classes.

Using the Ultralytics library, a new YOLOv8 model was initialized using the 'yolov8n.yaml' configuration file, selected for its balance between speed and accuracy. The model was then trained on the prepared drone image dataset for 10 epochs, employing a batch size of 1 and an image size of 640 pixels. This initial training process yielded positive results, with the model successfully detecting objects within the drone images.

To further understand the influence of model configurations and training parameters on performance, the experiment continued by loading a pre-trained 'yolov8n' model. This model, having been previously trained on a large dataset, offered a baseline for comparison. Additionally, the batch size was increased to 2 during training to assess its impact on speed, memory usage, and model accuracy.

The experiment revealed valuable insights into the behavior of YOLOv8 for object detection in drone imagery. The use of pre-trained models and the adjustment of batch size demonstrably impacted training dynamics. However, a more comprehensive analysis is needed to quantify these effects and determine the optimal configuration for this specific task.

Future research will delve deeper into comparing different YOLOv8 model configurations, such as 'yolov8s.yaml' and 'yolov8m.yaml', to identify the most suitable architecture for drone image analysis. A systematic study of various batch sizes will be conducted to establish the ideal balance between training speed, memory efficiency, and model accuracy. Finally, rigorous performance evaluation will be performed using quantitative metrics like precision, recall, and F1-score to objectively assess the effectiveness of different model configurations and batch sizes.

This experimental analysis lays the groundwork for optimizing YOLOv8 models for specific object detection tasks in drone imagery. The findings will guide future research and development, ultimately enhancing the performance and reliability of object detection in this domain.

Report score: .

Instructor's signature: .